MASTERARBEIT

Entwicklung einer netzwerkfähigen Steuereinheit für digitale Spiegelreflexkameras

ausgeführt am
Institut für Elektronik
der Technischen Universität Graz
Leiter: Univ.-Prof. DI Dr.techn. Wolfgang Bösch

unter Anleitung von Ass.Prof. Dipl.-Ing. Dr.techn. Stöckler Gerhard

durch

Nikolaus Schicker
Matr.-Nr. 0430585

im
SS 2012

Eidesstattliche Erklärung

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

Graz, am ....
(Unterschrift)

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitely marked all material which has been quotes either literally or by content from the used sources.

....
date (signature)

Zusammenfassung

In dieser Arbeit wird der Entwurf einer netzwerkfähigen Steuereinheit für digitale Spiegelreflexkamera behandelt. Der Entwurf beginnt mit der Spezifikation und dem Konzept und endet mit dem fertigen Prototyp. Die benötigte Hard- und Software wurde selbstentwickelt und dokumentiert. Ausgelegt ist es für Kameras der Modellserie EOS der Firma Canon, es kann aber durch kleine Anpassungen auch für Kameras anderer Hersteller verwendet werden. Das entwickelte System wird dabei über Bluetooth oder das Ethernet konfiguriert und gesteuert. Bei einer vorherigen Konfiguration ist es auch in der Lage ohne Netzwerk selbständig Steuerungen vorzunehmen. Die Steuerungssoftware wurde für Android und Windows entwickelt. Zur Steuerung der Kamera wird der 2,5 mm Klinkenbuchsen- und der USB Eingang verwendet, womit z.B. auch die Belichtung, Blende und Verschlusszeit der Kamera geändert werden kann. Dafür wurde das von der Kamera verwendete Picture Transfer Protocol und die zuvor eruierten spezifischen Anweisungen und Parameter der Firma Canon in einem Vinculum 2 Host Controller implementiert. Durch das Vorhandensein von 2 USB Anschlüssen können bis zu zwei Kameras in das System integriert werden. Als zentraler Controller für die restlichen Module kommen ATmegas der Firma Atmel zum Einsatz. Die Versorgung des Systems erfolgt entweder über eine 9V Blockbatterie oder mit Hilfe von Power over Ethernet. Zum Einsatz kommen hierbei PoE Module der Firma Silvertel. Das aus dieser Arbeit resultierende offene Steuerungssystem kann um beliebige Sensoren und Aktoren erweitert werden.

Abstract

This work is about developing a network compatible digital single lens reflex camera system. The design starts with specifications and the proof of the concept and ends with a prototype. All the necessary hard- and software will be discussed in this document. It is designed for Canon cameras but with small adjustments it could also be used for cameras of other brands. The configuration of this system could be done over Bluetooth or the Ethernet with the self-developed software for Windows and Android. A once configured system could be used as a stand-alone solution. For controlling the camera the 2.5 mm jack plug and the USB connector is used. With the last connector it is possible to change the exposure, shutter speed and aperture of the used camera. For doing this it was necessary to implement the used Picture Transfer Protocol with the special used commands and parameters from Canon on a Vinculum 2 USB host controller. This controller has two USB ports for interaction with two DSLR cameras. The ATmega controller from Atmel is used for the other developed parts of the system. As power supply a 9 V battery or a power over Ethernet module from Silvertel could be used. The resulted open system with the defined in/outputs could be used with any sensor or actuator.

Danksagung

An dieser Stelle möchte ich bei einigen Personen und Firmen für ihre Unterstützung bei der Erstellung dieser Arbeit bedanken. Vielen Dank an meinen Betreuter Dipl.-Ing. Dr.techn. Stöckler Gerhard vom Institut für Elektronik für die Unterstützung und das Sponsern der Platinen. Für die PoE Module und den raschen Versand derselben, bedanke ich mich beim Herrn Steve Edwards (Managing Director, Silvertel) und Herrn Michael Schrutka (Produktmanager, Codico). Für das Bestellen und vor allem der Bezahlung der verwendeten Bauteile, bedanke ich mich beim Ing. Bernhard Rath von der Firma COPY RATH. Zum Schluss bedanke ich mich noch bei Suzy Cohan von der Canon Foundation für ihre Hilfe bei den von Canon implementierten PTP Befehlen und ihr Bemühen einen geeigneten Ansprechpartner innerhalb von Canon zu finden.

Inhaltsverzeichnis

1 Einleitung

Die Idee für diese Arbeit entstand aus der Verknüpfung der Hobbybereiche Fotografie und Mikrocontrollersysteme. Eine moderne Spiegelreflexkamera besitzt eine Fülle von Einsatz- und Einstellungsmöglichkeiten, die nur selten von einem Anwender wirklich ausgereizt werden. Es kommt auch vor, dass die vom Hersteller zur Verfügung gestellte Firmware auf der Kamera nicht ausreicht um alle Bedürfnisse des Endkunden zu erfüllen. Das kann zum Beispiel eine fehlende Einstellmöglichkeit in der Software, eine fehlende Interaktion mit der Kamera oder eine schlicht unkomfortable Bedienung sein. Das hier gezeigt Steuerungssystem hat das Ziel diese Schwächen zu beseitigen und eine Schnittstelle für beliebige weitere externe Erweiterungen bereitzustellen. Um das zu erreichen wird die Spiegelreflexkamera in ein System bestehend aus aktuellen Technologien integriert, womit sich eine Fülle neuer Anwendungsmöglichkeiten ergibt. Zwar gibt es auf dem Markt bereits Produkte, die Teilbereiche dieser Arbeit abdecken, doch diese sind in sich geschlossen und somit nicht erweiterbar.
In den letzten Jahren hat sich für die Interaktion des Benutzers mit einem Embedded System ein neuer Trend entwickelt. Sehr beliebt ist die Verwendung eines Smartphones oder Tablet Rechner mit dem Betriebssystem Android zur Bereitstellung von Informationen und Einstellmöglichkeiten in einer eigens dafür geschriebenen Anwendung. Die Daten werden dabei über die Schnittstellen des Android Gerätes (USB, Bluetooth, WiFi, etc.) mit dem Embedded System ausgetauscht. Unterstützt wird dieser Trend durch die weite Verbreitung der Geräte und die teilweise geringen Beschaffungskosten, sowie durch eine gute Dokumentation der benötigten API Befehle des Betriebssystems und eine einfache Programmierung in einer freien Entwicklungsumgebung. Ein weiterer Vorteil ist die Verringerung der Gerätekosten durch das Wegfallen der Bauteile für die Ausgabe (z.B LCD) und Eingabe (z.B. Taster). Die Kosten für die zusätzlichen benötigten Schnittstellenbauteile, betragen üblicherweise nur ein Bruchteil des Einkaufpreises der eingesparten Bauteile. Ein Smartphone bietet außerdem die Möglichkeit das System mit einem geringen Aufwand um weitere Funktionen wie z.B. GSM, Webserver, etc. zu erweitern.
In dieser Arbeit wird zusätzlich zur Steuerungsmöglichkeit über Android mittels Bluetooth, ein Windows Programm entwickelt, das über Ethernet mit dem Embedded System kommunizieren kann. Als Besonderheit kann die Schaltung zusätzlich über Ethernet mit dem der Hilfe von PoE versorgt werden, womit ein eigenes Netzteil überflüssig wird.

Diese Masterarbeit ist in die Stationen einer Produktentwicklung untergliedert und um einen Theoriebereich und eine Projektauswertung erweitert. Im Theorieteil befinden sich genauere Informationen zum theoretischen Hintergrund. Die Projektauswertung beinhaltet ein Resümee, die entstandenen Kosten und einen Ausblick auf mögliche zukünftige Erweiterungen.
In Abbildung 1.1 ist der Prozess einer Produktentwicklung ersichtlich, den jedes elektronische Gerät durchläuft. Ausgangspunkt eines jeden Produktes ist eine Idee. Ist diese gefunden, werden alle Anforderungen für ihre Umsetzung definiert, was als Spezifikation bezeichnet wird. Gleichzeitig wird auch ein Konzept erstellt, um die technische Realisierbarkeit zu überprüfen.
Das darauffolgende Glied ist die Schaltungsumsetzung, in der die elektronischen Bauteile in einem Schaltplan miteinander verbunden werden. Dies geschieht mit sogenannten EDA-Tools (Bsp. Orcad, Eagle), in denen meistens auch anhand des eingegebenen Schaltplans die Schaltungsumsetzung durch Positionieren der Bauteile auf der Platine und Routen der Leiterbahnen entsteht.

Zur Fertigung gehören die Erzeugung und die Bestückung der Platine. Zum Schluss erfolgt die Inbetriebnahme, in der die Funktionalität der Schaltung überprüft wird. Werden dabei alle Anforderungen erfüllt, ist das Produkt fertig entwickelt.
Die Verifikation begleitet den ganzen Entwicklungsprozess und stellt sicher, dass in jeder Station die Übereinstimmung mit den Spezifikationen garantiert ist. Treten Abweichungen auf, springt man in jene Station zurück, in der man den Fehler korrigieren kann. Im schlimmsten Fall muss man die Spezifikationen bzw. das Konzept ändern und den Prozess von vorne beginnen. Es ist wichtig so viel Zeit wie möglich in die Verifikation zu investieren, denn je früher ein Fehler entdeckt wird, desto leichter ist seine Beseitigung und desto weniger Zeit beansprucht die Korrektur [Winzker M. 2008].

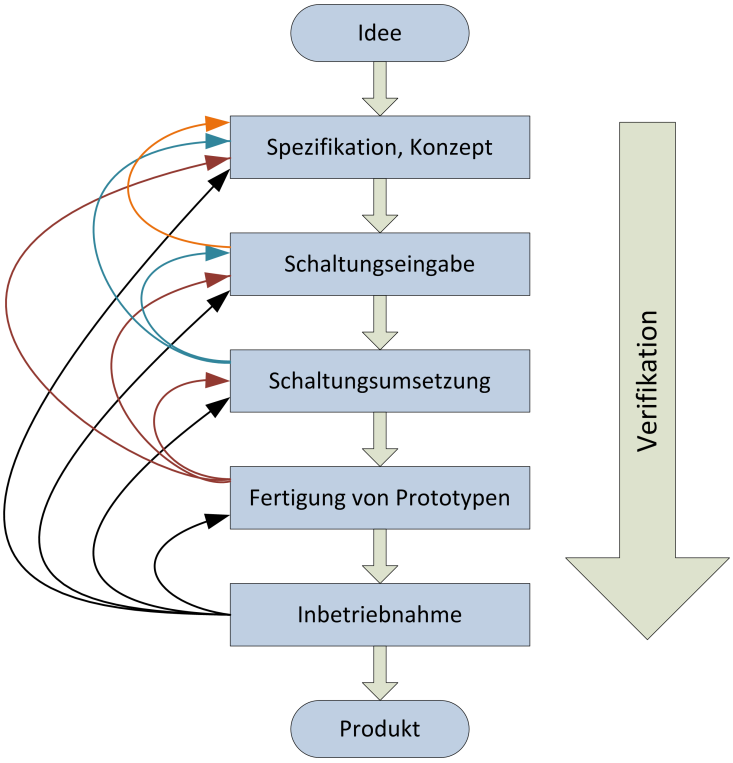


Abbildung 1.1: Ablauf der Produktentwicklung

2 Spezifikation und Konzept

2.1 Spezifikation

Das zu entwerfende netzwerkbasierende Sensor- und Steuerungssystem sollte die vielseitigen Verwendungsmöglichkeiten moderner Spiegelreflexkameras ausnützen um verschiedene Aufgabenstellungen zu realisieren. Die Anforderungen sind:

- Software gesteuerte Fernauslösung mehrerer Kameras
- Bewegungsgesteuerte Aufnahmen
- Machbarkeit eines 3D Farbscanners
- 360° momentan Fotos mit Hilfe mehrerer Kameras
- Ermöglichung von Zeitrafferaufnahmen
- Aufnahme von 3D und 3D HDR Bildern
- Anschlussmöglichkeit von externen Sensoren

Ziel dieser Arbeit ist die Entwicklung der dazugehörigen Schaltungen für die Realisierung der Anforderungen inklusive der benötigten Software. Als Spiegelreflexkamera soll das Modell Canon EOS 500D verwendet werden.

2.2 Konzept

Mit Hilfe des in der Abbildung 2.1 ersichtlichen Konzeptes, sollte es möglich sein alle notwendigen Aufgabenstellungen zu realisieren. Das System besteht aus mehreren Modulen, die jeweils einer von 6 Klassen angehören und grob in die 4 Typen Eingabe, Steuereinheit, Aktor und Sensor unterteilt werden können. Mit steigender Klassennummer erhöhen sich üblicherweise auch die Stückkosten eines Moduls, weil mehr Bauteile benötigt werden. Das hängt natürlich auch von den Stückkosten der verwendeten Bauteile ab. Im nachfolgenden Teil wird beginnend bei den niedrigsten Klassen auf die Eigenschaften der einzelnen Module eingegangen.
Sogenannte Klasse I und Klasse II Module können abhängig von ihrer Funktion dem Typ Aktor oder Sensor zugeordnet werden. Sensoren wandeln physikalische oder chemische Größen in ein elektrisches Signal um und stellen diese Information den höheren Klassen zur Verfügung (z.B. Temperatursensor, Lichtschranke, Geräuschsensor, etc.). Aktoren bekommen Informationen von den höheren Klassen und führen anhand dessen Aktionen aus (z.B. Schalter schließen, Kamera auslösen, etc.). Die Unterscheidung zwischen den Klassen I und II erfolgt durch die Art des Bezugs bzw. der Bereitstellung der Information durch die Module. Für die Klasse I steht nur eine Leitung für die Interaktion bereit, während die Klasse II die Informationen über die TWI Schnittstelle (Atmels Äquivalent zu I^2C) austauscht. Dadurch können mehrere Module über einen Bus miteinander verbunden werden. Die Spannungsversorgung kann dabei über ein Modul einer höheren Klasse erfolgen, was zumindest für Klasse I Module nicht zwingend der Fall ist. Bei eigener Spannungsversorgung ist jedoch auf die Stromkreistrennung zu achten.
Klasse III und IV Module gehören zum Typ Steuereinheit und verarbeiten die Informationen von den Sensoren oder geben Aufträge an die Aktoren weiter. Dafür wird ein μC der Firma Atmel verwendet, dessen Interrupt Pins ideal für Ereignisse benutzt werden

können, die eine zeitnahe Bearbeitung erfordern. Durch den integrierten Analog Digital Konverter ist er auch in der Lage, eine größere Anzahl von unterschiedlichen Klasse I Modulen zu unterstützen. Es besteht auch die Möglichkeit die normalen Ein-/Ausgabe-Pins zu benutzen. Durch das Hinzufügen der Netzwerkfähigkeit, wird aus einem Klasse III ein Klasse IV Modul und es können Daten mit der Umgebung ausgetauscht werden. Der Austausch der Daten findet hierbei entweder über Ethernet oder Bluetooth statt. Der Vorteil von Ethernet ist, dass gleichzeitig die Spannungsversorgung über PoE für das Modul erfolgen kann und somit keine eigene Spannungsversorgung notwendig ist.
Ein Klasse V Modul ist ebenfalls vom Typ Steuereinheit und hat die Besonderheit, dass zwei USB Anschlüsse für die Interaktion mit einer Canon EOS 500D Kamera vorhanden sind. Über diesen Anschluss können die Einstellungen an der Kamera geändert werden. Module der Klasse I und II können direkt an ein Modul der Klasse V angeschlossen werden, wodurch schnell auf Ereignisse reagiert werden kann. Die Kommunikation mit anderen Modulen findet über die Ethernet oder Bluetooth Schnittstelle statt. Ein Klasse V Modul besitzt wie die Klasse III oder IV eine eigene Spannungsversorgung oder nutzt das PoE der Ethernet Verbindung.
Bei der Klasse VI handelt es sich um eine Software, die auf einer Fremdhersteller Hardware läuft und mit den darunterliegenden netzwerkfähigen Klasse Modulen interagieren kann. Sie entspricht dem Typ Eingabe. Um die Software verwenden zu können, muss die Hardware über eine Bluetooth oder Ethernet Verbindung verfügen. Mit Hilfe dieser Software werden Einstellungen am System vorgenommen z.B. wann eine Aktion durchzuführen ist oder Werte an der Kamera geändert. Zusätzlich können Daten vom vorhandenen Netzwerk ausgelesen und dem Benutzer zur Verfügung gestellt werden. Als Beispiel wird diese Software für die 2 verschiedenen Betriebssysteme Windows (Desktop PC) und Android (Tablet, Smartphone) implementiert, wobei bei jedem eine andere Art der Verbindungsmöglichkeit benutzt wird.
Die Anzahl der jeweils vorkommenden Klassenmodule ist nur durch die physischen Ressourcen der höheren Klassenmodule (z.B. verfügbare Port Pins, Spannungsversorgung) oder durch die Netzwerkeigenschaften (z.B. maximale TWI Teilnehmer, maximale Anzahl Piconetz Teilnehmer bei Bluetooth) beschränkt.

2.3 Verifikation

2.3.1 Fernauslöser EOS 500D

Die EOS 500D verfügt über eine 2,5 mm Stereo Klinkenbuchse, mit deren Hilfe die Kamera fernausgelöst werden kann. Das Anschlussbild ist in der Abbildung 2.2 b zu sehen.

Durch verbinden des Focus- mit dem GND-Anschluss wird das Bild fokussiert, was dem halb gedrückten Auslöseknopf entspricht. Durch schließen des Kontaktes vom Trigger und GND wird die Aufnahme ausgelöst (Auslöseknopf voll durchgedrückt).
Messungen haben ergeben, dass zwischen Trigger und GND bzw. zwischen Focus und GND jeweils 2,987 V anliegen und beim Schließen jeweils ein Strom von 68 μA fließt.

2.3.2 Zugriff auf die EOS 500D

SDK von Canon
Canon stellt für die Integrierung ihrer Produkte in andere Projekte eine umfangreiche Sammlung an Informationen (Quellcodes, Beschreibungen, Beispielen, etc.) für Mitglieder des Digital Imaging Developer Programms zur Verfügung (Registrierung erforderlich). Für die DSLR Kameras der EOS Serie ist aktuell die SDK v2.10 zum Download erhält-

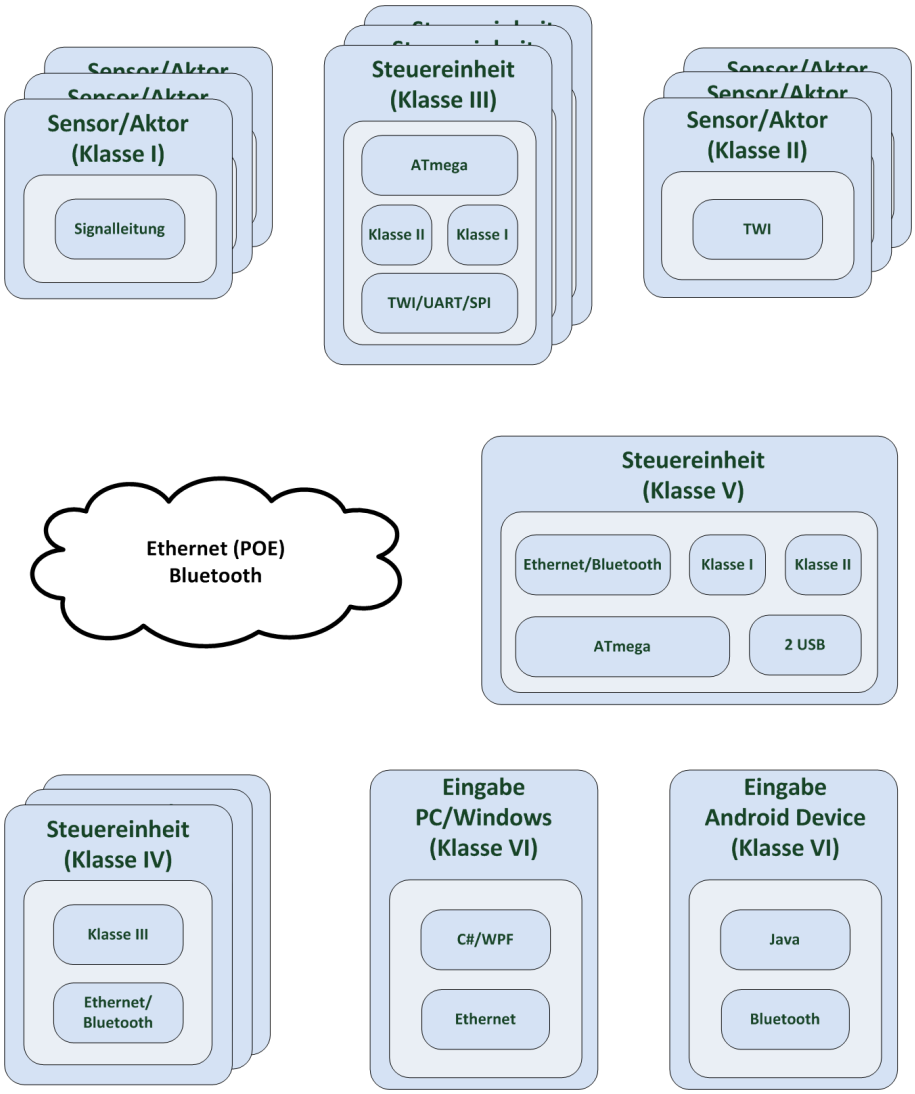
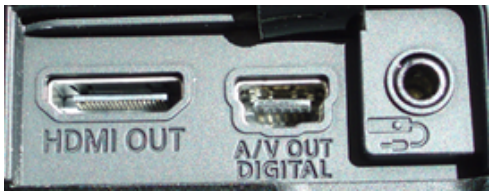


Abbildung 2.1: Blockdiagramm des Konzeptes

lich, womit von einer selbstgeschriebenen Anwendung auf die Kamera zugegriffen werden kann. In der SDK befinden sich die benötigten DLL Dateien, eine Anleitung über den Kommunikationsablauf und die verwendbaren API Befehle und Beispielanwendungen in den Sprachen Visual Basic, Visual C und C#.
Leider wurde für die Sprache C# keine fertige Anwendung bereitgestellt, sondern nur die Klasse „EDSDK.cs“ mit den Konstanten- und den Prototypdefinitionen der API Aufrufe. Diese Klasse muss man in die eigene Anwendung integrieren und die benötigten DLL's von der SDK in den Ausführordner kopieren. Für die Erstellung einer C# Anwendung empfiehlt es sich die Beispielanwendung „CameraControl“ im Visual C Ordner genauer zu betrachten und die Dokumentation „EDSDK_API“ zu lesen. Darin ist auch zu lesen, dass für die Kommunikation mit dem PC das PTP Protokoll verwendet wird, dessen Beschreibung unter 8.9 nachzulesen ist.
Um sich einen Überblick über die vorhandenen Geräteeigenschaften zu verschaffen, wurde der DeviceInfo Datensatz mit dem GetDeviceInfo (Operation Code 0x1001) Befehl ausgelesen. Das komplette USB Protokoll der Datenphase mit einer Auswertung befindet sich im Anhang E. Man erkennt, dass die EOS 500D sehr viele herstellerspezifischen Erweiterungen verwendet (MSN des Operation Codes ist 1001). Um die Bedeutung der einzelnen Operation Codes herauszufinden wurde bei Canon nachgefragt, jedoch konnte sie keine



(a) Schnittstellen der EOS 500D

(b) Anschlussbild des 2,5 mm Klinkensteckers für den Fernausgang [Elektor Oktober 2010]

Abbildung 2.2: Verifikation der Fernauslösung einer EOS 500D

Definition der Erweiterungen bereitgestellt werden. Deshalb musste der Weg des „Reverse Engineering“ gewählt werden.
Da die Kamera das PTP Protokoll verwendet, ist die Grundstruktur der Datenübertragung bekannt und es müssen nur die einzelnen Parameter und die Antworten der Kamera herausgefunden werden. Dazu eignet sich das Programm USBlyzer 2.0, das in einer 33 tägigen Freeware Version verfügbar ist. Mit Hilfe dieses Programmes können die einzelnen USB Ports eines PCs analysiert und die Datenübertragung überwacht werden. Ziel dieser Arbeit ist es die Kamera über den USB Anschluss fernauszulösen, den TV und AV Wert zu setzen, den Fokus zu bedienen und die Belichtungsdauer zu ändern. All diese Funktionen lassen sich auch anhand der mit der Kamera mitgelieferten EOS Utility Fernsteuerungssoftware von Canon einstellen. Während die benötigten Einstellungen in der Software der Reihe nach ausgeführt wurden, protokollierte USBlyzer den Traffic über den USB Port, an dem die Kamera angesteckt wurde. Das Ergebnis war jedoch nicht zufriedenstellend, da eine Fülle von Transfers stattgefunden hat und keine Möglichkeit bestand, festzustellen welcher Transfer durch welches Ereignis stattgefunden hat. Grund für die hohe Anzahl an Transaktion ist, dass die EOS Utility Software periodisch zusätzliche Daten von der Kamera abfragt (z.B. Aktuelle Einstellungen, Batteriestand, etc.).
Um dennoch eine detaillierte Analyse durchzuführen wurde mit Hilfe der SDK Dokumentation eine Windows Anwendung geschrieben, die nacheinander die geforderten Befehle ausführt. Das Protokoll der Auswertung ist im Anhang F zu finden. Die Abfrage von Events (z.B. Foto ist bereit zum Abholen, Änderung Batteriestand, etc.) wurde hierbei nicht verwendet, es wird einfach eine genügend lange Zeit gewartet. Die Einstellungen für TV, AV, Belichtungsdauer und Fokus wurden jeweils mit 2 unterschiedlichen Werten durchgeführt, um die Veränderungen der Parameter zu verfolgen. Es wurden auch gezielt nicht verfügbare Einstellungsmöglichkeiten übertragen (z.B. +3 Belichtung wird von der EOS 500D nicht unterstützt), um die Reaktion der Kamera auf diese Befehle zu testen.
Zu Beginn der Kommunikation mit der Kamera muss die SDK mit der Funktion „EdsInitializeSDK“ initialisiert werden. Danach wird über „EdsGetCameraList“ eine Liste der vorhandenen verbundenen Kameras am PC herausgefunden. Die Anzahl wird dabei mit der Funktion „EdsGetChildCount“ bestimmt. Wenn mindestens eine Kamera bereit steht, erhält man mit „EdsGetChildAtIndex“ Zugriff auf die ausgewählte Kamera. Die Interaktionen mit der Kamera müssen innerhalb einer Session stattfinden („EdsOpenSession“ und „EdsCloseSession“). Befehle (z.B. Auslösen der Kamera oder LiveView starten) werden mit der Funktion „EdsSendCommand“, Einstellungen (z.B. TV, AV und Belichtung) mit „EdsSetPropertyData“ übertragen. Die Änderung des Fokuses kann nur innerhalb des LiveView erfolgen. Die möglichen Werte von der Einstellung sind dabei sehr gut in der SDK Dokumentation ersichtlich. Es sei noch einmal darauf hingewiesen, dass die hier geschriebene Windowsanwendung auf die DLL's der SDK zugreift und diese von Canon

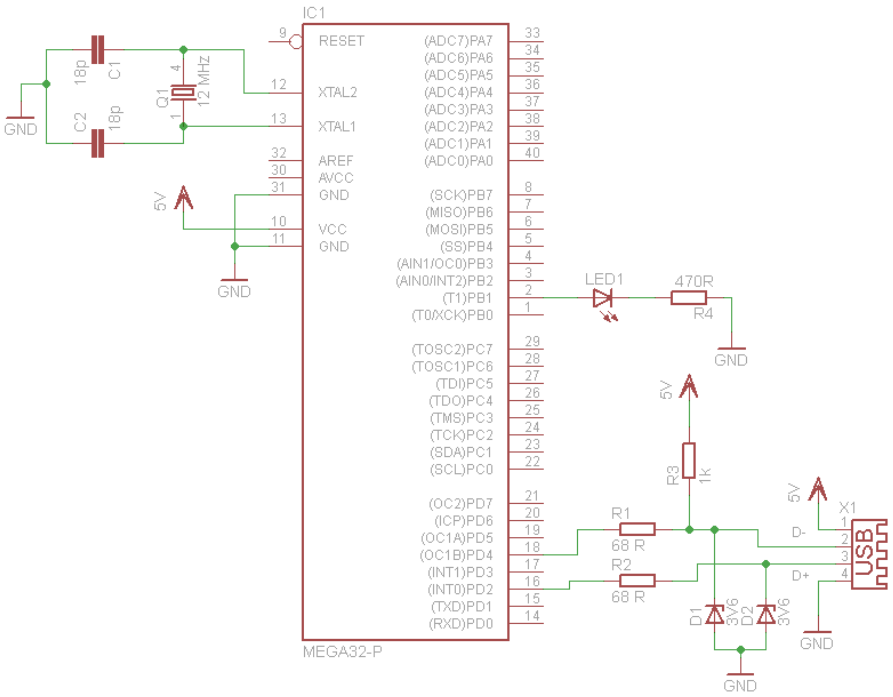


Abbildung 2.3: Testschaltung für die softwareseitige Lösung einer USB Verbindung mit einem ATmega. Statt dem ATmega32 wurde ein ATmega16 verwendet.

geschützt sind und nur Benutzern des Digital Imaging Developer Programm zur Verfügung stehen. Daher dürfen sie nicht ohne Erlaubnis kopiert werden.

Exkurs: Verschiedene Arten der Implementierung einer USB Kommunikation mit einem Atmel μC

Hier wird nur auf die Hardwareseite eingegangen, die Probleme der Treiberentwicklung für ein Betriebssystem werden nicht näher erörtert. Für den interessierten Leser empfehle ich jedoch den Artikel „Universeller USB-Treiber“ unter [Elektor März 2007] zu lesen.

Software Lösung

Eine sehr einfache Möglichkeit einen „normalen“ ATmega Controller um eine Kommunikation über die USB Schnittstelle zu erweitern besteht in der Verwendung einer Softwareseitigen Lösung. Der Aufwand für die extern benötigte Schaltung hält sich dabei in Grenzen, sodass diese auf einem Steckbrett realisiert (siehe Abbildung 2.3) werden kann. Die beiden Widerstände R1 und R2 dienen zur Strombregrenzung bei einem Kurzschluss und zur Verminderung von Störreflexionen. Zu beachten ist, dass der D+ Ausgang des USB Gerätes mit dem INT0 Einganges des μC verbunden ist. Mit Hilfe von R3 wird dem Host mitgeteilt, dass es sich hierbei um ein Low Speed Gerät handelt. Die Taktfrequenz des ATmegas muss mindestens 12 MHz betragen, damit diese Lösung durchgeführt werden kann. Die Unterstützung anderer Frequenzen ist in der Dokumentation der Library beschrieben. Die Zenerdioden dienen zur Spannungsanpassung für den Host, weil der μC mit 5V vom USB Anschluss versorgt wird und es nicht garantiert werden kann, dass der Host Controller mit den Spannungspegel arbeiten kann. Am Ausgang des Portes PB1 wird eine LED angesteuert, die über den USB Anschluss ein- und ausgeschaltet wird.

Für eine Implementierung dieser Lösung in eine eigene Anwendung müssen die benötigten Include Dateien unter [V-USB] heruntergeladen werden. Danach sind die Dateien aus dem Ordner „usbdrv“ in das eigene Projekt einzubinden. In der Datei „usbconfig.h“

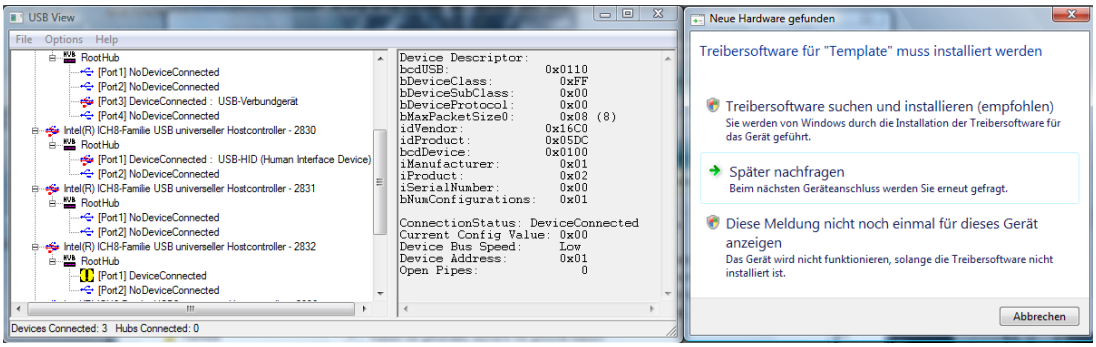


Abbildung 2.4: Ergebnis der Enumeration des USB Devices.

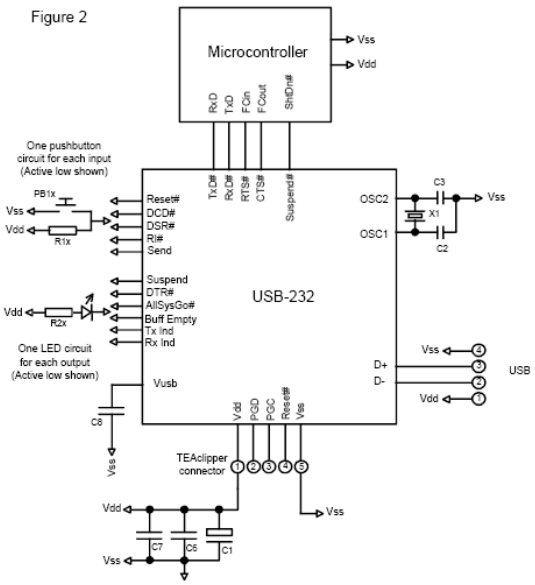
sind die benötigten Parameter für die Schaltung anzugeben (Port Pins für die D+ und D- Leitungen, Taktfrequenz, etc.).
Wenn man die kompilierte Anwendung auf den Controller geladen hat und die Schaltung über den USB Anschluss mit einem PC verbindet, sollte das Betriebssystem melden, dass eine neue Hardware gefunden wurde. Der Benutzer wird daraufhin zur Auswahl eines Treibers aufgefordert. Jetzt muss man den selbstgeschriebenen Treiber auswählen um den Datenaustausch über USB zu ermöglichen [Elektor März 2007]. Mit Hilfe des Programmes „USBView“, was ein Teil des Windows Driver Kit ist, kann man die Informationen über das angeschlossene Gerät auslesen.

Verwendung einer USB Bridge

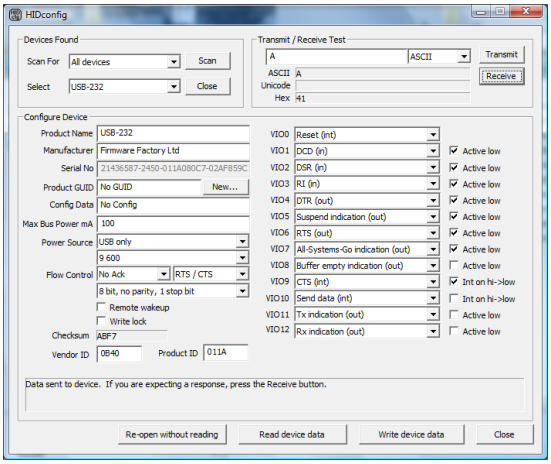
Eine andere Variante ist die Verwendung eines zusätzlichen speziellen IC, in dem das USB Protokoll bereits integriert ist. Der μC kann dann über die häufig vorhandenen Standard Schnittstellen (USART, I2C, SPI) auf diesen IC zugreifen. Diese IC's werden zum Beispiel von der Firma HexWax angeboten und sind zum Testen auch im praktischen DIL Gehäuse erhältlich. Als Beispiel wird hier der USB 232 Bridge IC dieser Firma als eine USB Erweiterung für einen μC gezeigt. Das Betriebssystem erkennt diesen IC als ein HID, d.h. es ist kein selbstgeschriebener Treiber für die Kommunikation notwendig. Als Anwendungsbeispiel wurde das im Datenblatt für ein Bus Powered Device empfohlene verwendet. Zu beachten ist, dass es sich hierbei um eine Full-Speed Verbindung handelt, d.h. man muss D+ mit einem Pullup Widerstand mit VDD verbinden 8.8.3. Als Controller dient ein ATmega32, dessen Pins von der USART Schnittstelle (RxD und TxD) für die Kommunikation mit der USB Bridge verbunden wurden. Als einfaches Demonstrationsbeispiel wird am Pin 7 des Ports D eine Leuchtdiode angesteuert, die für alle empfangenen Zeichen $<$ 0x5B leuchtet, für höherwertige Zeichen jedoch nicht mehr. Der Hersteller HexWax stellt auf der Homepage die Software „HIDConfig“ für die Konfiguration des IC's kostenlos zur Verfügung. Damit ist es schnell möglich den IC auf Funktionalität zu überprüfen, weil man beliebige Zeichen übertragen und auslesen kann (siehe Abbildung 2.5).

Externer Host Controller

Als externer Host Controller bietet sich das Vinculum V2 32 Pin Modul von FDTI an (siehe Abbildung 2.6). Es ist für den schnellen Prototypenbau ausgelegt, wenn eine USB Host Funktionalität erforderlich ist. Das Modul beinhaltet einen Vinculum V2 Controller der anhand der Toolchain in der Programmiersprache C programmiert werden kann. Nicht alle Sprachelemente von C werden unterstützt, deswegen empfiehlt es sich den „User Guide“ durchzulesen, wenn man die Fehlermeldung des Compilers nicht nachvollziehen kann. Die Fehlersuche im Programm erwies sich zeitweise als frustrierend, weil sich das Programm



(a) Schematic der Beschaltung von der USB 232 Bridge [Datenblatt USB-232]



(b) Konfigurationssoftware von HexWax

Abbildung 2.5: Kommunikation über USB mit einer USB 232 Bridge

im Debugger nicht nachvollziehbar verhält. Eine genaue Kontrolle der Interpretation des Compilers durch eine Schrittweise Ausführung des Programmes ist zwingend notwendig.
Die vorhandene Firmware des Controllers ist in 4 Schichten aufgeteilt. In der letzten Schicht („Application Layer“) befindet sich die eigentliche Anwendung, die auf verschiedene Funktionen der unteren Schicht zugreifen kann. Der Sinn hinter dieser Aufteilung ist die Abstraktion der teilweise komplizierten Abläufe für eine leichte Integration in eigene Anwendungen durch einfache API Aufrufe. So läuft auf der untersten Ebene („VOS Kernel“) ein multi-tasking Betriebssystem, das die Hardware Ressourcen verwaltet. Diese Ebene muss in jeder Anwendung inkludiert werden. Die anderen zwei Schichten sind die „FTDI Drivers“, damit kann auf die verschiedenen Schnittstellen des Controllers zugegriffen werden (USB Host/Slave, UART, SPI etc.), und „FTDI Libraries“ für die Standard C Bibliotheken (z.B. string.h, stdio.h).
Für die Übertragung des Programms ist das Zusatzmodul „VNC2 Debugger/Programmer Module“ (Abbildung 2.6 rechts) notwendig. Beim VNC2 handelt es sich um einen leistungsstarken 16 Bit Mikrocontroller mit 2 2.0 Full- und Low-Speed USB Host/Slave Anschlüssen, SPI, UART, PWM und noch vielen anderen Möglichkeiten. Für die Verwendung des Debugger/Programmer Zusatzmoduls ist die Installation zusätzlicher Treiber notwendig, die auf der Hersteller Homepage heruntergeladen werden können. Gleichzeitig mit der „Toolchain“ werden Beispielprojekte mit installiert, anhand derer die Funktionsweisen und Eigenschaften des Controllers demonstriert werden. Ein Beispiel Projekt verwendet das PTP Protokoll um die Aufnahme einer Kamera auszulösen und das Bild von der Kamera auf einen Massenspeicher, der am zweiten USB Anschluss steckt, zu kopieren. FDTI liefert dafür einen Still Image Class Treiber mit, der aber nur bedingt zu gebrauchen ist, weil damit nicht der volle Umfang der PTP Kommunikation mit einer EOS 500D genützt werden kann [VNC2 User Guide].
Für diese Zwecke wurde das PTP ausprogrammiert und befindet sich in den Dateien „PTP.c“ (siehe Anhang H) mit der dazugehörigen Header Datei „PTP.h“ (siehe Anhang H). Entscheidend bei der Nutzung ist die Verwendung des richtigen Operation Codes mit der richtigen Anzahl an Parametern und Werten, weil das Programm sonst in eine

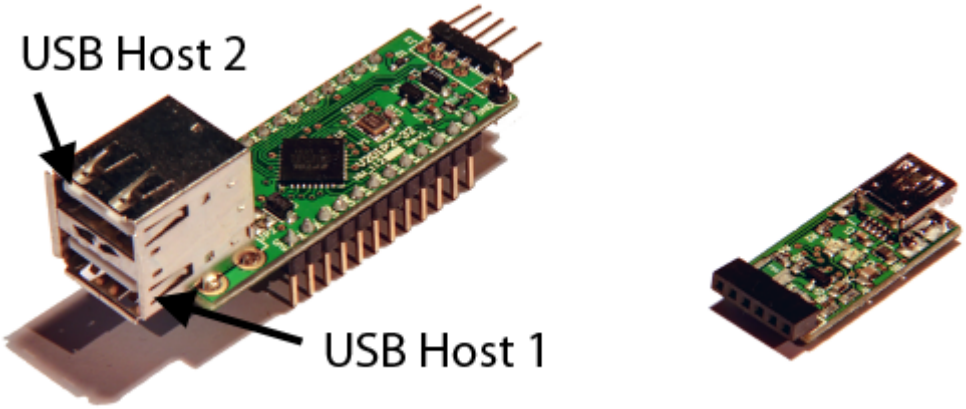


Abbildung 2.6: Im Bild links zu sehen das VNC2 Vinculum double USB Module mit 32 Pins von FDTI mit passendem Debugger/Programmer Modul rechts.

Endlosschleife geraten kann. Die Reihenfolge der zu erwartenden PTP Abläufe und der Informationsrichtung der einzelnen USB Endpunkte muss ebenfalls bekannt sein.

Integrierter USB Host Controller
Von der Firma Atmel sind einige 32 Bit Controller der Serie AT32UC3 in der Lage USB Host Controller Funktionalität zu übernehmen. Leider sind diese Controller nur in SMD Bauform vorhanden. Deswegen wurde zum Testen des AT32UC3B1256 Controllers, eine Adapter Platine entworfen, deren Schaltplan unter 3.4 zu finden ist. Diese kann für den Aufbau einer Testschaltung auf einem Steckbrett verwendet werden. Bei Versuchen an dieser Testschaltung wurden versehentlich die Spannungspole vertauscht, wodurch der Controller zerstört wurde. Das Programmiergerät konnte keine Verbindung mehr zu diesem aufbauen, womit keine weiteren Tests möglich waren. Da aber die Implementierung mit Hilfe eines externen Host Controller 2.3.2 ohne Probleme funktioniert, wird diese Variante in dieser Arbeit weiter verwendet.

Festellen ob EOS 500D ein Host Controller ist
Eine Verbindung über das USB Protokoll findet immer zwischen einem Host und einem Device statt. Deshalb ist es notwendig in Erfahrung zu bringen, ob die EOS 500D als Host oder als Device arbeitet. Das lässt sich leicht überprüfen, indem man das USB Kabel an die Kamera anschließt und auf der anderen Seite die Spannung zwischen den Anschlüssen VDD und GND misst. Beträgt diese 5 V, ist das ein Indiz dafür, dass in der Kamera ein Host Controller eingebaut ist. Es wurde, wie zu erwarten war, eine Spannungsdifferenz von 0 V gemessen, was bedeutet, dass es sich hier um einen Device Controller handelt. Die USB Steckerform gibt übrigens keine Auskunft über die Host bzw. Device Funktionalität. Von den vorher vorgestellten Möglichkeiten einen μC der Firma Atmel mit der EOS 500D zu verbinden, bleibt nur die Variante des bereits integrierten USB Anschlusses für die Auswahl 2.3.2 oder die Verwendung eines externen Host Controllers 2.3.2 übrig.

2.3.3 Aufnahme von HDR Fotos

Für die Erzeugung einer HDR Aufnahme ist es notwendig eine Serie von Fotos mit unterschiedlichen Belichtungszeiten vom selben Motiv zu erstellen (siehe 8.3). Die Canon EOS 500D verfügt, wie die meisten anderen Kameras in diesem Preissegment, über die Möglichkeit einer Belichtungsreihenautomatik [Bedienungsanleitung EOS 500D]. Das in

(a) -2 EV (b) 0 EV (c) +2 EV

Abbildung 2.7: Aufnahmen mit Hilfe der Belichtungsautomatik

(a) Adobe Photoshop CS3 (b) Photomatix Pro 4.0

Abbildung 2.8: Gegenüberstellung Tone Mapping Ergebnis

dieser Arbeit verwendete Klasse V Modul wird die Auslösung von HDR Aufnahmen nicht über die Verwendung der Belichtungsreihenautomatik Funktion erzeugen, sondern durch direktes ändern der Belichtungswerte. Ein manuelles aktivieren der Belichtungsautomatik kann durch den Benutzer jedoch jederzeit vorgenommen werden, dadurch werden durch die Betätigung der Auslösefunktion mehrere Fotos aufgenommen.
Um die geschossenen Fotos in ein HDR Bild umzuwandeln wurden die 2 Bildbearbeitungsprogramme Adobe Photoshop CS3 [Adobe] und Photomatix Pro 4.0 [Photomatix] verwendet. Dabei wurden jeweils 3 Fotos mit einem Abstand von 2 EV aufgenommen (siehe Abbildung 2.7) und mit diesen zwei Programmen zu einem HDR zusammengefügt und anschließend mit Tone Mapping in ein JPG umgewandelt (siehe Abbildung 2.8). Insgesamt konnten mit dem Photomatix Pro 4.0 bessere Ergebnisse erzielt werden. Vor allem die dort vorhandene Geisterbildkorrektur Funktion hat sich bei Aufnahmen mit nicht starren Objekten als sehr nützlich erwiesen. Mit der Standardeinstellung lieferten jedoch beide Programme in etwa dasselbe Ergebnis.

2.3.4 Zeitrafferaufnahmen

Zeitrafferaufnahmen stellen mit den bereits beschriebenen Möglichkeiten zur Fernauslösung der Kamera kein großes Problem dar. Dafür wird das unter 3.1 beschriebene Fernauslöse Modul mit einem Ausgang des μC verbunden, der gesteuert von der Timereinheit des Controllers in bestimmten Intervallen ein Foto auslöst. Anstelle der Timereinheit, kann der Impuls für das Auslösen eines Fotos auch von einem Klasse VI Modul oder einem Sensor kommen. Die einzelnen geschossenen Fotos werden mit Hilfe einer geeigneten Video Software zu einem Video hintereinander gefügt. Als Beispiel wurde, über den Zeitraum von ca. 8 Stunden, alle 2 Minuten ein Foto vom Blick aus dem Fenster aufgenommen

(siehe Youtube - Zeitraffer Canon EOS 500D). Es empfiehlt sich das Display der Kamera zu deaktivieren, damit der Akku länger hält.

2.3.5 3D Scanner

Um die Möglichkeit der Durchführung einer 3D Scannung mit Hilfe von mehreren Aufnahmen mit unterschiedlichen Fokuseinstellungen zu testen, wurde eine Serie von Fotos von einem Objekt mit unterschiedlichen Schärfeebenen aufgenommen. Die Änderung des Fokuses erfolgte mit Hilfe des Programms EOS Utility von Canon. Dazu muss die EOS 500D über den USB Anschluss mit dem PC verbunden sein. Anhand eines selbst geschriebenen Scripts in Matlab (siehe A) wurde versucht ein 3D Modell des Objektes dazustellen. Dafür muss zuerst eine Kantendetektion für jedes der geschossenen Fotos durchgeführt werden, um danach die Fotos übereinander zu legen. Das Objekt wird aus den schärfsten Kanten einer Fotoserie aufgebaut. Für das einfache Beispiel, ist das Ergebnisbild in der Abbildung 2.9 zu sehen.
Anhand der ersten Tests wurde festgestellt, dass man einen Schwellwert einführen muss, ab wann eine Kante als scharf erkannt wird. Eine Glättung des Kantenbildes ist ebenfalls notwendig, weil die Störungen zu groß waren. Es hilft, wenn man das Objekt gut ausleuchtet um die Aufnahmen mit einem geringen ISO Wert durchzuführen. Damit ein korrekter Abstand zu jedem Foto einer Serie zugeordnet werden kann, wäre es sinnvoll eine Art Kalibrierungslinie neben dem Objekt abzubilden.
Wie man anhand von den Ergebnissen der Simulation sehr gut erkennt, kann durch Anwendung dieser Methode durchaus ein ungefähres 3D Modell eines Objektes angefertigt werden. Um ein Farbmodell zu erzeugen, könnten die Farben vom Punkt der stärksten Kante übernommen werden. Für ein vollständiges 3D Modell muss die Fotoserie aus mehreren verschiedenen Blickwinkeln aufgenommen werden. Die Ergebnisse der einzelnen Serien müssen daraufhin mit Hilfe von 3D Transformationen zu einem Objekt zusammengefügt werden.
Um Aufnahmen aus unterschiedlichen Blickwinkeln zu erzeugen, könnte sich das Objekt z.B. auf einer rotierenden Plattform befinden. Wodurch die Kameraposition nicht geändert werden muss.

2.3.6 3D Aufnahmen

Zum Erzeugen von 3D Aufnahmen sind, wie unter 8.5 beschrieben, zwei Fotos aus unterschiedlichen Positionen notwendig. Moderne Kameras verfügen für die Verbindung mit einem Stativ über ein 1/4 Zoll UNC Gewinde im Gehäuse. Die benötigten Schrauben sind leider im europäischen Raum nur sehr schwer zu bekommen, weil hier vorrangig das metrische ISO Gewinde verwendet wird. Die für die selbstentwickelten Konstruktion (siehe Abbildung 2.10 und 2.11) verwendeten Schrauben wurden vom Astro Online Shop bezogen.
Beim Verbindungsstück zum Stativ befindet sich auf der Unterseite eine Versenkung für die Verbindungsschraube zur Schiene. Die einzelnen Löcher befinden sich jeweils in der Mitte. Gezeichnet wurden die Skizzen mit Hilfe des Programmes Google SketchUp 8.

2.3.7 360° momentan Fotos

Wie unter 8.2.2 erklärt, hängt der von einer Kamera aufgenommen Bildwinkel von der eingestellten Brennweite und von der Größe des Bildaufnehmers der Kamera ab. Um ein

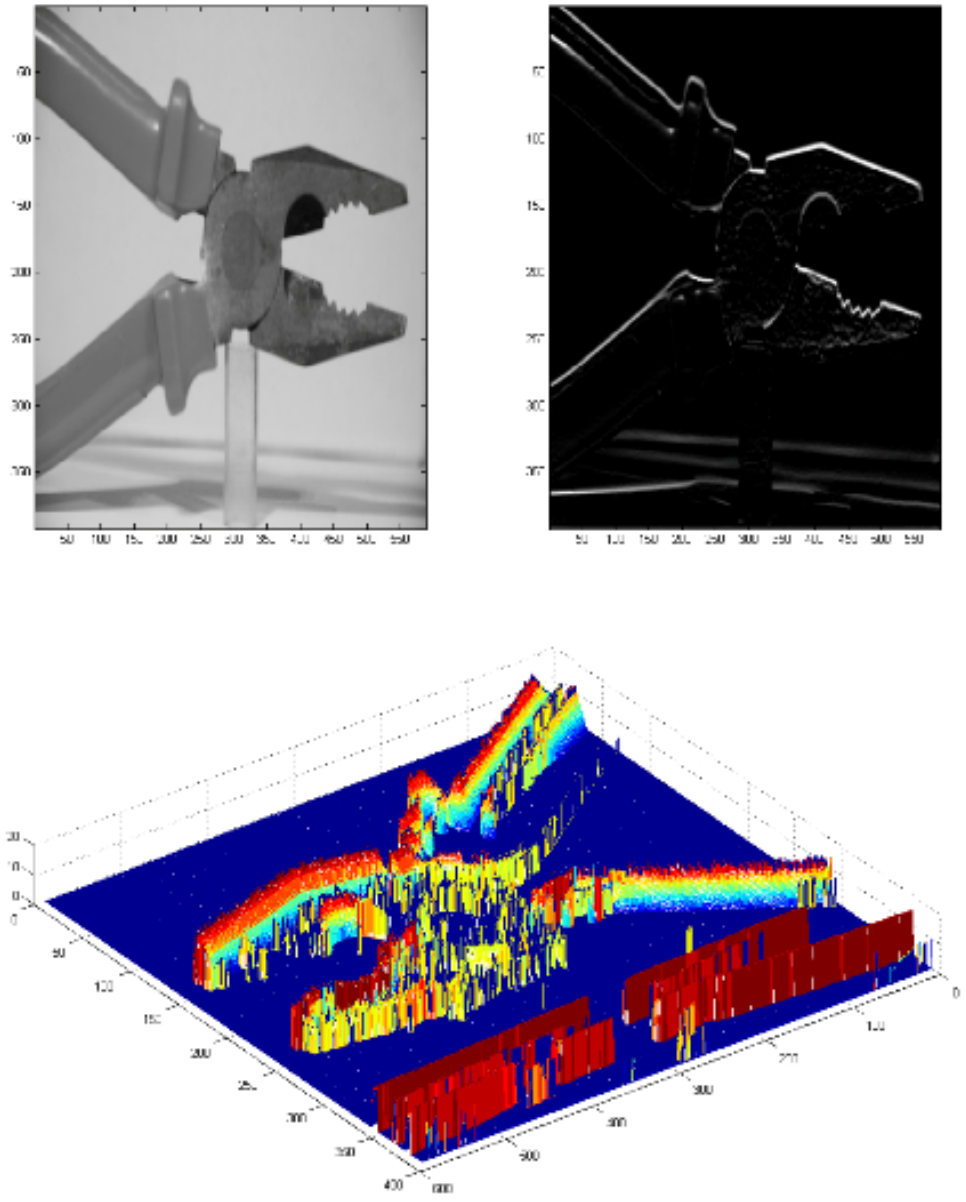
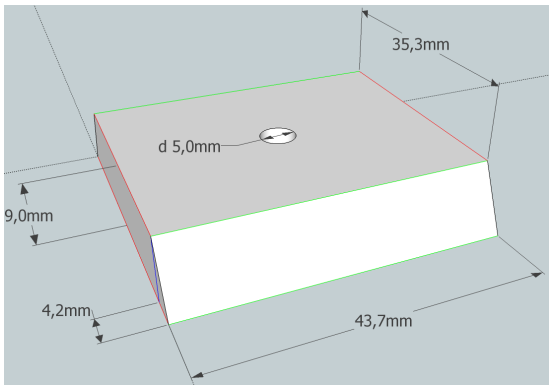
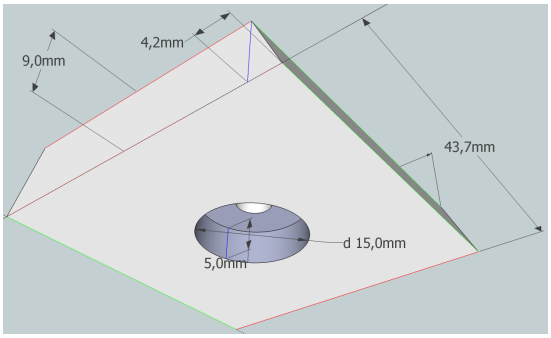


Abbildung 2.9: Ergebnis des Tests einer 3D Scannung durch Aufnahmen einer Zange mit unterschiedlichen Schärfeebenen. Die Bilder oben zeigen eine Aufnahme inklusive der Kantendetektion und das Bild unten zeigt das zusammengefügte Endergebnis.



(a) Stativ Aufsatz Oberseite



(b) Stativ Aufsatz Unterseite

Abbildung 2.10: Verbindungsstück für ein Cullmann ALPHA 2500 Stativ.

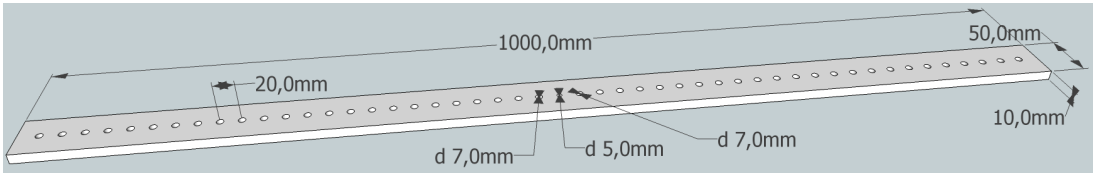


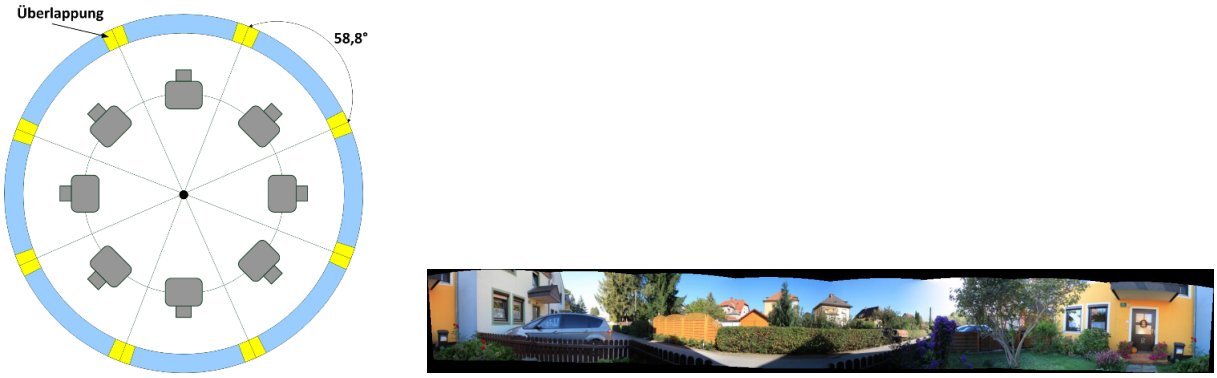
Abbildung 2.11: Schiene für die 3D Aufnahme Konstruktion.

360° Foto zu erzeugen sind mehrere Kameras notwendig, die in unterschiedliche Richtungen ausgerichtet sind und in Summe einen Bildwinkel von 360° aufnehmen. Anfangsüberlegte Beschränkungen für einen maximalen minimal aufzunehmenden Bildwinkeln wurden verworfen, weil zwar die Anzahl der zu nutzenden μC Pins beschränkt ist, aber für ein momentan Foto nur ein Ausgang notwendig ist, von dem alle eingesetzten Fernauslöserlemente parallel angesteuert werden. Durch diese Lösung kann eine beliebige Anzahl von Kameras verwendet werden. Wenn die Ausgangslast für den μC Pin zu groß werden sollte, muss ein Treiber verwendet werden. Der aufgenommenen Bildwinkel sollte größer als der minimale unter 1 berechnete sein, damit die Stitching Software aus den sich überlappenden Bereichen das gewünschte Bild erzeugen kann.
Für die Verifikation wurde ein Versuch im eigenen Vorgarten vorgenommen. Dafür wurden 8 Fotos mit der EOS 500D bei einer Brennweite von 24 mm (Bildwinkel = 58,8°) aufgenommen und mit Hilfe des Programmes AutoStitch zu einem 360 Foto zusammengefügt 2.12.
Ein interessantes Projekt zum Thema 360° Grad momentan Fotos, ist die werfbare Ballkamera. Dabei handelt es sich um einen Ball mit 36 integrierten Kameras und einem Beschleunigungssensor. Anhand dessen man den höchsten Punkt in der Wurfbahn bestimmt und alle Kameras zu gleich auslöst.

$$\text{minimaler Bildwinkel} = \frac{360°}{\text{verfügbarer Fotopositionen}} = \frac{360°}{8} = 45° \tag{1}$$

2.3.8 Test der Auslöseverzögerung

Damit ein Objekt zum gewünschten Zeitpunkt aufgenommen wird, sind mehrere Faktoren auschlaggebend die in der Abbildung 2.13 ersichtlich sind.
Nehme man an, dass man einen Wassertropfen kurz vor dem Aufschlag fotografieren möchte. Ein Auslösen zum gewünschten wahrgenommenen Zeitpunkt würde kein zufrie-



(a) Skizze Versuchaufbau

(b) Ergebnisbild mit Hilfe des Programmes AutoStitch zusammengefügt.

Abbildung 2.12: Verfikation einer 360° Aufnahme. Foto Links Versuchsaufbau mit überlappenden Bildbereichen. Foto Rechts das fertige Ergebnis

denstellendes Ergebnis erzielen. Der Tropfen wäre am Bild schon am Boden aufgeschlagen. Der Grund dafür sind eine Reihe von Verzögerungen bis das tatsächliche Bild entsteht. Diese entstehen durch die Reaktionszeit des Entscheiders, dabei kann es sich um einen Menschen, eine Lichtschranke oder ein anderes technisches System handeln, und zum anderen die Zeit, die die Kamera für die Auslösung benötigt.

Um die Verzögerung[1] der Kamera zu messen, wurde eine Testschaltung aufgebaut, die den unter 2.3.1 beschriebenen Fernauslöser verwendet (Abbildung 2.14). Sie besteht aus einem ATmega μC, getaktet mit 12 MHz, an dessen Ports A und C insgesamt 16 LEDs angeschlossen sind. Verwendet wurde die Timereinheit 1 des Controllers mit einem Vorteiler von 1, der Vergleichswert wurde so eingestellt, dass alle 1 ms ein Interrupt ausgelöst wird (Berechnung siehe 2). In der Interrupt Service Routine wird der Ausgangswert um 1 erhöht und beginnend beim PIN C0 in binärform ausgegeben. Beim Auswerten des Ausgabewertes ist zu beachten, dass die einzelnen Pins des Port A umgekehrt zur Reihenfolge des Port C angeordnet sind. Die Auslösung der Kamera wird durch die Timer 0 Einheit gesteuert, in dem bei jedem Überlauf die char Variable „Timer0Value“ erhöht wird. Wenn diese Variable den Wert 200 erreicht hat (es kommt immer zu einem gewollten Überlauf), wird der Status des PIN B1 gewechselt (ca. alle 5 Sekunden) und bei einer positiven Flanke beginnt eine neue Messung. Da die Portleitungen PC3, PC4 und PC5 für die Ausgabe verwendet werden und sich gleichzeitig die JTAG Schnittstelle dort befindet, muss man das JTAGEN FUSE Bit deaktivieren, weil dieses standardmäßig bei den ATmega Modellen gesetzt ist und man die Pins sonst nicht als normale I/O Pins benutzen kann. In der Firmware wurde gleichzeitig die Software für die Messung der Impulsbreite implementiert, deren Beschreibung unter 2.3.9 nachzulesen ist.

$$\text{einzustellender Wert} = \frac{\text{Taktfrequenz}}{\text{Vorteiler} * \text{Auslösefrequenz}} = \frac{12\ \text{MHz}}{1 * 1\ \text{kHz}} = 12000 \qquad (2)$$

Die Kamera wird so eingestellt, dass alle LEDs am aufgenommenen Bild gut sichtbar sind. Als Einstellungsprogramm empfiehlt sich der Tv-Bewegungsaufnahmen („Time Value“) Modus mit einer Verschlusszeit die kleiner der Hälfte der gewünschten zeitlichen Auflösung. Dadurch wird verhindert, dass mehrere Ausgangszustände im Bild abgebildet werden (gewählt wurde 1/4000). Die Anzahl der geschossenen Fotos kann im Programm

[1] englische Bezeichnung dafür ist „shutter lag“

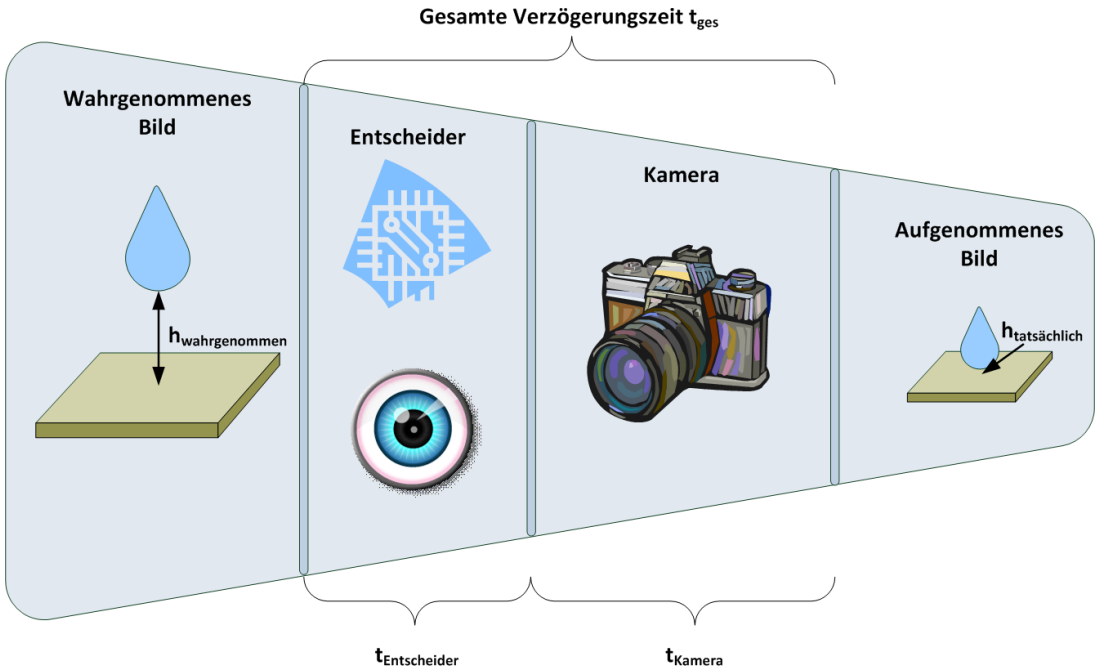


Abbildung 2.13: Durch Verzögerungen entstehender Unterschied zwischen einem wahrgenommenen und dem entstandenen Bildes.

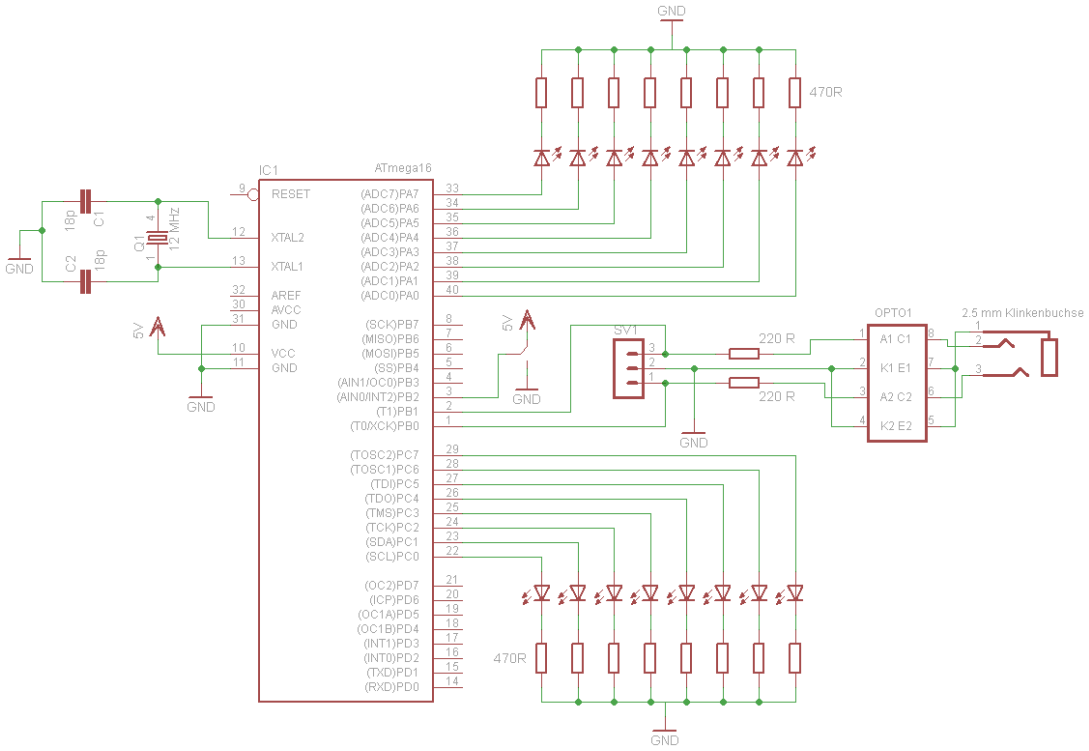
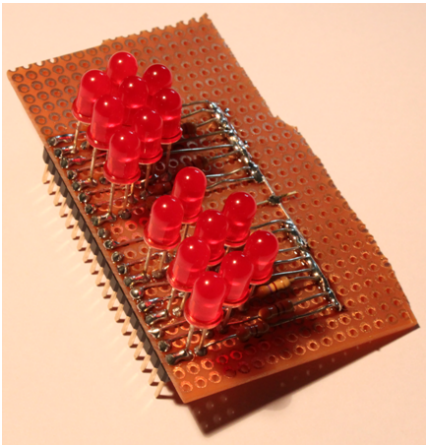


Abbildung 2.14: Eagle-Schaltung zur Messung der Auslöseverzögerung und der minimalen Trigger Impulsbreite.



(a) LED Platine für die Ausgabe der Auslöseverzögerung. Die Erweiterungsplatine lässt sich durch die Stiftleiste optimal an das DIL Gehäuse des ATmega anbinden.

(b) Kameraeinstellung an der EOS 500D

Abbildung 2.15: Kameraeinstellung und LED Testmodul für Messung der Auslöseverzögerung

mit Hilfe des DEFINE Wertes „NUMBERPHOTOS“ festgelegt werden, hier wurden 20 Fotos gewählt. Für den Test sind keine Fotos in hoher Bildqualität notwendig, deswegen kann die Bildgröße auf S herabgesetzt werden (siehe Abbildung 2.15 Bild b).

Ein visuelles Auswerten der geschossenen Fotos hat sich als zu zeitaufwendig erwiesen, deshalb wurde eine kleine Windows WPF Anwendung erstellt (ShutterSpeedTester.exe - Anhang J), die diese Aufgabe abnimmt. Dabei handelt es sich um kein fertiges Programm und es kann durchaus sein, dass einige Fehler nicht abgefangen werden. Wenn man sich an die kurze Anleitung (Anhang D) hält, wird man aber sicher ein zufriedenstellendes Ergebnis erzielen. Um Rechenzeit bei der Auswertung zu sparen, werden die Bilder beim Laden automatisch auf eine Breite von 500 Pixel transformiert.
Der verwendete Testaufbau ist gemeinsam mit einem geschossenen Test Foto in der Abbildung 2.16 zu sehen. Als Testmodell stand die Canon EOS 500D (EF-S 18-55mm IS Objektiv) zur Verfügung. Mit ihr wurden zweimal vier Messreihen mit unterschiedlichen Einstellungen an der Kamera durchgeführt:

- Auto-Fokus: Spiegelverriegelung ausgeschaltet (AF-Saus)
- Auto-Fokus: Spiegelverriegelung eingeschaltet (AF-Sein)
- Manueller-Fokus: Spiegelverriegelung ausgeschaltet (MF-Saus)
- Manueller-Fokus: Spiegelverriegelung eingeschaltet (MF-Sein)

Die Messreihen wurden an zwei Tagen mit unterschiedlichen Lichtverhältnissen durchgeführt, womit die verschiedenen Ergebnisse beim Auto-Fokus erklärbar sind. Bei den Messreihen mit eingeschalteter Spiegelverriegelung wurden nur 10 Aufnahmen gemacht (Es wird ein zusätzliche Trigger Impuls für die Spiegelverriegelung benötigt). In der Tabelle 2.1 sind die Ergebnisse der Messungen abgebildet. Es ist zu erkennen, dass bei eingeschalteter Spiegelverriegelung die Auslöseverzögerung beim MF und AF sehr geringe Werte aufweisen. Die wahrscheinliche Ursache dafür ist die Tatsache, dass die Fokussierung

(a) Messaufbau für die Messung der Auslöseverzögung und der Triggerimpulsbreite.

(b) Testergebnisbild der AF Messung mit ausgeschalteter Spiegelverriegelung (217 ms).

Abbildung 2.16: Messung der Auslöseverzögerung

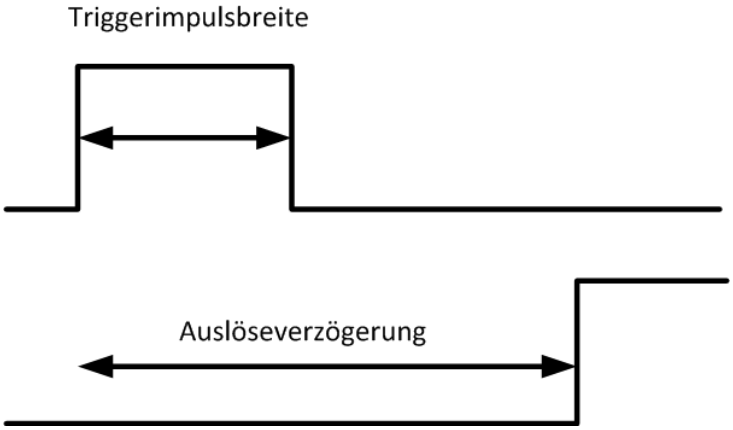


Abbildung 2.17: Skizze der Trigger Impulsbreite und der Auslöseverzögerung.

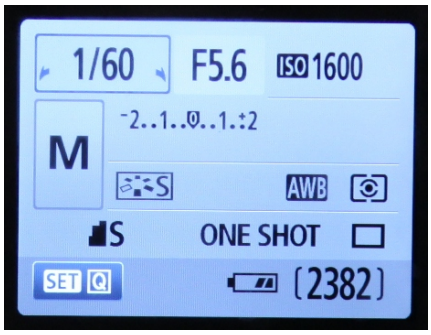
beim ersten Trigger Impuls durchgeführt wird, der auch die Spiegelverriegelung auslöst. Warum bei der zweiten Messreihe die Auslöseverzögerung bei eingeschalteter Spiegelverriegelung geringer ausfiel, als die Vergleichsmessung beim MF, konnte nicht eruiert werden. Wie zu erwarten war, kommt es bei der Verwendung des Auto-Fokus und ausgeschalteter Spiegelverriegelung zu einer beträchtlichen Verlängerung der Auslöseverzögerung. Dieser Wert ist aber vom verwendeten Objektiv abhängig, weil bei Canon Kameras die AF Einheit im Objektiv eingebaut ist.

2.3.9 Test der Trigger Impulsbreite

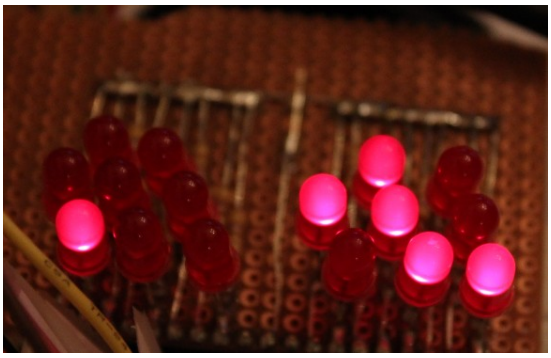
Beim messen der Auslöseverzögerung hat es sich herausgestellt, dass der Trigger Impuls eine gewisse Breite besitzen muss, damit eine Aufnahme ausgelöst wird. Um die Breite, in Abhängig von den gewählten Einstellungen, zu messen wurde die Schaltung von der Abbildung 2.14 verwendet. Anhand des Zustandes des PIN B2 wird unterschieden ob die Auslöseverzögerung- oder die Trigger Impulsbreite-Messreihe durchgeführt werden soll (Low = Trigger Impulsbreite, high = Auslöseverzögerung). Die Überprüfung geschieht beim Programmstart, d.h. man muss den Controller reseten, um den Testmodus zu än-

	Messreihe 1		Messreihe 2	
Messung	**Messergebnis**	**Anzahl**	**Messergebnis**	**Anzahl**
AF-Saus	*221,15 ms*	*20*	*372,70 ms*	*20*
	216,00 ms	11	216,00 ms	1
	217,00 ms	7	298,00 ms	1
	249,00 ms	1	297,00 ms	1
	279,00 ms	1	355,00 ms	1
			391,00 ms	3
			376,00 ms	1
			380,00 ms	1
			383,00 ms	1
			394,00 ms	2
			388,00 ms	2
			399,00 ms	1
			406,00 ms	1
			405,00 ms	1
			395,00 ms	1
			407,00 ms	1
			400,00 ms	1
AF-Sein	*122,20 ms*	*10*	*107,00 ms*	*10*
	122,00 ms	8	90,00 ms	5
	123,00 ms	2	147,00 ms	1
			92,00 ms	1
			132,00 ms	1
			95,00 ms	1
			154,00 ms	1
MF-Saus	*152,00 ms*	*20*	*152,00 ms*	*20*
	152,00 ms	20	152,00 ms	20
MF-Sein	*122,60 ms*	*10*	*124,50 ms*	*10*
	122,00 ms	8	122,00 ms	1
	123,00 ms	1	123,00 ms	5
	127,00 ms	1	127,00 ms	4

Tabelle 2.1: Ergebnisse der Verzögerungsmessungen an EOS 500D (EF-S 18-55mm IS Objektiv)



(a) Kameraeinstellung an der EOS 500D.



(b) Beispiel eines Testergebnisbild der Trigger Impulsbreite (2. Foto und 185 ms Impulsbreite).

Abbildung 2.18: Messung der Trigger Impulsbreite

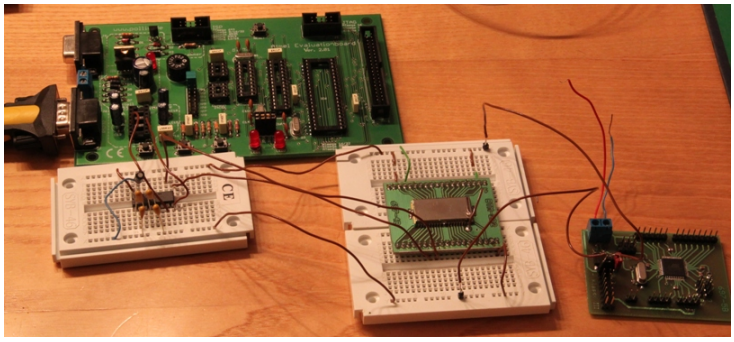
dern. Für die Erzeugung der Impulsbreite wird die Timereinheit 1 des ATmega mit den gleichen Einstellungen wie unter 2.3.8 beschrieben benötigt. In einer Schleife, bleibt der Ausgang PIN B1 so lange gesetzt, bis die Variable „Outputvalue“, die vom Timer jede ms um eins erhöht wird, der gewünschten Breite entspricht. Pro eingestellte Breite werden 5 Fotos geschossen. Den Start- und Endpunkt der Messung und die Schrittweite wird über die Variablen „STARTWITH“ und „MAXWITH“ bzw. „STEPSWITH“ im Programmcode festgelegt.
Bei der Kamera wurde diesmal das Programm „M“-Manuelle Belichtung benutzt. Die Einstellungen sind im Bild a der Abbildung 2.18 zu sehen. Am Port A wird bei dieser Messung der Index des Fotos ausgegeben und am Port C die Trigger Impulsbreite.
Bei der Messung mit dem Auto-Fokus wurde eine Schrittweite von 5 verwendet (Messbereich Trigger Impulsbreite 150 ms - 220 ms). Die Messung unter Manuellen-Fokus wurde mit einer Schrittweite von 1 ms durchgeführt (Messbereich Trigger Impulsbreite 30 ms -50 ms). Die Messergebnisse sind in der Tabelle 2.2 ersichtlich. Wenn der Trigger Impuls zu schmal ist, wird nur bei jedem zweiten Impuls ein Foto aufgenommen (deswegen abwechselnd 40% und 60% bei der MF Messung).

2.3.10 BTM-222 Bluetooth Funkmodul

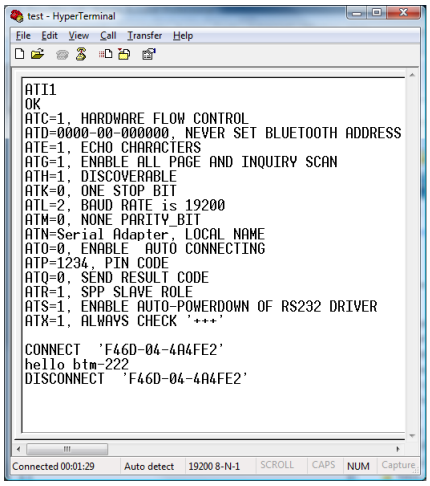
Für die Kommunikation über Bluetooth bietet sich das BTM-222 Bluetooth Modul der Firma Rayson an, das besonders durch seinen günstigen Preis zu empfehlen ist (ca. 11 €). Darüber hinaus handelt es sich um ein Klasse 1 Modul, womit eine theoretische Reichweite von 100 m überbrückt werden kann, und es unterstützt das SPP Profil (siehe 8.6). Um das Modul zu testen wurde die Schaltung 3.5 verwendet und die DIL Testplatine 4.4 erzeugt. Als Antenne dient ein Stück Draht mit der Länge $\lambda/4$ (ca. 31 cm). Versorgt wird das Modul mit 3.3 Volt und die Kommunikation findet über die serielle Schnittstelle des ATmegas μC mit den Verbindungseinstellungen 19200 Bit/s, 8 Datenbit, keine Parität und 1 Stop Bit statt. Eine Flusskontrolle ist nicht notwendig. Laut Datenblatt werden auch verschiedene Stromspar-Verbindungen unterstützt.
Die Einstellungen werden mit Hilfe von AT Kommandos durchgeführt, die im Datenblatt zu finden sind. Leider beschränken sich die Informationen im Datenblatt nur auf ein Minimum.
Um die Kommunikationsabläufe mit dem Modul besser zu verstehen, wurde dieses zuerst über einen RS232-USB Adapter an den Laptop angeschlossen und mit dem Hyperterminal darauf zugegriffen. Damit das Programm Hyperterminal auch unter neueren Windows

Messung	Trigger Impulsbreite	Anzahl geschossenen Fotos	Prozent vom Soll
AF-Saus		*59*	*84 %*
	150	4	80%
	155	3	60%
	160	4	80%
	165	3	60%
	175	2	40%
	180	4	80%
	185	4	80%
	190	5	100%
	195	5	100%
	200	5	100%
	205	5	100%
	210	5	100%
	215	5	100%
	220	5	100%
MF-Saus		*67*	*64 %*
	30	2	40%
	31	3	60%
	32	2	40%
	33	3	60%
	34	2	40%
	35	3	60%
	36	2	40%
	37	3	60%
	38	2	40%
	39	3	60%
	40	2	40%
	41	3	60%
	42	2	40%
	43	3	60%
	44	2	40%
	45	5	100%
	46	5	100%
	47	5	100%
	48	5	100%
	49	5	100%
	50	5	100%

Tabelle 2.2: Ergebnisse der Trigger Impulsbreite an EOS 500D (EF-S 18-55mm IS Objektiv)



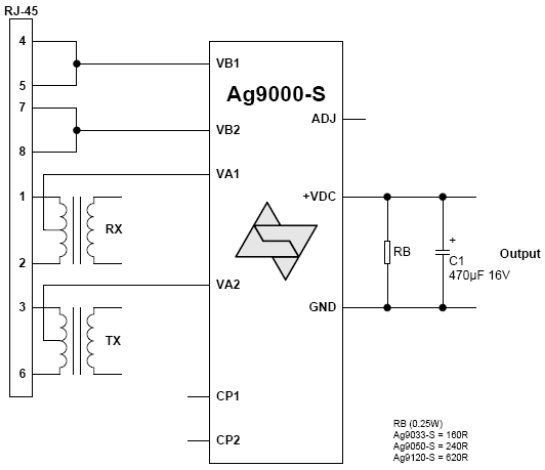
(a) Versuchsaufbau inklusive des MAX3232 mit der BTM-222 Testplatine 4.4. Die 3V3 Spannungsversorgung wurde vom Klasse III Modul genommen. Als RS232 Buchse diente das Atmel Evaluation Board von Pollin, dessen RxD und TxD Signale zum MAX3232 geleitet wurden. Bei weiteren Tests erwies sich diese Methode als ungeeignet, weil es zu Masseproblemen gekommen ist. Die Signale RxD, TxD und Masse wurde dann direkt über eine RS-232 Buche bezogen.



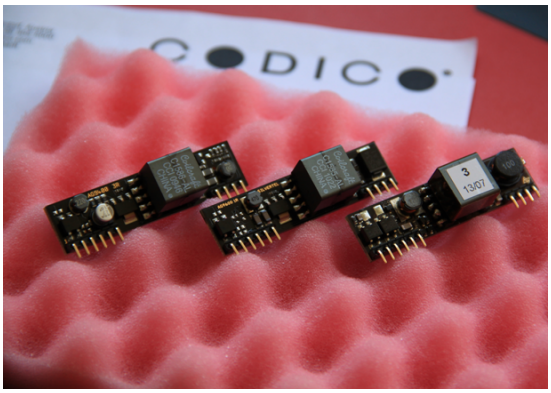
(b) Hyperterminalausgabe einer Testverbindung. Über den Befehl „ATI1“ lassen sich die aktuellen Einstellungen aufrufen.

Abbildung 2.19: Versuchsaufbau für das BTM-222 Modul und der Kommunikationsaufbau im Hyperterminal.

Versionen (ab Vista) verwendet werden kann, muss die „hypertrm.exe“ und die zwei DLL's „hticons.dll“ und „hypertrm.dll“ von einer älteren Windows Version kopiert werden, weil es standardmäßig nicht installiert ist. Hier wurden sie von einer Virtual PC XP Image Datei kopiert.
Für die unterschiedlichen Signalpegel wurde der MAX3232 von MAXIM verwendet. Der im Unterschied zum MAX232 auch mit 3V3 als Betriebsspannung und Signalpegel arbeiten kann. Die Schaltung für die Umsetzung wurde auf einem Steckbrett mit weiteren 5 Kondensatoren mit je 0,1 μF aufgebaut (siehe Datenblatt).
Für die erstmalige Bluetooth-Verbindung (Pairing) muss die Codefolge „1234“ übertragen werden, das ist aber nicht für die Kommunikation über die serielle Schnittstelle notwendig. Standardmäßig ist das BTM-222 so eingestellt, dass alle empfangenen Zeichen als Echo zurücksendet werden. Ein Befehl wird mit 0x0D (<CR>) abgeschlossen. Nicht akzeptierte Befehle werden mit „<CR><LF>ERROR<CR><LF>“ beantwortet (<LF> = 0x0A). Zum testen kann man den Befehl „AT<CR>“ an das Modul schicken, das mit „<CR><LF>OK<CR><LF>“ antworten muss. Die aktuellen Einstellungen können mit „ATI1<CR><LF>“ abgerufen werden. Sobald eine Verbindung mit dem BTM-222 Modul aufgebaut ist, sendet es „CONNECT <BluetoothID><CR><LF>“. Im jetzigen Modus werden die empfangen und gesendeten Daten durchgereicht. Sobald die Verbindung unterbrochen wird, sendet das Modul „DISCONNECT <BluetoothID><CR><LF>“ und es können wieder AT Befehle bearbeitet werden.
In der Abbildung 2.19 ist die Testschaltung des Bluetooth Moduls und ein Screenshot von Hyperterminal Kommunikation zu sehen. Als Bluetooth Partner fungierte das ASUS EEE Pad Transformer TF101, auf dem die selbstgeschrieben Android Software 5.5 läuft.



(a) Typische Schaltungsumsetzung [Datenblatt Ag9000-S]



(b) Die PoE Module im SIL Format von links nach rechts: Ag9412-2BR, AG9605-2BR, Ag9033-S

Abbildung 2.20: PoE Module der Firma Silvertel

2.3.11 PoE-Module

Für die Implementierung des PoE Protokolls und die Bereitstellung der Spannungsversorgung aus dem Ethernet, eignen sich die Module der Firma Silvertel. Sie sind kurzschlussgesichert und kostengünstig im praktischen SIL Format erhältlich. Außerdem erfüllen sie den IEEE 802.3af Standard und verfügen über eine hohe Isolationsspannung von 1500V zwischen Eingang und Ausgang. Der Spannungseingangsbereich kann zwischen 36V und 57V liegen. Die bereitgestellte Ausgangsspannung hängt vom gewählten Modul ab und kann je nach verwendeter Serie bis zu 24V betragen. Mit Hilfe eines Widerstandes können zusätzlich Anpassungen vorgenommen werden. Für Anwendungen mit einem höheren Leistungsbedarf empfiehlt sich die Verwendung der Module Ag5700 in Kombination mit Ag6700. Damit kann ein PD mit bis zu 200W versorgt werden.
Dank der Unterstützung von Herrn Steve Edwards (Managing Director, Silvertel) und Herrn Michael Schrutka (Produktmanager, Codico), standen jeweils 2 Module des Typs Ag9033-S, Ag9412-2BR und Ag9605-2BR zur Verfügung. Die letzten beiden Ziffern der Typenbezeichnung geben jeweils die bereitgestellte Ausgangsspannung der Module an (3.3V, 5V und 12V). Nur bei der Ag9000 Serie ist es möglich die PD Geräteklasse mit Hilfe eines externen Widerstandes zwischen den Pins CP1 und CP2 festzulegen. Bei den anderen zwei Serien wird die Signatur der Klasse 0 verwendet, wodurch maximal 12.95W zur Verfügung stehen.
Im Bild links der Abbildung 2.20 ist der typische Schaltungsaufbau ersichtlich. Wie man anhand der Beschaltung für die Pins VB1, VB2 und VA1, VA2 erkennt, kann das Modul die Ausgangsspannung sowohl von einer Midspan, als auch von einer Endspan Lösung erzeugen. R_B ist dabei der erwähnte Widerstand für die Anpassung der Ausgangsspannung.

2.3.12 ENC28J60

Um die Kommunikation über Ethernet zu ermöglichen wird der Ethernet Controller ENC28J60 von Microchip verwendet. Die Verbindung zum μC wird dabei über eine SPI Schnittstelle hergestellt. Versorgt wird er mit 3.3 V, die Eingänge sind auch 5 V tolerant. Für den Betrieb wird ein externer Quarz mit 25 MHz benötigt. Aus dieser Frequenz kann ein CLK Signal für andere Teilnehmer mit unterschiedlichen Prescaler generiert werden.

In dieser Arbeit wird davon aber nicht Gebrauch gemacht.
In der Abbildung 2.22 ist eine Übersicht über die wichtigsten Elemente des IC zu sehen. In ihm ist bereits eine PHY Einheit integriert, die für die richtigen Spannungspegel auf der Ethernet Leitung sorgt. Die korrekten Abläufe des MAC Protokolls ebenfalls direkt vom IC durchgeführt. Er beinhaltet 3 unterschiedliche Arten von Speicher:

- Control Register
- Ethernet Buffer
- PHY Register

Letztere werden in dieser Arbeit nicht verwendet, weil sie automatisch durch den Anschluss der LED B festgelegt werden (Half-Duplex). Ansonsten geben sie Auskunft über Zustände auf der Leitung bzw. kontrollieren das Verhalten der LED A und B. Über die Control Register werden die Einstellungen des ENC28J60 verwaltet, er benutzt dazu 4 Speicherbänke mit je 5 Bit Adressen. Die zu sendeten bzw. empfangenen Daten sind im 8 kByte großem Ethernet Buffer als FIFO Speicherkette abgelegt. Die Größe der einzelnen FIFO Ketten wird über das Control Register festgelegt. Der SPI Bus kann nur im Modus 0 verwendet werden. Das dazugehörige Zeitdiagramm ist in der Abbildung 2.23 zu sehen. Insgesamt können 7 Instruktionen an den ENC28J60 übertragen werden, die in der Tabelle in der Abbildung 2.26 zu sehen sind. Die ersten 3 Bits legen fest, um welche Instruktion es sich dabei handelt. Die letzten 5 Bits des Byte 0 geben die Speicheradresse an oder sind konstante Werte. Danach können mehrere Datenbytes folgen. Der genau Aufbau der einzelnen Instruktionen mit dem SPI Datendiagramm ist im Kapitel 4.2 SPI Instruction Set [ENC28J60 Datenblatt] nachzulesen.
Bei der Initialisierung des Controllers muss auch die verwendete MAC Adresse angegeben werden. In dieser Arbeit wird hierfür der Wert „00-1E-EC-06FB-E6“ verwendet. Es ist darauf zu achten, dass innerhalb eines Netzwerkes, die MAC Adressen einzigartig sind. Für kommerzielle Anwendungen muss ein eigener MAC Bereich unter IEEE registriert werden.
Der Aufbau eines Ethernet Frames ist in der linken Abbildung unter 2.21 zu sehen. Die Preamble und den Start-Frame Delimiter werden dabei nicht in den Receive Buffer kopiert, zusätzlich werden 2 Bytes für einen Zeiger auf das nächste Paket und 4 Bytes als Statusinformation hinzugefügt (siehe Abbildung 2.24). Beim Auslesen der Daten ist darauf zu achten, dass die Zeiger im Register „ERXRDPT“ und „ERDPT“ am Schluss auf den Beginn des nächsten Datenpaketes gesetzt werden, damit der Speicher wieder freigeben wird. In dieser Arbeit wird das automatische Inkrementieren des „ERDPT“ Registers verwendet, damit eventuelle Überläufe des FIFO nicht berücksichtigt werden müssen.
Die restlichen benötigten Schichten des OSI Modelles müssen in der Software des μC implementiert werden. Als Transportprotokoll zwischen dem Windows Programm und dem ENC28J60 wurde UDP (Schicht 4 im OSI Modell) ausgewählt (Aufbau siehe Tabelle 2.4) [Wikipedia - UDP]. Dabei handelt es sich um ein verbindungsloses Protokoll, ohne eine Garantie, dass das Paket beim Empfänger ankommt. Da UDP auf IPv4 (Schicht 3 im OSI Modell) aufbaut, ist dieses ebenfalls zu implementieren (siehe Tabelle 2.3) [Wikipedia - IP]. Jedoch wird diese nur in der für dieser Arbeit benötigten Umfang implementiert (keine Fragmentierung).

Erste Tests haben ergeben, dass moderne Betriebssysteme (z.B. Windows Vista) einen ARP Request ausführen, bevor sie ein UDP Paket senden, um ihre ARP Tabellen zu aktualisieren. Falls keine ARP Antwort eintrifft, wird das UDP nicht gesendet. Dadurch muss dieses Protokoll ebenfalls implementiert werden. Der Aufbau eines ARP Paketes ist

Feld	Größe (Bit)	Beschreibung
Version	4	IP Version (IPv4 = 4)
IP Header Length	4	Größe des IP Headers * 4 Byte
Type of Service	8	Feld für Priorisierung
Total Length	16	Länge des gesamten IP Paketes (inkl. IP Kopf)
Identification	16	Einzigartige ID der fragmentierten Pakete.
Flags	3	Kontrolle für die Fragmentierung (In dieser Arbeit immer 010 -> keine Fragmentierung)
Fragment Offset	13	Position der Daten innerhalb einer fragmentierten Übertragung (In dieser Arbeit immer 0)
Time to Live	8	Gibt die Lebensdauer eines IP Paketes an
Protocol	8	Verwendetes Protokoll für die höhere Schicht (UDP = 17)
Header Checksum	16	Checksumme für den Header
Source Address	32	IP Adresse des Senders
Destination Address	32	IP Adresse des Empfängers
Options und Padding (optional)	32	maximal 40 Bytes immer ein vielfaches von 4 Bytes
Daten	n	maximal 2^{16}

Tabelle 2.3: Aufbau des IPv4

Feld	Größe (Bytes)	Beschreibung
Quell-Port	2	Bestimmt die Anwendung die das Paket gesendet hat
Ziel-Port	2	Bestimmt für welche Anwendung das Paket bestimmt ist
Längenfeld	2	Länge des UDP Paketes (Header + Daten)
Prüfsummenfeld	2	Es kann eine Prüfsumme gesendet werden (Wert 0, wenn keine notwendig ist)
Daten	n	maximal $2^{16} - 8$

Tabelle 2.4: Aufbau des UDP

Feld	Größe (Bytes)	Beschreibung
Hardwareadresstyp	2	Typ der MAC-Adresse (1 für Ethernet)
Protokolladresstyp	2	angeforderter Protokolltyp (IPv4, IPv6, etc.)
Hardwareadressgröße	1	Größe der MAC Adresse (6 für Ethernet)
Protokolladressgröße	1	Größe des Protokolls (4 für IPv4)
Operation	2	Unterscheidung ob es eine Anforderung (1) oder Antwort(2) ist
Quell-MAC-Adresse	6	MAC Adresse des Gerätes der Anforderung, bzw. Antwort
Quell-IP-Adresse	4	IP Adresse des Gerätes der Anforderung, bzw. Antwort
Ziel-MAC-Adresse	6	Für Anforderung nicht definiert, bei Antwort die Quell-MAC-Adresse vom Anforderungs ARP
Ziel-IP-Adresse	4	IP Adresse des gesuchten Hosts bei Anforderung, bei Antwort die IP-Adresse vom Anforderungs ARP

Tabelle 2.5: Aufbau des ARP

in der Tabelle 2.5 zu finden. Die Daten befinden sich in den Nutzdaten des Ethernetframes [Wikipedia - ARP]. Unterschieden wird zwischen IPv4 und ARP durch den Wert im Type Feld des Ethernet Frames (0x0800 IPv4, 0x0806 ARP).
Der ENC28J60 besitzt auch die Möglichkeit nach eingehenden Paketen zu filtern. Dadurch kann der beschränkte Receive Buffer besser genutzt werden. In dieser Arbeit wird jedoch kein Filter verwendet (Control Register ERCFCON = 0x00), weil davon ausgegangen wird, dass im Test-Netzwerk kein hoher Traffic stattfindet.
Beim Senden werden von der MAC Einheit des ENC28J60 die Preamble und der Start Frame Delimiter automatisch erzeugt. Er wurde zusätzlich so konfiguriert das die Auffüll Bytes automatisch hinzugefügt werden, wenn die Datenmenge 46 Bytes nicht übersteigt. Außerdem soll der ENC28J60 das Paket auf einmal versenden und die Frame Check Sequence vom Ethernet Paket selber berechnen und automatisch anfügen. Zusätzlich zu den Datenbytes muss ein Control Byte der Sendeeinheit übergeben werden, mit dessen Hilfe die Übertragungseinstellungen durchgeführt werden (siehe Abbildung 2.25). Sobald das Senden abgeschlossen ist, geben 7 Status Bytes genauere Auskunft über den Erfolg bzw. eventuelle Fehler. Die IP Adresse wurde in ein char Feld gespeichert, damit sie für spätere Anwendungen auch im laufenden Betrieb geändert werden kann. Aktuell hat sie den Wert „192.168.20.50“.

Zum testen der gesendeten bzw. empfangenen Signale über das Ethernet, eignet sich Wireshark, das kostenlos zur Verfügung steht. Damit können alle empfangen bzw. gesendet Ethernet Pakete von einem Interface angezeigt werden. Vor allem beim kontrollieren der vom Modul gesendeten Pakete hat es sich als sehr nützlich erwiesen, weil Fehler im Protokoll sofort angezeigt werden.

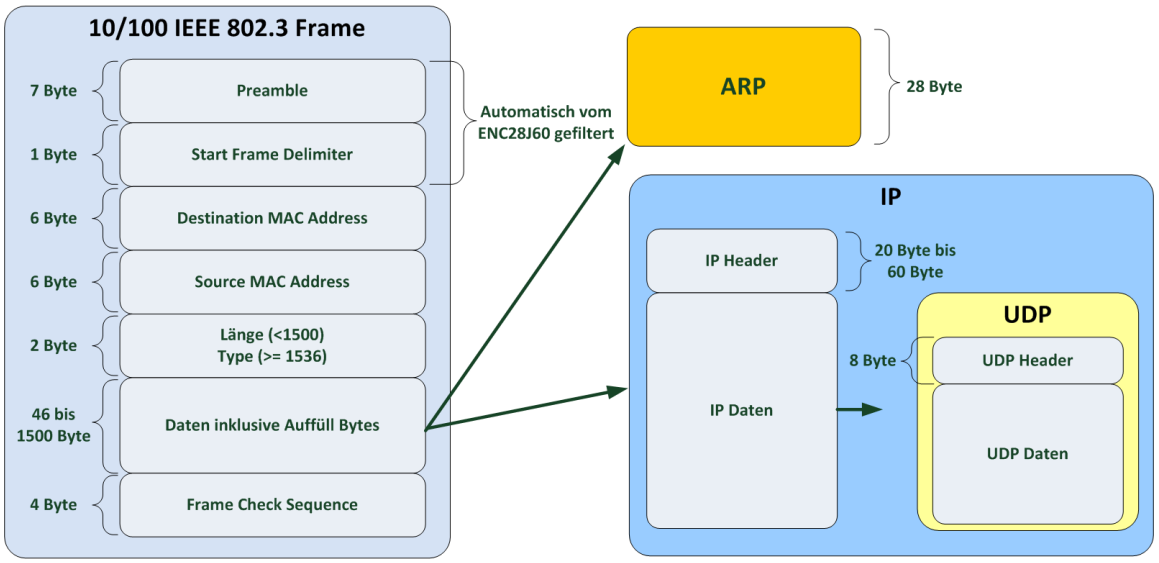


Abbildung 2.21: Aufbau eines Ethernet Frames

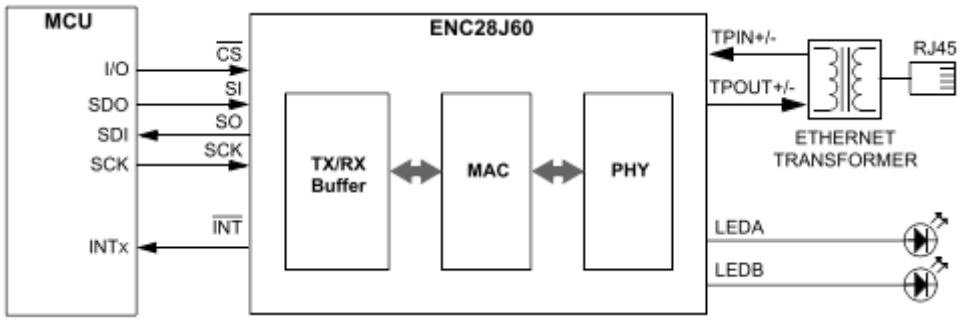
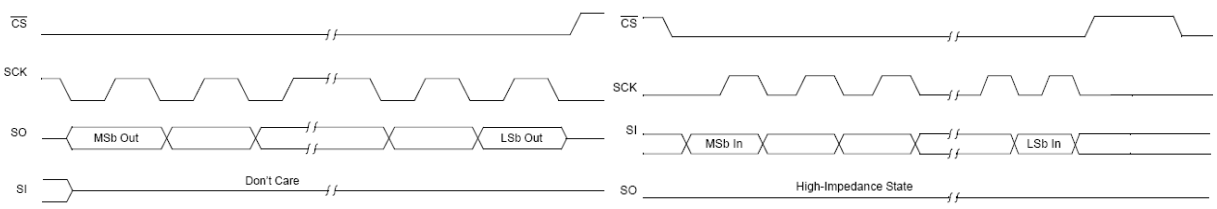


Abbildung 2.22: Schaltungsübersicht ENC28J60 [ENC28J60 Datenblatt]



(a) Read (b) Write

Abbildung 2.23: SPI Verbindung des ENC28J60 [ENC28J60 Datenblatt]

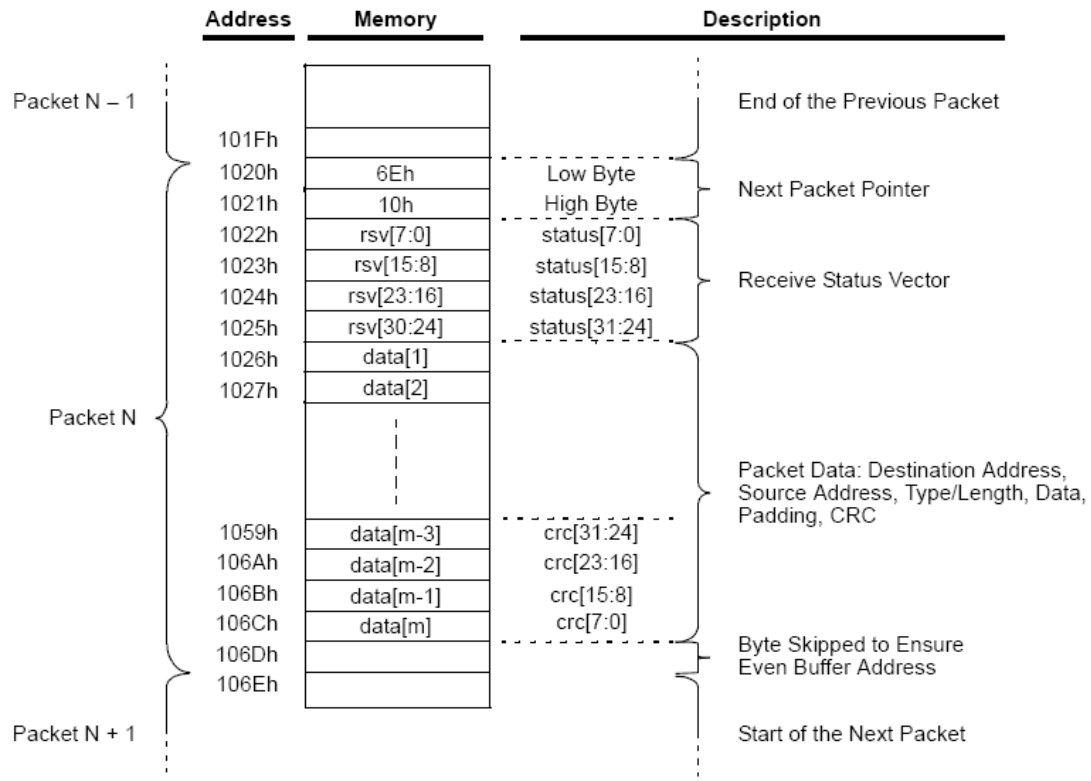


Abbildung 2.24: Receive Buffer Struktur des ENC28J60 [ENC28J60 Datenblatt]

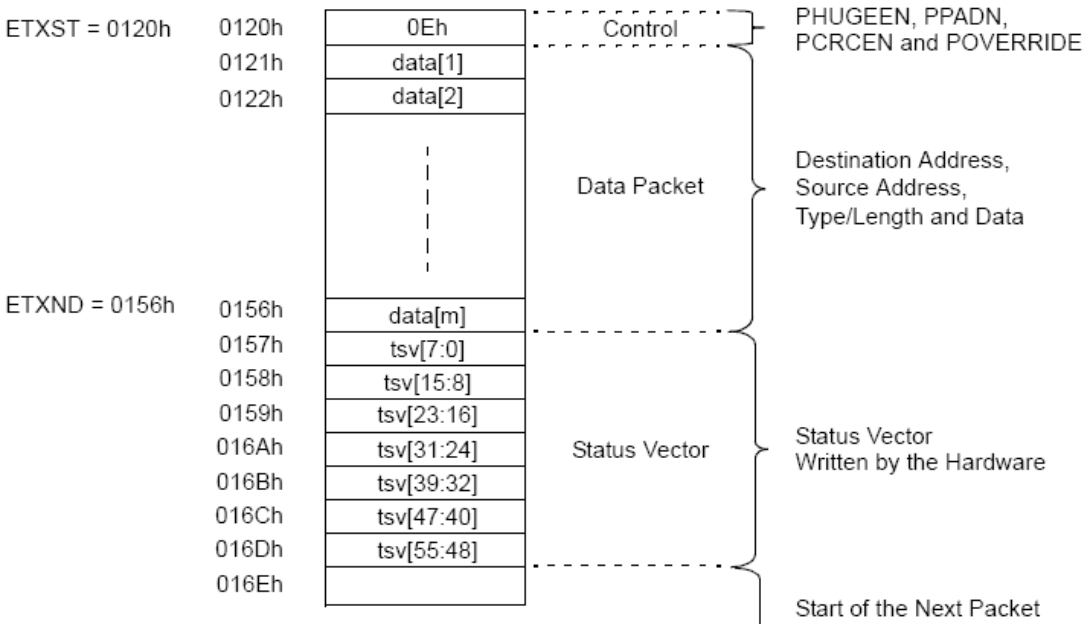


Abbildung 2.25: Transmit Buffer Struktur des ENC28J60 [ENC28J60 Datenblatt]

Instruction Name and Mnemonic	Byte 0								Byte 1 and Following							
	Opcode			Argument					Data							
Read Control Register (RCR)	0	0	0	a	a	a	a	a	N/A							
Read Buffer Memory (RBM)	0	0	1	1	1	0	1	0	N/A							
Write Control Register (WCR)	0	1	0	a	a	a	a	a	d	d	d	d	d	d	d	d
Write Buffer Memory (WBM)	0	1	1	1	1	0	1	0	d	d	d	d	d	d	d	d
Bit Field Set (BFS)	1	0	0	a	a	a	a	a	d	d	d	d	d	d	d	d
Bit Field Clear (BFC)	1	0	1	a	a	a	a	a	d	d	d	d	d	d	d	d
System Reset Command (Soft Reset) (SRC)	1	1	1	1	1	1	1	1	N/A							

Legend: a = control register address, d = data payload.

Abbildung 2.26: SPI Instruktionen des ENC28J60 [ENC28J60 Datenblatt]

3 Schaltungseingabe

3.1 Fernauslöser Aufsatz

Bei dem Fernauslöser Aufsatz handelt es sich um einen Aktor der Klasse I, der an die zur Verfügung stehenden Portleistungen des μC eine Klasse III Steuereinheit angeschlossen wird. Sobald am Ausgang ein High Pegel anliegt, wird der Transistor im Optokoppler durch die LED Diode leitend und die Kontakte der 2,5 mm Klinkenbuchse werden geschlossen.

Für die Berechnung der Vorwiderstände wird ein typischer Strom in Flussrichtung von $I_f = 20\ mA$ und ein Spannungsabfall an der Sendediode von $U_f = 1.2\ V$ angenommen.

$$R_v = \frac{5\ V\ -\ 1.2\ V}{20\ mA} = 190\ \Omega \text{ gewählt wurden } 220\ \Omega$$

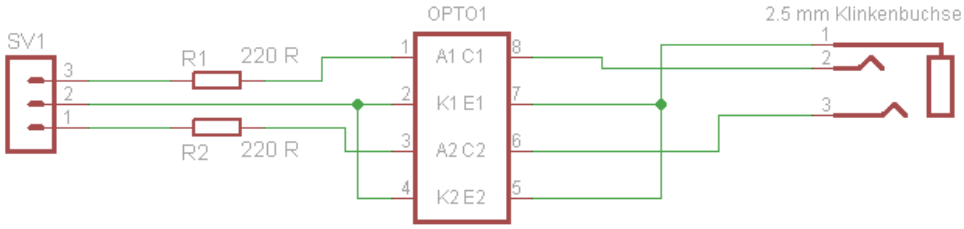


Abbildung 3.1: Schaltung für den Fernauslöseraufsatz

3.2 Ethernet mit PoE Modul

Die ersten Versuche einen Controller über das Ethernet mit gleichzeitiger Bereitstellung einer PoE Spannungsversorgung zu verbinden, wurden mit der Schaltung 3.2 durchgeführt. Die Berechnung für die Vorwiderstände der integrierten LEDs in der Netzwerkbuchse erfolgt äquivalent zur Berechnung aus dem Kapitel 3.1. Im Datenblatt der Netzwerkbuchse wird angegeben, dass beide Durchflussspannungen der LEDs jeweils 2.1 V betragen und dabei ein Strom in Flussrichtung von 20 mA fließt.

$$R_v = \frac{3.3\ V\ -\ 2.1\ V}{20\ mA} = 60\ \Omega \text{ gewählt wurden } 62\ \Omega$$

Die Schaltung arbeitet mit 3V3 und kann über verschiedene Spannungsquellen gespeist werden. Die unterschiedlichen Quellen werden über Jumper mit den Versorgungsleitungen der Schaltung verbunden.

- Direkt vom PoE Modul bei der Verwendung des Ag9033-S
- Über eine externe Quelle
- Über den 3V3 Spannungsregler wenn ein anderes PoE Modul verwendet wird

Für die Taktfrequenz des μC wurde ein 12 MHz Quarz verwendet.

3.3 Adapter für RJ45-Buchse

Um leichter die Funktionalität der Ethernet Schaltung zu testen, wurde diese zuerst auf einem Steckbrett aufgebaut. Weil sich die RJ45-Buchse nicht direkt für das Steckbrett verwendet werden konnte, wurde ein Adapter entwickelt 3.3.

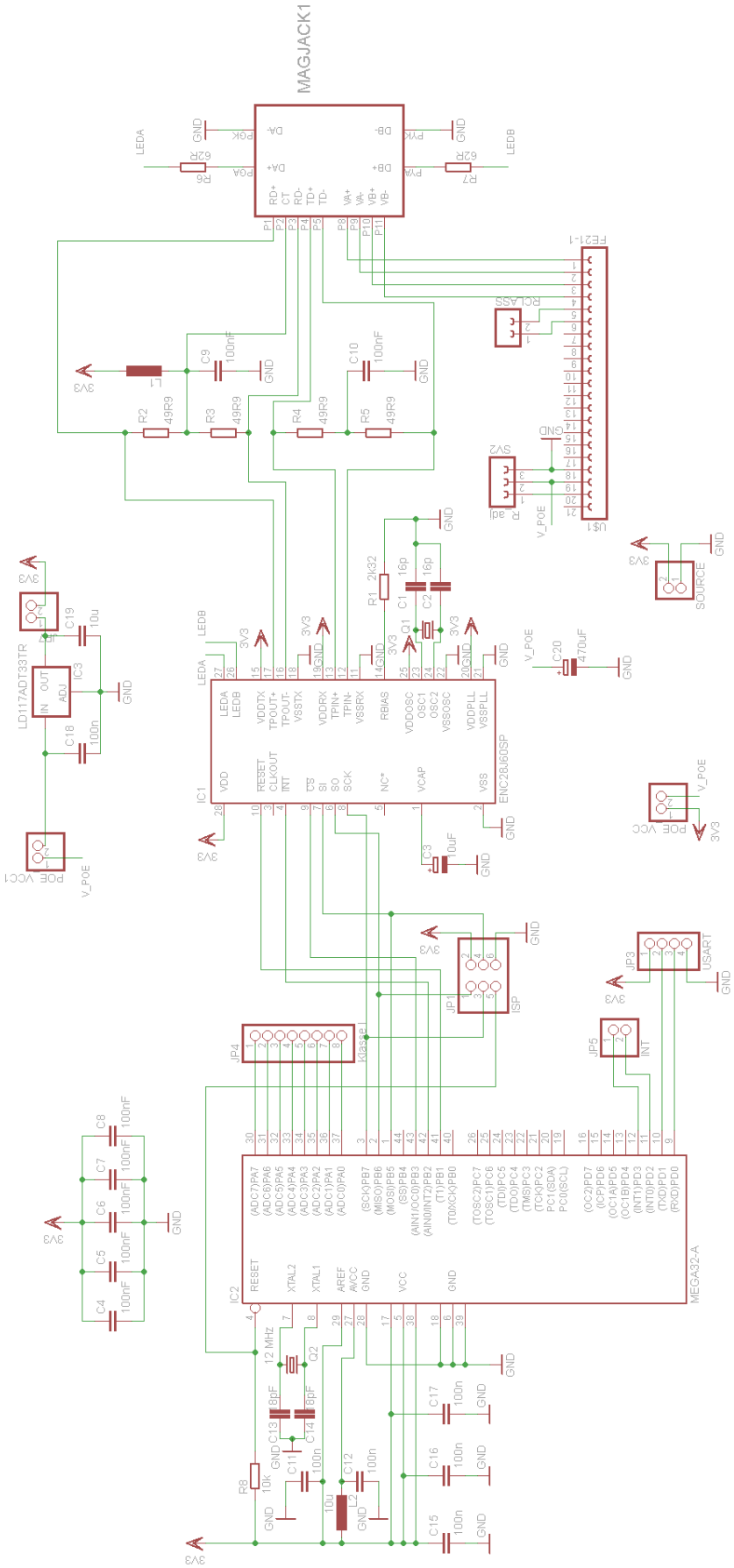


Abbildung 3.2: Schaltung für ein Ethernet Modul

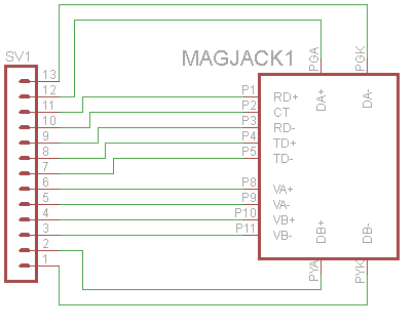


Abbildung 3.3: Schaltung für den Adapter der RJ45-Buchse

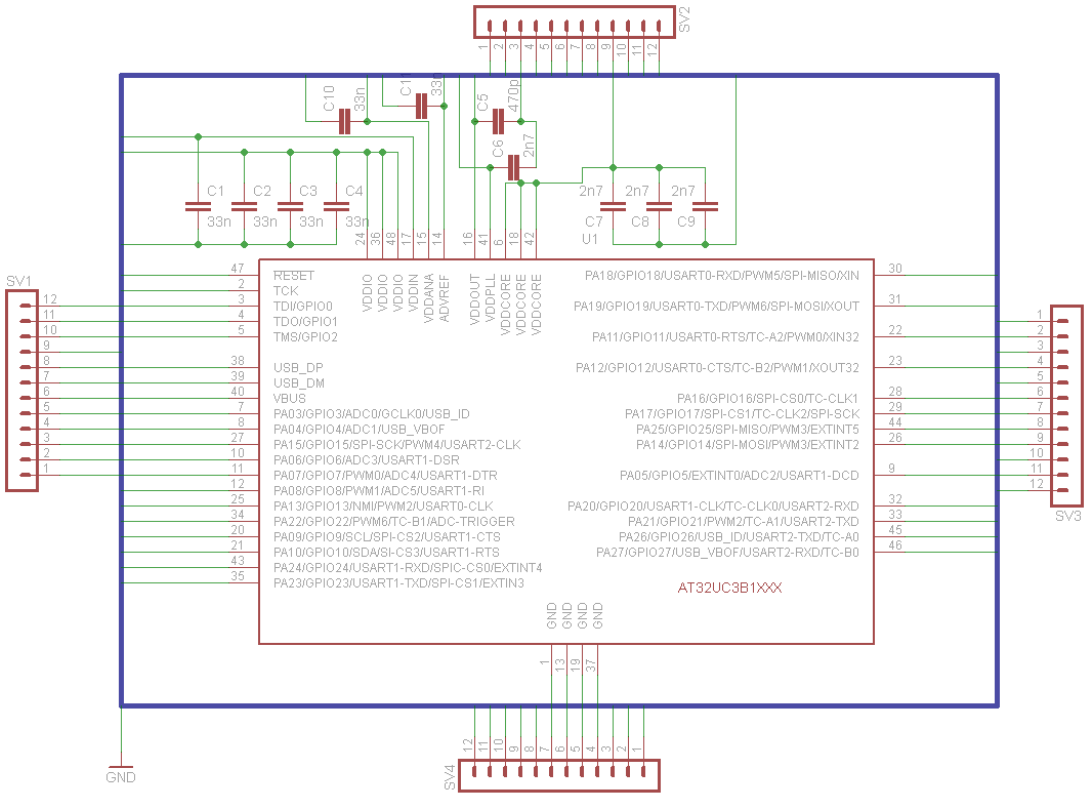


Abbildung 3.4: Schaltung für den AT32UC3B1256 Adapter

3.4 Adapter für AT32UC3B1256

Für die Tests am AT32UC3B1256 wurde die Schaltung in der Abbildung 3.4 benutzt. Grund für die Entwicklung einer Platine war die Tatsache, dass der μC nur im unhandlichen SMD Gehäuse erhältlich ist. Als Informationsquelle diente dabei das AVR32 UC3B Schematic Checklist Dokument [Atmel - AVR32715]. Versorgt wird die Schaltung mit einer 3V3 Spannungsquelle.

3.5 Bluetooth Testschaltung

Für Tests mit des BTM-222 Moduls wurde folgende Schaltung und die dazugehörige Platine 4.4 entwickelt.

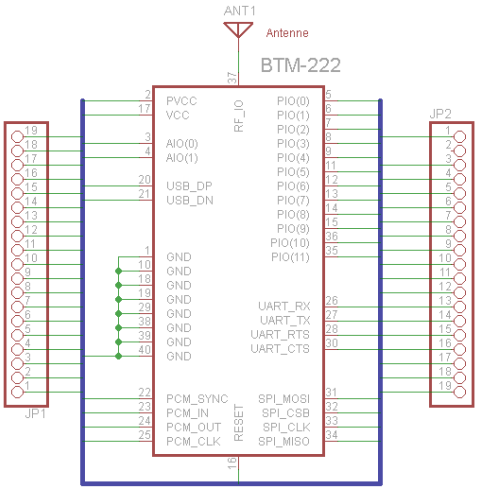


Abbildung 3.5: Schaltung für den BTM-222 Adapter

JP8	JP9	Resultat
L	L	Klasse IV (Ethernet)
L	H	Klasse IV (Bluetooth)
H	L	Klasse V (Ethernet)
H	H	Klasse V (Bluetooth)

Tabelle 3.1: Übersicht Jumpereinstellungen

3.6 Klasse III

Klasse III Module haben die Aufgaben Portleitungen für Klasse I, eine TWI Schnittstelle für Klasse II und eine Schnittstelle für Erweiterungen zu einer höheren Klasse bereitzustellen. Die dazugehörige Schaltung ist in der Abbildung 3.6 zu finden. Als Taktfrequenz für die μC wurde ein 12 MHz Quarz verwendet und die Versorgungsspannung wurde mit 3V3 gewählt. Sie wird über einen Spannungsregler geliefert. Der ATmega324a verfügt über 2 serielle Schnittstellen. Eine wird für die Bereitstellung von Debug Information oder als zusätzliche Interruptquelle genutzt und mit Hilfe der zweiten erfolgt ein Datenaustausch zu einem Bluetooth 3.5 oder Ethernet Modul 3.2. Die Jumper 8 und 9 dienen zur Einstellung der Klasse vom Modul (siehe Tabelle 3.1). Um die Funktionalität auch bei nicht angesteckten Jumper zu garantieren, werden die interne Pull Up Widerstände des ATmega324a verwendet. Diese machen theoretisch auch eine VCC Verbindung der Jumper überflüssig. Für Klasse I Module stehen die 8 Portleitungen des Portes A zur Verfügung. Bei der Verwendung von Bluetooth als Klasse IV Gerät, muss jedoch eine Leitung für das Reset (Pin A7) verwendet werden.

Für ein Klasse V Modul wurde die SPI Schnittstelle auf eine Stiftleiste geführt. Gleichzeitig wird das SPI auch für den ISP Adapter genutzt, für den eine eigene 2x3 Pinleiste bereitsteht. Externe Module können über die Stiftleiste JP16 oder über die Versorgungspinpaare der einzelnen Schnittstellen versorgt werden.

3.7 Stücklisten

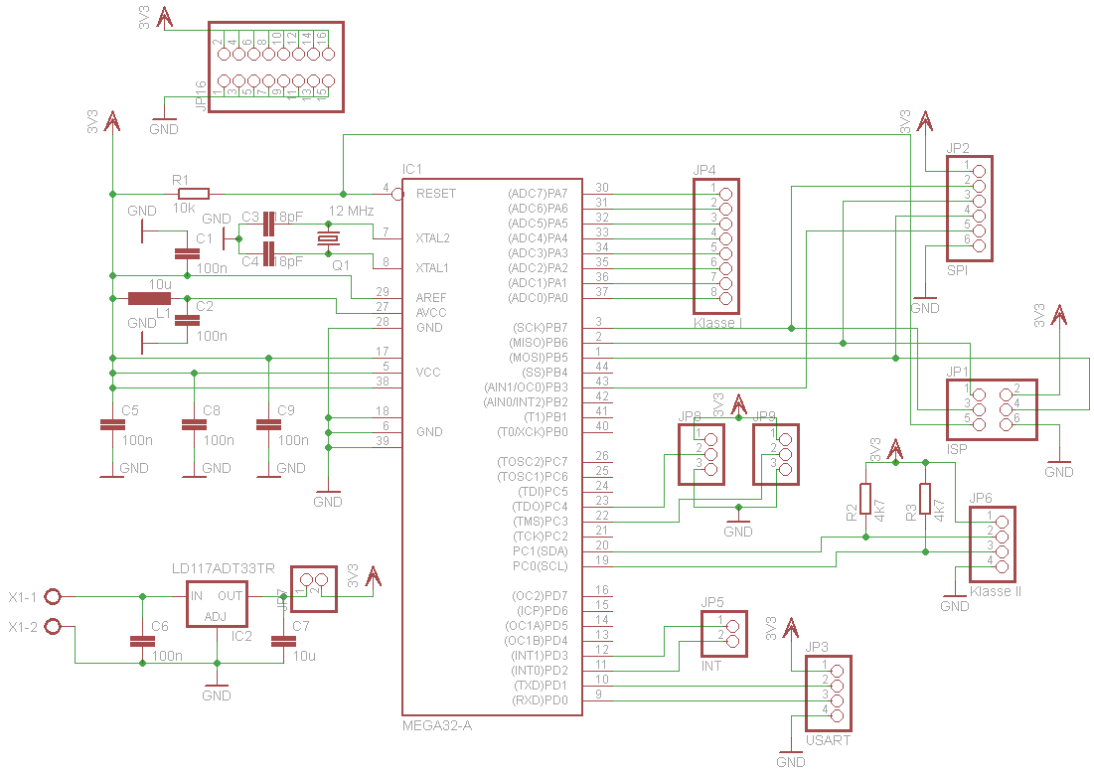


Abbildung 3.6: Schaltung für ein Klasse III Modul

Anzahl	Bezeichnung	Preis pro Stück	Link
1	Optokoppler (ACPL-227-500E)	0,486 €	RS Components
2	Widerstände 220 Ω (0805)	0,021 €	RS Components
1	Klinkenbuchse 2,5 mm	0,898 €	Farnell
3 Pins	Stiftleiste		

Tabelle 3.2: Stückliste Fernauslöser Aufsatz

Anzahl	Bezeichnung	Preis pro Stück	Link
1	RJ45 POE Buchse (SI-52008-F)	8,08 €	Farnell
13 Pins	Stiftleiste		

Tabelle 3.3: Stückliste RJ45-Buchse Adapter

Anzahl	Bezeichnung	Preis pro Stück	Link
1	BTM-222	13,95 €	CSD Electronics
38 Pins	Stiftleiste		

Tabelle 3.4: Stückliste Bluetooth Testschaltung

Anzahl	Bezeichnung	Preis pro Stück	Link
1	AT32UC3B1256	9,04 €	RS Components
6	Kondensator 33nF (0805)	0,075 €	RS Components
1	Kondensator 470pF (0805)	0,056 €	RS Components
4	Kondensator 2,7nF (0805)	0,011 €	Farnell
48 Pins	Stiftleiste		

Tabelle 3.5: Stückliste AT32UC3B1256 Adapter

Anzahl	Bezeichnung	Preis pro Stück	Link
1	Atmega32 (TQFP 44)	4,63 €	RS Components
1	3V3 (LD117ADT33TR - DPAK)	0,42 €	RS Components
1	Widerstand 10k (0805)	0,02 €	RS Components
2	Widerstand 4k7 (0805)	0,009 €	RS Components
1	Kondensator 10uF (0805)	0,062 €	RS Components
1	Spule 10uH (0805)	0,128 €	RS Components
1	12 MHz (HC49S)	0,541 €	RS Components
2	Kondensator 18pF (0805)	0,031 €	RS Components
6	Kondensator 100nF (0805)	0,036 €	RS Components
61 Pins	Stiftleiste		

Tabelle 3.6: Stückliste Klasse III

Anzahl	Bezeichnung	Preis pro Stück	Link
1	Atmega32 (TQFP 44)	4,63 €	RS Components
1	3V3 (LD117ADT33TR - DPAK)	0,42 €	RS Components
1	Widerstand 10k (0805)	0,02 €	RS Components
1	Kondensator 10uF (0805)	0,062 €	RS Components
1	Spule 10uH (0805)	0,128 €	RS Components
1	12 MHz (HC49S)	0,541 €	RS Components
2	Kondensator 18pF (0805)	0,031 €	RS Components
6	Kondensator 100nF (0805)	0,036 €	RS Components
1	25 MHz (HC49S)	0,518 €	RS Components
1	Spule (EMV)	0,17 €	RS Components
1	RJ45 POE Buchse (SI-52008-F)	8,08 €	Farnell
1	Widerstand 2k32 (0805-1%)	0,318 €	RS Components
1	Kondensator 10uF (Radial EE)	0,766 €	RS Components
2	Kondensator 16pF (0805)	0,019 €	RS Components
4	Widerstand 49R9	0,044 €	RS Components
2	Widerstand 62R (0805)	0,014 €	RS Components
1	ENC28J60SP	3,3 €	RS Components
7	Kondensator 100nF (Radial)	0,103 €	RS Components
1	Kondensator 470uF (Radial EE)	0,994 €	RS Components
28 Pins	Stiftleiste		
26 Pins	Buchsenleiste		

Tabelle 3.7: Stückliste Ethernet mit PoE Modul

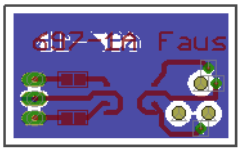


Abbildung 4.1: Layout des Fernauslöser Aufsatzes

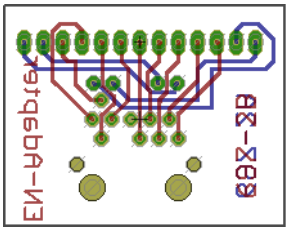


Abbildung 4.2: Layout RJ45-Buchse Adapter

4 Schaltungsumsetzung

4.1 Layout für Fernauslöser Aufsatz

Die Unterseite wurde als Massefläche ausgeführt und mit dem PIN 2 der Stiftleiste verbunden.
Beim anlöten des Optokopplers wurde festgestellt, dass ein falsches Layout des Gehäuses verwendet wurde und sich die Pads zu knapp am IC Gehäuse befanden. Damit die Platine für Tests trotzdem verwendet werden konnte, wurde mehr Lötzinn verwendet, um eine leitende Verbindung mit den Pads zu erzeugen. Wenn das nicht gereicht hat, wurde der Pin direkt an die Leiterbahn angelötet. Dazu musste das Laminat heruntergekratzt werden.

4.2 Layout für RJ45-Buchse Adapter

4.3 Layout Adapter für AT32UC3B1256

Die Unterseite der Platine wurde als Massefläche ausgeführt.

4.4 Layout für Bluetooth Testplatine

In der Abbildung 4.4 ist das fertige Layout zu sehen. Dabei wurden die Oberseite und die Unterseite der Platine als Massefläche verwendet. Als $\lambda/4$ Antenne dient ein Stück Draht (ca. 31 cm).

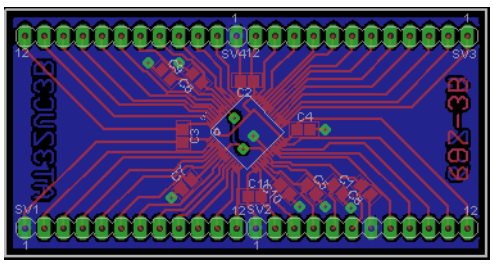
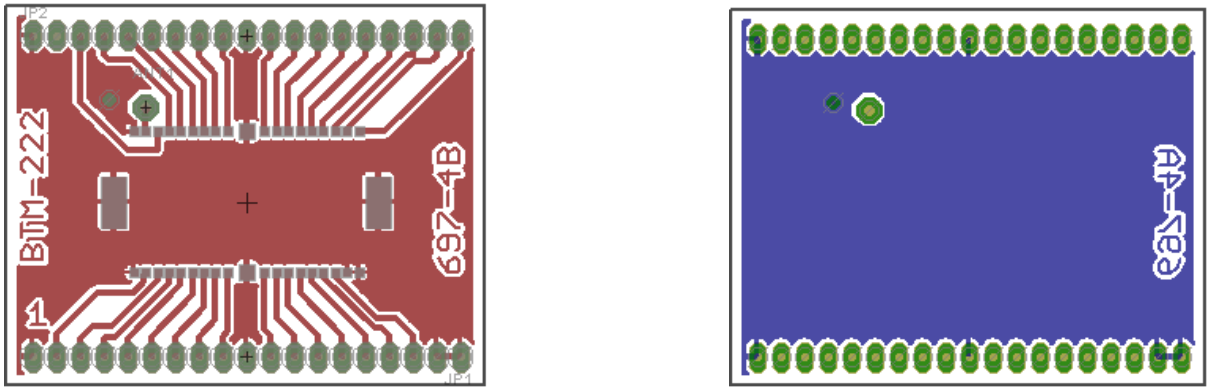


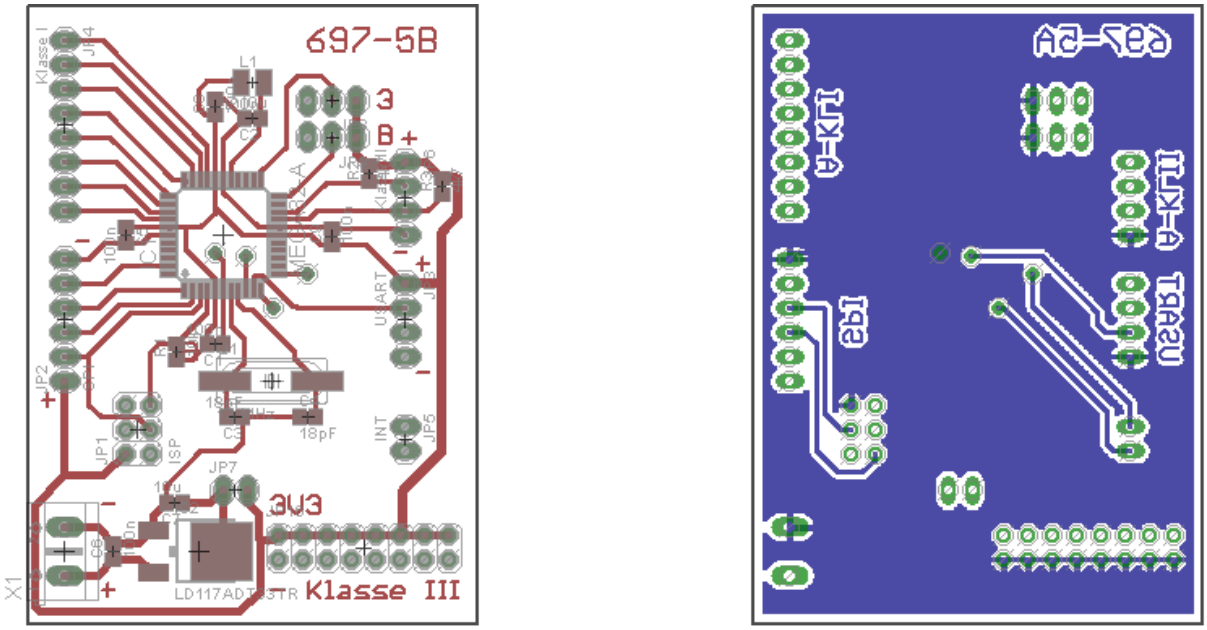
Abbildung 4.3: Layout AT32UC3B1256 Adapter



(a) Platinenoberseite

(b) Platinenunterseite

Abbildung 4.4: Layout für BTM-222 Testplatine



(a) Platinenoberseite

(b) Platinenunterseite

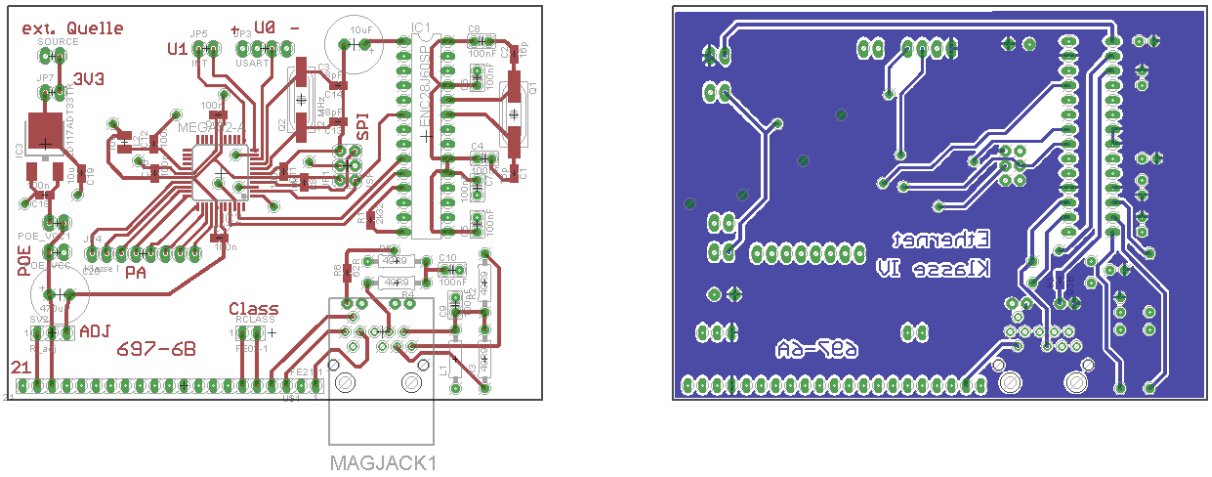
Abbildung 4.5: Layout für ein Klasse III Modul

4.5 Layout für Klasse III Modul

In der Abbildung 4.5 ist das fertige Layout zu sehen. Die Unterseite wurde als Massefläche verwendet.

4.6 Layout für Ethernet Modul

In der Abbildung 4.6 ist das fertige Layout zu sehen. Die Unterseite wurde als Massefläche verwendet. Wie unter [ENC28J60 Datenblatt] beschrieben, wurde ein SMD Widerstand für den Ausgang des RBIAS Pins verwendet und so nahe wie möglich am Pin positioniert, damit keine kapazitiven Einkopplungen von benachbarten Leitungen entstehen. Auch die Sützkondensatoren an den Spannungseingängen wurden so knapp wie möglich an die Pins positioniert.
Das Modul wird mit 3,3 Volt versorgt, die auf verschiedenen Arten bereitgestellt werden



(a) Platinenoberseite

(b) Platinenunterseite

Abbildung 4.6: Layout für das Ethernet Modul

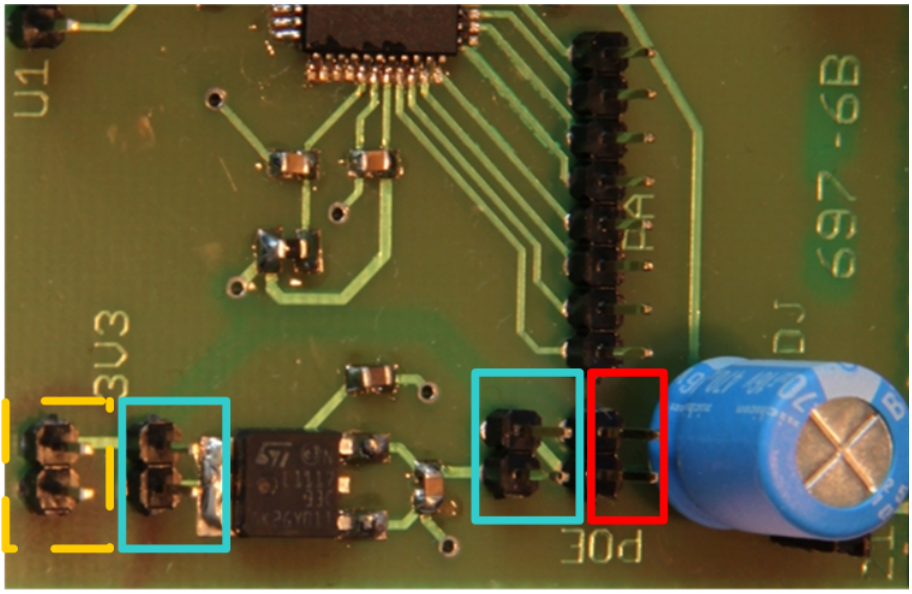


Abbildung 4.7: Spannungsversorgung am Ethernet Modul

können. Eine Möglichkeit ist eine externe Spannungsversorgung (z.B. ein Klasse III Modul), dazu dienen die 2 Pins für V und GND in der gelben Markierung der Abbildung 4.7. Für die Speisung über ein PoE fähiges Netzwerk gibt es zwei Möglichkeiten. Wenn direkt 3,3 Volt vom PoE Modul erzeugt werden (Ag9033-S), wird die Versorgungsleiterbahn der Platine durch eine Brücke zwischen den Pins in der roten Markierung mit dem Ausgang des PoE Moduls verbunden. Für die zwei anderen Module (Ag9412-2BR und AG9605-2BR) müssen die Pins in der blauen Markierung mit einem Jumper verbunden werden. Dabei wird die höhere Spannung vom PoE mit Hilfe eines Spannungsreglers auf 3,3 Volt heruntergeregelt.

5 Fertigung vom Prototyp

5.1 Protokoll

Um Daten zwischen den Klassenmodulen III - VI auszutauschen werden diese in Container verpackt und übermittelt. Es werden dabei die 2 Arten Command und Response Container unterschieden, deren Strukturen in der Tabelle 5.1 ersichtlich sind. Die Kommunikation erfolgt in 3 Phasen 5.2, wobei die Data Phase optional ist, damit ähnelt es der PTP Kommunikation 8.9.
Auch hier wird eine 16 Bit TransaktionID verwendet, um die einzelnen Antworteten den richtigen Aufträgen zuzuweisen. Für die Vergabe der TransaktionsID ist jedes Steuersystem selber verantwortlich. Durch den Datenaustausch zwischen den Steuer- und Eingabeeinheiten kann es vorkommen, dass eine TransaktionID doppelt vergeben wird. Das stellt jedoch kein Problem da, weil zwischen ein- und ausgehenden Transaktionen unterschieden wird. Eingehende Transaktionen sind Befehle die von einem anderen Klassenmodul gesendet werden (z.B. Befehl zum Setzen eines Port Pins) und ausgehende Transaktionen wurden vom Modul selbst initiiert (z.B. Änderung der Belichtungsdauer).
Der Command Code ist in vier Bereiche von je einem Byte aufgeteilt. Im vierten Byte befindet sich die unter 5.2 beschriebene DeviceID des Auftragsmoduls. Das dritte Byte beinhaltet die Zielklasse des Befehles (Mögliche Werte 1 - 6). Die Werte des zweiten Bytes werden abhängig vom dritten Byte interpretiert und es ist im Moment nur für die Klassen I, II und V definiert (Tabelle 5.2). Anhand des ersten Bytes wird unterschieden welcher Befehl ausgeführt werden soll.
Das letzte Byte des Response Codes gibt an ob es sich um eine Daten (Byte = 0) oder eine Response (Byte = 0xFF) Phase handelt. Die restlichen 3 Bytes geben zusätzliche Informationen zurück.
Eine Liste der verwendeten Response und Command Codes ist im Anhang M zu finden. Die Anzahl und der Aufbau der benötigten Parameter sind in den Funktionen „CheckCommand“ und „LogicUnit“ der jeweiligen Protokoll.c Datei implementiert. Innerhalb eines Containers werden die einzelnen Felder, wie beim PTP 8.9, im little Endian Format übertragen.

Feld	Bytes	Datentyp
TransaktionsID	2	UINT16
CommandCode	4	UINT32
Anzahl Parameter	2	UINT16
n Parameter	1	beliebig

Feld	Bytes	Datentyp
TransaktionsID	2	UINT16
ResponseCode	4	UINT32
Anzahl Parameter	2	UINT16
n Parameter	1	beliebig

Tabelle 5.1: Aufbau des Command (links) und des Response (rechts) Containers.

Klasse	Bedeutung
Klasse I	Port Pin
Klasse II	TWI Adresse
Klasse V	USB Host Nummer

Tabelle 5.2: Bedeutung des 2. Bytes des CommandCodes in Abhängig vom 3. Byte

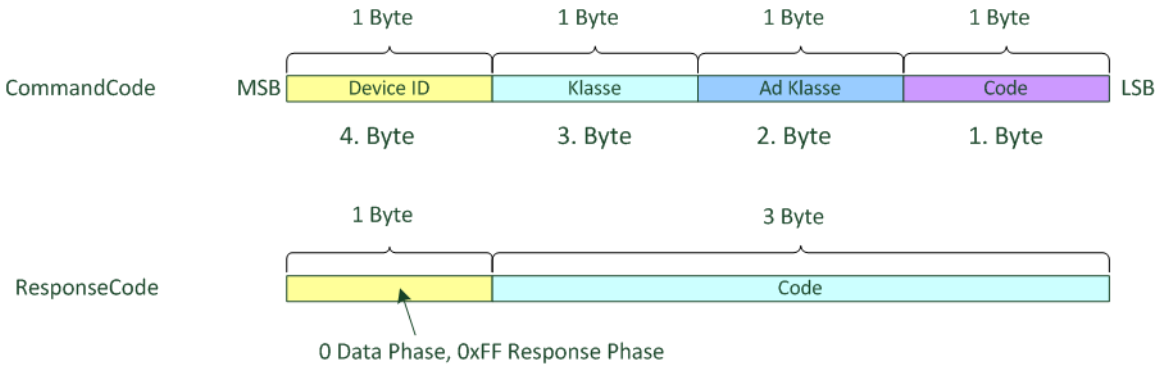


Abbildung 5.1: Aufbau des Command- und ResponseCodes

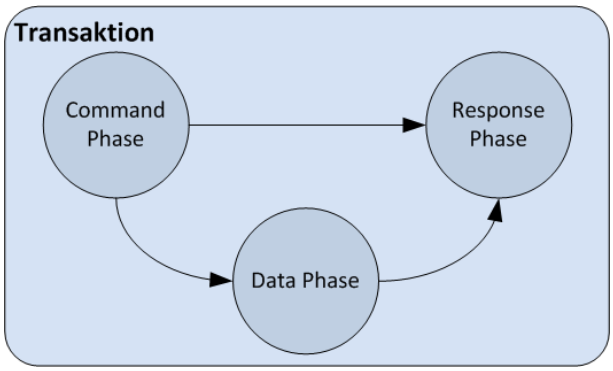


Abbildung 5.2: Protokoll der Kommunikation

5.2 DeviceID

Um eine bessere Rekonstruktion der einzelnen Abläufe zu ermöglichen ist ein Byte im Command Code für das jeweilige Auftragsmodul reserviert. Diese sollte für jedes Modul unterschiedlich definiert sein. Aktuell dient sie nur für Debug Informationen und wird teilweise fix im Command Code eingebaut. Die Werte 0 und 0xFF können nicht vergeben werden, weil anhand dessen die Unterscheidung zwischen Data und Response Phase vorgenommen wird.

5.3 Grundstruktur

Um die sehr umfangreiche Software zu vereinfachen, wurden die einzelnen Bereiche in Module unterteilt. Für jedes System (Windows, Android, ATmegas, VNC II) werden die gleiche Module verwendet, wodurch die Grundstruktur der Software immer gleich aufgebaut ist (siehe Abbildung 5.3). Natürlich ist es notwendig auf die einzelnen Gegebenheiten des jeweiligen Systems in der Firmware zu berücksichtigen (z.B. Art des Netzwerkes, unterschiedliche Bausteinprotokolle, Buffergrößen etc.).
Der Hauptteil wird als „Control Response Block“ bezeichnet. In ihm wird die Grundstruktur des Systems festgelegt (klassische Mainfunktion, inklusive der Initialisierung und Systemabläufe). Bei einem μC würde dort z.B. die Eingänge, der Empfangsbuffer und Interrupts abgefragt (Aktor, Sensor, Kamera) und entschieden wann etwas zu tun ist. Klasse VI Geräte hätten dafür eine Anwendungsmaske, in der der Benutzer verschiedene Aktionen durchführen kann.
Das „Read Container“ Modul empfängt die über die verschiedenen Netzwerke (Bluetooth, SPI, Ethernet) empfangen Daten und rekonstruiert einen Übertragungscontainer des ver-

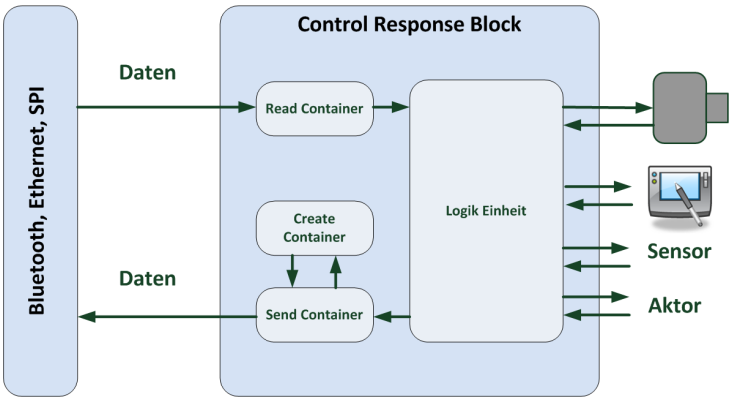


Abbildung 5.3: Um die Softwareentwicklung zu vereinfachen wurde ein Programm in Module unterteilt.

wendeten Protokolls 5.1. Das Logik Modul wertet den empfangen Container aus oder nimmt Aufträge vom „Control Response Block“ entgegen. Die einzelnen Abläufe der zur Verfügung stehenden Commands bzw. Data und Response Phasen sind innerhalb des Moduls definiert.
Um die Daten wieder in das Netzwerk zu senden, müssen diese zuerst mit Hilfe des Modules „Create Container“ in die richtige Form gebracht werden und darauffolgend im Modul „Send Container“ um die übertragungsspezifischen Eigenschaften erweitert werden.

5.4 Software Windows

Das Ethernet Modul hat standardmäßig die IP Adresse „192.168.20.50“, somit muss sich der Windows Rechner im selben Netzwerk befinden. Die Programmierung wurde so vorgenommen, dass auch mehrere Netzwerkkarten in einem System kein Problem darstellen. Die UDP bzw. ARP Pakete werden automatisch von der richtigen Netzwerkkarte gesendet. Damit das passiert, muss jedoch die verwendete Netzwerkkarten IP eingetragen werden (IP Feld in der Abbildung 5.4). Das zu verwendende Port muss ebenfalls beim Start der Software festgelegt werden. Programmiert wurde die Software in der Sprache C# unter Verwendung von WPF im Microsoft Visual Studio 2008 Professional Edition.
Die Empfangsroutine für ein UDP Paket befindet sich in einem eigenen Thread, weil diese sonst das restliche Programm blockieren würde. Verwendet wurde dabei die UdpClient-Klasse im Namespace System.Net.Sockets. Durch drücken des Buttons "Broadcast"wird eine UDP Broadcast Nachricht (UDP Dateninhalt = „Tell me!“) in das Netzwerk geschickt und alle verfügbaren Ethernet Module antworten. Die Antworten werden über ein BeginInvoke im Listenfeld des UI Threads dargestellt. Um direkte Befehle an ein bestimmtes Modul zu senden, muss die gewünschte IP in das Feld „Ziel IP“ eingetragen werden.
Aktuell können folgende Befehle über die Windows Anwendung gesendet werden:

- COMMAND_GETCLASS: Findet die Klasse des Moduls heraus.
- COMMAND_PIN_READ: Liest die Pin Struktur 5.8 des ausgewählten Tabs vom Modul aus.
- COMMAND_PIN_SET: Überträgt die eingestellten Werte der Pin Struktur an das Modul
- COMMAND_PIN_STORE: Modul speichert die einzelnen Pin Struktur Werte in das interne EEPROM.

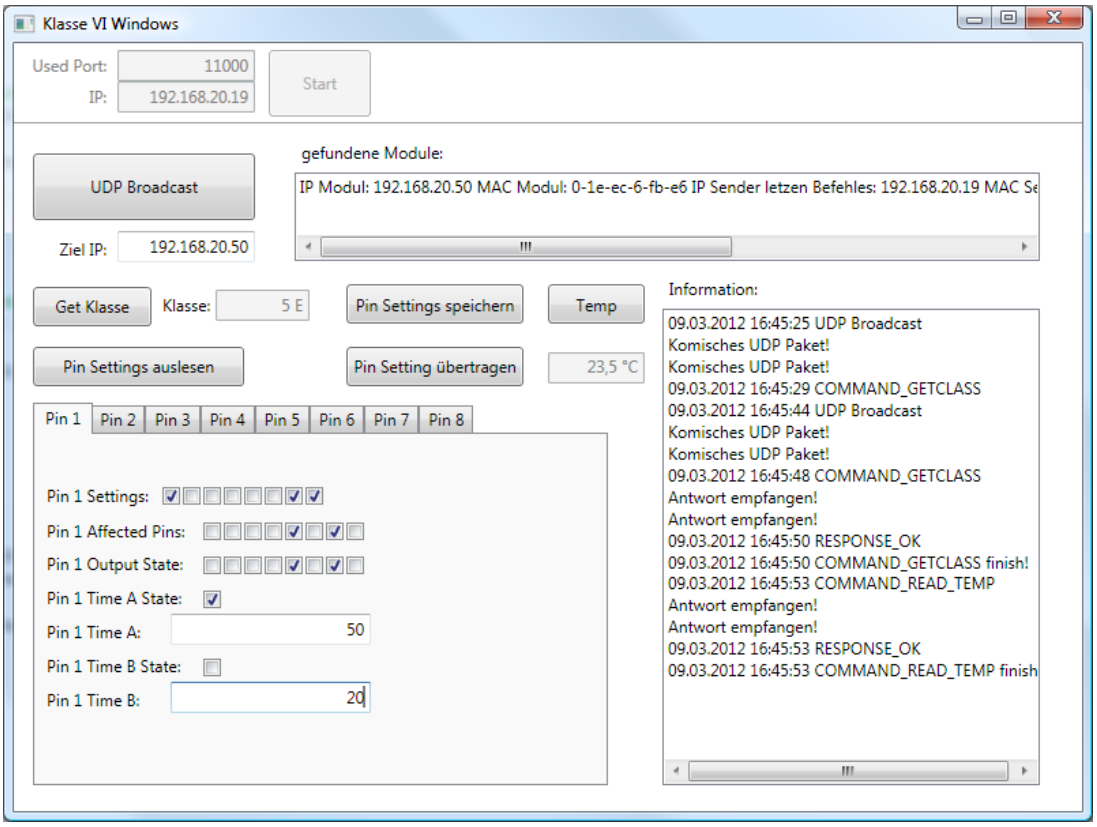


Abbildung 5.4: Klasse VI Implementation Windows

- COMMAND_READ_TEMP: Liest die aktuelle Temperatur des DS1621 aus.

5.4.1 Programmstruktur

Das Windows Programm besteht aus folgenden Dateien:

- Commands.cs: Definition der verwendeten Parameter
- Window1.xaml: WPF Formular
- Window1.xaml.cs: Dazugehöriger Code inklusive dem UDP Server, Sende- und Empfangsfunktionen

5.5 Software Android

Die Software für Android wurde in der Programmiersprache Java im Eclipse IDE 3.6 mit dem Android Development Tool entwickelt. Java weißt einige Unterschiede im Vergleich zur Programmiersprache C und C++ auf. Einer der wichtigsten ist das Fehlen von #Define Anweisungen, sowie das nicht Unterstützen von unsigned Typen. Das Gegenstück des C Typ „char“ ist in Java „byte“. In Java werden für einen „char“ Typen 2 Bytes, anstatt 1 Byte reserviert.
Die entwickelte Anwendung unterstützt mehrere Sprachen. Aktuell wurde zusätzlich die Sprache „Englisch“ implementiert. Die dazugehörige strings.xml Datei befindet sich im Ordner „values-en“. Damit die Anwendung gestartet werden kann, muss das Gerät über eine Bluetooth Schnittstelle verfügen. Falls diese nicht aktiviert ist, wird der Benutzer zur Aktivierung aufgefordert.

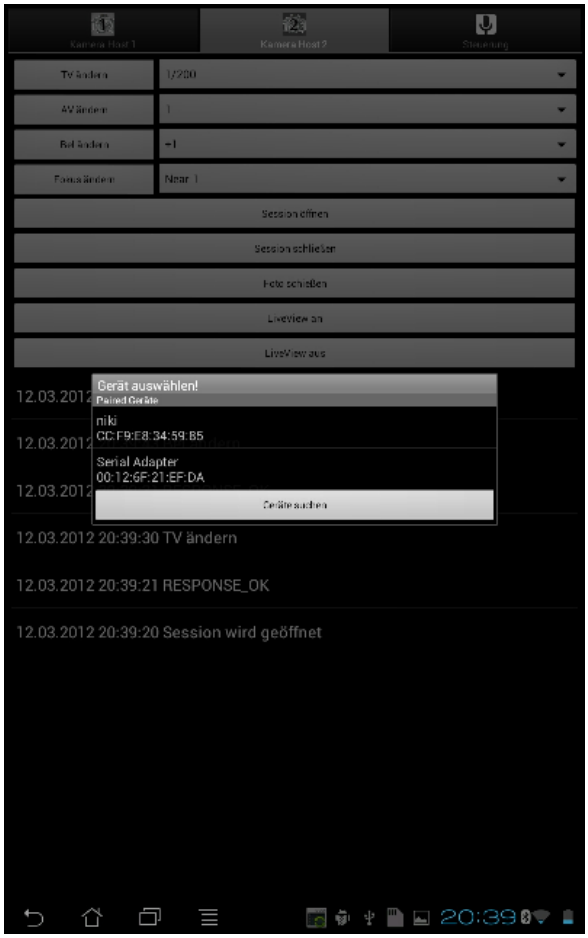
Eine Verbindung zu einem anderen Bluetooth Gerät wird über den Punkt „verbinden“ im Optionsmenü hergestellt. In ihm werden alle bereits gepairden Geräte angezeigt, bzw. besteht die Möglichkeit noch nach weiteren Geräten zu suchen. Sollte keine aktuelle Verbindung zu einem Gerät bestehen, kann kein Befehl gesendet werden und der Benutzer bekommt eine Nachricht angezeigt. Die Anwendung ist in 3 Tabs unterteilt. Mit Hilfe der ersten beiden Tabs können die Einstellungen einer angeschlossenen EOS 500D am jeweiligen USB Host des 2.3.2 verändert werden (siehe linkes Bild der Abbildung 5.5). Der Kommunikationsablauf wird dabei im Listenfeld angezeigt. Im dritten Tab sind aktuell folgende verschiedene Funktionen implementiert (siehe rechtes Bild der Abbildung 5.5):

- Durchführung einer 3D Scannung
- Durchführung von Zeitraffer Aufnahmen
- DDR und Pin Register vom Port A auslesen
- Ermitteln ob aktuell ein PTP am USB Host 1 bzw. 2 angesteckt ist
- Die Klasse des Moduls herausfinden
- Aufnahme einer 3D HDR Aufnahme mit 2 Kameras (5 Fotos - Belichtung -2 bis 2)

Die Kommunikation über die Bluetooth Schnittstelle wird über die Klasse „BluetoothChatService“ durchgeführt. Um sicher zu gehen, dass genügend Zeit für die Verarbeitung der empfangenen Zeichen zur Verfügung steht, wird nach jedem empfangenen Zeichen ’o’ (dezimal 111) an das Steuerungssystem zurückgesendet. Erst danach wird ein neues Zeichen übertragen. Das erleichtert die Rückgewinnung des übertragenen Datenpaketes, weil es schrittweise rekonstruiert werden kann, hat aber den Nachteil, dass inzwischen keine neuen Befehle an das Steuerungssystem übertragen werden können. Die zu sendenden bzw. empfangenen Daten werden zwischen den einzelnen Klassen über Intents ausgetauscht. Aktuell steht die Software nur im Porträt Format zur Verfügung, weil durch einen Bildschirmwechsel die OnCreate Methode der Klasse noch einmal aufgerufen wird und die aktuellen Einstellungen nicht übernommen werden. Im Internet gibt es mehrere Lösungen für dieses Problem, jedoch wurde aus Zeitmangel die Fixierung auf eine Bildschirmausrichtig durchgeführt.

5.5.1 Bluetooth

Für die Bluetooth Kommunikation stellt Android ein Framework mit verschiedenen API Funktionen zur Verfügung. Um diese zu nützen muss das Package „android.bluetooth“ in die eigene Anwendung integriert werden. Die wichtigste Klasse in diesem Package ist der Bluetooth Adapter. Er repräsentiert den lokalen Bluetooth Adapter mit seinen aktuellen Einstellungen. Mit ihm kann herausgefunden werden ob Bluetooth vom Gerät unterstützt wird und ob es aktiviert ist. Außerdem kann man auf die Liste der aktuell gepairden Geräte (eine Authentifizierung hat schon einmal stattgefunden) zugreifen und nach zusätzlichen Geräten suchen (Discovery).
Die Bluetooth Device Klasse repräsentiert die einzelnen gefundenen Geräte. Anhand ihr kann man z.B. die Netzwerkadresse oder den Namen herausfinden und eine Verbindung aufbauen. Für die Verbindung ist ein Bluetooth Server Socket notwendig, der auf eingehende Anforderungen wartet und einen Bluetooth Socket zurückgibt, wenn eine Verbindung zu einem Gerät aufgebaut wurde. Über diesen Socket werden auch die Daten als „Input Stream“ und „Output Stream“ ausgetauscht.



(a) Kamera Tab

(b) Steuerungs Tab

Abbildung 5.5: Klasse VI Implementation Android

Um Bluetooth in der Anwendung zu nützen muss das entsprechende Recht im Manifest festgelegt werden. In der Software ist es wichtig, dass die Lese- und Schreibbefehle auf den Bluetooth Socket in jeweils eigene Threads verpackt werden, weil diese sonst das restliche Programm blockieren würden und das Betriebssystem es automatisch beendet.
Das Tutorial von Android Developers beschreibt den Zugriff auf die Bluetooth Schnittstelle des Android Gerätes [Android Bluetooth]. Die Beispiel App „Bluetooth Chat“ konnte mit geringen Modifikation übernommen werden.

5.5.2 UUID

Android verwendet für den lokalen Bluetooth Server Socket eine 128 Bit UUID, die man beim Erstellen angeben muss. Die UUID muss die vom BTM-222 entsprechen, damit das Modul sich mit dem Android Gerät über den richtigen Kanal verbinden kann. In der Hilfe zur Methode createRfcommSocketToServiceRecord der BluetoothDevice Klasse ist diese für SPP Geräte mit „00001101-0000-1000-8000-00805F9B34FB“ angegeben. Nach einer Änderung sollte eine Verbindung möglich sein.

5.5.3 Klassenübersicht

Das Android Programm besteht aus folgenden Java Dateien:

- BluetoothChatService.java: Beinhaltet alle benötigten Bluetooth Funktionen
- CameraControl.java: Beinhaltet die Daten für Tab 1 und Tab 2 (PTP Kameras)
- Commands.java: Beinhaltet die Definitionen der Rückgabewerte und Befehle
- DeviceListActivity.java: Optionsmenu für das Verbinden mit einem Bluetooth Gerät
- Settings.java: Beinhaltet die Daten vom Tab 3 (Steuerung)
- TabMain.java: Hauptfenster mit Nachrichtenverwaltung

Auf die verwendeten Ressource Dateien (drawables, layout, menu, values) sowie auf das Manifest wird nicht näher eingegangen.

5.6 Firmware VNC 2

Die VNC 2 Firmware beinhaltet die SPI Verbindungsteuerung zum μC, die USB Host Funktionalität (Enumeration des angesteckten USB Devices und Feststellung ob es sich um ein PTP fähiges Gerät handelt) und das selbst programmierte PTP Protokoll, mit den im Anhang F beschriebenen Abläufen. Aktuell werden die PTP Befehle Open und Close Session (Standard Befehl), sowie die herstellerspezifischen Erweiterungen zur Remote Steuerung, Fernauslösung, Live View de-/aktivierung, Belichtungs-, AV-, TV- und Fokusänderung unterstützt. Die fixen Parameter der einzelnen PTP Befehle sind dabei direkt im Code angegeben, d.h. der μC muss nur die variablen Datenparameter übertragen. Eine Implementierung von weiteren Befehlen ist jedoch ohne Probleme möglich, weil nur der Switch-Case-Zweig um den jeweiligen Eintrag, inklusive der Parameter und Rückgabewerte, erweitert werden muss.
Für die logische Abfolge des PTP ist die Software des „Auftraggebers“ verantwortlich. Es erfolgt keine Überprüfung ob dieser Befehl im aktuellen Modus ausgeführt werden kann

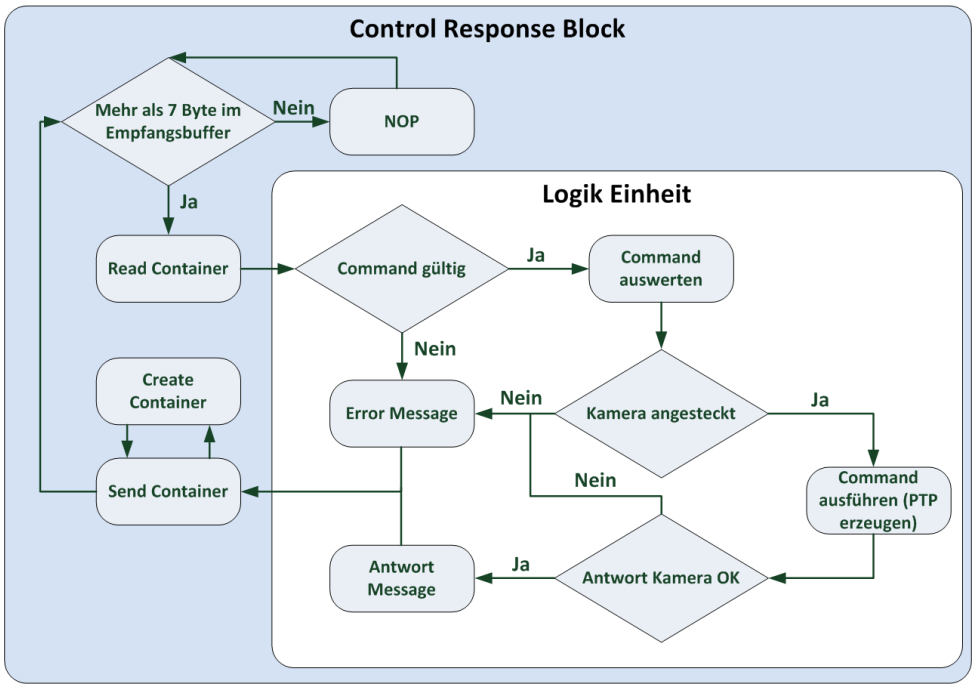


Abbildung 5.6: Flussdiagramm der Firmware am VNC

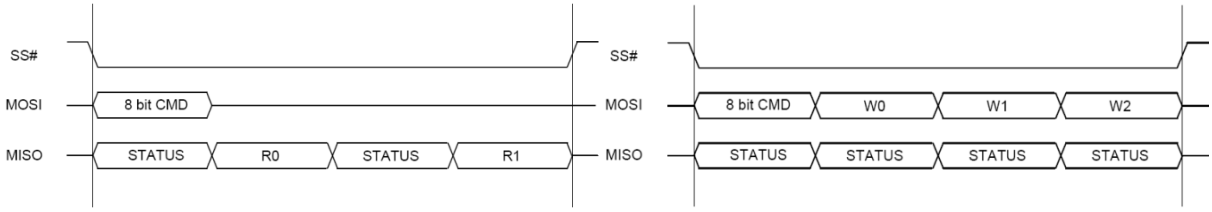
(Bsp.: Live View Aktivierung bevor der Fokus verändert wird). Nur die PTP TransaktionsID wird in der Firmware verwaltet und fortlaufend erhöht. Auf eine separate TransaktionsID für die USB Host 1 und 2 wird verzichtet. Um sicherzustellen dass ein Befehl korrekt vom PTP Gerät ausgeführt wurde, wird der Response Code überprüft (0x2001 entspricht ok). Bei einem Fehler wird „RESPONSE_PTP_ERROR“ als Code in der Response Phase zurückgesendet.
Als zukünftige Erweiterungen würde sich eine Echo Funktion für die einzelnen PTP Daten anbieten, d.h. die PTP Kommunikation könnte direkt von einem Klasse VI Modul abgewickelt werden und das Klasse V Modul dient nur als Zwischenglied. Darüber hinaus könnte man auch Fotos bzw. andere Dateien kopieren. Aktuell wird ca. die Hälfte des 256k Byte großen Speichers genutzt.

5.6.1 Flussdiagramm

Das Flussdiagramm ist in der Abbildung 5.6 ersichtlich. Solange sich nicht mehr als 7 Bytes im SPI Empfangsbuffer befinden, wird keine Aktion durchgeführt. Das ist sinnvoll, weil erst danach die Anzahl der gesamten Bytes des zu empfangenen Command Befehles festgestellt werden kann. Der eingelesene Befehl wird später auf Gültigkeit überprüft (Unterstützung des Codes, Anzahl der Parameter, Klasse- und Kontrollphase richtig). Nicht gültige Befehle werden mit einer Fehlernachricht in der Response Phase quittiert.
Falls ein gültiger Command erkannt wurde, wird zuerst eruiert um welchen USB Host es sich handelt und ob eine gültige Verbindung zu einen PTP Gerät vorhanden ist. Darauffolgend wird der PTP Befehl ausgeführt und die Antwort der Kamera abgewartet. Nach senden der Antwort an das Klasse V Gerät, wiederholt sich die ganze Prozedur.

5.6.2 SPI Slave Verbindung

Um Daten mit dem ATmega Controller auszutauschen wird die SPI Slave Schnittstelle des VNC 2 im Full Duplex Betrieb benutzt. Der VNC 2 erwartet zu Beginn einer Übertra-



(a) Read (b) Write

Abbildung 5.7: SPI Slave Full Duplex Diagramm des VNC 2 [VNC2 Datenblatt]

Z	Z	Z	Z	TXE	RXF	ACK	Z

Tabelle 5.3: Aufbau Status Byte

gung ein Command Byte (Tabelle 5.3), das angibt, ob es sich um einen Lese- oder einen Schreibzugriff handelt (R/W Bit: 1 schreiben 0 lesen). Erst danach können die Daten bzw. Dummybytes übertragen werden. Die restlichen Bits sind für diese Arbeit nicht relevant. Wie in der Abbildung 5.7 ersichtlich, antwortet der VNC 2 während einer Übertragung mit einem Status Byte, dessen Struktur in der Tabelle 5.4 abgebildet ist. Das ACK Bit sollte immer gesetzt sein, weil der VNC 2 nicht in einer Multi Slave Umgebung eingesetzt wird. Ein gesetztes TXE Bit zeigt an, ob sich Daten im Sendebuffer befinden (1: keine Daten, 0: Daten vorhanden) und RXF informiert darüber ob der VNC 2 Empfangsbuffer Daten beinhaltet (1: Daten vorhanden, 0: Buffer leer). Eine SPI Verbindung ist so lange aktiv, bis das „Chip Select“ wieder auf High gesetzt wird.
Beim Senden von Daten über die SPI Schnittstelle wartet der VNC 2 solange bis der Empfänger alle zu sendenden Bytes ausgelesen hat. Deshalb ist es wichtig immer die kompletten Bytes eines bereitgestellten Paketes zu lesen, da sonst die Firmware des VNC 2 neugestartet werden muss.

5.6.3 Programmstruktur

Die Firmware besteht aus folgenden Dateien:

1. Commands.h: Beinhaltet die Definitionen der Rückgabewerte vom VNC 2 und die unterstützen Befehle.
2. VNC_SPI.h: Definitionen für den VNC 2 (Verwendete USB Hosts und SPI Schnittstelle, inklusive Einbindung der benötigten Header-Dateien)
3. VNC_SPI.c: Initialisierung des VNC 2. Teile des Control Response Blocks (Main Funktion, Logic Block)
4. PTP.h: Definitionen für die PTP Kommunikation
5. PTP.c: Generieren, senden und empfangen der PTP Pakete und Verwaltung der PTP TransaktionsID
6. Host.h: Definition der verwendeten USB Host Funktionen

A2	A1	A0	R/W	Z	Z	Z	Z

Tabelle 5.4: Aufbau Command Byte

7. Host.c: USB Host Verbindungsverwaltung. Kontrolle ob ein Gerät angesteckt ist und ob es sich um ein PTP Gerät handelt.

8. Protokoll.h: Definitionen der vorhandenen Funktionen

9. Protokoll.c: Beinhaltet die restlichen Blöcke der Grundstruktur. Senden und empfangen der Daten über die SPI Schnittstelle. Kontrolle ob ein Befehl unterstützt wird und ob die Parameter richtig angegeben wurden. Senden und empfangen der entsprechenden PTP Pakete.

Es wird bewusst darauf verzichtet die einzelnen Funktionen näher zu beschreiben, weil es sonst den Rahmen dieser Arbeit sprengen würde. Der Quellcode wurde jedoch gut dokumentiert und sollte ohne Probleme nachvollziehbar sein. Die Hauptarbeit findet in der Funktion „LogicUnit“ in der Datei „Prototkoll.c“ statt. Zu erwähnen ist noch, dass vor jeder Ausführung eines Befehles die Verbindung überprüft wird. Falls ein Gerät am entsprechenden USB Host erkannt wurde, wird solange gewartet, bis die Enumeration vollständig abgeschlossen wurde. Erst danach kann man feststellen ob es sich um ein PTP fähiges Gerät handelt.

5.7 Firmware Controller

In der Firmware für den Controller muss unterschieden werden für welche Klasse dieser verwendet wird. Abhängig davon werden unterschiedliche Funktionen für den Controller freigeschaltet (Einlesen der Pins PC3 und PC4). Eine Differenzierung zwischen der Klasse III und höheren Klassen findet nicht statt, weil es programmtechnisch nicht relevant ist. Der serielle Datenbuffer wird durch das Fehlen einer Verbindung nicht befüllt, womit der betroffene Programmzweig nicht ausgeführt wird. Um eine schnelle Anpassung Möglichkeit des Steuerungssystemes auf geänderte Bedürfnisse zu ermöglichen, sollten die einzelnen Pins des Port A dynamisch zum konfigurieren sein ohne das der Controller dazu mit einem neuen Programm geflasht werden muss. Dafür stehen jedem Pin des Port A 9 Bytes zur Verfügung, die auch im EEPROM abgelegt werden können. Deren Aufbau ist in der Abbildung 5.8 ersichtlich. Beim Programmstart werden diese Werte aus dem EEPROM ausgelesen und in das RAM abgelegt. Durch Aufrufen des Befehles „COMMAND_PIN_STORE“ werden die aktuellen Einstellungen wieder in das EEPROM gespeichert. Jeder Pin des Portes A bekommt einen Modus zugeordnet, der in den ersten 6 Bit des Bytes Pin Setting gespeichert wird:

- Inaktiv (0)
- Aktiv (1)
- Interrupt (2)
- Pulsweitenmodulation (3)

Im 8. Bit wird angegeben ob es sich um einen Eingang oder einen Ausgang handelt (1=Ausgang, 0=Eingang). Bei der Verwendung des Pins als Ausgang haben die restlichen Bits keine Relevanz, die Ausnahme ist hier die Verwendung des Pins als Pulsweitenmodulation. Nicht verwendete Pins sollten als Ausgang gespeichert werden.
Das 7. Bit gibt Auskunft darüber, ab wann eine Reaktion des Steuergerätes erfolgen soll. Wenn es gesetzt ist, wird die Aktion ausgeführt wenn der Pegel am entsprechenden Pin High ist. Abhängig vom verwendeten Modus ergeben sich unterschiedliche Bedeutungen

dafür. Im Interrupt Modus gibt es die Signalflanke an, für die ein Interrupt ausgelöst werden soll (H->L 7. Bit nicht gesetzt, L->H 7. Bit gesetzt) (siehe 5.7.8). Für den Modus „Aktiv“ beschreibt es den Zustand des Pins. Für die Pulsweitenmodulation hat es keine Bedeutung. Der Modus „Inaktiv“ wird für Pins verwendet, wenn kein Gerät angeschlossen ist und die Firmware den Pin nicht abfragen soll. „Aktive“ Pins werden vom Programm im Block „Pins im Modus Aktiv abfragen“ auf Erfüllung der Bedingung kontrolliert. Falls die Bedingung zutrifft (7. Bit Pin Setting) werden die Zustände der betreffenden Pins geändert. Welche Pins das sind, wird in den Bits 16 bis 23 in den 4 Bytes der „Pin Reaction“ Variable definiert. Wenn das entsprechende Bit mit 0 gesetzt wurde, wird dieser Pin nicht geändert. Welchen Ausgangspegel die betreffenden Pins annehmen, ist in den ersten 8 Bit definiert, wobei 1 High bedeutet. Diese Einstellung ist besonders für nicht zeitkritische Anwendungen gedacht. Wenn möglichst schnell auf Ereignisse reagiert werden soll, eignet sich der Modus „Interrupt“. Die Zustände der betreffenden Pins werden dabei sofort in der Interruptroutine geändert. Bei den Modi „Aktiv“ und „Interrupt“ ist die Bedingung nur für eine bestimmte Zeitperiode aktiviert, deren Dauer in den 15 Bit des Time A Bereiches in den 4 Bytes der Pin Time Variable abgelegt (kleinste Einheit ist 0,5 ms) wird. Danach werden die betroffenen Pins wieder auf den negierten Soll-Ausgangswert gesetzt. Die gewünschte Zeit wird dafür in den 15 Bit des Time B Feldes gespeichert und bei jedem Timer Interrupt um eins verringert wird. Die maximale Dauer einer Bedingung ist somit auf 16384 s festgelegt. Die Dauer 0 bedeutet, dass die betreffenden Pins nicht mehr rückgesetzt werden. Für Pins im Modus „Interrupt“ wird das Time B Feld bei jedem Eintreten auf den Ausgangswert gesetzt, was bei „Aktiv“ Pins nicht der Fall ist.
Die letzte Einstellung ist die „Pulsweitenmodulation“. Damit lässt sich eine periodische Signalform erzeugen, d.h. der Pin ist für eine bestimmte Zeit High und in der restlichen Zeit Low. Die Summe der beiden Zeiten ist die Periodendauer. Dafür wird nicht die bereits integrierte Pulsweitenfunktion des ATmegas verwendet, sondern ein fortlaufender Timer. Dieser wurde so eingestellt, dass alle 0,5 s ein Überlauf stattfindet und ein Interrupt ausgelöst wird. In der Interruptroutine wird eine 16 Bit Zählervariable erhöht, die in der Hauptschleife abgefragt werden kann. Außerdem wird in der ISR die PWM durchgeführt. Dafür wird eine Hilfsvariable zuerst auf die gewünschte Bereichsdauer gesetzt und bei jedem ISR Aufruf um eins verringert. Sobald sie den Wert 0 erreicht hat, kommt es zu einem Bereichswechsel und der entsprechende Pin wird anhand des Wertes des Bit 31 bzw. 15 der Time Variable geändert. Mit dieser Methode können alle Pins mit unterschiedlichen PWM Einstellungen arbeiten. Die Dauer der einzelnen Impulse und der Ausgangspegel des Pins sind in den 4 Byte der Time Variablen gespeichert. Daraus ergibt sich eine maximale Periodendauer von:

$$max\ Periodendauer = \frac{2 * 15\ Bit}{2}\ s = 32767\ s \approx 546\ min \approx 9\ h$$

Die aktuell implementierten Befehle sind in der Tabelle 5.5 ersichtlich.

5.7.1 Flussdiagramm

In der Abbildung 5.9 ist das grobe Flussdiagramm der Firmware zu sehen. Die serielle Schnittstelle ist auch für Controller der Klasse III freigeschaltet, damit diese z.B. über die serielle Schnittstelle eines PC's konfiguriert werden können. In dieser Arbeit erfolgt die Konfiguration jedoch nur mit Hilfe des Bluetooth oder dem Ethernet Modul.

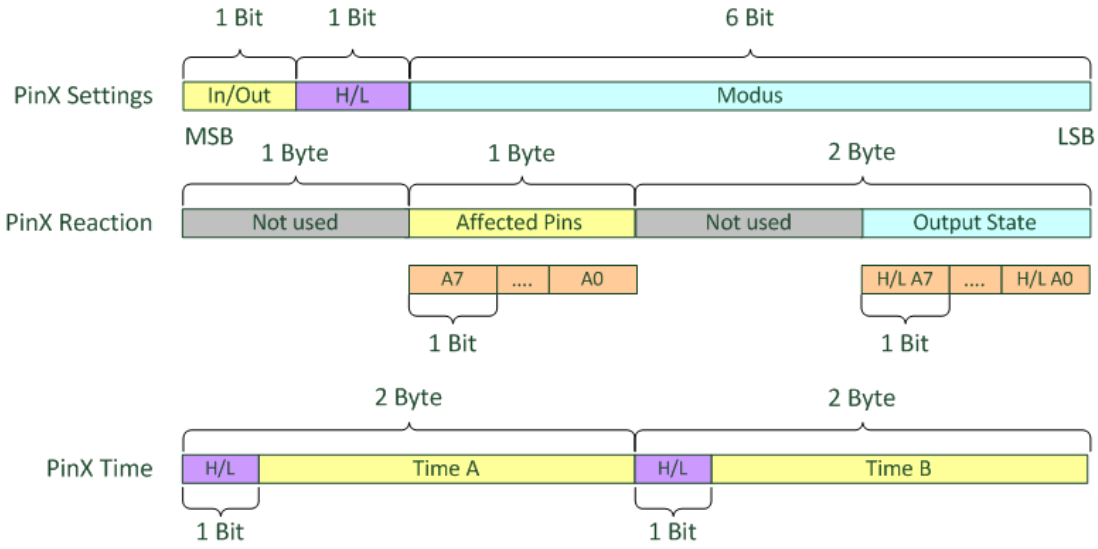


Abbildung 5.8: Definition einer Pin Struktur

Modul	Name	Beschreibung
Klasse I	COMMAND_PIN_STORE	Speichert die aktuellen Pin Einstellungen in das EEPROM
Klasse I	COMMAND_PIN_STATE	Gibt das aktuelle DDRA und PINA zurück
Klasse I	COMMAND_GETCLASS	Gibt die aktuelle Geräteklasse anhand den Jumpern JP8 und JP9 zurück
Klasse I	COMMAND_PIN_READ	Sendet die gewünschten Pin Einstellungen zurück
Klasse I	COMMAND_SET_OUTPUT	Ändert das DDRA und PORT A Register anhand den gesendeten Werten
Klasse I	COMMAND_PIN_SET	Übernimmt die gesendete Pin Einstellungen in das RAM
Klasse II	COMMAND_READ_TEMP	Gibt die aktuell eingelesen Temperatur des DS1621 zurück
Klasse V	COMMAND_PTP_SESSION_OPEN	Öffnet eine PTP Session
Klasse V	COMMAND_PTP_SESSION_CLOSE	Schließt eine PTP Session
Klasse V	COMMAND_PTP_SHOOT	Schießt ein Foto
Klasse V	COMMAND_PTP_GETCONNECTION	Liest aus ob sich an den USB Hosts ein PTP Gerät befindet
Klasse V	COMMAND_PTP_LIVE_VIEW_ON	Live View einschalten
Klasse V	COMMAND_PTP_LIVE_VIEW_OFF	Live View ausschalten
Klasse V	COMMAND_PTP_CHANGE_TV	Tv Wert ändern
Klasse V	COMMAND_PTP_CHANGE_AV	Av Wert ändern
Klasse V	COMMAND_PTP_CHANGE_BELICHTUNG	Belichtung ändern
Klasse V	COMMAND_PTP_FOKUS	Fokus ändern

Tabelle 5.5: Unterstütze Befehle des Klasse III Modules

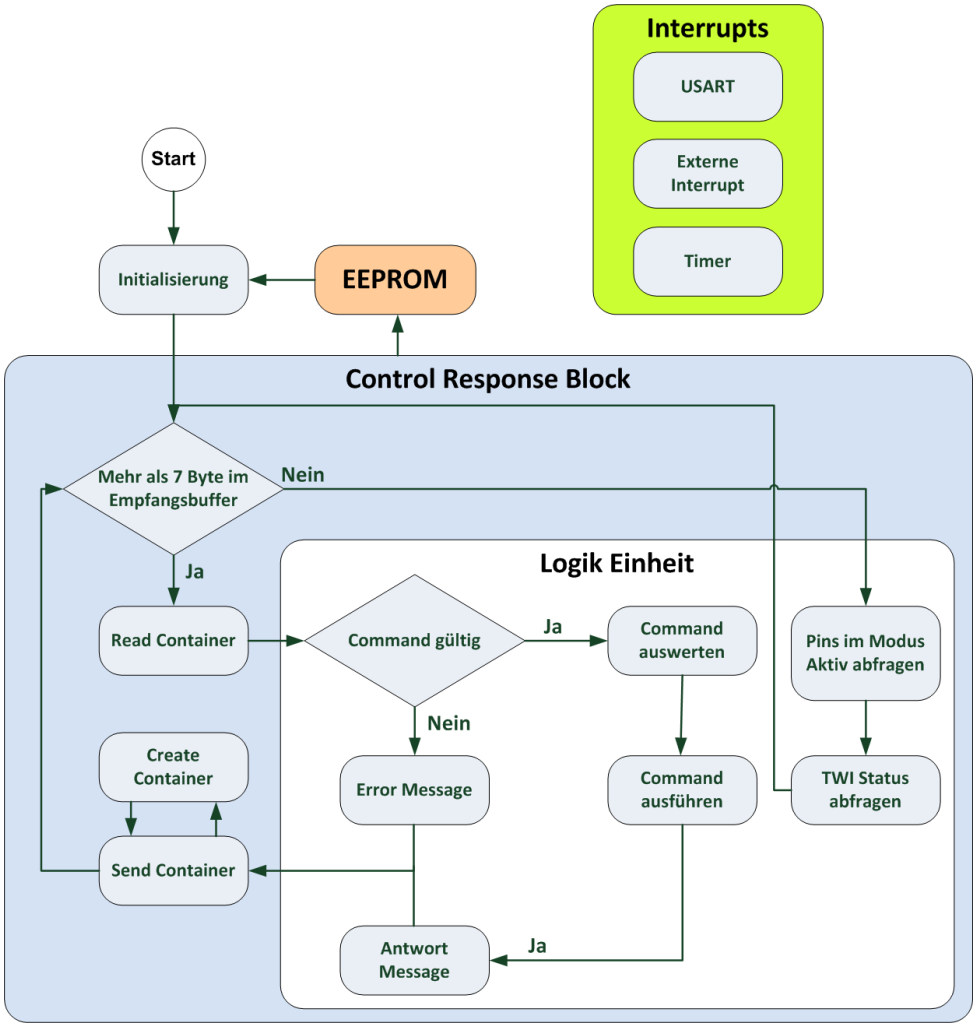


Abbildung 5.9: Flussdiagramm der Firmware eines Controllers

5.7.2 EEPROM

Um Systemeinstellungen auch nach einer Versorgungsunterbrechung zu behalten, wird der 1024 Byte große EEPROM Speicher des ATmega324a verwendet. Der Zugriff erfolgt über die in der Library eeprom.h inkludierten Methoden „eeprom_write_block“ und „eeprom_read_block“. Vor jedem Aufruf sollte mit Hilfe der Methode „eeprom_busy_wait“ überprüft werden, ob der EEPROM Speicher bereit ist. Die Anzahl der möglichen Schreibvorgängen auf den EEPROM Speicher ist mit ca. 100.000 beschränkt. Das sollte für einen Prototypen jedoch ausreichend sein. Da Schreibvorgänge relativ viel Zeit beanspruchen (einige ms), werden diese nur durchgeführt, wenn sich die Werte auch tatsächlich geändert haben und der entsprechende Befehl aufgerufen wird [Schäffer 2008].

5.7.3 Bluetooth Schnittstelle

Für den Kommunikationsablauf mit dem BTM-222 Modul wurden eigene c und Header Dateien erzeugt, die in das Projekt mit eingebunden werden müssen. Wie unter 2.3.10 beschrieben, kann das Modul über AT Befehle angesprochen werden, solange noch keine Verbindung aufgebaut wurde. Danach werden bis zur Beendigung der Verbindung die empfangenen und gesendeten Bytes durchgeleitet. Um den Beginn bzw. das Ende einer Bluetooth Verbindung festzustellen übernimmt die Funktion „CheckConnection“, die in der Empfangsinterruptroutine der seriellen Schnittstelle aufgerufen wird. Sobald sich der Zustand ändert (Disconnect -> Connect bzw. Connect -> Disconnect) müssen alle restlichen Zeichen im Empfangsbuffer ausgelesen werden.
Die Standardeinstellungen des BTM-222 Moduls werden nicht geändert. Damit das Modul genug Zeit für die Verarbeitung eines gesendeten Zeichens hat, wird auf das Echo gewartet. Das Modul muss innerhalb von bestimmten Zeitspannen auf die Befehle vom μC antworten. Dadurch wird verhindert, dass das Programm endlos auf eine Reaktion wartet. Anhand der Antwort auf den Befehl „AT“ wird festgestellt ob das Modul verfügbar ist.

5.7.4 USART Schnittstellen

Die Verbindung des μC zum BTM-222 bzw. dem Ethernet Modul erfolgt über die USART 0 Schnittstelle mit den Einstellungen wie unter 2.3.10 beschrieben. Die USART 1 Schnittstelle dient im Moment nur zur Bereitstellung von Debug Informationen, kann aber auch für zukünftige Projekte als zusätzliche Interrupt Pins dienen. Die Einstellungen sind dieselben wie für die USART 0 Schnittstelle. Als Empfangsspeicher dient ein Ringbuffer mit 80 Byte Größe, der in der Empfangsinterruptroutine des UART 0 gefüllt wird. Es ist dafür Sorge zu tragen, dass nicht mehr als 80 ungelesene Bytes vorhanden sind, weil sonst keine neuen gespeichert werden. Die benötigten Funktionen sind in den Daten uart.h und uart.c zu finden.

5.7.5 Fuse Bits

Da die Portleitungen PC3 und PC4 für das Einlesen des Klassentyps vom Modul verwendet werden und sich gleichzeitig die JTAG Schnittstelle dort befindet, muss das JTAGENFUSE Bit deaktivieren werden, weil dieses standardmäßig bei dem ATmega Modellen gesetzt ist und man die Pins sonst nicht als normale I/O Pins benutzen kann.
Außerdem ist darauf zu achten, dass das CLKDIV8 Fusebit nicht gesetzt ist, weil sonst die serielle Datenübertragung bei einem 12 MHz Quarz nicht funktioniert.

5.7.6 Timer

Es wird die Timer 1 Einheit des ATmega324a mit einem Vergleichswert von 0x16E3 und einem Vorteiler von 1024 verwendet, damit alle 0,5 Sekunden ein Interrupt ausgelöst wird. Die Berechnung erfolgt mit der unter 2 verwendeten Formel.

5.7.7 SPI Schnittstelle

Bei der SPI Übertragung wird zuerst das MSB mit 1/64 des Controller Taktes gesendet. Die Schnittstelle wird dabei im SPI Mode 0 betrieben (Das Bit für die Clock Polarity und Clock Phase ist im Control Register nicht gesetzt). Die benötigten Funktionen sind in den Daten SPI_Master.h und SPI_Master.C zu finden. Um einen Takt am SPI Bus auch beim Lesen zu erzeugen wird 0 übertragen.

5.7.8 Externe Interrupts

Als Interruptquelle stehen alle Pins des Port A zur Verfügung (PCINT7:0). Sobald sich der Zustand an einem dieser Pins ändert, wird der PCI0 ausgelöst. Die Einstellung der Pins als externe Interruptquellen werden im PCMSK0 Register vorgenommen. Die Interruptüberwachung erfolgt bei den ATmega Modellen asynchron, d.h., sie können zum Aufwachen aus dem Sleep Modus verwendet werden. Damit die Interruptserviceroutine aufgerufen wird, muss eine Aktivierung des PCI0 im PCICR Register vorgenommen werden.
Da jede Änderung des Zustandes einen Interrupt verursacht und im ATmega kein Register die benötigten Informationen zur Feststellung des betroffenen Pins liefert, wird in der Variable „PinChangeState“ die aktuellen Eingangspegels des Ports A gespeichert. Die Information welcher Pin von der Änderung betroffen ist, wird in der Variable „PinChange“ abgelegt. In der Interruptserviceroutine erfolgt eine Exklusiv-Oder Verknüpfung mit dem aktuellen Eingangspegel und dem Wert davor.

```
PinChange = PCMSK0 & (PinChangeState ^ (PINA & PCMSK0))
```

5.7.9 Programmstruktur

Die Firmware besteht aus folgenden Dateien:

1. Commands.h: Beinhaltet die Definitionen der Rückgabewerte vom Klasse III bzw. Klasse V Modul und die unterstützen Befehle.
2. BTM_222.h: Definition der Funktionen für das BTM-222.
3. BTM_222.c: Status, Reset, Senden und Überwachung ob eine Verbindung aufgebaut wurde.
4. DS1621.h: Definitionen der Funktionen für den DS1621.
5. DS1621.c: Initialisierung, Anstarten einer Konvertierung, Temperatur auslesen, Daten senden
6. EEPROM_PIN.h: Beinhaltet die Pin Struktur und die Definitionen der EEPROM Funktionen.
7. EEPROM_PIN.c: Pin_A Array und Funktionen für das Schreiben und Lesen des EEPROM.

8. Protokoll.h: Definitionen der vorhandenen Funktionen.
9. Protokoll.c: Beinhaltet die restlichen Blöcke der Grundstruktur. Senden und empfangen der Daten über die entsprechende Schnittstelle (SPI, USART, TWI). Kontrolle ob ein Befehl unterstützt wird und ob die Parameter richtig angegeben wurden.
10. RS232_ATmega324a.h: Definitionen der vorhanden Funktionen.
11. RS232_ATmega324a.c: Main Funktion, Timer Interrupt, Externe Interrupt, Initialisierung, Klasse überprüfen.
12. SPI_Master.h: Definition der SPI Funktionen.
13. SPI_Master.c: SPI initialisieren, Senden und Empfangen.
14. TWI.h: Defintion der Funtkionen für das TWI.
15. TWI.c: Initialisierung des TWI, Start-Stop Signal und Senden-Empfangen.
16. uart.h: Definition der Funktionen für beide serielle Schnittstellen.
17. uart.c: Einstellung der Baudrate, Buffergröße, Interruptempfangs- und Einleseroutine, Initialisierung der seriellen Schnittstellen.

5.7.10 AVR Memory Usage

Debug Modus wurde aktiviert.

```
Program:   15228 bytes (46.5% Full)
(.text + .data + .bootloader)
Data:       1517 bytes (74.1% Full)
(.data + .bss + .noinit)
EEPROM:       72 bytes (7.0% Full)
(.eeprom)
```

5.8 Firmware Ethernet Modul Controller

Der Controller am Ethernet Modul dient in erster Linie als Ethernet - USART Wandler. Damit die IP und MAC Adresse eines Moduls einfacher herauszufinden sind, wird zusätzlich der UDP Befehl „Anforderung“ verwendet. Dabei handelt es sich um eine Broadcast UDP Nachricht mit dem Dateninhalt „Tell me!“, worauf das Modul mit seiner IP, MAC Adresse und der IP, MAC Adresse und Ziel-Port des Absenders des letzten direkten UDP Paketes antwortet (siehe Tabelle 5.6). In der Abbildung 5.10 ist das grobe Flussdiagramm der Firmware zu sehen. Die SPI bzw. USART Einstellungen sind analog zu 5.7.7 bzw. 5.7.4.
Als externer Interrupt wird INT2 verwendet, damit wird vom ENC28J60 angezeigt, dass ein Ethernet Paket empfangen wurde. Ein Softwarefilter auf eingehende Pakete kontrolliert, dass nur direkt[1] adressierte oder Broadcast[2] Pakete bzw. ARP oder UDP Pakete weiter verarbeitet. Die Eigenschaften des ENC28J60 sind unter 2.3.12 näher beschrieben.

[1] MAC Adresse entspricht die des Moduls.
[2] niederwertigstes Bit des höchstwertigsten Bytes der MAC Adresse des Empfängers ist 1.

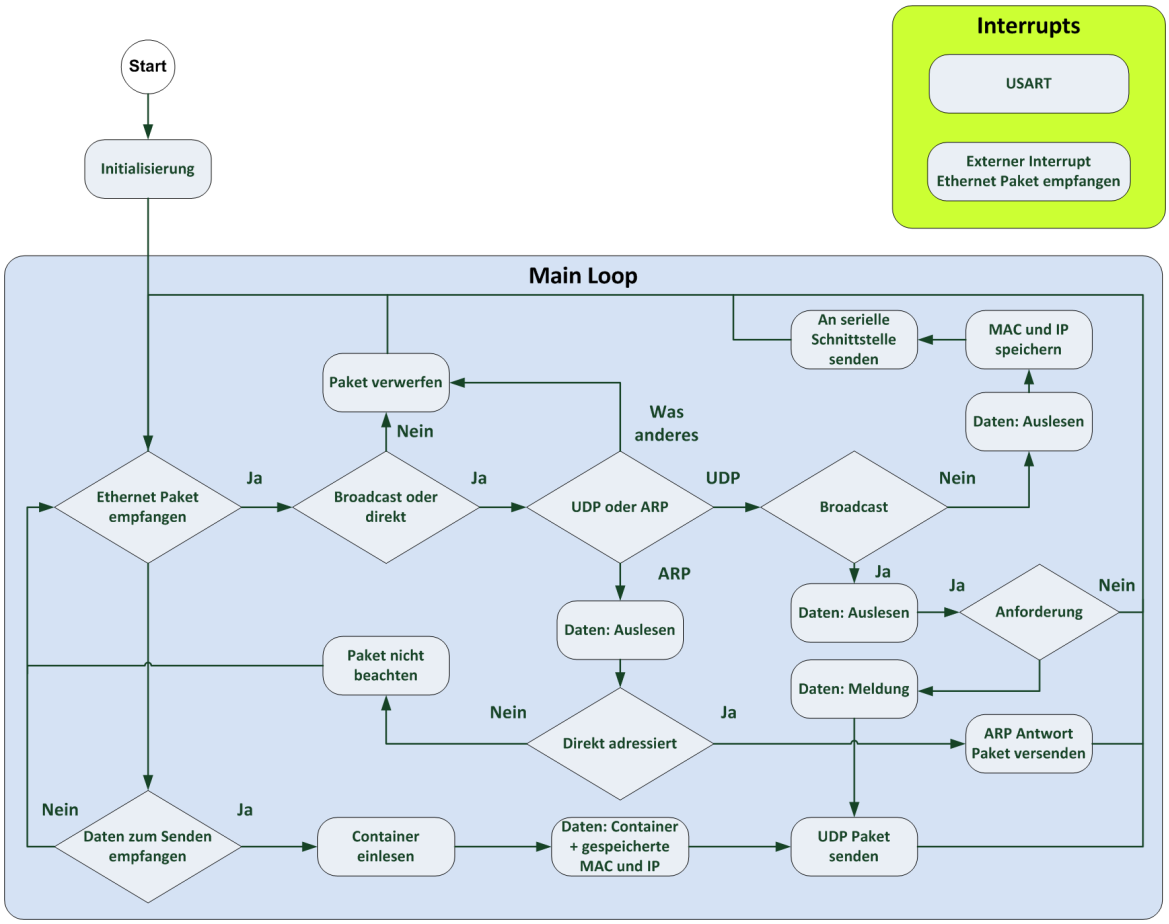


Abbildung 5.10: Flussdiagramm der Firmware des Controllers für das Ethernet Modul

Feld	Größe (Bytes)
IP Modul	4
MAC Modul	6
IP Senders letzten Befehles	4
MAC Senders letzten Befehles	6
Ziel-Port letzten Befehles	2

Tabelle 5.6: Aufbau des Datenpaketes einer „Tell me!“ UDP Broadcast Antwort

5.8.1 Programmstruktur

Die Firmware besteht aus folgenden Dateien:

1. Atmega324a_Ethernet.h: Funktionsdefinitionen
2. Atmega324a_Ethernet.c: Interrupt für Ethernet Pakete, Main Loop, Initialisierung.
3. uart.h: Definition der Funktionen für beide serielle Schnittstellen.
4. uart.c: Einstellung der Baudrate, Buffergröße, Interruptempfangs- und Einleseroutine, Initialisierung der seriellen Schnittstellen.
5. SPI_Master.h: Definitionen SPI Pins und Funktionen.
6. SPI_Master.c: SPI initialisierung, Empfangs- und Senderoutine.
7. ENC28J60.h: Definitionen für den ENC28J60.h (vorhandene Funktionen, MAC Adresse, SPI Instruktionen OpCodes, Receive Buffer, Register Definition).
8. ENC28J60.c: IP Adresse, Initialisierungsroutine des ENC28J60, Ethernet Pakete empfangen und senden, Generation der ARP und UDP Pakete.

5.8.2 AVR Memory Usage

Debug Modus wurde aktiviert.

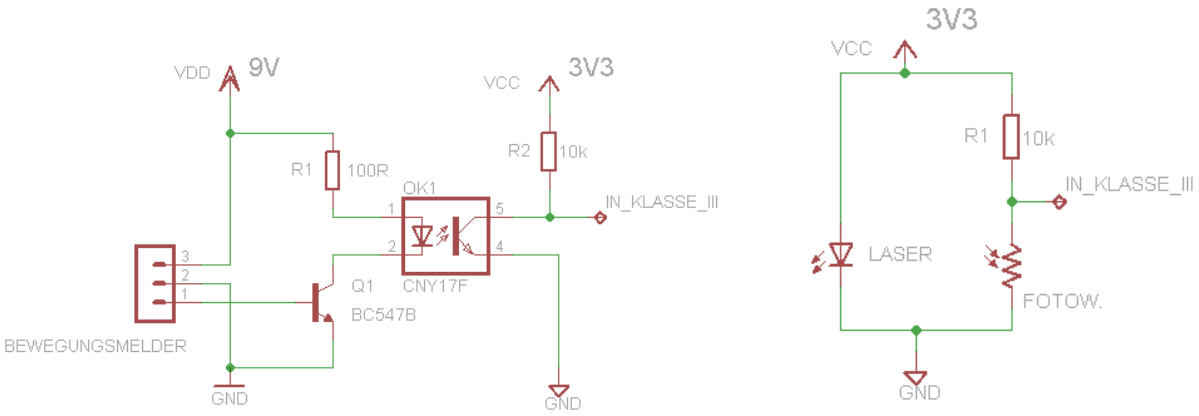
```
Program:   15542 bytes (47.4% Full)
(.text + .data + .bootloader)
Data:        799 bytes (39.0% Full)
(.data + .bss + .noinit)
```

5.9 ATmega324a Problem im AVR Studio 5

Als μC wurde das Modell ATmega324a gewählt. Bei den ersten Versuchen eine Kommunikation über die serielle Schnittstelle mit dem PC aufzubauen, kam es jedoch beim Senden von Strings zu unerwarteten Problemen, die lange für Kopfzerbrechen sorgten. Derselbe Code funktionierte auf einem ATmega16 ohne Probleme, er wurde nur für eine der zwei seriellen Schnittstellen des ATmega324a angepasst. Nach lange Suche im Internet konnte das Problem jedoch gelöst werden „AVR Freaks Forum - Writing to global variables not happening?“. Durch einen Fehler im hinterlegten Modell des ATmega324a im AVR Studio 5.0, werden die Variablen bei der Erstellung des Programmes im falschen RAM Bereich des Controllers abgelegt. Man umgeht das Problem indem man für die Erstellung das Ziel Modell ATmega324pa auswählt, dessen Aufbau ident zum ATmega324a ist.

5.10 Klasse II - TWI Sensor

Für den Test eines Klasse II TWI Sensors wurde der DS1621 Temperaturfühler von Maxim im DIP Gehäuse gewählt. Der dazugehörige Code befindet sich in der Firmware des Controllers 5.7 und kann über „#define DS1621“ eingebunden werden. Dabei wird die aktuelle Temperatur alle 20 Sekunden ausgelesen. Das Anstoßen der Messung erfolgt 10 Sekunden vor dem Auslesen, damit dem DS1621 genug Zeit für die Umwandlung zur Verfügung steht.



(a) Schaltung Bewegungsmelder

(b) Schaltung Lichtschranke

Abbildung 5.11: Schaltungen von Klasse I Modulen

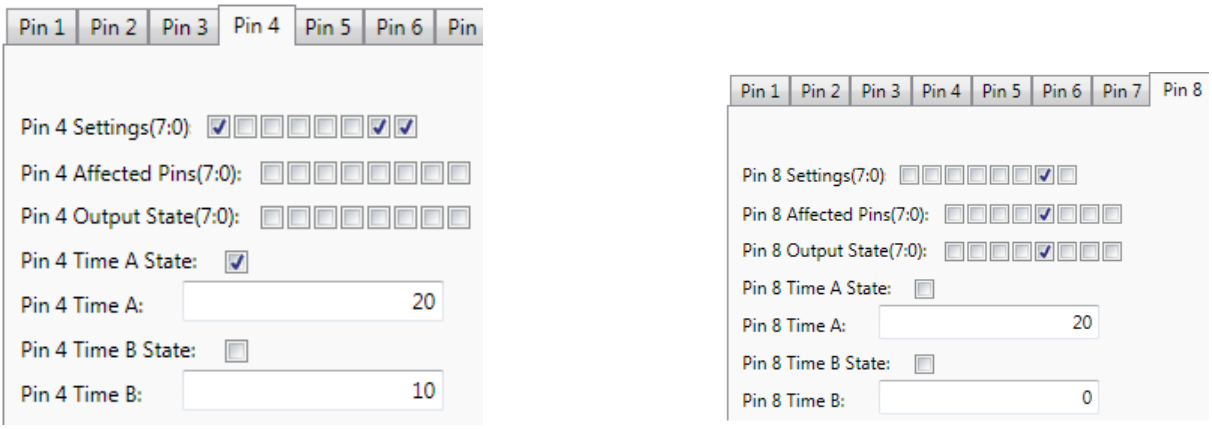
In dieser Arbeit wird eine Temperatur mit einer Auflösung von 0,5 °C (2 Bytes mit MSB zuerst) ausgelesen. Die Erfassung der Temperatur soll nur auf Kommando erfolgen (1SHOT = 1). Den Alarm Ausgang T_{OUT} könnte man zusätzlich mit einem Pin des Portes A des Steuergerätes als Interrupt Eingang verbinden um bei einer gewünschten Temperatur ein Ereignis auszulösen. Die Adresse des Slaves (A_0, A_1, A_2) wurde auf 101 festgelegt. Versorgt wird der IC über Spannung des Klasse III Modules. Die genauen Einstellungen des TWI Protokolls sind unter [DS1621 Datenblatt] nachzulesen.

5.11 Klasse I - Bewegungsmelder

Als Beispiel für ein Klasse I Modul eignet sich das PIR Bewegungsmelder Fertigmodul von conrad. Es hat eine Reichweite von 6 m und einen Erfassungswinkel von 110°. Als Test bekommt es eigene Stromversorgung über einen 9V-Volt-Block. Der Meldeausgang des Bewegungsmelder wird über einen Optokoppler mit einem Eingang des Klasse III Moduls verbunden. Sobald eine Bewegung festgestellt wurde, wird der Meldeausgang High und der Eingang des Klasse III Moduls auf Low gezogen. Weil der Meldeausgang nicht genug Strom für die Sendediode des Optokopplers liefert, wird ein Transistor als Verstärker verwendet (siehe linke Abbildung 5.11).

5.12 Klasse I - Lichtschranke

Ein weiteres Beispiel für ein Klasse I Modul ist ein Lichtschranke. Dabei wird eine Laserdiode verwendet, die auf einen Fotowiderstand leuchtet. Kommt es zu einer Unterbrechung des Strahles, wird der Fotowiderstand hochohmig (einige $M\Omega$) und der Eingang des Klasse III Modules wird über den Pull-Up Widerstand auf High gelegt. Um den Einfluss des Umgebungslichtes auf den Fotowiderstand zu minimieren, wurde er in ein ca. 10 cm langes Rohr mit einem 6 mm Durchmesser gesteckt. Als Lasterdiode wird das Laser Modul LP-705 verwendet.
Dieses Klasse I Modul wurde auch für den Versuchsaufbau der Verzögerungsmessung benutzt 6.3.



(a) Pulsweitenfunktionstest

(b) Interruptfunktionstest

Abbildung 6.1: Pinstruktur des Pulsweiten- und Interruptfunktionstests

6 Inbetriebnahme

6.1 Funktionstest

Nachfolgend werden einzelne Tests beschrieben, um einerseits eine Rekonstruktion zu ermöglichen und um die Funktionalität näher zu beschreiben. Damit die Abläufe im Programmcode besser verfolgt werden können, wurden die Debugausgaben der zweiten seriellen Schnittstelle am Hyperterminal verfolgt.

Pulsweitenmodulation Dabei wurde am PINA 4 das Fernauslösemodul (siehe 3.1) angeschlossen und mit dem Windowsprogramm die Pinstruktureinstellungen über das Ethernet an das Modul übertragen (siehe linkes Bild der Abbildung 6.1).

EEPROM Speichern Durch drücken des Knopfes „Pin Settings speichern“ werden die aktuellen Pinstruktur-Werte des Modules in das EEPROM gespeichert. Wird nun die Spannungsversorgung unterbrochen und erneut angelegt, sollte die Pulsweitenfunktion weiter aktiv sein.

Interrupt Für diesen Test wird am PINA 8 der Bewegungsmelder 5.11 angeschlossen, der das Fernauslösemodul am PINA 4 ansteuert. Zuvor muss die PINA 4 Struktur auf Ausgang und Modus Inaktiv geschaltet werden. Die benötigten Einstellungen sind im rechten Bild der Abbildung 6.1 zu sehen (Die Time A und B States werden nur bei der Pulsweitenfunktion benötigt). Am Bit 6 des Settingsbyte ist erkennbar, dass der Bewegungsmelder als Aktiv Low betrieben wird. Sobald eine Bewegung erfasst wurde, wird der der Ausgang des PINA 4 für 10 Sekunden aktiviert. Sollte innerhalb der 10 Sekunden eine weitere Bewegung erfasst werden, wird die Wartezeit erneut auf 10 Sekunden gesetzt und kein Foto ausgelöst.
Wenn nun PINA 8 im Modus „Aktiv“ (Setting = 0x01) betrieben wird und eine weitere Bewegung erfasst wurde. Kommt es zu keinem erneuten setzen der Wartezeit auf 10 Sekunden. Womit ein erneutes Foto nach 10 Sekunden geschossen wird.

PoE Die PoE Funktionalität wurde anhand einer Endspan Lösung mit Hilfe des Switches TP-LINK TL-SF1008P getestet, der über 4 PoE Ports verfügt und aktuell der günstigste PoE fähige Switch am Markt ist (43,99 €). Einmal wurden die benötigten 3.3 V direkt

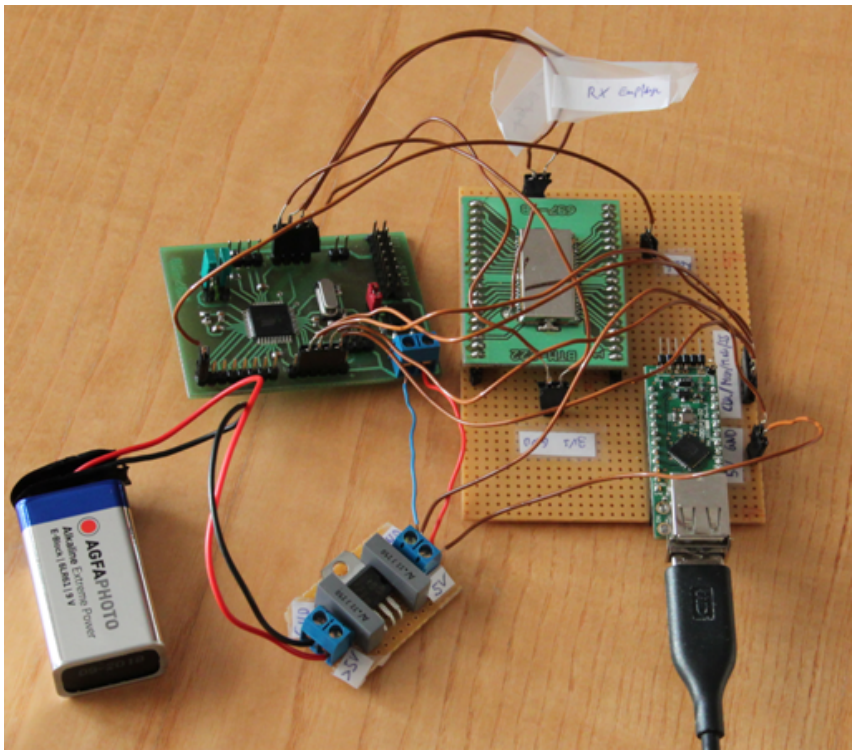


Abbildung 6.2: Klasse III Gerät inklusive Bluetooth und VNC2

vom Ag9033-S Modul bezogen und bei den anderen zwei Modulen (Ag9412-2BR und Ag9605-2BR) über den Spannungsregler am Ethernet Modul auf 3.3 V herunter geregelt. Als Spannungsversorgung für die restlichen Module diente das Klasse IV Ethernet Modul.

Sonstige Spannungsversorgung Für die Übertragung mit Bluetooth wurde die Platinen (Klasse III, VNC2, BTM-222) mit einer 9 V Blockbatterie versorgt. Da der VNC2 eine 5 V Spannungsversorgung benötigt, wurden die 9 V zuerst mit einem linearen Spannungsregler auf 5 V heruntergeregelt, der sich auf einer extra Entwurfsplatine befindet. Der Testaufbau ist in der Abbildung 6.2 zu sehen.

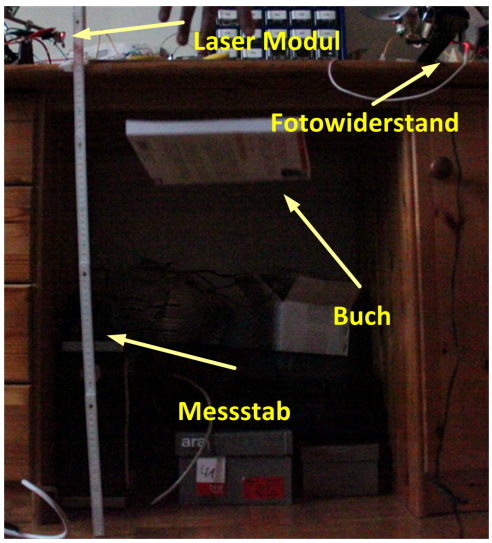
6.2 Stromverbrauch

Für die Messung des Stromverbrauches des Steuerungssystemes wurde ausgehend beim Klasse III Modul der Strom gemessen und schrittweise auf eine höhere Klasse erweitert. Externe Sensoren oder Aktoren wurden dabei nicht verwendet. Das Ergebnis der Messung ist in der Tabelle 6.1 zu finden. Für den Stromverbrauch der Klasse V hat es keinen Unterschied gemacht, ob eine Kamera an einem der USB Hosts angesteckt ist oder nicht. Ebenso konnte für den ENC28J60 kein Unterschied im Stromverbrauch zwischen Ruhezustand und einer Übertragung gemessen werden.

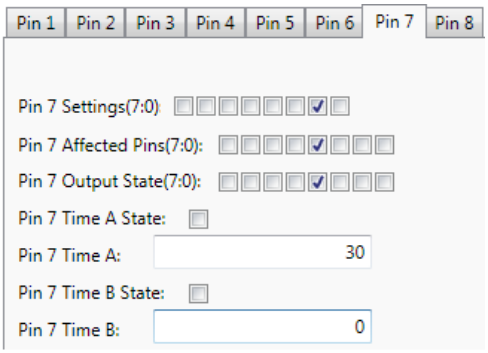
Wie man anhand der Strommessung sieht, eignet sich der ENC28J60 durch seinen hohen Stromverbrauch nicht für einen Betrieb mit Batterie oder Akku. Deshalb bietet sich eine PoE Versorgung bei der Art dieses Netzwerkes an. Stromsparend hingegen ist der BTM-222, dank seiner bereits integrierten automatischen Abschaltung, wenn keine Übertragung stattfindet.

Messung	Strom
Klasse III	16,3 mA
Klasse IV Bluetooth (Übertragung)	45 mA
Klasse IV Bluetooth (keine Übertragung)	19 mA
Klasse IV Ethernet	157,7 mA
Klasse V mit Bluetooth (keine Übertragung)	45 mA
Klasse V mit Bluetooth (Übertragung)	92 mA
Klasse V mit Ethernet	182,7 mA

Tabelle 6.1: Strommessung bei einer Versorgung über eine 9 V Batterie



(a) Versuchsaufbau der Verzögerungsmessung



(b) Einstellung des Pin 7

Abbildung 6.3: Messung der Verzögerung des Prototypen

6.3 Verzögerungsmessung

Für das Eruieren der Verzögerungszeit des Prototypen wird eine Aufnahme der Kamera mit Hilfe des Lichtschranke Klassenmoduls 5.12 ausgelöst. Der Versuchsaufbau ist in der linken Abbildung 6.3 zu sehen. Die Lichtschranke befindet sich am PINA 7 und löst eine Aufnahme durch das Fernauslösemodul aus (Pinstruktur siehe rechte Abbildung 6.3). Für den Test wurde ein Buch von der Tischkante (Entfernung zum Boden 74 cm) fallen gelassen und der Abstand zum Boden auf der Aufnahme abgelesen. Der Auslöseimpuls wird als Activ Low ausgeführt, womit die Lichtschranke zum Zeitpunkt t_0 durch das Buch unterbrochen ist. Sobald das Buch losgelassen wird, erfolgt ein Interrupt, weil das Licht vom Laser Modul auf den Fotowiderstand (A 9060) trifft. Es wurden 10 Aufnahmen mit TV - MF 1/320 mit einer ausgeschalteten Spiegelverriegelung aufgenommen, deren Ergebnisse in der Tabelle 6.2 ersichtlich sind.
Die Reaktionszeit wurde durch folgende Formel berechnet:

$$t_1 = \sqrt{\frac{(\text{Ausgangshöhe-Abstand zum Boden auf der Aufnahme}) * 2}{g}}$$

Wie unter 2.3.8 beschrieben, setzt sich die gesamte Verzögerungszeit aus der Zeit des Entscheiders und der Verzögerung der Kamera zusammen. Die Verzögerungszeit der

Aufnahme	Abstand zum Boden	Verzögerung
1	47 cm	235 ms
2	50 cm	221 ms
3	53 cm	207 ms
4	50 cm	221 ms
5	48 cm	230 ms
6	52 cm	212 ms
7	51 cm	217 ms
8	58 cm	181 ms
9	52 cm	212 ms
10	52 cm	212 ms
	Durchschnitt:	215 ms

Tabelle 6.2: Verzögerung Prototyp

Kamera beträgt für die verwendeten Einstellungen 152 ms (siehe 2.1), womit für die restlichen 63 ms das Klasse III und I Modul verantwortlich sind. Wobei die Ansprechzeit des Fotowiderstandes praktisch 99,9 % dieser Verzögerung ausmacht. Für eine kürzere Ansprechzeit eignen sich Fotodioden oder Fototransistoren.

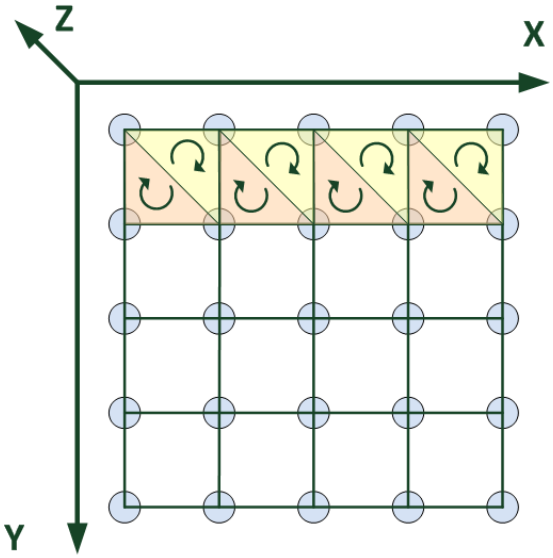
6.4 3D HDR Aufnahmen

Mit Hilfe der Android Klasse VI Software und der Konstruktion von 2.3.6 wurden eine Serie von HDR Aufnahmen von einem Objekt in verschiedenen Abständen von 2 Kameras durchgeführt. Das Ergebnis ist im Internet unter http://www.schniko.at/Bilder3DHDR/ zu finden. Um einen Wackeleffekt zu erzeugen, muss mit der Maus über das entsprechende Bild gefahren werden. Zum Einsatz kamen dabei eine EOS 400D mit einem TAMRON 18-270mm Objektiv und eine EOS 500D mit einem 18-135mm Objektiv. Verwendet wurde bei beiden Kameras das Av Programm mit einer Blenden von 5,6. Fokussiert wurde jeweils der gleiche Punkt. Die HDR Bilder sind aus 5 Aufnahmen mit einer Belichtung von -2 bis 2 erzeugt worden („Start 3D HDR Button“ im dritten Tab der Android Anwendung). Das Zusammenfügen zu einer Tonemapping Datei geschah mit [Photomatix]. Um die Wirkung des 3D Effektes im Abstand zum Objekt zu testen, wurde dieses aus 2 m, 4 m und 6 m fotografiert. Aus meiner Sicht, konnten die Aufnahmen aus 4 m Entfernung des besten 3D Effekt erzielen.
Die beiden Kameras wurden in einem Abstand von 30 cm auf der Schiene positioniert. Durch die Größe der Gehäuse beider Kameras kann kein natürlicher Abstand von zwei Augen nachgebildet werden (65 mm, siehe 8.5). Auf der Konstruktion wurde im Abstand von 2 cm ein Befestigungsloch erzeugt, damit der Abstand beider Kameras auf der Schiene abhängig vom aufgenommenen Objekt eingestellt werden kann.

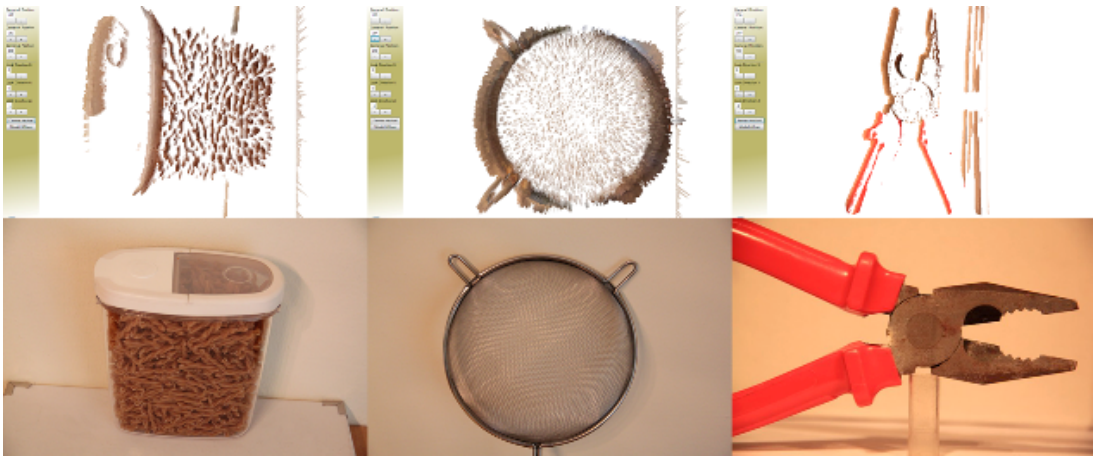
6.5 3D Farbscanner

Die Erstellung der Fotoserie wird mit Hilfe des Android Klasse VI Gerätes durchgeführt. Durch drücken des 3D Scanner Start Knopfes im Tab Settings wird eine Serie von Fotos mit unterschiedlichen Fokuseinstellungen geschossen. Damit sichergestellt werden kann, dass sich der Fokus vor Beginn der Serie am Anschlag befindet, wird 20-mal der Wert „0x03 (Near 3)“ als Parameter im Befehl „COMMAND_PTP_FOKUS“ übertragen. Die Anzahl der unterschiedlichen Schärfeebenen ist im Textfeld vom Benutzer einzutragen.

Nach jedem geschossenen Foto wird diese Anzahl um eins reduziert und der Fokus um den Wert „Far 2 (0x8002)“ weitergestellt.
Für den Prototypen eines 3D Farbscanners wurde der Code des unter 2.3.5 verwendeten Konzeptes übernommen und nur um die zusätzliche Speicherung des RGB Wertes des als Kante erkannten Punktes erweitert (siehe Anhang B). Auf die Anzeige des Ergebnisbildes der Kantendetektion wurde ebenfalls verzichtet. Da die gewünschte grafische 3D Darstellung des erzeugten Modells in Matlab nicht möglich ist, werden die einzelnen Werte des Ergebnisbildes in die angegebene Datei gespeichert. Die Werte aus der Datei dienen dazu ein Netz von Dreiecken zu generieren, wobei die Farbe des Dreieckes den Mittelwerten der Farben der 3 Eckpunkte entspricht. Dargestellt wird diese Objekt in einem Viewport3D Steuerelement einer C# WPF Anwendung. Einen guten Einstieg in die Behandlung von 3D Modellen mit WPF ist unter [WPF 3D] zu finden, wo auch ein Teil des Codes übernommen wurde.
Zu Beginn der Datei sind die Anzahl der Punkte auf der X-Achse (1. Zeile) und die Anzahl von den Punkten auf der Y-Achse (2. Zeile) zu speichern, damit die Dreiecke korrekt erzeugt werden können. Darauf folgen die einzelnen Punkte im Format „X,Y,Z;R,G,B“ von links nach rechts und von oben nach unten, jeweils in einer eigenen Zeile. Für die Bestimmung des orthogonalen Vektors auf die Ebene eines Dreieckes ist die Reihenfolge der Bekanntgabe der Punkte von einem Dreieck zu beachten. Sie müssen wie im Bild a der Abbildung 6.4 zu sehen ist, gegen den Uhrzeigersinn angegeben werden. Die Position der perspektivischen Kamera kann der Benutzer selber wählen. Sie sollte jedoch so gewählt werden, dass sie ca. der realen Kamera entspricht. Als Farbmodell wurde „AmbientLight“ verwendet, damit die Flächen bei jeder perspektivischen Kameraposition farblich angezeigt werden. Abhängig von der Größe des Bildes kann die Ausführung vom Programm längere Zeit beanspruchen. Die Berechnung eines kompletten 3D Modells anhand mehrerer Fotoserien wurde wegen Zeitmangels nicht durchgeführt.
Mehrere Ergebnisbilder sind im Bild b der Abbildung 6.4 zusehen. Man erkennt, dass diese Methode kein optimales Ergebnis liefert, weil die Fläche zwischen den Kanten leer bleibt. Was vielleicht bei einem kompletten 3D Modell nicht weiter störend ist, wenn die Flächen anhand der gefundenen Kanten geschlossen werden. Um ein besseres Ergebnis zu erzielen, müsste die Schrittweite der Fokussierung verringert werden. Gut gelingt bei dieser Methode die Abgrenzung des Objektes vom Hintergrund. Dabei ist jedoch auf die richtige Anzahl der geschossenen Fotos zu achten.



(a) Dreieckerzeugung



(b) Ergebnisbilder eines 3D Scans (von links nach rechts: Nudelbox, Sieb, Zange)

Abbildung 6.4: Prototyp eines 3D Farbscanners. Das Bild a zeigt wie die Daten für das Viewport3D eingelesen werden, um daraus die einzelnen Dreiecke zu bilden. Im Bild b sieht man Ergebnisbilder einer Fotoserie, erstellt mit dem selbstgeschriebenen Programm Bilder3D.exe.

Modul	Kosten
Klasse III	6,1 €
Klasse IV Bluetooth	13,95 €
Klasse IV Ethernet	21,2 €
Klasse V VNC inklusive Debugger	36 €
PoE Module	ca. 10 €
Gesamte Steuerungseinheit	87,25 €

Tabelle 7.1: Kosten für ein Steuerungsgerät

7 Projektauswertung

7.1 Resümee

Es konnte erfolgreich ein Prototyp einer Steuereinheit für digitale Spiegelreflexkameras entwickelt werden. Wobei das entworfene System nicht nur auf diesen Einsatzbereich beschränkt ist. Die Steuerung einer Spiegelreflexkamera ist in diesem Falle nur eine zusätzliche Erweiterung. Aufbauend auf dieses Steuersystem können beliebige Sensoren und Aktoren entwickelt werden, die mit dem System interagieren können. Die unter 2.1 gewünschten Anforderungen können alle mit diesem System abgebildet werden.
Bei dem Umfang dieses Projektes ist eine gute Planung und Durchführung der Spezifikation und Konzept Phase unerlässlich. Nur so können große Änderungen in der Hard- und Software vermieden werden. Anscheinend hat es in diesem Fall sehr gut funktioniert, weil keine Änderungen notwendig waren. Verbesserungen bzw. Vereinfachungen sind im Zuge des Testens entstanden, diese können aber in den nächsten Prototyp einfließen. Den größten Zeitaufwand in der Entwicklung hat sicher die Software und das Testen dieser benötigt. Wobei in der aktuellen Version nur der minimale Teil implementiert ist.
Die Interaktion verschiedener Systeme (Windows, Android, Controller) und Controller (Atmel, VNC2), erfordert es die Software in den jeweiligen Entwicklungssystemen und Sprachen (C, C#, Java) zu entwickeln. Dabei hat es sich gezeigt, dass sich die Unterschiede meistens nur auf den Syntax beschränken und die Semantik gleich bleibt. Womit ein gut konstruierter Code ohne viel Aufwand von einem System in das andere portieren lässt. Deshalb sollte bei der Entwicklung der Software darauf Wert gelegt werden, einzelne Bereiche in Funktionen zu kapseln.
Persönlich hat mir diese Arbeit sehr viel Spaß bereitet, weil sie durch den großen Umfang (Elektronik, Softwareentwicklung, Kommunikationsnetzwerke) einen breiten Bereich des Studiums Telematik abdeckt. Während dem Entstehen dieser Arbeit, sind mehrere Embedded System Projekte in Fachzeitschriften oder dem Internet veröffentlicht worden, die für die Interaktion des Benutzers ebenfalls das Android Betriebssystem verwenden. Ein Grund dafür ist sicher die weite Verbreitung, die gute Dokumentation und einfache Programmierung. Es ist davon auszugehen, dass dieses Thema noch eine wichtige Rolle in der Zukunft spielen wird.

7.2 Kosten

In den Kosten sind die einzelnen Platinen nicht berücksichtigt, weil diese direkt vom Institut für Elektronik bereitgestellt wurden und deshalb kein Preis bekannt ist.

Modul	Kosten
Klasse I Fernauslöser	1,43 €
Klasse I Bewegungsmelder	ca. 13 €
Klasse I Lichtschranke	ca. 6 €
Klasse II DS1620	6,19 €

Tabelle 7.2: Kosten Sensoren

7.3 Zukünftige Erweiterungen

Als zukünftige Firmwareerweiterungen für Klasse III, IV und V Geräte, wäre ein Eventlog interessant. Dabei könnte das Auftreten gewünschter Ereignisse in einem Ringbuffer abgelegt werden, um sie später auszulesen. Außerdem würde sich die Verwendung eines Betriebssystems als Firmware Grundlage anbieten. Der jetzige sequentielle Ablauf kann in diesem auf verschiedene Threads aufgeteilt werden, womit das gleichzeitige Bearbeiten mehrerer Befehle ermöglicht wird. Die Firmware des Ethernet Modul könnte noch für die TCP/IP erweitert werden, damit eine garantierte Übertragung stattfindet.
Die möglichen Erweiterungen für die Software der Klasse VI Geräte sind keine Grenzen gesetzt. Jedoch sollten beide alle Funktionen eines Klasse III bzw. V Moduls ausnützen können. Für den täglichen Gebrauch eignen sich die entworfenen Schaltungsplatinen noch nicht. Dafür sind die Steckverbindungen nicht ausgelegt und es fehlt noch ein passendes Gehäuse. Die entworfenen Schaltungen sind außerdem noch auf EMV Richtlinien zu überprüfen. In den aktuellen Softwareversionen wurde keine Energieoptimierung (deaktivieren nicht benötigter Funktionen) durchgeführt, was aber für den mobilen Einsatz des Steuerungssystems unbedingt notwendig ist, um den Energieverbrauch zu minimieren.

8 Theorie

8.1 Bildaufnehmer

Als Bildaufnehmer bezeichnet man den Sensor der die empfangenen Lichtstrahlen in elektrische Information umwandelt. Heutzutage werden 2 unterschiedliche Technologien (CMOS und CCD) verwendet, die jedoch beide auf den gleichen physikalischen Effekt beruhen.
Äußerlich hat der Bildaufnehmer eine rechteckige Form. Das Seitenverhältnis entspricht meistens einem verwendeten Aufnahmeformat (z.B. 3:2 für das Kleinbildformat).

8.1.1 Physikalisches Prinzip

Um den physikalischen Vorgang zu beschreiben wird zunächst kurz das Bohrsche[1] Atommodell erklärt. Ein Atom besteht aus den 3 Elementen Protonen, Neutronen und Elektronen. Im Kern befinden sich Protonen und Neutronen, um diesen Kern kreisen die Elektronen in verschiedenen Umlaufbahnen (Atomhüllen). Die äußerste Hülle des Elementes wird Valenzband genannt. Auf der Welt gibt es verschiedene Atome, die im Periodensystem aufgelistet sind. Für den Leitungstransport ist es notwendig Elektronen in das Leitungsband zu befördern. Bei einigen Elementen ist das kein Problem (Leiter), bei anderen wiederum ist das nicht möglich (Isolatoren). Halbleiter verfügen nur über geringe freien Elektronen, mit Hilfe von gezielten Verunreinigungen (Dotierung) mit anderen Elementen, kann man die Eigenschaft des Ladungstransportes beeinflussen.
Licht kann man sich als einen Photonenstrahl vorstellen, die abhängig von der Farbe, mit einer unterschiedlichen Frequenz schwingen. Je nach Frequenz besitzen diese Photonen eine unterschiedliche Energie, die z.B. als Wärme empfunden werden kann (Sonnenstrahl). Die geringste Energie besitzt rotes Licht, am meisten Energie besitzt blaues Licht. Das weiße Licht ist eine Kombination aus verschiedenen Frequenzen. Wenn nun ein Photonenstrahl auf ein Halbleiterelement trifft, werden mit Hilfe dieser Energie die Elektronen aus dem Valenzband in das Leitungsband befördert. Dieses Prinzip wird auch photovoltaischer Effekt genannt. Dadurch stehen mehr Ladungsträger für die Leitung des elektrischen Stromes zur Verfügung. Anhand der Anzahl der Ladungsträger im Leitungsband, die in einer gewissen Zeit entstanden sind, kann man auf den empfangenen Photonenstrahl rückschließen [Winzker M. 2008].

8.1.2 CMOS

Bei der Herstellung eines CMOS Bildsensors, bekommt jeder Pixel einen eigenen Schalttransistor und eine eigene Ladungs-Spannungsumwandlungseinheit. Damit wird es möglich die Pixel einzeln anzusprechen und man kann das Auslesen auf bestimmte Regionen (Region of Interest, Windowing) beschränken, wodurch die Auslesegeschwindigkeit erhöht wird. Zusätzlich können bei der Herstellung andere Bauelemente daneben integriert werden, wodurch sich die restliche benötigte externe Beschaltung auf ein Minimum reduziert. Eine höhere Anzahl zusätzlicher integrierter Bauelemente vergrößert allerdings den Füllfaktor.
Zum Auslesen der einzelnen Pixelinformationen unterscheidet man zwei Verfahren 8.1.

[1] Benannt nach Niels Henrik David Bohr(1885-1962): dänischer Wissenschaftler und Nobelpreisträger für Physik

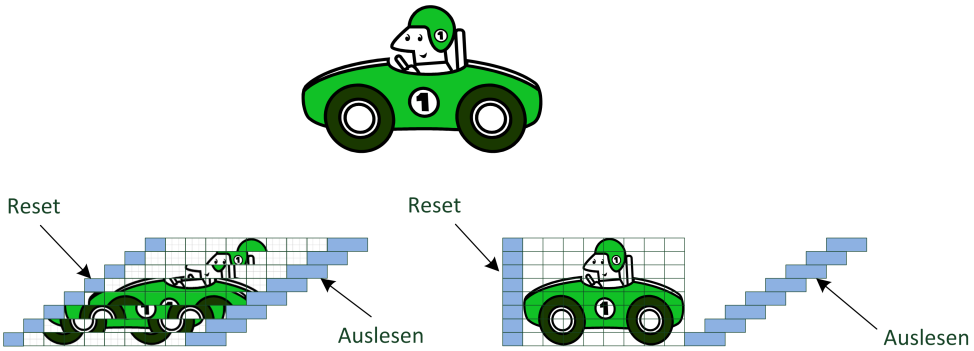


Abbildung 8.1: Bildinformationsausleseverfahren bei einem CMOS Sensor. Rolling Shutter (links) - Global Shutter (rechts)

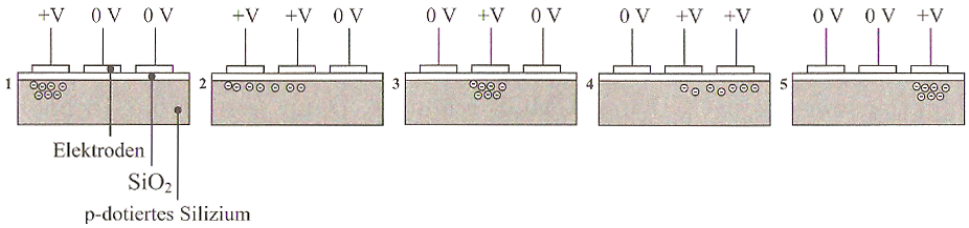


Abbildung 8.2: Skizze eines Ladungstransportes bei einem CCD Bildsensor (Eimerprinzip) [Azad P., Gockel T., Dillmann R. 2009]

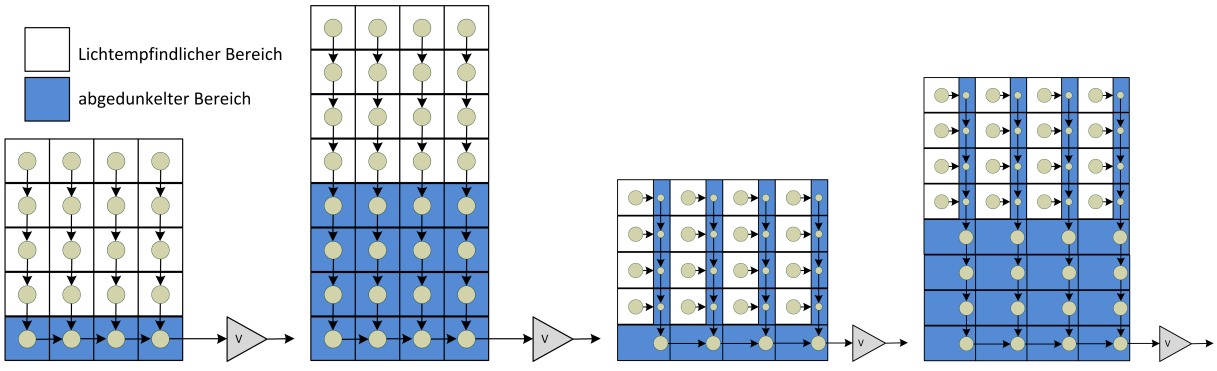
Beide Verfahren setzen zuerst ein Pixel auf 0, integrieren eine bestimmte Zeitspanne (Belichtungsdauer), die pro Pixel gleich ist, und lesen danach die Information aus.

Der Rolling Shutter geht dabei Pixel für Pixel vor. Was dazu führt, dass bewegte Motive verzerrt wirken, weil jedes Pixel zu unterschiedlichen Zeiten die momentane Information speichert.
Der Global Shutter umgeht dieses Problem, indem er alle Pixel zeitgleich auf 0 setzt und bei allen zeitgleich die Belichtung beendet. Dafür ist aber ein zusätzlicher Schalttransistor notwendig [Azad P., Gockel T., Dillmann R. 2009].
In der Canon EOS 500D Kamera sitzt ein eigenentwickelter CMOS Chip mit 15,1 Megapixel auf einer Fläche von 22,3 x 14,9 mm (APS-C Format). Die einzelnen Pixel haben nur einen Durchmesser von 4,7 μm.
Sogenannte Mikrolinsen erhöhen die Lichtausbeute und sorgen dafür, dass die einzelnen Pixel gapless[1] angeordnet werden können [Schwabe M., 2010]. Zum Auslesen wird das Rolling Shutter Verfahren verwendet, was man mit Hilfe der Videofunktion feststellen kann. Durch eine schnelle seitliche Bewegung treten Verzerrungen im Bild auf.

8.1.3 CCD

Ursprünglich wurde CCD als Speicherbausteintechnologie entwickelt, jedoch wurde festgestellt, dass die Chips sehr lichtempfindlich sind, wodurch man auf die Idee gekommen ist, sie als Bildsensoren zu verwenden.
Durch den photovoltaischen Effekt sammeln sich Ladungsträger in sogenannten Potentialmulden. Diese können durch anlegen eine Spannung von Speicherzellen zu Speicherzelle verschoben werden und so seriell ausgelesen werden (Eimerprinzip) 8.2.

[1]Die Linsen bündeln das Licht so, dass kein Lichtstrahl in die Zwischenräume der Pixel gelangt und damit 100% der Lichtstrahlen am Sensor auftreffen. Damit wird effektiv ein höherer Füllfaktor erreicht.



(a) Full Frame (b) Frame Transfer (c) Interliner Transfer (d) Frame Interline Transfer

Abbildung 8.3: Verschiedene Methoden des Auslesens bei CCD Kameras

Während dem Auslesen dürfen keine weiteren Ladungen hinzugefügt werden, weil sonst das Ergebnis verfälscht wird (Smear Effekt). Das serielle Auslesen benötigt eine gewisse Zeit und deswegen gibt es verschiedene Methoden wie man den Smear Effekt verringert 8.3.

Beim Full Frame (siehe Abbildung 8.3 Bild a) wird die Information in der Zelle zuerst sequentiell und dann seriell mit Hilfe eines externen Taktes ausgelesen. Der Vorteil von dieser Methode ist, dass der Füllfaktor gering ist, was den Sensor billig macht. Jedoch verlangsamt diese Technik den Vorgang und der lichtempfindliche Bereich muss inzwischen abgedunkelt werden.

$$\text{Füllfaktor} = \frac{\text{Fläche lichtempfindlicher Bereich}}{\text{Fläche abgedunkelter Bereich}} \tag{3}$$

Besser ist hingegen die Frame Transfer Methode (siehe Abbildung 8.3 Bild b). Hier wird die Information schrittweise parallel in einen abgedunkelten Bereich gespeichert, was den Smear Effekt verringert. Der Sensor ist durch die große Fläche jedoch teurer.
Die verbreiterte Technik ist die Interline Transfer Methode (siehe Abbildung 8.3 Bild c). Hier befindet sich die abgedunkelte Speicherzelle direkt neben dem Sensorelement, wodurch die empfangene Information schneller in den geschützten Bereich gelangt. Eine vollständige Abdunkelung ist aufgrund der geringen Größe aber nicht möglich, aber der gute Füllfaktor hebt den geringen Smear Effekt auf. Am besten ist die Frame Interline Transfer Methode, die die vorherigen zwei kombiniert und den geringsten Smear Effekt aufweist.
Der Nachteil der CCD Technologie ist die Notwendigkeit von mehreren externen Bauteilen (Taktgenerierung, A/D Wandlung, Verstärkung, verschiedene Spannungsversorgungen, etc.) [Azad P., Gockel T., Dillmann R. 2009].

8.2 Objektiv

Ein wichtiger Bestandteil in der Bildverarbeitung ist neben dem Bildaufnehmer 8.1 das Objektiv. Wenn man das Objektiv von außen betrachtet fällt einem auf, dass es die Form eines Zylinders hat und die Linsen kreisförmig sind. Weil die Sensoren in der Regel eine rechteckige Form besitzen, gelangt nur ein Teil des kreisförmigen Bildausschnittes auf den Sensor. Damit ein Gegenstand vollständig und scharf abgebildet wird, ist es wichtig das richtige Objektiv zu verwenden.

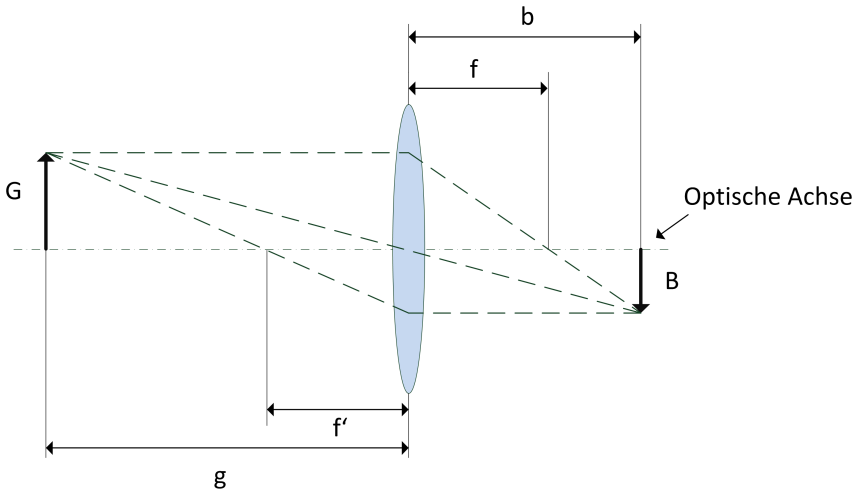


Abbildung 8.4: Optische Abbildung mit einer Sammellinse

Mathematisch wird die optische Abbildung durch die Linsengleichung und den Strahlensatz beschrieben (siehe Abbildung 8.4).

Strahlensatz:

$$V = \frac{G}{B} = \frac{g}{b} = \frac{f}{b-f} \tag{4}$$

Linsengleichung nach Descartes [1]:

$$\frac{1}{f} = \frac{1}{b} + \frac{1}{g} \tag{5}$$

Die wichtigen Kenngrößen von Objektiven werden nachfolgend beschrieben.

8.2.1 Brennweite

Die Brennweite gibt den Abstand der Linse zum Brennpunkt an. Es handelt sich dabei um einen fixen Wert, d.h. dieser ändert sich nicht, egal welche Bildaufnehmer verwendet wird (siehe dazu auch den Bildwinkel 8.2.2). Typische Brennweiten sind in der Tabelle 8.2 ersichtlich.
Anhand der Entfernung zum Objekt, dessen Größe und den Abmessungen der Bildaufnehmers, kann man die benötigte Brennweite durch umformen der Gleichung 4 berechnen.

$$f = \frac{b * G}{B + G} \tag{6}$$

8.2.2 Bildwinkel

Der Bildwinkel gibt die Größe des vorhandenen Sichtfeldes an und ist von der Brennweite und der Größe des Bildaufnehmers abhängig.
Wie in der Abbildung 8.5 ersichtlich, lässt sich der Bildwinkel mit folgender Formel berechnen (f: Brennweite, B_{max} : Breite, Höhe oder Diagonale des Bildaufnehmers):

$$\theta = 2 * arctan\ \frac{B_{max}}{2 * f} \tag{7}$$

[1] René Descartes (1596-1650): französischer Philosoph, Mathematiker und Naturwissenschaftler

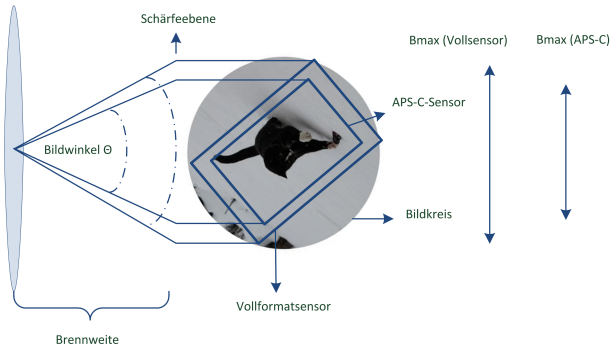


Abbildung 8.5: Zusammenhang Bildwinkel und Sensorgröße

Weitwinkelobjektiv	Normalobjektiv	Teleobjektiv
$\theta > 60°$	$35° <= \theta <= 60°$	$\theta < 35°$

Tabelle 8.1: Einteilung des Objektives anhand des Bildwinkels

Anhand der Formel 7 erkennt man, dass es sich dabei um eine redundante Information handelt, dennoch wird der Bildwinkel meistens zusätzlich
angegeben [Azad P., Gockel T., Dillmann R. 2009].
Mit Hilfe des Bildwinkels wird auch unterschieden ob das Objektiv ein Weitwinkel-, Normal- oder Teleobjektiv ist 8.1. Ein Normalobjektiv hat ca. das gleiche Sichtfeld wie das menschliche Auge, mit einem Teleobjektiv kommt einem alles vergrößert vor und mit einem Weitwinkelobjektiv erfasst man mehr Informationen.

In der Abbildung 8.5 sieht man, dass die Größe des Bildaufnehmers einen Einfluss auf den Bildwinkel hat. Dadurch entsteht der sogenannte Cropfaktor in der Digitalfotografie, wenn man einen kleineren Sensor als den Vollformatsensor verwendet, weil nicht der komplette Bildausschnitt auf den Sensor trifft. Der Faktor beträgt bei einem Sensor (APS-C) mit 27mm Durchmesser (22,5mm * 15mm, Seitenverhältnis ca. 3:2), 1.6 wenn der Durchmesser auf den Durchmesser vom Vollformat ($27mm * 1.6 \approx 43mm$ Durchmesser, Größe 36mm * 24mm) bezogen wird. D.h. ein Objektiv mit einer Brennweite von 50mm auf einem APS-C-Sensor, liefert den gleichen Bildausschnitt wie ein 80mm Objektiv auf einem Vollformatsensor [Schwabe M., 2010].
In der Tabelle 8.2 sind die entstehenden unterschiedlichen Bildwinkel zusammengefasst. Die Werte wurden mit der Formel 7 berechnet, als B_{max} wurde die Diagonale des Sensors verwendet. Besonders störend wirkt sich der Cropfaktor bei Weitwinkelobjektiven aus, positiv hingehen bei Fernobjektiven.

8.2.3 Blende

Die Blendezahl gibt die Größe des Lichtdurchlasses im Objektiv im Vergleich zur Brennweite an. Je kleiner die Blende, desto mehr Licht gelangt auf den Bildaufnehmer. Objektive mit einem Blendenwert < 2.8 werden als lichtstark bezeichnet, d.h. sie können schon bei wenig Umgebungslicht gute Ergebnisse erzielen.

$$k = \frac{f}{\varnothing \text{ des Lichtdurchlasses im Objektiv}} \tag{8}$$

Typische Blendenwerte sind 1.4, 1.8, 2, 2.8, 4, 5.6, 8, 11, 16, 22, 32. Der Blendenwert hat einen Einfluss auf die Schärfentiefe, näheres dazu im Punkt 8.2.6.

Brennweite	θ Vollformat	θ APS-C
14 mm	114,2°	88,0°
20 mm	94,5°	68,1°
24 mm	84,1°	58,8°
28 mm	75,4°	51,6°
35 mm	63,4°	42,2°
50 mm	46,8°	30,3°
70 mm	34,3°	21,9°
80 mm	30,3°	19,2°
85 mm	28,6°	18,1°
100 mm	24,4°	15,4°
135 mm	18,2°	11,4°
200 mm	12,3°	7,7°
300 mm	8,2°	5,2°
400 mm	6,2°	3,9°
500 mm	5,0°	3,1°
600 mm	4,1°	2,6°

Tabelle 8.2: Gegenüberstellung der unterschiedlichen Bildwinkel die durch die Verwendung eines Vollformat- bzw. APS-C-Sensors entstehen

Abbildung 8.6: Zusammenspiel von Sensorgröße und Bildwinkel. Nur der grüne Bildbereich würde bei einem kleineren Bildaufnehmer abgebildet werden (Cropfaktor = 1.6).

Einige Objektive erlauben eine manuelle oder automatische Einstellung, viele billigen bieten aber keine Möglichkeit [Azad P., Gockel T., Dillmann R. 2009].

8.2.4 Fokus

Mit Fokussieren bezeichnet man das Scharfstellen. Physikalisch gesehen wird dadurch die Bildweite b verändert. Bei einer Fokussieren von Gegenständen die ∞ weit weg sind, nähert sich die Bildweite b der Brennweite f an. Nahe Objekte können nur bis zur MOD scharfgestellt werden.
Bei einigen Objektiven geschieht das Fokussieren automatisch oder man kann es manuell durchführen. Es gibt aber auch Objektive die eine fixe Fokussierung besitzen [Azad P., Gockel T., Dillma

8.2.5 Minimale Objektdistanz

Die minimale Objektdistanz gibt an, wie weit das abzubildende Objekt mindestens von der vordersten Linse entfernt sein muss, damit es noch scharf abgebildet werden kann. Diese Einschränkung entsteht durch die Fokussiermechanik im Objektiv und hängt von der Brennweite ab.
Durch einen Zwischenring kann die MOD unterschritten werden, weil die Bildweite b erhöht wird (siehe Abbildung 8.7). Man muss aber beachten, dass durch die Verwendung eines Zwischenringes die Abbildungsfehler zunehmen [vision-doctor 2011].
Mit der Formel 9 kann die benötigte Dicke des Zwischenringes berechnen werden (Index zr bedeutet Zwischenring).

$$MOD = g_{min}, \quad \frac{g}{b} = \frac{f}{b-f}, \quad b_{zr} = b + d$$

$$\frac{MOD}{b} = \frac{f}{b-f}$$

$$MOD * b - MOD * f = b * f$$

$$b * (MOD - f) = MOD * f$$

$$b = \frac{MOD * f}{MOD - f}$$

Die gleiche Formel kann man nun auf die Verwendung eines Zwischenringes anwenden:

$$b_{zr} = \frac{MOD_{zr} * f}{MOD_{zr} - f}$$

$b_{zr} = b + d$ eingesetzt und umgeformt ergibt:

$$d = \frac{MOD_{zr} * f}{MOD_{zr} - f} - \frac{MOD * f}{MOD - f} \tag{9}$$

8.2.6 Schärfentiefe

Auf dem Sensor wird nur die Ebene scharf abgebildet, die parallel zur Linse liegt. Andere Ebenen erzeugen einen Unschärfekreis, auch Zerstreuungskreis genannt. Wenn die Unschärfe größer als ein Pixel vom Sensor ist, geht Auflösung verloren und die Abbildung erscheint unscharf [Azad P., Gockel T., Dillmann R. 2009].

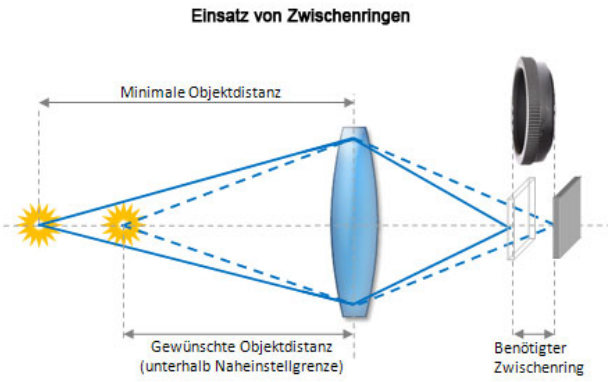


Abbildung 8.7: Durch einfügen von Zwischenringen, kann die minimale Objektdistanz unterschritten werden [vision-doctor 2011].

Für die Berechnung der Schärfentiefe muss man die Punkte g_v und g_h , die vor bzw. hinter der abgebildeten Ebene mit dem Abstand g liegen und gerade noch scharf abgebildet werden, berechnen. Die Differenz von diesen Punkten entspricht die Schärfentiefe.

$$\text{Schärfentiefe} = g_h - g_v$$

$$\text{Schärfentiefe} = \frac{g}{1 - \varnothing\ des\ Pixels\ * k * \frac{g-f}{f^2}} - \frac{g}{1 + \varnothing\ des\ Pixels\ * k * \frac{g-f}{f^2}} \tag{10}$$

Für die Herleitung wird die Linsengleichen 5, die Formel für die Blende 8 und die Winkelfunktion für den tan benötigt.

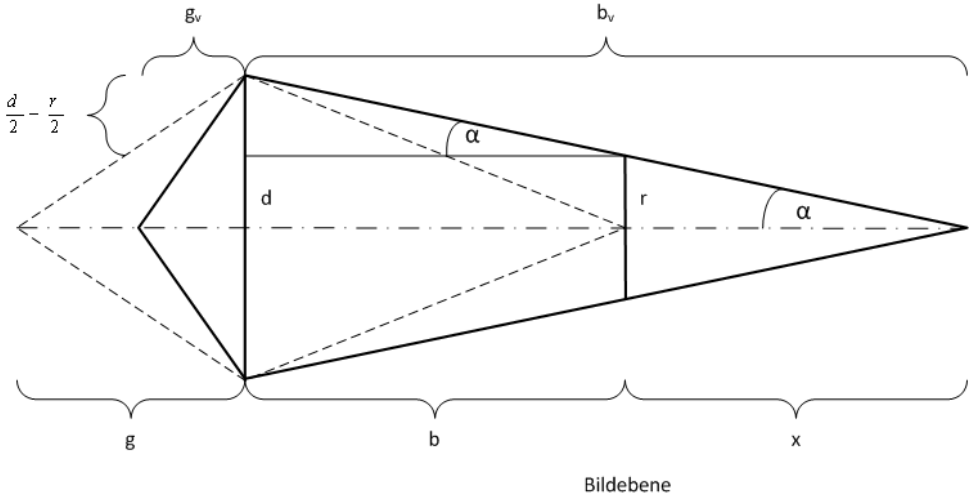


Abbildung 8.8: Skizze für die Herleitung von g_v

$$g_v = \frac{f * b_v}{b_v - f}; \quad b = \frac{f * g}{g - f}; \quad b_v = b + x; \quad \tan\alpha = \frac{GK}{AK}$$

$$\tan\alpha = \frac{\frac{r}{2}}{x} = \frac{r}{2 * x} \Rightarrow x = \frac{r}{2 * \tan\alpha}$$

$$k = \frac{f}{\varnothing \text{ des Lichtdurchlasses im Objektiv}} = \frac{f}{d} \Rightarrow d = \frac{f}{k}$$

$$\tan\alpha = \frac{\frac{d-r}{2}}{b} = \frac{d-r}{2*b} = \frac{\frac{f}{k}-r}{2*b}$$

$$x = \frac{r}{2*\tan\alpha} = \frac{r}{2*\frac{\frac{f}{k}-r}{2*b}} = \frac{r*b}{\frac{f}{k}-r} = \frac{r*\frac{f*g}{g-f}}{\frac{f}{k}-r}$$

$$b_v = b + x = \frac{f*g}{g-f} + \frac{r*\frac{f*g}{g-f}}{\frac{f}{k}-r}$$

$$g_v = \frac{f*b_v}{b_v-f} = \frac{f*(\frac{f*g}{g-f}+\frac{r*\frac{f*g}{g-f}}{\frac{f}{k}-r})}{\frac{f*g}{g-f}+\frac{r*\frac{f*g}{g-f}}{\frac{f}{k}-r}-f} = \frac{f*(\frac{f*g}{g-f}+\frac{r*\frac{f*g}{g-f}}{\frac{f-r*k}{k}})}{\frac{f*g}{g-f}+\frac{r*\frac{f*g}{g-f}}{\frac{f-r*k}{k}}-f} = \frac{f*(\frac{f*g}{g-f}+\frac{r*k*f*g}{(f-r*k)*(g-f)})}{\frac{f*g}{g-f}+\frac{r*k*f*g}{(f-r*k)*(g-f)}-f}$$

$$= \frac{\frac{f*g}{g-f}+\frac{r*k*f*g}{(f-r*k)*(g-f)}}{\frac{g}{g-f}+\frac{r*k*g}{(f-r*k)*(g-f)}-1} = \frac{f*g*(f-r*k)+r*k*f*g}{g*(f-r*k)+r*k*g-(f-r*k)*(g-f)}$$

$$= \frac{f^2*g}{f^2+r*k*(g-f)} = \underline{\underline{\frac{g}{1+r*k*\frac{g-f}{f^2}}}}$$

r entspricht im Rechenbeispiel dem Pixeldurchmesser. Für g_h verläuft die Herleitung nach dem gleichen Prinzip, nur wird für $b_v = b - x$ angenommen.
Anhand der Formel 10 kann man erkennen, dass sowohl die Brennweite als auch die Blende einen Einfluss auf die Schärfentiefe besitzen. Der Einfluss von der Brennweite ist jedoch quadratisch, wodurch man mit einem Weitwinkelobjektiv eine größere Schärfentiefe erzeugen kann.

8.3 HDR Aufnahmen

High Dynamic Range Bilder sind aus dem Grund entstanden, weil das menschliche Auge einen Kontrastumfang von ca. 1.000.000:1 unterscheiden kann, währenddessen moderne Bildaufnehmer nur einen Kontrastumfang von 10.000:1 darstellen (EOS 500D hat einen 14 Bit A/D Wandler = ca. 16.000:1). Der Kontrast ist das Verhältnis des hellst möglich abzubildenden Punktes zum dunkelsten Punktes. Bei den Bildsensoren wird der Kontrastumfang durch das Auflösungsverhältnis des A/D Wandlers verursacht. Der Signalprozessor von der Kamera muss sich bei der Wahl der korrekten Belichtung auf einen Helligkeitspunkt im Bild festlegen, wodurch andere Bereiche eventuell über- bzw. unterbelichtet werden. Das geschossene Foto erscheint dadurch weniger Detailreich und die Farben entsprechen nicht der visuellen Wahrnehmung eines Menschen.

Die Lösung des Problems ist das Erzeugen mehrere Aufnahmen mit unterschiedlichen Belichtungszeiten. Dadurch wird sichergestellt, dass jeder Bereich des Bildes einmal korrekt belichtet wurde. Mit Hilfe von speziellen Softwareprodukten kann man diese Serie von Aufnahmen dann zu einem einzigen HDR Bild zusammenfügen. Das ganze Prozedere funktioniert natürlich nur einwandfrei, bei der Aufnahme von nicht bewegten Objekten, weil diese sonst unscharf dargestellt würden.

Echte HDR Fotos besitzen 32 Bit pro Farbkanal, was ca. einem Kontrastumfang von 4.000.000.000:1 entspricht. Da für diese Bilder keine Ausgabemedien existieren, muss man die Werte mit Hilfe eines „Tone-Mappings“ in ein sogenanntes LDR Bild umwandeln. Diese LDR Bilder können dann wieder in ein übliches Bildformat (JPG, PNG, etc.) gespeichert

(a) -2 EV (b) 0 EV (c) +2 EV

Abbildung 8.9: Aufnahmen mit unterschiedlicher Belichtung

und angezeigt werden.

8.3.1 LDR

Für die Erzeugung eines LDR Bildes gibt es im Internet bereits verschiedene Algorithmen. Diese können in drei Gruppen unterteilt werden:

- Globale
- Lokale
- Gradientenbasierent

Eine gute Übersicht über die jeweils zu den Gruppen zugehörigen Algorithmen ist unter Wikipedia - Tone Mapping zu finden. Im Prinzip wird immer ein Indikatorbereich (z.B. Helligkeit, Luminaz, etc.) des HDR Bildes bestimmt, danach wird der Wert des Pixels in den Bereich 0-255 in Abhängig seiner Position im Indikatorfeld des HDR Bildes transformiert. Die Unterschiede entstehen durch die Tatsache wie man den Dynamikbereich des HDR Bildes festlegt. Bei globalen Algorithmen wird er aus den Informationen von allen Pixeln des HDR Bildes gewonnen. Bei lokalen wird nur ein gewisser Bereich um den zu transformierenden Pixel betrachtet. Gradientenbasierente Operationen bilden zuerst die Gradienten vom HDR Bild und erzeugen das LDR Bild durch die Verwendung der Information über die Stärke der einzelnen Gradienten.

8.4 Bildbearbeitungshardware

Hier werden nur 2 Arten von Bildbearbeitungsbausteinen behandelt. Andere verwendete Hardware wie z.B. PCs, ASICs oder GPOs werden nicht beschrieben.

8.4.1 DSP

Digitale Signalprozessoren sind Anfang der 80er Jahre im Zuge der Speicherung analoger Audiosignale in digitaler Form entwickelt worden. Damals wurden sie zum Noise-Shaping eingesetzt. Heutzutage findet man sie in allen Bereichen die eine Echtzeitausführung von Befehlen auf einen empfangenen Datenstrom erfordern, unter anderem auch in der Bildverarbeitung.
Gegenüber eines normalen μC besitzen sie den Vorteil, dass sie über mehrere Rechenwerke verfügen und bei den meisten ist ein Multiplikationsakkumulator vorhanden. Wodurch Operationen der Form A' = A + B * C direkt in der Hardware realisiert sind. Intern wird

Abbildung 8.10: Mit Hilfe von Tone-Mapping im Programm [Photomatix] erzeugtes LDR Bild.

eine sogenannte Registerstruktur verwendet, das bedeutet, dass die Operanden und die Ergebnisse in Registern gespeichert werden. Dadurch kann z.B. das Ergebnis einer Filterberechnung direkt beim nächsten Rechenschritt wiederverwendet werden. Die vorhandene Peripherie in einem DSP unterscheidet sich praktisch nicht von einem μC, beide verfügen über Timer, Watchdog, Interrupt und interne Taktgeneratoren.
Üblicherweise sind die Programme für einen DSP kurz, deswegen besitzen sie auch nur einen kleinen Programmspeicher. Außerdem sollten Programmsprünge vermieden werden, weil sonst die parallele Ausführung in einer Pipeline nicht möglich ist. Als Unterschied zu einem μC verwenden DSP die Harvard Architektur, d.h. der Programm- und der Datenspeicher sind voneinander getrennt. Wodurch die Geschwindigkeit erhöht wird, weil der Adress- und Datenbus voneinander getrennt sind. Für die Programmierung stehen, wie bei einem μC, ein Satz von Befehlen vom Hersteller bereit. Im Handel werden die digitalen Signalprozessoren in die zwei Kategorien Floating Point (Zahlen werden in Exponentialschreibweise gespeichert) und Fixed Point (Zahlen bestehen aus einer fixen Anzahl von Ziffern und auch die Position des Kommas ist vorgegeben) unterteilt ([Kisačanin B., Bhattacharyya S., Chai S. 2009] und [Elektor Mai 2011]).

8.4.2 FPGA

Im Allgemeinen besteht ein FPGA aus Logik-, Ein- und Ausgabeblöcken, einem Blockspeicher und einer Takterzeugung inklusive einem Takttreiber. Die einzelnen Blöcke sind untereinander in einem Netzwerk verbunden und können beliebig verbunden werden. Die sogenannte Konfigurationsdatei kann, abhängig vom gewählten FPGA, direkt im Flash bzw. im nicht flüchtigen Speicher des FPGA Chips gespeichert werden und ist somit direkt nach dem Einschalten verfügbar oder sie wird über einen μC bzw. einem externen Speicherbaustein geladen. Ein Logikblock besteht aus den 3 Grundelementen LUT (für die Implementierung der logischen Grundelemente z.B. And, Or, XOR, etc.), einem D-FlipFlop und einem Multiplexer 8.11.

FPGAs werden mit Hilfe von sogenannten Hardwaresprachen (Verilog, VHDL, etc.) logisch programmiert, d.h. die Funktion und die Verschaltung der einzelnen Gatter wird direkt beschrieben, womit man ganze logische Schaltungen auf einem Chip implementie-

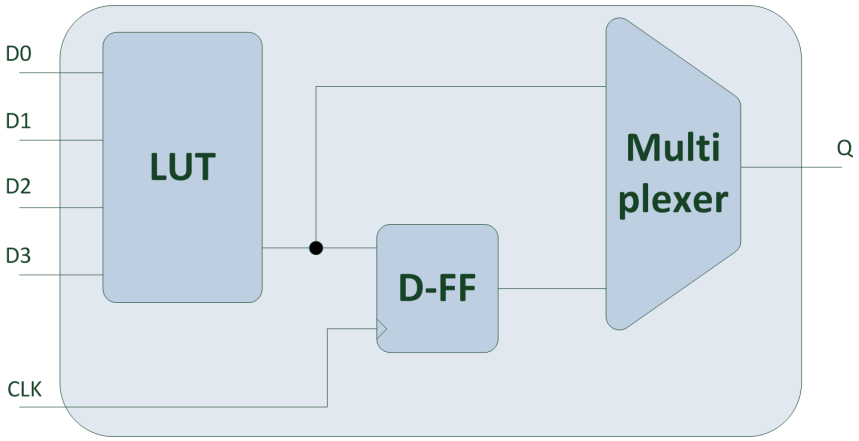


Abbildung 8.11: Schematischer Aufbau eines FPGA Logikblockes.

ren kann. Damit sind Platinen mit einer Fülle von logischen ICs überflüssig.
Weitere Vorteile von FPGA gegenüber μC sind die große Anzahl von Ein- und Ausgängen und die Möglichkeit parallele Abläufe zu implementieren und das durch die Verwendung einer Hardwaresprache ein Herstellerwechsel leichter durchzuführen ist. Was aber gleichzeitig zu großen Schwierigkeiten bei der Programmierung (Umdenken von einer sequentiellen in eine parallele Ausführung) und zu hohen Anforderungen an die Simulationssoftware führt [Sauer P., 2010].
Es ist ohne Probleme möglich die logische Beschreibung eines μC in einem FPGA zu „packen“ und diesen dann wie einen μC zu verwenden. Eine solche logische Schaltungsbeschreibung wird „Soft Core“ genannt und kann z.B. unter http://opencores.org kostenlos heruntergeladen werden.
Die maximale Frequenz eines FPGAs entspricht ca. der Frequenz von einem DSP, jedoch benötigt der FPGA mehr Strom. Für viele Computer Vision Element wird der FPGA in Verbindung mit einem DSP eingesetzt um beide Vorteile zu kombinieren. Der FPGA übernimmt dabei die I/O Aufgaben und stellt dem DSP die empfangenen Daten auf parallelen Leitungen zur Verfügung. Der DSP übernimmt dann die benötigten Berechnungen [Kisačanin B., Bhattacharyya S., Chai S. 2009].

8.5 3D Bilder

Mit unseren 2 Augen sind wir im Stande binokular zu sehen, das bedeutete, dass wir unsere dreidimensionale Umgebung auch in 3D wahrnehmen können um z.B. zielsicher Objekte zu greifen. Der Mensch ist in der Lage selbst beim Betrachten von 2D Bildern, Informationen auf die räumliche Verteilung der Objekte Rückschlüsse zu ziehen. Dabei benutzt er die bereits erworbenen Erfahrungen im täglichen Leben über die abgebildeten Objekte (z.B. übliche Größe des Objektes), wie die Objekte angeordnet sind (z.B. Verdeckung eines Objektes durch ein anderes) oder die Perspektive (z.B. Zusammenlaufen von parallelen Geraden in der Ferne). Auch mit einem Auge sind wir im Stande Dreidimensional zu sehen. Durch leichte Kopfbewegungen („Bewegungsparallaxe“), kann das Gehirn die wahrgenommen Bilder aus leicht unterschiedlicher Perspektiven zu einem 3D Modell vereinen.
In der Abbildung 8.12 ist der Vorgang des binokulare Sehens anhand eines Kegels ersichtlich. Wenn man die Spitze des Kegels fokussiert (Punkt F), wird diese scharf auf der

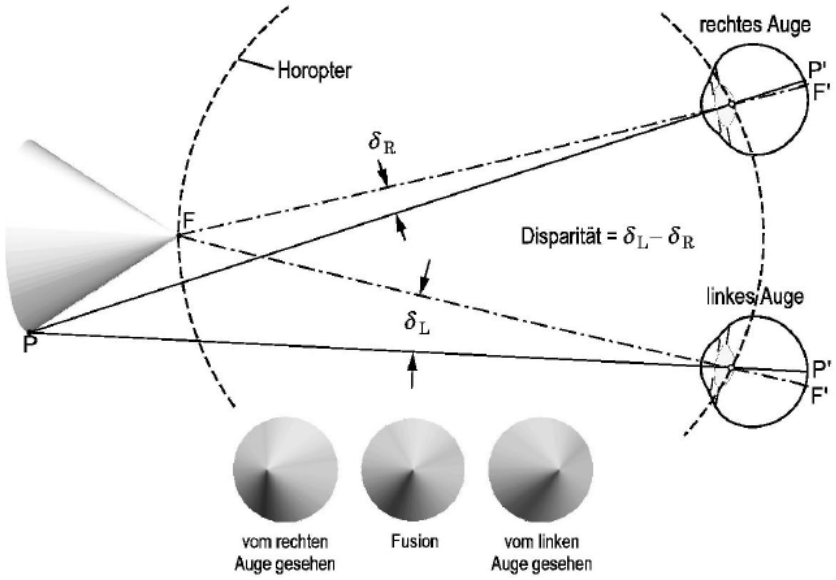


Abbildung 8.12: Binokulares Sehen [Mahler G. 2005]

Fovea [1] abgebildet (Konvergenz). Die Punkte neben der Kegelspitze treffen nicht auf den gleichen Bereich der Netzhaut, sie weißen einen horizontalen Abstand auf (horizontale Disparation), das wird durch den Abstand zwischen den Augen (ca. 65 mm) und der Entfernung zum Objekt verursacht. Der Punkt P wird in beiden Augen unterschiedlich abgebildet, was man anhand des Winkels zwischen F und P vom linken bzw. rechten Auge erkennt. Dadurch unterscheiden sich auch die wahrgenommenen Bilder von den einzelnen Augen. Wenn die Differenz von den Winkeln Disparität $= \delta_R - \delta_L$ nicht zu groß und auch nicht zu klein ist, kommt es zu einer Fusion (Es entsteht ein räumliches Gesamtbild). Der Winkelbereich indem es zu einer Fusion kommen kann, wird als „Panum-Bereich“ bezeichnet. Für weit entfernte Objekte ist die entstehende Disparität zu klein, womit es für diese Objekte keinen Unterschied macht, ob man sie mit dem linken oder rechten Auge betrachtet. Der Horopter-Kreis kennzeichnet jenen Bereich, wo ein betrachtetes Objekt des gleichen Winkeln in den Augen erzeugt (Disparität = 0).

Für die Aufnahme von 3D Bildern werden zwei Kameras benötigt, die das gleiche Objekt in einem horizontalen Abstand, der dem des Auges entsprechen soll, aufnehmen. Die geschossenen Fotos müssen dem jeweilig richtigen Auge zugeführt werden (Bildtrennung oder auch Stereoskopie genannt). Das kann z.B. durch das abwechselnde Zeigen eines Bildes erzeugt werden, wenn der Mensch eine Brille benutzt, die über einen sogenannten Shutter verfügen, der zum richtigen Zeitpunkt das Bild für das entsprechende Auge durchlässt („3D-Shutter-Brillen“). Eine andere Möglichkeit wäre das gleichzeitige projizieren beider Bilder in komplementär Farben (Anaglyphen-Verfahren). Hier wird eine Brille mit Farbfilterfolien benötigt (Typische 3D Brille mit einer roten und grünen Folie). Das Verfahren ist aber für Farbbilder nicht gut geeignet. Besser ist hier das Polarisationsverfahren. Die Bilder werden unterschiedlich polarisiert und die Brille enthält für das entsprechende Auge den richtigen Polarisationsfilter (Verfahren im Kino). Einen Einfluss auf den 3D Effekt hat bei diesem Verfahren die Kopfhaltung und die Rückstrahleigenschaft der Projektionsfläche.

Bei all diesen Methoden wurde das 3D Bild mit zwei Kameras aus einer Perspektive aufgenommen, wodurch keine Bewegungsparallaxe möglich ist, was aber nicht als störend

[1] Bereich des schärfsten Sehens auf der Netzhaut

empfunden wird [Mahler G. 2005].

8.6 Bluetooth

Um eine komfortable Datenübertragung zwischen mobilen Endgeräten zu ermöglichen, wurde 1999 das Drahtlose Standard Protokoll Bluetooth im Standard 1.0B entwickelt. Inzwischen wurde das Protokoll um mehrere Funktionen (z.B. bessere Verschlüsslung und höhere Datenrate) erweitert. Die momentan aktuellste Version ist 2.1. Der Standard wurde jedoch so ausgelegt, dass jedes Bluetooth Gerät zu den älteren Standards abwärtskompatibel ist.
Für die Datenübertragung steht eine Bandbreite von 1 MHz im 2.4 GHz ISM Band zur Verfügung. Bis zur Bluetooth Version 2.0 konnten Daten auf einem Kanal mit maximal einer Geschwindigkeit von 780 kBit/s übertragen werden, danach wurde das Modulationsverfahren geändert, womit Übertragungsraten von 2.178 MBit/s möglich wurden (Enhanced Data Rate mit DQPSK und 8PSK Modulation). Mit Hilfe von Bluetooth können mehrere Teilnehmer zu einem sogenannten Piconetzwerk zusammengeschlossen werden. Diese Teilnehmer teilen sich die zur Verfügung stehende Datenrate, womit die tatsächlich erreichte von der Aktivität im Netzwerk und von der Anzahl der Teilnehmer abhängt. In jedem Netzwerk gibt es einen Master, der die Verbindung verwaltet und bestimmt wann jemand Daten senden darf, und bis zu 7 Slave. Die Übertragung kann hierbei bidirektional erfolgen und es ist nicht ausgeschlossen, dass die Rollen von den einzelnen Endgeräten getauscht werden (Master wird zu Slave und umgekehrt). Jedem Gerät stehen für die Datenübertragung ein sogenannter Slot mit einer Länge von 625 μs bereit, wovon maximal 5 hintereinander von einem Gerät belegt werden dürfen. Typische Piconetzwerke bestehen jedoch nur aus einem Master und einem Slave.
Weil der verwendete Frequenzbereich auch von anderen Bluetooth Piconetzwerken oder z.B. Wlan verwendet wird und es dadurch zu Störungen bei der Datenübertragung kommen kann, ist es notwendig ein Verfahren zu verwenden, was die Wahrscheinlich des Auftretens einer Störung minimiert und eventuell eine Fehlerkorrektur durchführt. Bluetooth ist in der Lage für jedes Datenframe einzelne Bits für die Fehlerkorrektur hinzufügen und bei nicht erfolgreicher Übertragung das Paket noch einmal zu senden. Außerdem wird bei diesem Standard das sogenannte „Frequency Hopping Spread Spectrum“ angewendet. Dabei findet die Kommunikation nicht konstant auf einer Sendefrequenz statt, sondern wird nach jedem Datenpaket (kann 1, 3 oder 5 Slots lang sein) gewechselt. Dieser Wechsel findet abhängig vom auftretenden Master in einem Verbindungsnetzwerk statt. Womit sichergestellt werden kann, dass zwei Piconetze nicht den gleichen Frequenzsprung durchführen. Insgesamt stehen 79 Sendefrequenzen zur Verfügung.
Bluetooth Geräte sind in 3 Leistungsklassen eingeteilt (siehe Tabelle 8.3), die sich zwischen der verwendeten Leistung und der damit verbundenen Reichweite unterscheiden. Die tatsächlich erreichbare Reichweite hängt jedoch immer von der Umgebung (Wände, Freiraum, etc.) und vor allem von der höchsten Leistungsklasse der beteiligten Endgeräte ab (hohe Klasse bedeutet geringere Reichweite). In der maximalen Leistungsaufnahme sieht man auch den Vorteil zu einer WLAN Verbindung, weil diese ca. 200 mW - 500 mW an Leistung benötigt.
Damit eine Bluetooth Verbindung stattfinden kann, muss zuerst die Umgebung nach vorhandenen Endgeräten abgesucht werden. Dieser Vorgang wird als „Inquiry Scan“ bezeichnet und der Master sendet dabei schnell auf mehreren Kanälen (ca. alle 11 ms ein anderer Kanal) ein entsprechendes Anfrage Paket (ID Pakete). Die empfangsbereiten Geräten horchen eine Frequenz auf „Anfrage Pakete“ ab und wechseln die Frequenz langsamer (ca. alle

Klasse	maximale Sendeleistung	Reichweite
1	100 mW	100 m
2	2.5 mW	50 m
3	1 mW	10 m

Tabelle 8.3: Leistungsklassen bei Bluetooth

1.3 Sekunden). Empfangene ID Pakete werden mit einem Frequency Hop Synchronization Paket beantwortet. Danach befindet sich die Verbindung im „Connection-Active“-Status und es können Daten ausgetauscht werden. Wenn keine Daten für eine Übertragung anfallen, besteht die Möglichkeit die Verbindung schlafen zu stellen, damit der Energieverbrauch reduziert werden kann (Connection - Hold, - Sniff, - Park).
Bluetooth unterstützt mehrere Anwendungen (sogenannte Profile) anhand derer Daten ausgetauscht werden. Durch diese Profile wird gewährleistet, dass Bluetooth Geräte von verschiedenen Herstellern ohne Probleme untereinander kommunizieren können. Die bekanntesten sind HID (Bluetooth Mäuse und Tastaturen), Dial Up Networking (Verbindung PC mit Bluetooth Gerät), Headset (Verbindung Bluetooth Headset mit Handy). Außerdem werden 3 grundlegende Profile genützt, von denen das „Serial Port Profile“ (SPP) interessant ist, weil es in dieser Arbeit verwendet wird. Dabei handelt es sich um einen Emulator für die serielle Schnittstelle der eingesetzt werden kann um die klassische kabelgebundene Übertragung zu ersetzen. Für die beteiligten Endgeräte ändert sich dabei nichts, weil diese weiterhin die Daten an die jeweiligen Bluetooth Geräte über die normale serielle Schnittstelle senden. Die Daten werden danach vom Bluetooth Gerät automatisch in ein passendes Paket verpackt und versendet [Sauter M. 2008].

8.7 Power over Ethernet

Unter PoE versteht man das gleichzeitige Nutzen eines Ethernets Verbund zum Datenaustausch und zur Spannungsversorgung. Als Standardsteckverbindung wird das sogenannte RJ 45 verwendet, was in der Norm ISO/IEC 11801 zu finden ist. Für die Übertragung stehen 8 Leitungen, die zu 4 Paaren verdrillt sind zur Verfügung. Abhängig von der Qualität des Kabels, können sie untereinander und auch zusammen durch einen Schirm geschützt werden.
In der Norm IEEE 802.3af sind die Einzelheiten zu PoE zu finden, damit die Datenintegrität und Netzwerksicherheit nicht gestört wird.

Ein PoE Gerät gehört einer von 2 Gruppen an. Die erste Gruppe wird PSE (Quellen) genannt und speist die Spannung auf das Verbindungskabel ein. Als 2. Gruppe gibt es die PD (Verbraucher), die die Versorgungsspannung über die Ethernet Verbindung beziehen und somit keine anderen Spannungsquellen benötigten. Die Versorgungsspannung wird hinter den Ethernet Eingängen abgegriffen und auf den gewünschten Spannungspegel geregelt.
Für die Verwendung eines PD ist es nicht notwendig ein bestehendes Netzwerk komplett umzurüsten. Es genügt einen Midspan zu verwenden, den man zwischen Switch und Verbraucher schaltet. Diese sind kostengünstig zu erwerben und erfordern keinen Austausch des Switches. Sie haben aber den Nachteil, dass nur eine PoE Verbindung möglich ist und das sie nicht für größere Übertragungsraten 1000 Mbit/s (1000 BaseT) geeignet sind. Die Spannung wird hier über die 2 nicht verwendete Adernpaare eingespeist (siehe Abbildung 8.13, Bild rechts).

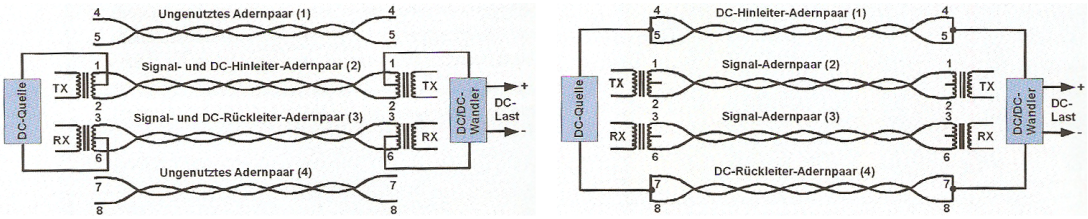


Abbildung 8.13: Verschiedene Möglichkeiten der Spannungseinspeisung. In Bild links wird die Endspan Methode gezeigt und im Bild rechts die Midspan [ELV Journal 4/2009].

Bei der Endspan Lösung muss der Switch PoE fähig sein. Die Spannung wird als Phantomspannung [1] auf die bereits genützten Adernpaare gelegt (siehe Abbildung 8.13, Bild links), die restlichen 2 bleiben für andere Zwecke frei. Durch intelligente Switches kann der Stromverbrauch den angeschlossenen PD angepasst und überwacht werden, womit eine Energieeffiziente Versorgung ermöglicht wird.
An einem Port des PSE darf laut 802.3af Standard eine Spannung zwischen 44 V_{DC} und 57 V_{DC}, bei einem maximalen Strom von 0,35 A eingespeist werden. Durch den Spannungsabfall über das Netzwerkkabel (Widerstand ca. 20 $\Omega/100\ m$), gelangen max. 50 V_{DC} ($0,35\ A * 20\ \Omega = 7\ V$ Spannungsabfall) bzw. minimal 37 V_{DC} zum PD, womit eine minimale Leistung von $37\ V_{DC} * 0,35\ A = 12,95\ W$ zur Verfügung steht. Weiter Verluste entstehen, wie in der Applicaton Note „ANX-PoE-Power von Silvertel“ beschrieben, durch die Verwendung von Brückengleichrichtern (Annahme Verlust von 2 V im schlechtesten Fall, $35\ V_{DC} * 0,35\ A = 12,25\ W$) und keine 100% Effizienz bei der Spannungsumwandlung (Annahme Effizienz im schlechtesten Fall 85%, $85\ \% * 35\ V_{DC} * 0,35\ A = 10,41\ W$). Für viele Geräte reicht das zum Beispiel schon aus (IP-Kamera, IP-Telefon, WLAN Access Point, RFID Leser). Falls das doch nicht reicht, wurde der Standard um „PoE plus“ erweitert, der einen maximalen Strom von 0,72 A erlaubt und um den Standard „PoE Ultra“, der eine maximale Leistung von 60 W zulässt.
Ein Problem der größeren Leistung beim „PoE plus“ ist die Erwärmung im Leiter und der Kontaktabbrand bei den Steckverbindungen. Gegen die Erwärmung empfiehlt es sich qualitativ hochwertige Kabel der höchsten Kategorie 7 zu verwenden, weil bei ihnen der Leiterquerschnitt größer ist, was zu einem geringeren Widerstand führt und weil durch die mehrfache Schirmung die Wärme besser abtransportiert werden kann. Der Kontaktabbrand lässt sich durch ein intelligentes Verhalten beim Abstecken verhindern (Gerät vorher abschalten oder Kontaktstecker anpassen, damit Spannung vor dem Abstecken unterbrochen wird).
Ein PD wird je nach Leistungsbedarf in verschiedene Klassen unterteilt. Damit kann ein PSE die über den Port einzuspeisende Energie anpassen. Die Feststellung zu welcher Klasse das angesteckte PD gehört, erfolgt über ein Protokoll (siehe Abbildung 8.14).
Zuerst wird mit Hilfe von 2 Spannungsrampen (2,8 V und 10,1 V) überprüft ob überhaupt ein Gerät angeschlossen ist. Danach erfolgt die Klassifizierung, um die Klasse des PD festzustellen. Hierfür wird ein Spannungsbereich von 14,5 V - 20,5 V durchlaufen und die Signatur vom PD ausgewertet. In IEEE 802.3af PoE wird ein Gerät einen von 4 Klassen zugeordnet (siehe Tabelle 8.4). Jetzt werden die Spannung und der Strom entsprechend der Leistungsklasse des PD eingestellt und überwacht. Bei einem Trennen des

[1] Bei der Phantomspannung wird die Spannung zwischen Tx und Rx angelegt. Womit kein Spannungsunterschied zwischen den Leitungen vom Adernpaar Tx bzw. Rx auftreten, jedoch zwischen den Adernpaaren Tx und Rx.

Klasse	Minimale Leistung	Maximale Leistung
0	0,44 W	12,95 W
1	0,44 W	3,84 W
2	3,84 W	6,49 W
3	6,49 W	12,95 W
4	reserviert	reserviert

Tabelle 8.4: Klasseneinteilung eines PD je nach Leistungsverbrauch

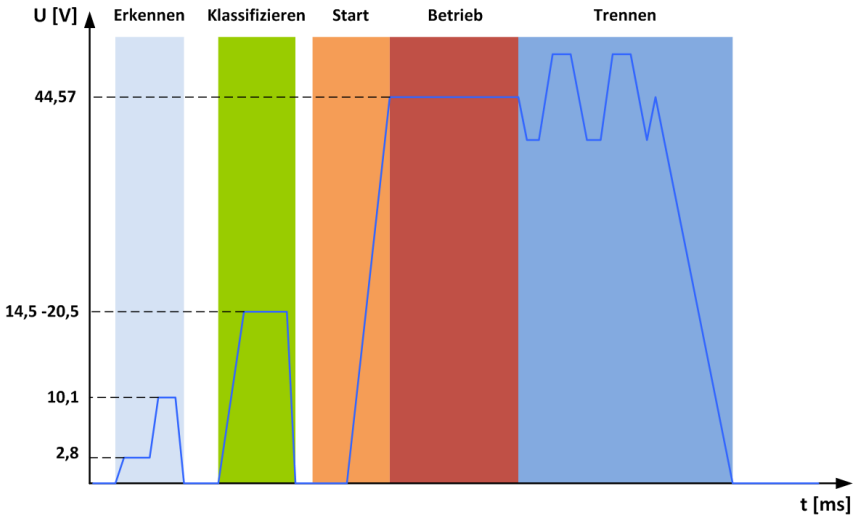


Abbildung 8.14: Protokoll einer PoE Verbindung

Kabels, wird diese Sequenz von Schritten noch einmal ausgeführt.

Problematisch können IDLE Modes der PD werden, weil dadurch die Stromaufnahme sinkt und das PSE das als einen nicht angesteckten Port interpretieren kann. Um das zu vermeiden wird eine kleine Wechselspannung der Gleichspannung überlagert, die einen konstanten Strom über den Verbraucher erzeugt, der noch über der Abschaltschwelle liegt [ELV Journal 4/2009].

8.8 USB

Der universelle serielle Bus (USB) ist im Jahr 1995 aus dem Wunsch heraus entwickelt worden, eine einfache, robuste, billige, echtzeitfähige und erweiterbare einheitliche Schnittstelle zwischen dem PC und den einzelnen Peripherie Geräten zur Verfügung zu stellen. Hier wird nur auf die Grundlagen eingegangen, die für diese Arbeit wichtig sind, eine genauere Beschreibung ist in der USB Spezifikation 2.0 unter zu finden. Die Verbindung zu den einzelnen Peripherie Geräten (Devices) erfolgt sternförmig und wird dabei von einem PC (Host) gesteuert. Ein Device kann im laufenden Betrieb hinzugefügt werden und wird automatisch vom Host erkannt und bekommt von ihm eine eindeutige Adresse zugewiesen. Insgesamt können bis zu 127 Devices verwaltet werden. Um den Anschluss von mehr USB Geräten zu ermöglichen, obwohl die vorhandene Anzahl an USB Anschlüssen vom Host beschränkt ist, können sogenannte USB Hubs zum Einsatz kommen. Mit diesen kann man jedoch, wegen der entstehenden Laufzeitverzögerung, nicht beliebig viele Ebenen erzeugen.
Ein weiterer Vorteil von USB ist, dass eine eingeschränkte Stromversorgung des Devices über die mitgeführten Spannungsleitungen möglich ist. Dabei steht dem Device zum Zeit-

Geschwindigkeitsklassen	Datenrate
Low-Speed	1,5 Mbit/s
Full-Speed	12 Mbit/s
High-Speed	480 Mbit/s
Super-Speed	5 Gbit/s

Tabelle 8.5: Übersicht über die verschiedenen USB Arten mit ihren Geschwindigkeiten

punkt der Enumeration 100 mA (0,5 W) zur Verfügung und danach 500 mA (2,5 W). Bei einem höheren Strombedarf wird das Gerät zur Sicherheit abgeschaltet, wobei einige Host Controller eine kurzzeitige Mehrbeanspruchung tolerieren [USB.org].

8.8.1 USB Arten

Der USB Standard wurde in den letzten Jahren immer weiterentwickelt und für höhere Datenraten angepasst, damit auch Peripheriegeräten mit einem größeren Bandbreitenbedarf über USB mit dem PC verbunden werden können. Die im Moment gebräuchlisten sind in der Tabelle 8.5 ersichtlich.
Für Low-, Full- und High-Speed Geräte gibt es auch einen USB On-The-Go Standard. Dieser ermöglicht eine Kommunikation über USB zwischen mobilen Endgeräten ohne Verwendung eines PC Hosts. D.h. ein Device kann auch eine eingeschränkte Host Funktionalität übernehmen [USB.org].

8.8.2 Datenfluss

Seit USB 2.0 werden unterschiedliche Transfertypen unterstützt:

- Control-Transfer: Verfügbar in jedem USB Device (Enumeration)
- Bulk-Transfer: Große Datenmengen, keine garantierte Bandbreite (z.B. Festplatten)
- Isochron-Transfer: garantierte Bandbreite, keine Fehlerkorrektur (z.B. Audiosignal)
- Interrupt-Transfer: garantierte Bandbreite, geringe Datenmenge (z.B. Maus, Tastatur)

Diese Transfertypen sind wiederum aus mehreren, zeitlich nicht trennbaren Transaktionen aufgebaut. Dabei kann der Datentransfer innerhalb einer Transaktion immer nur in eine Richtung erfolgen (Ausnahme Control-Transfer). Die Richtung des Datentransfers wird aus der Sicht eines Hosts angegeben. „In“ bedeutet, dass die Daten vom Host gelesen werden und „Out“, dass die Daten in das Device geschrieben werden.
Ein Device kann mehrere Schnittstellen besitzen, die wiederum verschiedene Endpunkte haben können. Der Endpunkt 0 ist allerdings bei jedem Device vorhanden, weil von diesem die Deskriptoren ausgelesen werden.
Eine Datenabfrage geht immer von einem Host aus. Wenn ein Device hinzugefügt wird, werden zuerst die Gerätedaten (Deskriptoren) mit einem Control-Transfer ausgelesen. Dieser Vorgang wird auch Enumeration genannt und spricht immer den Endpunkt 0 des Devices an. Jetzt bekommt das Device eine eindeutige Adresse vom Host zugewiesen, an die die USB Pakete gesendet werden. Die Deskriptoren werden in 4 Arten unterteilt. In jeder werden devicespezifische Daten ausgelesen, die für die Kommunikation wichtig

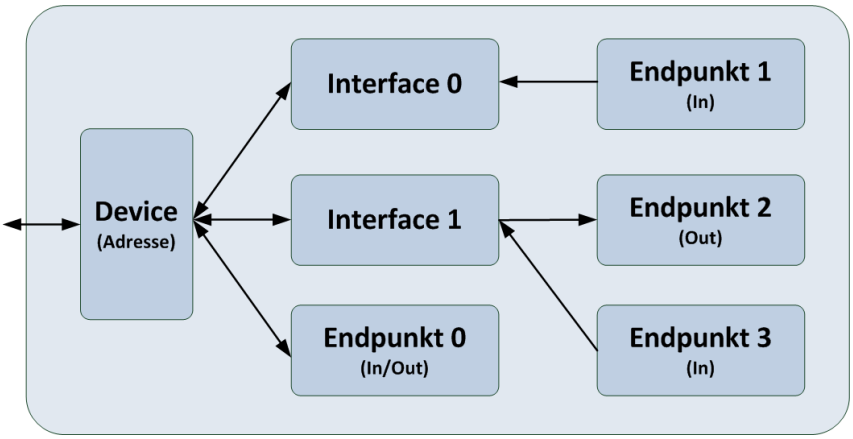


Abbildung 8.15: Übersicht über die verschiedenen Deskriptoren in einem USB Device.

sind (Anzahl Endpunkt und Datenrichtung, Größe der Buffer, Schnittstellen, Vendor und Produkt ID, Paktgröße, etc.) [Elektor Juni 2011]:

- Geräte Deskriptor: Gibt es nur einmal
- Konfigurations Deskriptor: Diese sind z.B. bei unterschiedlichen Stromaufnahmen sinnvoll. Beispiel: Die 100 mA in der Enumeration reichen für das Anlaufen der Festplatte nicht, deswegen gibt es dort verschiedene Konfigurationen
- Schnittstellen Deskriptor: Devices können über mehrere Schnittstellen verfügen z.B. die Schnittstelle Speicher, Audio Wiedergabe, Drucken, etc.
- Endpunkte Deskriptor: Auskunft über die vorhanden Endpunkte (Richtung, Buffer, etc.).

Ein grafisches Beispiel über die verschiedenen Deskriptoren ist in der Abbildung 8.15 zu finden. Im Anhang C befinden sich die Informationen aus den Deskriptoren der Canon EOS 500D.

8.8.3 Physikalische Ebene

In einem USB Kabel für Low, Full und High Speed Datenraten (USB 2.0 und abwärts) befinden sich 4 Adern mit unterschiedlichen Farben (Schwarz - GND, Rot - 5V, Weiß - D-, Grün - D+). 2 sind für die Spannungsversorgung vorgesehen und 2 für die Datenübertragung (Abbildung 8.17 Bild a). Die Leitungen für die Datenübertragung sind zusätzlich verdrillt, um Störeinflüsse zu verringern. Eine Übersicht der Steckverbindung ist in der Abbildung 8.16 ersichtlich, wobei bei PCs standardmäßig der Stecker vom Typ A verwendet wird. Als Low Cost Variante ist es auch möglich die USB Anschlüsse als Leiterbahnen auf der Platine zu implementieren (siehe Abbildung 8.17 Bild b).

Es wird eine differentielle Datenübertragung im NRZI Verfahren durchgeführt, wodurch die Taktfrequenz mitübertragen wird und die verwendete Codierung auch wieder rückgewonnen werden kann. Die Länge eines USB Kabeln ist auf wenige Meter beschränkt. Beim Anschluss eines USB Devices, wird die verwendete Geschwindigkeit über den Status der D+ bzw. D- Leitung festgestellt 8.18. Im Host sind die Leitungen D+ und D- über Pull-Down Widerstände auf Masse gelegt. Wenn das Device über einen Pull-Up Widerstand die D+ Leitung auf „high“ zieht, handelt es sich um ein Full oder High Speed fähiges Device, wenn der Pull-Up Widerstand sich auf der D- Leitung befindet, um ein

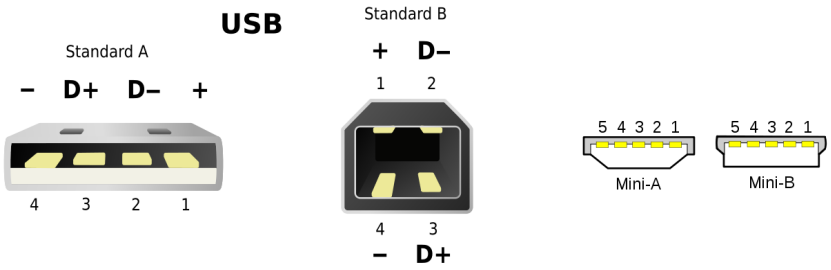
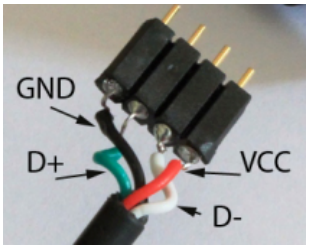
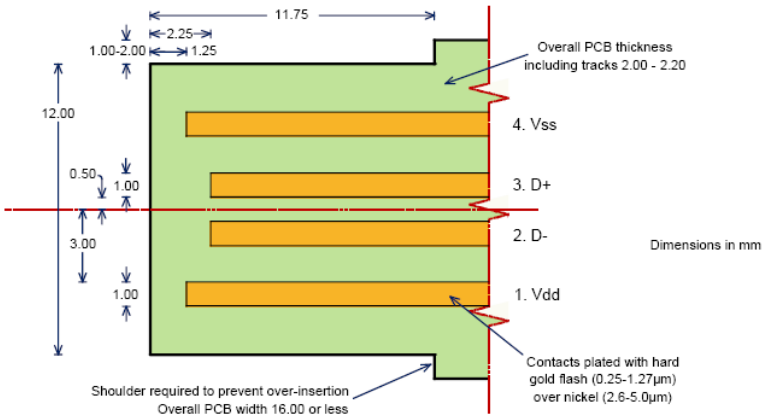


Abbildung 8.16: Übersicht über die verschiedenen USB Steckverbindungen [Wikipedia - USB].

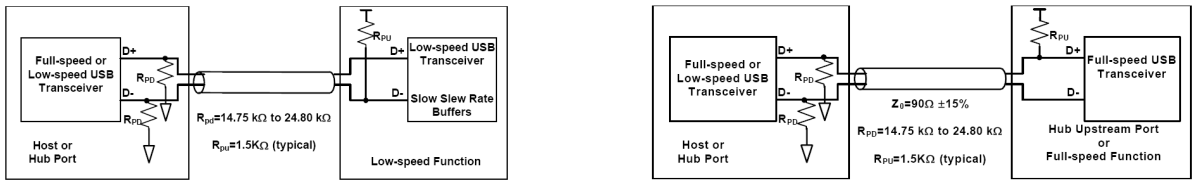


(a) Die 4 Adern einer USB Leitung



(b) Leiterbahnen als USB Anschluss auf einer Platine [Datenblatt USB-232]

Abbildung 8.17: USB Leitungen und Layout für einen USB Anschluss auf einer Leiterplatte.



(a) Low Speed (b) Full Speed

Abbildung 8.18: Feststellung der Geschwindigkeitsklasse anhand von Pull-Up Widerständen auf den Datenleitungen D+ und D- [USB.org].

Low Speed Device. Die Unterscheidung zwischen Full und High Speed trifft der Host indem er Pakete im High Speed Modus zum Device schickt („J and K chirp“). Schlägt die Übertragung fehl, handelt es sich um in Full Speed Device. Für eine Super Speed Übertragungsgeschwindigkeit sind zusätzliche Datenleitungen notwendig [USB.org].

8.9 Picture Transfer Protocol

Für die Kommunikation eines PC einer Digitalkamera hat sich im Laufe der Zeit das Picture Transfer Protocol als Plattform- und Transportunabhängiger Standard entwickelt. Es beinhaltet die Standardbefehle, Konfigurationen, Events und Verhaltensweise, die ein PTP fähiges Gerät implementiert haben muss. Die genauen Definitionen sind unter ISO 15740:2005 nachzulesen. Im Internet ist häufig noch der ältere Standard PIMA 15740 zu finden. Die Vorteile für die Implementierung des PTP in einem Produkt sind:

- Es ermöglicht Fremdentwicklern die Kamera in ihre Produkte zu integrieren
- Das PTP Protokoll ist in vielen Betriebssystem schon standardmäßig integriert (Apple OS, in Windows ab XP als Media Transport Protokoll), d.h. der Benutzer muss keinen speziellen Treiber für jedes Produkt installieren
- Eine Treiber- und Weiterentwicklung ist nicht mehr nötig, dadurch werden die Entwicklungskosten gesenkt
- Es sind nur eine geringe Anzahl von Standardbefehlen zu implementieren
- Das Protokoll kann um eigene Befehle erweitert werden

Diese Vorteile sind dafür verantwortlich, dass es bereits in 95% der weltweit verkauften Digitalkameras verwendet wird. Zusätzliche herstellerspezifische Erweiterungen können bei der I3A angegeben werden und werden durch eine spezielle 32 Bit lange VendorID angezeigt, anhand derer die Unterscheidungen getroffen werden. Eine Liste der momentan vorhanden ist unter 8.6 zu finden.

Aktuell ist PTP für die Transportebenen USB, Firewire, Infrarot, RS232C und TCP/IP definiert, wovon mindestens eine in jedem PTP fähigen Gerät vorhanden sein muss [I3A]. Außerdem muss jedes Gerät Thumbnails und asynchrone Ereignisse (z.B. Batteriestand, Speicherkarte entfernt, Modus umgestellt, etc.) unterstützen und zusätzlich definierte Bild- und Datenformate bereitstellen. Es wird auch empfohlen folgende optionale Funktionen zu implementieren:

- Möglichkeit direkt auf den Speicher vom Gerät zu schreiben
- Hierarchische Dateistruktur

0x00000001	Eastman Kodak Company
0x00000002	Seiko Epson
0x00000003	Agilent Technologies, Inc
0x00000004	Polaroid Corporation
0x00000005	Agfa-Gevaert
0x00000006	Microsoft Corporation
0x00000007	Equinox Research Ltd
0x00000008	ViewQuest Technologies
0x00000009	STMicroelectronics
0x0000000A	Nikon Corporation
0x0000000B	Canon, Inc
0x0000000C	FotoNation, Inc
0x0000000D	PENTAX Corporation
0x0000000E	Fuji Photo Film Co., LTD

Tabelle 8.6: aufgelistete Vendor Erweiterungen

- Möglichkeit von Multisessions
- Handles sollen auch außerhalb einer Session noch gültig sein
- Unterstützung mehrere Bildformate
- Unterstützung mehrere Thumbnail Bildformate
- Möglichkeit eines Schreibschutzes
- Möglichkeit einer Übertragung von Bildteilbereichen
- Unterstützung mehrere nicht Standardfunktionen (z.B. Sleep Modus)

Im PIMA wird die Übertragung von Daten und Bildelementen definiert, jedoch wird nicht festgelegt wie diese Elemente physisch im Gerät gespeichert werden. Die dabei verwendeten Datenformate sind in der Tabelle 8.10 ersichtlich und werden dort kurz erklärt. Einzelne Enumerationen sind in der angegebenen Quelle [PIMA, Inc.] nachzulesen. Die Transportebene muss in der Lage sein Anschluss Events (Gerät wird angesteckt oder abgesteckt) festzustellen, die Übertragung zu kontrollieren inklusive einer Übertragungsfehlerkorrektur und sie soll asynchrone Events unterstützen.
Die Kommunikation erfolgt immer zwischen einem Initiator (verwaltet die Session und sendet auszuführende Operationen und Events) und einem Responder (führt Operationen aus, gibt ein Feedback zurück und sendet Events). Es ist möglich, dass ein Gerät beide Rollen abwechselnd übernehmen kann. Bis auf den GetDeviceInfo, in dem der DeviceInfo Datensatz des Responders ausgelesen wird, müssen alle anderen Operationen in einer Session stattfinden. In dieser sind die einzelnen Datensätze und Handels zu den Dateien konstant. Innerhalb dieser Session finden sogenannte Transaktionen, deren Aufbau im Bild links in der Abbildung 8.19 zu sehen ist, und Events statt. Jede Transaktion bekommt eine fortlaufenden ID zugewiesen und besteht immer aus der „Operation Request“ und der „Response“ Phase, die „Data“ Phase ist optional. In ihr können die Daten vom Initiator zum Responder (Push) oder umgekehrt (Pull) versendet werden. In der „Operation Request“ und in der „Response“ Phase werden jeweils 30 Bytes übertragen, deren Aufbau in der Tabelle 8.7 ersichtlich ist. Als Beispiel wird die Kommunikation für das Auslesen aller Objekte eines Responders in der Abbildung 8.19 Bild rechts dargestellt. Ein Event kann

Feld	Größe (Bytes)	Datentyp
OperationCode (ResponseCode)	2	UINT16
SessionID	4	UINT32
TransactionID	4	UINT32
Parameter1	4	Any(Special)
Parameter2	4	Any(Special)
Parameter3	4	Any(Special)
Parameter4	4	Any(Special)
Parameter5	4	Any(Special)

Tabelle 8.7: Aufbau eines Operation bzw. Response (abweichende Werte in Klammer) Datensatzes

Feld	Größe (Bytes)	Datentyp
EventCode	2	UINT16
SessionID	4	UINT32
TransactionID	4	UINT32
Parameter1	4	Any
Parameter2	4	Any
Parameter3	4	Any

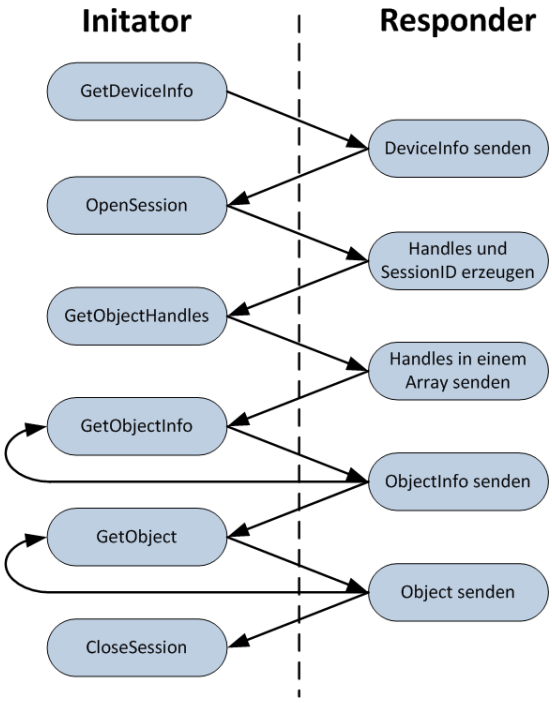
Tabelle 8.8: Aufbau eines Event Datensatzes

entweder vom Initiator oder vom Responder ausgelöst werden. Dabei wird ein Datensatz übertragen, der in der Tabelle 8.8 ersichtlich ist. Die Transaktions ID erlaubt es auf eine vorherige Transaktion Bezug zu nehmen. [PIMA, Inc.]

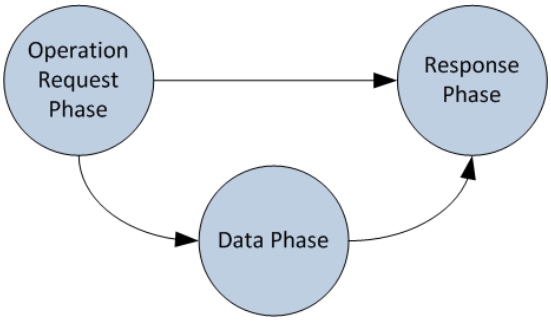
Im nachfolgenden Teil wird kurz auf die USB Transportebene eingegangen, weil diese in dieser Arbeit verwendet wird. Für die Feststellung ob ein USB Device eine Kommunikation über das PTP zulässt, muss es das Standard USB Protokoll unterstützen. Beim Auslesen und Auswerten des Device Descriptors muss der „class code“, „sub-class code“ und der „protocol code“ auf 0 gesetzt sein und zumindest ein Interface mit „class code = 0x06“, „sub-class code = 0x01“ und „protocol code = 0x01“ vorhanden sein. Dieses Interface muss über 3 Endpunkte verfügen:

- Ein Interrupt Endpunkt (IN)
- Ein Bulk Endpunkt (OUT)
- Ein Bulk Endpunkt (IN)

Über den Interrupt Endpunkt werden die asynchronen Ereignisse durchgeführt, die Bulk Endpunkte dienen zur Übertragung der Bilder bzw. deren Informationen und um Geräteeinstellungen abzufragen bzw. zu ändern. Nicht unterstütze oder unbekannte Befehle werden mit einer STALL Anforderung beantwortet werden. Wie asynchrone Ereignisse behandelt werden ist abhängig vom Zustand der aktuellen USB Verbindung, die einzelnen Fälle sind in der Tabelle 8.11 zu sehen. Es ist nicht notwendig eine neue Session zu starten, falls die USB Verbindung zum Device suspended ist. Die Kommunikation zu den PTP Endpunkten erfolgt über Container deren Aufbau in der Tabelle 8.9 ersichtlich ist. Es ist darauf zu achten, dass die Daten von den jeweiligen Container Elementen im litt-



(a) Beispiel einer PTP Session für das Auslesen aller Objekt Informationen eines PTP Responders.



(b) Aufbau einer einzelnen Transaktion im PTP Protokoll.

Abbildung 8.19: Kommunikationsablauf im PTP Protokoll

le Endian Format [1] übertragen werden. Im Anhang C sieht man, dass alle notwendigen Bedingungen für die Implementierung vom PTP über USB bei einer Canon EOS 500D erfüllt sind [USB Device Working Group - PTP].

[1] Das LSB wird an der niedrigsten Speicheradresse gespeichert.

Größe (Bytes)	Feld	Beschreibung
4	Länge des Containers	Gibt die Größe des Containers in Byte an
2	Container Typ	Command, Daten, Response oder Event
2	Code	Zum Container dazu gehöriger Code aus der PTP Definition
4	TransactionID	
n*4	Parameter	Es können n Parameter, abhängig vom Container Typ und Code, folgen.

Tabelle 8.9: USB Übertragungscontainer des PTP Protokolls

Name	Größe (Bytes)	Format	Beschreibung
OperationCode	2	Datacode (UINT16)	Gibt an welche Operation ausgeführt werden soll (Session öffnen, Daten auslesen, etc.)
ResponseCode	2	Datacode (UINT16)	Antwort vom Gerät (Ok, Fehlerinformation, etc.)
EventCode	2	Datacode (UINT16)	Gibt Auskunft über die Art des Events (Speicher voll, Speicher entfernt, etc.)
DevicePropCode	2	Datacode (UINT16)	Gibt Auskunft über die eingestellen Geräteeinstellungen (Bildgröße, Batteriestatus, etc.)
ObjectFormatCode	2	Datacode (UINT16)	Format des Daten Objektes (JPG, TXT, MOV, etc.)
StorageID	4	Special (UINT32)	Ein Gerät kann über mehrere Speicher verfügen, die über unterschiedliche StorageIDs angesprochen werden. Diese setzt sich aus einer PhysicalStorageID (2 Byte) und einer LogialStorageID (2 Byte) zusammen.
ObjectHandle	4	Handle (UINT32)	Objekte werden über Handles angesprochen, diese sind mindestens während einer Session konstant und einheitlich.
DateTime	Variabel	String	
DeviceInfo	Variabel	Dataset	Der Datensatz beinhaltet Informationen über das Gerät. Der Aufbau ist in der Tabelle 8.12 ersichtlich
StorageInfo	Variabel	Dataset	Der Datensatz beinhaltet Informationen über den Speicher. Der Aufbau ist in der Tabelle 8.13 ersichtlich
ObjectInfo	Variabel	Dataset	Der Datensatz beinhaltet Informationen über das Objekt. Der Aufbau ist in der Tabelle 8.14 ersichtlich
DevicePropDesc	Variabel	Dataset	Der Datensatz beinhaltet Informationen über die Einstellungen des Gerätes und in welchen Beriechen diese verändert werden können. Auf eine detaillierte Betrachtung wird hier verzichtet, kann aber unter 13.2 [PIMA, Inc.] nachgelesen werden
DevicePropDescEnum	Variabel	Enumeration	Im Zusammenhang mit dem Datensatz DevicePropDesc relevant
DevicePropDescRange	Variabel	Bereich	Im Zusammenhang mit dem Datensatz DevicePropDesc relevant
Object	Variabel	Variabel	

Tabelle 8.10: Übersicht über die verschiedenen Datenformate im PIMA Protokoll

	USB suspended	USB not suspended
Remote Wakeup eingeschaltet	1. Remote Wakeup wird ausgeführt 2. PIMA Event wird ausgeführt, wenn USB wieder aktiviert ist	PIMA Event wird über den Interface Endpunkt mitgeteilt
Remote Wakeup ausgeschaltet	1. Device löst ein „UnreportedStatus“ Event aus, sobald USB wieder aktiviert ist. PIMA Session bleibt aktiviert. 2. Der Host muss das Device auf Statusänderungen überprüfen.	PIMA Event wird über den Interface Endpunkt mitgeteilt

Tabelle 8.11: PIMA Events Behandlung bei einem USB Device

Feld	Pos.	Größe (Bytes)	Datentyp
Standard Version	1	2	UINT16
Vendor Extension ID	2	4	UINT32
Vendor Extension Version	3	2	UINT16
Vendor Extension Desc	4	Variabel	String
Functional Mode	5	2	UINT16
Operations Supported	6	Variabel	OperationCode Array
Events Supported	7	Variabel	EventCode Array
Device Properties Supported	8	Variabel	DevicePropCode Array
Capture Formats	9	Variabel	ObjectFormatCode Array
Image Formats	10	Variabel	ObjectFormatCode Array
Manufacturer	11	Variabel	String
Model	12	Variabel	String
Device Version	13	Variabel	String
Serial Number	14	Variabel	String

Tabelle 8.12: Device Info Datensatz

Feld	Pos.	Größe (Bytes)	Datentyp
StorageType	1	2	UINT16
FilesystemType	2	2	UINT16
AccessCapability	3	2	UINT16
MaxCapacity	4	8	UINT64
FreeSpaceInBytes	5	8	UINT64
FreeSpaceInImages	6	4	UINT32
StorageDescription	7	Variabel	String
VolumeLabel	8	Variabel	String

Tabelle 8.13: StorageInfo Datensatz

Feld	Pos.	Größe (Byte)	Datentyp
StorageID	1	4	StorageID
ObjectFormat	2	2	ObjectFormatCode
ProtectionStatus	3	2	UINT16
ObjectCompressedSize	4	4	UINT32
ThumbFormat	5	2	ObjectFormatCode
ThumbCompressedSize	6	4	UINT32
ThumbPixWidth	7	4	UINT32
ThumbPixHeight	8	4	UINT32
ImagePixWidth	9	4	UINT32
ImagePixHeight	10	4	UINT32
ImageBitDepth	11	4	UINT32
ParentObject	12	4	ObjectHandle
AssociationType	13	2	AssociationCode
AssociationDesc	14	4	AssociationDesc
SequenceNumber	15	4	UINT32
Filename	16	Variabel	String
CaptureDate	17	Variable	DateTime
ModificationDate	18	Variabel	DateTime
Keywords	19	Variabel	String

Tabelle 8.14: ObjectInfo Datensatz

Abkürzungsverzeichnis

μC ..... Mikrocontroller

API ..... Application Programming Interface

APS-C ..... Advanced Photo System Classic

ARP ..... Address Resolution Protocol

ASIC ..... Application Specific Integrated Circuit

CCD ..... Charge Coupled Device

CE ..... Communautè Europèenne

CMOS ..... Complementary Metal Oxide Semiconductor

DIL ..... Dual In-Line

DLL ..... Dynamic Link Library

DQPSK ..... Differential Quaternary Phase Shift Keying

DSLR ..... Digital Single-Lens Reflex

DSP ..... Digital Signal Processor

EDA ..... Electronic Design Automation

EEPROM ..... Electrically Erasable Programmable Read-Only Memory

EMV ..... Elektromagnetische Verträglichkeit

EV ..... Exposure Value

FIFO ..... First In First Out

FPGA ..... Field Programmable Gate Arrays

GPU ..... Graphics Processing Unit

GSM ..... Groupe Spécial Mobile

HDR ..... High Dynamic Range

HID ..... Human Interface Device

I3A ..... International Imaging Industry Association

IC ..... Integrated Circuit

IDE ..... Integrated Development Environment

IEC ..... Internationale Elektrotechnische Kommission

IEEE ..... Institute of Electrical and Electronics Engineers

IP ..... Internet Protocol

ISA ..... Instruction Set Architecture

ISM ..... Industrial Scientific and Medical

ISO ..... International Organization for Standardization

ISR ..... Interrupt Service Routine

JTAG ..... Joint Test Action Group

LCD ..... Liquid Crystal Display

LED ..... Light Emitting Diode

LSB ..... Least Significant Bit

LUT ..... Look Up Table

MAC ..... Media Access Control

MOD ..... Minimale Objektdistanz

MSN ..... Most Significant Nibble

OSI ..... Open Systems Interconnection

PC ..... Personal Computer

PD ..... Power Device

PHY ..... Physical Layer

PIR ..... Passive Infrared

PoE ..... Power over Ethernet

PSE ..... Power Source Equipment

PSK ..... Phase Shift Keying

PTP ..... Picture Transfer Protocol

RAM ..... Random-Access Memory

RJ ..... Registered Jack

SIL ..... Single In-Line

SPI ..... Serial Peripheral Interface

UDP ..... User Datagram Protocol

UI ..... User Interface

USART ..... Universal Synchronous/Asynchronous Receiver Transmitter

USB ..... Universal Serial Bus

UUID ..... Universally Unique Identifier

VHDL ..... Very High Speed Integrated Circuit Hardware Description Language

WiFi ..... Wireless Fidelity

WLAN ..... Wireless Local Area Network

WPF ..... Windows Presentation Foundation

Abbildungsverzeichnis

Literatur

[vision-doctor 2011] www.vision-doctor.de:
Minimale Objektdistanz und Zwischenringe
Available online at http://www.vision-doctor.de/optische-grundlagen/mod-zwischenringe.html visited on 15th January 2011.

[Adobe] www.adobe.com:
Adobe Photoshop CS3 (aktuelle Version CS5)
Available online at http://www.adobe.com visited on 8th June 2011.

[V-USB] Objective Development V-USB:
Softwareseitige Implementierung einer USB Kommunikation mit einem ATmega
Available online at http://www.obdev.at/products/vusb/index.html visited on 25th June 2011.

[I3A] International Imaging Industry Association:
Picture Transfer Protocol (PTP)
Available online at http://www.i3a.org/technologies/digitalimaging/ptp/ visited on 1st August 2011.

[Photomatix] www.photomatix.de:
Photomatix Pro 4.0
Available online at http://www.photomatix.de/ visited on 8th June 2011.

[WPF 3D] WPF 3D Tutorial:
Windows Presentation Foundation (WPF) 3D Tutorial
Available online at http://kindohm.com/technical/WPF3DTutorial.htm visited on 4th December 2011.

[VNC2 User Guide] Vinculumn II User Guide:
Future Technology Devices International Ltd. 12.05.2011
Available online at http://www.ftdichip.com/Support/Documents/AppNotes/AN_151%20Vinculum%20II%20User%20Guide.pdf visited on 23th September 2011.

[VNC2 Datenblatt] VINCULUM-II EMBEDDED DUAL USB HOST CONTROLLER IC Datasheet:
Future Technology Devices International Limited
Available online at http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_Vinculum-II.pdf visited on 30th September 2011.

[ENC28J60 Datenblatt] ENC28J60 Data Sheet:
Stand-Alone Ethernet Controller with SPI Interface
2008 Microchip Technology Inc.
Available online at http://ww1.microchip.com/downloads/en/DeviceDoc/39662c.pdf visited on 14th Februar 2012.

[DS1621 Datenblatt] DS1621 Data Sheet:
Digital Thermometer and Thermostat
2012 Maxim Integrated Products
Available online at http://datasheets.maxim-ic.com/en/ds/DS1621.pdf visited on 9th March 2012.

[USB.org] www.usb.org:
USB Implementers Forum
Available online at http://www.usb.org/ visited on 29th June 2011.

[Wikipedia - USB] www.wikipedia.org:
Universal Serial Bus Eintrag auf der deutschen Wikipediaseite
Available online at http://de.wikipedia.org/wiki/Universal_Serial_Bus visited on 30th June 2011.

[Wikipedia - ARP] www.wikipedia.org:
Address Resolution Protocol Eintrag auf der deutschen Wikipediaseite
Available online at http://de.wikipedia.org/wiki/Address_Resolution_Protocol visited on 21st February 2012.

[Wikipedia - IP] www.wikipedia.org:
IP-Paket Eintrag auf der deutschen Wikipediaseite
Available online at http://de.wikipedia.org/wiki/IP-Paket visited on 21st February 2012.

[Wikipedia - UDP] www.wikipedia.org:
User Datagram Protocol Eintrag auf der deutschen Wikipediaseite
Available online at http://de.wikipedia.org/wiki/User_Datagram_Protocol visited on 21st February 2012.

[Atmel - AVR32715] www.atmel.com:
AVR32715: AVR32 UC3B Schematic Checklist
Available online at http://www.atmel.com/dyn/resources/prod_documents/doc32095.pdf visited on 29th July 2011.

[Bedienungsanleitung EOS 500D] Canon Inc. 2009:
Bedienungsanleitung EOS 500D
Available online at http://files.canon-europe.com/files/soft33608/Manual/EOS500D_DE_Flat.pdf visited on 21st July 2011.

[USB Device Working Group - PTP] USB Device Working Group. 2000:
Universal Serial Bus Still Image Capture Device Definition
Available online at http://www.usb.org/developers/devclass_docs/usb_still_img10.pdf visited on 8th August 2011.

[PIMA, Inc.] Photographic and Imaging Manufacturers Association, Inc. 2000:
PIMA 15740:2000
Photography ñ Electronic still picture imaging -
Picture Transfer Protocol (PTP)
for Digital Still Photography Devices
Available online at http://www.broomscloset.com/closet/photo/exif/PIMA15740-2000.PDF visited on 8th August 2011.

[Winzker M. 2008] Winzker Marco, 2008:
Elektronik für Entscheider
1. Auflage, Vieweg & Sohn Verlag

[ELV Journal 4/2009] ELV Journal, 4/2009:
Mehr Wissen in Elektronik : Power over Ethernet
Ausgabe Aug./Sept. 2009, ELV Elektronik AG

[Elektor März 2007] Odenwald Michael, Keller Michael, Elektor Zeitschrift, März 2007:
AVR steuert USB
Ausgabe M“arz 2007, Elektor-Verlag GmbH

[Elektor Oktober 2010] Jean-Pierre Gauthier, Elektor Zeitschrift, Oktober 2010:
Intervalltimer für Digitalkameras
Ausgabe Oktober 2010, Elektor-Verlag GmbH

[Kisačanin B., Bhattacharyya S., Chai S. 2009] Kisačanin Branislav, Bhattacharyya Shuvra S., Char Sek, 2009:
Embedded Computer Vision
2009, Springer Verlag

[Mahler G. 2005] Mahler Gerhard, 2005:
Die Grundlagen der Fernsehtechnik
2005, Springer Verlag

[Sauter M. 2008] Sauter Martin, 2008:
Grundkurs Mobile Kommunikationssysteme
3. Auflage 2008, Friedr. Vieweg & Sohn Verlag GWV Fachverlage GmbH

[Azad P., Gockel T., Dillmann R. 2009] Azad Pedram, Gockel Tilo, Dillmann R¸diger, 2008:
Computer Vision - Das Praxisbuch
2. Auflage, Elektor-Verlag GmbH

[Schäffer 2008] Schäffer F., 2008:
Hardware und C-Programmierung in der Praxis
2. Auflage: Elektor Verlag GmbH, Aachen

[Schwabe M., 2010] Schwabe Martin, 2010:
Canon EOS 500D
Markt+Technik Verlag

[Sauer P., 2010] Sauer Peter, 2010:
Hardware-Design mit FPGA
2010, Elektor-Verlag GmbH

[Datenblatt USB-232] Firmware Factory, 2010:
USB-232: Driver-free USB to asynchronous serial UART interface
London 2010, Firmware Factory Ltd

[Datenblatt Ag9000-S] Silver Telecom, 2010:
Ag9000-S: Power-Over-Ethernet Module
Oktober 2009, Silver Telecom Available online at http://www.silvertel.com/DataSheets/Ag9000-Sv2-22.pdf visited on 15th August 2011.

[EOS SDK v2.10] Canon Inc., 2011:
Canon Digital Imaging Developer Programme

March 2011, Canon Inc. Available online at https://www.didp.canon-europa.com/ visited on 25th September 2011.

[Android Bluetooth] Android Developers, 2011:
Bluetooth
March 2011, Canon Inc. Available online at http://developer.android.com/guide/topics/wireless/bluetooth.html visited on 12th November 2011.

[Elektor Mai 2011] Alexander Potchinkov, Mai 2011:
Elektor Zeitschrift: Audio-DSP-Kurs Teil 1: Audiosignalverarbeitung mit einem DSP
Ausgabe Mai 2011, Elektor-Verlag GmbH

[Elektor Juni 2011] Von Guy Weiler, Juni 2011:
Elektor Zeitschrift: Inside USB
Ausgabe Juni 2011, Elektor-Verlag GmbH

Anhang

A MATLAB Code Konzept für 3D Scanner

```
%Testen: 3D Scanner anhand Schärfentiefe
close all;
%Anzahl der Bilder
n = 15;
%Filtermaske
Hs = fspecial('sobel');
Hg = fspecial('gaussian');
%erstes Bild einlesen, damit die Größe bekannt ist
[Zeilen Spalten RGB] = size(imread(['Ausgangsbilder4\1.JPG']));
%3D Bild
DreiDImg = zeros(Zeilen, Spalten);
DreiDImgKanten = zeros(Zeilen, Spalten);
%Schwellwert ab wann es im 3D Bild angezeigt werden soll
Schwelle = 50;
%Schleife über alle Bilder zum einlesen und Kanten detektieren
for i = 1:n,
    %Bilder einlesen und als Grauwertbild umspeichern
    Img = rgb2gray(imread(['Ausgangsbilder4\' int2str(i) '.JPG']));
    figure(i)
    hold on
    subplot(1,2,1);
    %Bild anzeigen
    image(Img);
    %Grauwertbild
    colormap(gray(256));
    subplot(1,2,2);
    %Differenz der Punkte berechnen
    ImgKanten = imfilter(imfilter(Img,Hs),Hg);
    image(ImgKanten);
    hold off
    %3D Bild berechnen
    for js = 1 : Spalten
        for jz = 1 : Zeilen
            if DreiDImgKanten(jz,js) < ImgKanten(jz,js)   ...
                    && ImgKanten(jz,js) > Schwelle
                DreiDImgKanten(jz,js) = ImgKanten(jz,js);
                DreiDImg(jz,js) = i;
            end
        end
    end
end
%3D Bild anzeigen
figure(n+1)
mesh(DreiDImg)
```

B MATLAB Code 3D Scanner Prototyp

```
%Testen: 3D Scanner anhand Schärfentiefe
close all;
%Anzahl der Bilder
n = 49;
%Dateiname
DATEI = 'Kugeln.txt';
%Filtermaske
Hs = fspecial('sobel');
Hg = fspecial('gaussian');
%erstes Bild einlesen, damit die Größe bekannt ist
[Zeilen Spalten RGB] = size(imread(['..\..\..\Test_Fotos\3DScan\
    Kugeln\1.JPG']));
Faktor = Spalten/Zeilen;
[Zeilen Spalten RGB] = size(imresize(imread(['..\..\..\Test_Fotos\3
    DScan\Kugeln\1.JPG']),[250 round(250*Faktor)]));
%3D Bild
DreiDImg = zeros(Zeilen, Spalten);
DreiDImgKanten = zeros(Zeilen, Spalten);
DreiDRGBKanten = zeros(Zeilen, Spalten,RGB);
%Schwellwert ab wann es im 3D Bild angezeigt werden soll
Schwelle = 50;
%Schleife über alle Bilder zum einlesen und Kanten detektieren
for i = 1:n,
    %Bilder einlesen und als Grauwertbild umspeichern
    Img_g = rgb2gray(imresize(imread(['..\..\..\Test_Fotos\3DScan\
        Kugeln\' int2str(i) '.JPG']),[Zeilen Spalten]));
    Img_rgb = imresize(imread(['..\..\..\Test_Fotos\3DScan\Kugeln\'
        int2str(i) '.JPG']),[Zeilen Spalten]);
    %Differenz der Punkte berechnen
    ImgKanten = imfilter(imfilter(Img_g,Hs),Hg);
    %3D Bild berechnen
    for js = 1 : Spalten
        for jz = 1 : Zeilen
            if DreiDImgKanten(jz,js) < ImgKanten(jz,js)  ...
                    && ImgKanten(jz,js) > Schwelle
                DreiDImgKanten(jz,js) = ImgKanten(jz,js);
                DreiDImg(jz,js) = i;
                DreiDRGBKanten(jz,js,1) = Img_rgb(jz,js,1);
                DreiDRGBKanten(jz,js,2) = Img_rgb(jz,js,2);
                DreiDRGBKanten(jz,js,3) = Img_rgb(jz,js,3);
            end
        end
    end
end
disp('creating file: ')
fid = fopen(DATEI,'w');
fprintf(fid,'%i\r\n',Zeilen);
fprintf(fid,'%i\r\n',Spalten');
for js = 1 : Spalten
    for jz = 1 : Zeilen
```

```
47          if DreiDRGBKanten(jz,js,1) == 0 && DreiDRGBKanten(jz,js,2) ==
                0 && ...
48                  DreiDRGBKanten(jz,js,3) ==0
49              fprintf(fid,'%i,%i,%i;%i,%i,%i\r\n',jz-1,js-1,DreiDImg(jz
                    ,js), ...
50              255,255,255);
51          else
52              fprintf(fid,'%i,%i,%i;%i,%i,%i\r\n',jz-1,js-1,DreiDImg(jz
                    ,js), ...
53              DreiDRGBKanten(jz,js,1),DreiDRGBKanten(jz,js,2), ...
54              DreiDRGBKanten(jz,js,3));
55          end
56      end
57  end
58  fclose(fid);
59  disp('Finish')
```

C Device Descriptor der Canon EOS 500D

Mit Hilfe des Programmes „USBView“ erstellt, was ein Teil des Windows Driver Kit ist.

```
Device Descriptor:
bcdUSB:             0x0200
bDeviceClass:         0x00
bDeviceSubClass:      0x00
bDeviceProtocol:      0x00
bMaxPacketSize0:      0x40 (64)
idVendor:           0x04A9 (Canon Inc. (Kosugi Office))
idProduct:          0x31CF
bcdDevice:          0x0002
iManufacturer:        0x01
0x0409: "Canon Inc."
iProduct:             0x02
0x0409: "Canon Digital Camera"
iSerialNumber:        0x00
bNumConfigurations:   0x01
ConnectionStatus: DeviceConnected
Current Config Value: 0x01
Device Bus Speed:     High
Device Address:       0x01
Open Pipes:              3

Configuration Descriptor:
wTotalLength:       0x0027
bNumInterfaces:       0x01
bConfigurationValue:  0x01
iConfiguration:       0x00
bmAttributes:         0xC0 (Bus Powered Self Powered )
MaxPower:             0x01 (2 mA)

Interface Descriptor:
bInterfaceNumber:     0x00
bAlternateSetting:    0x00
bNumEndpoints:        0x03
bInterfaceClass:      0x06
bInterfaceSubClass:   0x01
bInterfaceProtocol:   0x01
iInterface:           0x00

Endpoint Descriptor:
bEndpointAddress:     0x81  IN
Transfer Type:        Bulk
wMaxPacketSize:     0x0200 (512)
bInterval:            0x00

Endpoint Descriptor:
bEndpointAddress:     0x02  OUT
Transfer Type:        Bulk
wMaxPacketSize:     0x0200 (512)
bInterval:            0x00

Endpoint Descriptor:
bEndpointAddress:     0x83  IN
Transfer Type:   Interrupt
wMaxPacketSize:     0x0008 (8)
bInterval:            0x0A
```

D Anleitung für ShutterSpeedTester.exe

1. Nach dem Programmstart sieht die Oberfläche wie in der Abbildung D.1 Bild a aus.

2. Zuerst müssen die Konfigurationsdaten eingestellt werden. Dazu drückt man den Button „Configuration“ um das Konfigurationsfenster zu öffnen. Durch klicken auf den Button „Open Image“ öffnet sich ein Dateiauswahlfeld, mit dem das Bild anhand dessen man die Konfiguration durchführt wird, ausgewählt werden kann (siehe Abbildung D.1 Bild b). Bilder werden durch das Programm automatisch auf eine Breite von 500 Pixeln beschränkt.

3. Jetzt werden die Kontrollpunkte („Checkpoints“) am Foto markiert. Dazu muss man einen Radius (Einheit Pixel) definieren und den Button „Define Check Points“ drücken. Der Radius kann für jeden Kontrollpunkt unterschiedlich sein (Abbildung D.1 Bild c). Die Farbinformation des Kontrollpunktes wird bei der späteren Auswertung über die Summe der einzelnen Farbkanäle der Pixel innerhalb des Radius gemittelt. Der Radius sollte dabei so gewählt werden, dass er nicht über den Rand einer Leuchtdiode herausragt, weil leichte Verwackelungen der Kamera die Messung verfälschen können.

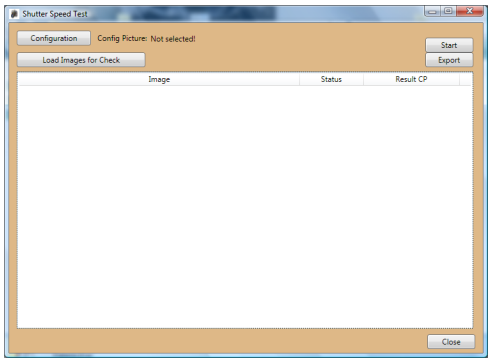
4. Die Reihenfolge des Anlegens, ist für die Position des Kontrollpunktes in der Auswertung ausschlaggebend. Der erste Punkt in der Liste befindet sich ganz Rechts in der Spalte „Result CP“. Ein Kontrollpunkt wird automatisch markiert, wenn man ihn in der Liste auswählt. Durch drücken der „Entf“-Taste kann ein Kontrollpunkt gelöscht werden (siehe Abbildung D.1 Bild d).

5. Für die Feststellung wann ein Kontrollpunkt als 1 markiert wird, ist es notwendig einen Schwellwert (Threshold) für die einzelnen Farbkanäle zu bilden. Dafür muss in den Reiter „Threshold“ gewechselt werden. Hier kann man die gemittelte Farbinformation aller Punkte innerhalb des Radius in den Feldern RGB sehen. Davor muss wieder einen Radius definiert werden und der Button „Set Threshold“ gedrückt werden. Nach einem Klick auf das Bild erscheint ein roter Kreis, der die beinhalteten Pixel markiert. Es empfiehlt sich die Farbinformation bei jedem Kontrollpunkt zu überprüfen, damit ein optimaler Schwellwert gefunden wird. Dieser ist dann in den Feldern „Threshold“ einzutragen (siehe Abbildung D.1 Bild e). Durch schließen des Konfigurationsfensters, werden die Werte automatisch gespeichert.

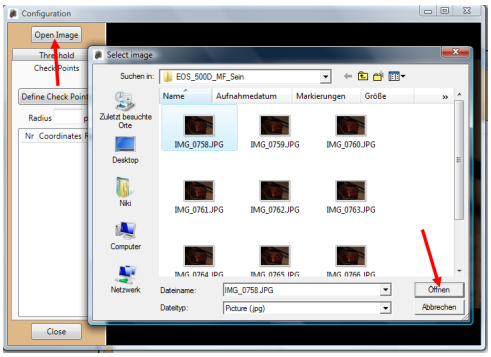
6. Jetzt müssen die auszuwertenden Bilder durch einen Klick auf den Button „Load Images for Check“ ausgewählt werden (siehe Abbildung D.1 Bild f).

7. Durch drücken auf den „Start“-Button wird die Berechnung gestartet. Das Ergebnis der Berechnungen ist nun in der Liste ersichtlich (siehe Abbildung D.1 Bild g).

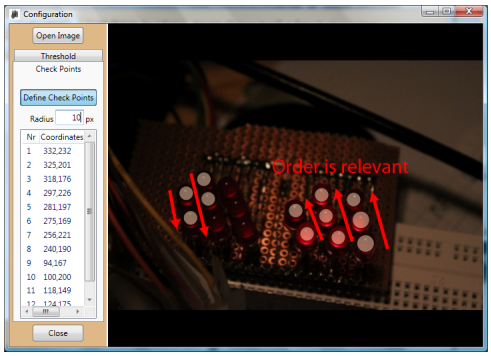
8. Es besteht die Möglichkeit die Messwerte als .csv oder .txt Datei durch Klick auf den Button „Export“ zu exportieren, um sie in einem anderen Programm zu bearbeiten (siehe Abbildung D.1 Bild h).



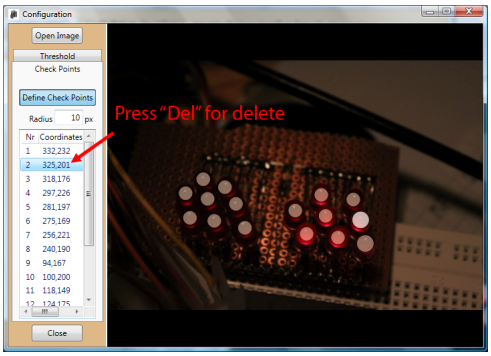
(a) Start



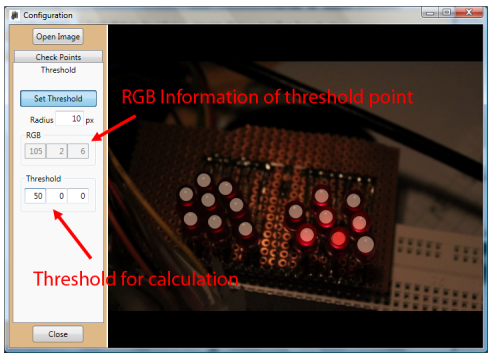
(b) Konfiguration Bild



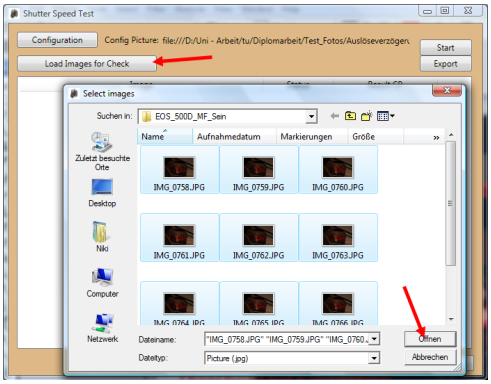
(c) Kontrollpunkte definieren



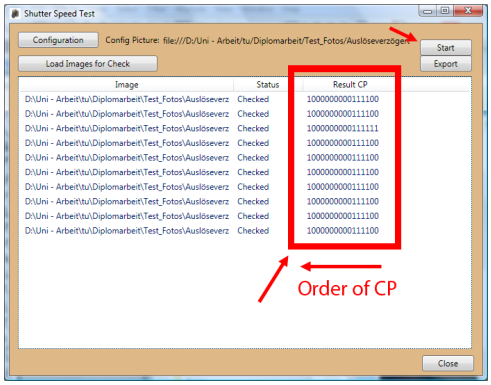
(d) Kontrollpunkte löschen



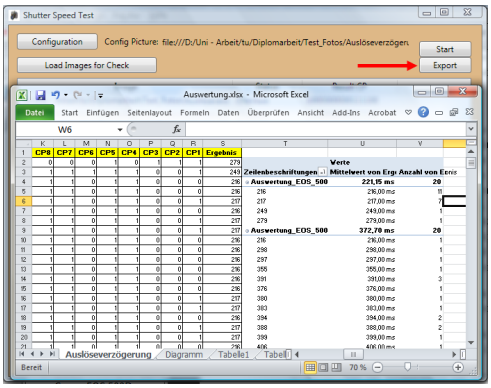
(e) Threshold definieren



(f) Bilder für Kontrolle laden



(g) Resultat der Berechnung



(h) Export

Abbildung D.1: Anleitung für ShutterSpeedTester.exe

Standard Version:	1.00
Vendor Extension ID:	6
Vendor Extension Version:	2.00
Vendor Extension Desc:	
Functional Mode:	0
Operation Supported:	79 Stück
Events Supported:	7 Stück
Device Properties Supported:	3 Stück
Capture Format:	1 Stück
Image Formats:	11 Stück
Manufacturer:	Canon Inc.
Model:	Canon EOS 500D
Device Version:	3-1.1.0
Serial Number:	920edfb4b212497d9e9f32265a30e0b1

Tabelle E.1: USB spezifische Information

E Device Info Datensatz

In der Abbildung E.2 ist die komplette Rückgabe der Datenphase einer PTP GetDeviceInfo Transaktion von einer Canon EOS 500D Kamera abgedruckt. Die Werte sind im Hex Format angegeben und werden innerhalb eines Containers im little Endian Format übertragen, wodurch das niedrigste Byte zuerst übertragen wird.
Die USB Informationen aus den einzelnen Containern ist in der Abbildung E.3 zu finden. Die Device Info ist in der Tabelle E.1 dargestellt. Für eine bessere Übersicht wurden die Bytes eines Containers farblich markiert. Das Device Info Datensatz wurde über USB Protokoll ausgelesen, was an den ersten Containern erkennbar, deren Aufbau in der Tabelle 8.9 zu finden ist. Insgesamt wurden 383 Bytes übertragen. Die Struktur des Device Info Dataset ist in der Tabelle 8.12 ersichtlich. Bei String Werten wird zu Beginn ein Byte übertragen, was die Anzahl der Zeichen eines Strings, inklusive des 0 Bytes am Schluss, angibt. Die maximale Anzahl eines Strings ist dadurch auf 255 Zeichen beschränkt. Die einzelnen Zeichen werden im Unicode Format (2 Bytes pro Zeichen) übertragen. Besteht ein String aus keinem Zeichen, wird nur ein Byte mit dem Wert 0 übertragen. Ähnlich wie bei Strings wird bei einem Array zu Beginn die Anzahl der nachfolgenden Array Elemente übertragen, nur stehen dort 4 Bytes zur Verfügung [PIMA, Inc.].
Auffallend ist, dass als Vendor Extension ID 6 (Microsoft) übertragen wird, obwohl man eigentlich B (Canon) erwarten würde. Das ist sich dabei um keine Fehler handelt, kann unter Device Info Datensatz - EOS 500D kontrolliert werden. Dort wurde ebenfalls der Wert 6 ausgelesen.

7F	01	00	00	02	00	01	10	00	00	00	00	64	00	06	00
00	00	C8	00	00	00	00	4F	00	00	00	14	10	15	10	16
10	01	10	02	10	03	10	06	10	04	10	01	91	05	10	02
91	07	10	08	10	03	91	09	10	04	91	0A	10	1B	10	07
91	0C	10	0D	10	0B	10	05	91	0F	10	06	91	10	91	27
91	0B	91	08	91	09	91	0C	91	0E	91	0F	91	25	91	26
91	15	91	14	91	13	91	16	91	17	91	20	91	F0	91	18
91	21	91	F1	91	1D	91	0A	91	1B	91	1C	91	1E	91	1A
91	53	91	54	91	60	91	55	91	57	91	58	91	59	91	5A
91	1F	91	FE	91	FF	91	28	91	29	91	2D	91	2E	91	2F
91	01	98	02	98	03	98	04	98	05	98	52	90	53	90	57
90	58	90	59	90	5A	90	5F	90	07	00	00	00	09	40	04
40	05	40	03	40	02	40	07	40	01	C1	03	00	00	00	02
D4	07	D4	06	D4	01	00	00	00	01	38	0B	00	00	00	01
30	02	30	06	30	0A	30	08	30	01	38	01	B1	03	B1	02
BF	00	38	04	B1	0B	43	00	61	00	6E	00	6F	00	6E	00
20	00	49	00	6E	00	63	00	2E	00	00	00	0F	43	00	61
00	6E	00	6F	00	6E	00	20	00	45	00	4F	00	53	00	20
00	35	00	30	00	30	00	44	00	00	00	08	33	00	2D	00
31	00	2E	00	31	00	2E	00	30	00	00	00	21	39	00	32
00	30	00	65	00	64	00	66	00	62	00	34	00	62	00	32
00	31	00	32	00	34	00	39	00	37	00	64	00	39	00	65
00	39	00	66	00	33	00	32	00	32	00	36	00	35	00	61
00	33	00	30	00	65	00	30	00	62	00	31	00	00	00	

Tabelle E.2: USB Device Info der Canon EOS 500D

Länge des Containers:	383 Byte
Container Typ:	Data Block
Code:	OK
TransaktionsID:	0

Tabelle E.3: USB spezifische Information

F Protokoll zur Auswertung der USB Pakete

Die nachfolgende USB Pakete wurden mit Hilfe des Programmes USBlyzer 2.0 festgestellt. Verwendet wurde dafür die selbstentwickelte Windows Applikation „CameraControl“. Im Debug Modus wurden die SDK Befehle [EOS SDK v2.10] einzeln ausgeführt und die gesendeten USB Pakete analysiert. Wie beim normalen PTP 8.9 dürfen Befehle an die Kamera nur innerhalb einer Session gesendet werden, sonst wird die Fehlermeldung „Session not Open“ vom Gerät zurück geliefert.

1. Session
   - Öffnen

Phase	Endpunkt	RAW Daten
Control	Bulk Out	10 00 00 00 01 00 14 91 0F 00 00 00 01 00 00 00
Response	Bulk In	0C 00 00 00 03 00 01 20 0F 00 00 00
Control	Bulk Out	10 00 00 00 01 00 15 91 10 00 00 00 01 00 00 00
Response	Bulk In	0C 00 00 00 03 00 01 20 10 00 00 00

   - Schließen

Phase	Endpunkt	RAW Daten
Control	Bulk Out	10 00 00 00 01 00 15 91 1E 00 00 00 00 00 00 00
Response	Bulk In	0C 00 00 00 03 00 01 20 1E 00 00 00
Control	Bulk Out	10 00 00 00 01 00 14 91 1F 00 00 00 00 00 00 00
Response	Bulk In	0C 00 00 00 03 00 01 20 1F 00 00 00

   - Erkenntnis Beim aktivieren einer Session wird 0x0001 als Control-Parameter übertragen und beim deaktivieren 0x0000.

Operationcode:	0x9115 und 0x9114
Anzahl Controlparameter:	1
Parameter 1:	an/aus
Anzahl Rückgabeparamter:	0

2. Kamera auslösen
   - erfolgreich

Phase	Endpunkt	RAW Daten
Control	Bulk Out	0C 00 00 00 01 00 0F 91 1A 00 00 00
Response	Bulk In	10 00 00 00 03 00 01 20 1A 00 00 00 00 00 00 00

   - nicht erfolgreich

Phase	Endpunkt	RAW Daten
Control	Bulk Out	0C 00 00 00 01 00 0F 91 18 00 00 00
Response	Bulk In	10 00 00 00 03 00 01 20 18 00 00 00 19 20 00 00

   - Erkenntnis

Operationcode:	0x910F
Anzahl Controlparameter:	0
Anzahl Rückgabeparamter:	1
Bedeutung Parameter:	Fehler bei Werten <> 0

3. TV Wert ändern

- 1/160 eingestellt (0x73 [EOS SDK v2.10])

Phase	Endpunkt	RAW Daten
Control	Bulk Out	0C 00 00 00 01 00 10 91 19 00 00 00
Data	Bulk Out	18 00 00 00 02 00 10 91 19 00 00 00 0C 00 00 00 02 D1 00 00 73 00 00 00
Response	Bulk In	0C 00 00 00 03 00 01 20 19 00 00 00

- 1/200 eingestellt (0x75 [EOS SDK v2.10])

Phase	Endpunkt	RAW Daten
Control	Bulk Out	0C 00 00 00 01 00 10 91 1A 00 00 00
Data	Bulk Out	18 00 00 00 02 00 10 91 1A 00 00 00 0C 00 00 00 02 D1 00 00 75 00 00 00
Response	Bulk In	0C 00 00 00 03 00 01 20 1A 00 00 00

- 1/8000 eingestellt (0xA0 [EOS SDK v2.10], wird von der Kamera nicht unterstützt)

Phase	Endpunkt	RAW Daten
Control	Bulk Out	0C 00 00 00 01 00 10 91 1D 02 00 00
Data	Bulk Out	18 00 00 00 02 00 10 91 1D 02 00 00 0C 00 00 00 02 D1 00 00 A0 00 00 00
Response	Bulk In	0C 00 00 00 03 00 01 20 1D 02 00 00

- Erkenntnis
  Wird ein von der Kamera nicht unterstützter Wert übertragen, wird von der Kamera der höchste bzw. niedrigste einstellbare Wert ausgewählt. Die hex Werte für den 3. Parameter sind in der Dokumentation von [EOS SDK v2.10] zu finden. Falls die Einstellung aufgrund des momentan eingestellten Modus nicht durchgeführt werden kann, sind keine Veränderungen in der Response Phase feststellbar.

Operationcode:	0x9110
Anzahl Controlparameter:	0
Anzahl Datenparameter:	3
Parameter 1:	0x000C
Parameter 2:	0xD102
Parameter 3:	einzustellender Wert
Anzahl Rückgabeparameter:	0

4. AV Wert ändern

- 5.0 eingestellt (0x2D [EOS SDK v2.10])

Phase	Endpunkt	RAW Daten
Control	Bulk Out	0C 00 00 00 01 00 10 91 2F 00 00 00
Data	Bulk Out	18 00 00 00 02 00 10 91 2F 00 00 00 0C 00 00 00 01 D1 00 00 2D 00 00 00
Response	Bulk In	0C 00 00 00 03 00 01 20 2F 00 00 00

- 11 eingestellt (0x40 [EOS SDK v2.10])

Phase	Endpunkt	RAW Daten
Control	Bulk Out	0C 00 00 00 01 00 10 91 3A 00 00 00
Data	Bulk Out	18 00 00 00 02 00 10 91 3A 00 00 00 0C 00 00 00 01 D1 00 00 2D 00 00 00
Response	Bulk In	0C 00 00 00 03 00 01 20 3A 00 00 00

- 91 eingestellt (0x70 [EOS SDK v2.10], wird von der Kamera nicht unterstützt)

Phase	Endpunkt	RAW Daten
Control	Bulk Out	0C 00 00 00 01 00 10 91 37 02 00 00
Data	Bulk Out	18 00 00 00 02 00 10 91 37 02 00 00 0C 00 00 00 01 D1 00 00 70 00 00 00
Response	Bulk In	0C 00 00 00 03 00 01 20 37 02 00 00

- Erkenntnis
  Wird ein von der Kamera nicht unterstützter Wert übertragen, wird von der Kamera der höchste bzw. niedrigste einstellbare Wert ausgewählt. Die hex Werte für den 3. Parameter sind in der Dokumentation von [EOS SDK v2.10] zu finden. Falls die Einstellung aufgrund des momentan eingestellten Modus nicht durchgeführt werden kann, sind keine Veränderungen in der Response Phase feststellbar. Es wird der gleiche Operationcode, wie für die Änderung des TV Wertes verwendet.

Operationcode:	0x9110
Anzahl Controlparameter:	0
Anzahl Datenparameter:	3
Parameter 1:	0x000C
Parameter 2:	0xD101
Parameter 3:	einzustellender Wert
Anzahl Rückgabeparameter:	0

5. Belichtung ändern

- +2 eingestellt (0x10 [EOS SDK v2.10])

Phase	Endpunkt	RAW Daten
Control	Bulk Out	0C 00 00 00 01 00 10 91 50 00 00 00
Data	Bulk Out	18 00 00 00 02 00 10 91 50 00 00 00 0C 00 00 00 04 D1 00 00 10 00 00 00
Response	Bulk In	0C 00 00 00 03 00 01 20 50 00 00 00

- -2 eingestellt (0xF0 [EOS SDK v2.10])

Phase	Endpunkt	RAW Daten
Control	Bulk Out	0C 00 00 00 01 00 10 91 51 00 00 00
Data	Bulk Out	18 00 00 00 02 00 10 91 51 00 00 00 0C 00 00 00 04 D1 00 00 F0 00 00 00
Response	Bulk In	0C 00 00 00 03 00 01 20 51 00 00 00

- +3 eingestellt (0x18 [EOS SDK v2.10], wird von der Kamera nicht unterstützt)

Phase	Endpunkt	RAW Daten
Control	Bulk Out	0C 00 00 00 01 00 10 91 52 00 00 00
Data	Bulk Out	18 00 00 00 02 00 10 91 52 00 00 00 0C 00 00 00 04 D1 00 00 18 00 00 00
Response	Bulk In	0C 00 00 00 03 00 01 20 52 00 00 00

- Erkenntnis
  Wird ein von der Kamera nicht unterstützter Wert übertragen, wird von der Kamera der höchste bzw. niedrigste einstellbare Wert ausgewählt. Die hex Werte für den 3 Parameter sind unter [EOS SDK v2.10] zu finden. Falls die Einstellung aufgrund des momentan eingestellten Modus nicht durchgeführt werden kann, sind keine Veränderungen in der Response Phase feststellbar. Es wird der gleiche Operationcode, wie für die vorherigen Änderung verwendet.

Operationcode:	0x9110
Anzahl Controlparameter:	0
Anzahl Datenparameter:	3
Parameter 1:	0x000C
Parameter 2:	0xD104
Parameter 3:	einzustellender Wert
Anzahl Rückgabeparameter:	0

6. Live View

- Live View aktivieren (notwendig, weil eine Änderung des Fokuses nur dort vorgenommen werden kann)

Phase	Endpunkt	RAW Daten
Control	Bulk Out	0C 00 00 00 01 00 10 91 19 00 00 00
Data	Bulk Out	18 00 00 00 02 00 10 91 19 00 00 00 0C 00 00 00 B0 D1 00 00 02 00 00 00
Response	Bulk In	0C 00 00 00 03 00 01 20 19 00 00 00

- Live View deaktivieren

Phase	Endpunkt	RAW Daten
Control	Bulk Out	0C 00 00 00 01 00 10 91 1D 00 00 00
Data	Bulk Out	18 00 00 00 02 00 10 91 1D 00 00 00 0C 00 00 00 B0 D1 00 00 00 00 00 00
Response	Bulk In	0C 00 00 00 03 00 01 20 1D 00 00 00

- Erkenntnis
  Für Änderungen an der Kamera wird immer der gleiche Operationcode verwendet. Beim aktivieren wird als 3. Parameter 0x0002 übertragen, beim deaktivieren 0x0000.

Operationcode:	0x9110
Anzahl Controlparameter:	0
Anzahl Datenparameter:	3
Parameter 1:	0x000C
Parameter 2:	0xD1B0
Parameter 3:	an/aus
Anzahl Rückgabeparameter:	0

7. Fokus ändern

- Fernfokussieren 3 (0x8003)

Phase	Endpunkt	RAW Daten
Control	Bulk Out	10 00 00 00 01 00 55 91 1A 00 00 00 03 80 00 00
Response	Bulk In	0C 00 00 00 03 00 01 20 1A 00 00 00

- Nahfokusieren 1 (0x0001)

Phase	Endpunkt	RAW Daten
Control	Bulk Out	10 00 00 00 01 00 55 91 1B 00 00 00 01 00 00 00
Response	Bulk In	0C 00 00 00 03 00 01 20 1B 00 00 00

- Erkenntnis Die möglichen Einstellungen für die Änderung der Fokussierung sind in der „EDSDKTypes.h“ Datei des Digital Imaging Developer Programm zu finden. Vollständigkeitshalber wurde die Enumeration im Anschluss abgebildet.

Operationcode:	0x9155
Anzahl Controlparameter:	1
Parameter 1:	Änderung für den Fokus
Anzahl Rückgabeparameter:	0

```
typedef enum
{
        kEdsEvfDriveLens_Near1  = 0x00000001,
        kEdsEvfDriveLens_Near2  = 0x00000002,
        kEdsEvfDriveLens_Near3  = 0x00000003,
        kEdsEvfDriveLens_Far1   = 0x00008001,
        kEdsEvfDriveLens_Far2   = 0x00008002,
        kEdsEvfDriveLens_Far3   = 0x00008003,
} EdsEvfDriveLens ;
```

G Firmware Ethernet Modul Controller

uart.h

```
1  #ifndef UART_H_
2  #define UART_H_
3
4  unsigned char ser_getc_1 (void);
5  void uart_putc_1(unsigned char);
6  void uart_puts_1 (char *);
7  void uart_ini_1 (void);
8
9  unsigned char ser_getc_0 (void);
10 void uart_putc_0(unsigned char);
11 void uart_puts_0 (char *);
12 void uart_ini_0 (void);
13
14 unsigned char Getrbuffcnt_0(void);
15
16 #endif
```

uart.c

```
1  #ifndef F_CPU
2  #define F_CPU 12000000UL
3  #endif
4
5  #include <avr/signal.h>
6  #include <avr/interrupt.h>
7  #include "uart.h"
8
9  #define UART_BAUD_RATE 19200L
10 #define UART_BAUD_CALC(UART_BAUD_RATE,F_CPU)  ((F_CPU)/((UART_BAUD_RATE)*16L
       )-1)
11
12 #define RBUFFLEN 80 //Pufferlänge für seriellen Empfang
13
14 volatile unsigned char rbuff_1[RBUFFLEN];                  // Ringpuffer
15 volatile uint8_t        rbuffpos_1,              // Position, die als
       nächstes gelesen werden muß im Ringpuffer
16                                       rbuffcnt_1,              // Anzahl
                                             zu lesender Zeichen im Puffer
17                                       udr_data_1;                        //
                                             Daten aus dem UART (volatile,
                                             damit nicht wegoptimiert wird
                                             vom Präprozessor)
18
19 volatile unsigned char rbuff_0[RBUFFLEN];                  // Ringpuffer
20 volatile uint8_t        rbuffpos_0,              // Position, die als
       nächstes gelesen werden muß im Ringpuffer
21                                       rbuffcnt_0,              // Anzahl
                                             zu lesender Zeichen im Puffer
22                                       udr_data_0;                        //
                                             Daten aus dem UART (volatile,
                                             damit nicht wegoptimiert wird
                                             vom Präprozessor)
23
24 // Interruptroutine, die Zeichen aus dem UART sofort ausliest, wenn
       empfangen
```

```
25 ISR (USART1_RX_vect)
26 {
27
28         //uart_puts_1("Interrupt 1");
29         udr_data_1= UDR1;        //Byte auf jeden Fall abholen, sonst
               Endlosinterrupt
30
31         if(rbuffcnt_1 < RBUFFLEN)                  // kein Zeichen in einem
               vollen Ringpuffer überschreiben
32               rbuff_1[(rbuffpos_1+rbuffcnt_1++) % RBUFFLEN] = udr_data_1;
                     // welche Position? Gelesene Zeichenpos + Anzahl
                    Zeichen MODULO Pufferlänge
33         // (von 0 wieder anfangen, wenn Ende erreicht)
34 }
35
36 // Nächstes zu lesendes Zeichen aus Ringpuffer zurückgeben
37 unsigned char ser_getc_1 (void)
38 {
39         unsigned char c;
40
41         //uart_puts_1("Waiting 1");
42         while(!rbuffcnt_1);              // Warte bis ein Zeichen vorhanden
               ist
43         //uart_puts_1("complete 1!");
44         cli(); // Interruptbehandlung kurz aussetzen. Ab jetzt muß es
               schnell gehen (wenig Befehle),
45         //damit Zeichen, die ab jetzt eintreffen nicht verloren gehen.
46         rbuffcnt_1--;                                 // anschl. ein Zeichen
               weniger zum ausgeben
47         c = rbuff_1 [rbuffpos_1++];      // Zeichen holen, was nach dem
               bereits gelesenen liegt
48         if (rbuffpos_1 >= RBUFFLEN)  rbuffpos_1 = 0;
49         // wenn hinterstes Zeichen (rechts im Puffer) gelesen wurde, dann
               wieder vorne anfangen
50
51         sei();           // Interruptbehandlung wieder aktivieren
52
53         return (c);              // Zeichen zurückgeben
54 }
55
56 // Ein Zeichen senden
57 void uart_putc_1(unsigned char c)
58 {
59     while(!(UCSR1A & (1 << UDRE1)));  // warte, bis UDR bereit
60
61     UDR1 = c;     // sende Zeichen
62 }
63
64 // Einen String senden
65 void uart_puts_1(char *s)
66 {
67           while (*s != '\0') { // send all chars except \0 (end of string)
68       uart_putc_1(*s);
69       s++; // increment pointer
70           }
71 }
72
73 // USART initialisieren
74 void uart_ini_1 ()
```

```
75 {
76         sei();  // Interruptbehandlung aktivieren
77
78         // Baudrate wählen
79         UBRR1H=(uint8_t)(UART_BAUD_CALC(UART_BAUD_RATE,F_CPU)>>8);
80         UBRR1L=(uint8_t)UART_BAUD_CALC(UART_BAUD_RATE,F_CPU);
81
82         UCSR1B |= (1 << TXEN1);                    // UART TX (senden)
               einschalten
83         UCSR1B |= (1 << RXEN1 );                   // UART RX (
               empfangen) einschalten
84         UCSR1B |= (1 << RXCIE1 );
85         UCSR1C  |= (1<<UCSZ10)|(1<<UCSZ10);      // Asynchron, 8N1
86         //UCSR1A |= (1<<U2X1);
87 }
88
89
90 // Interruptroutine, die Zeichen aus dem UART sofort ausliest, wenn
       empfangen
91 ISR (USART0_RX_vect)
92 {
93
94         //uart_puts_0("Interrupt 0");
95         udr_data_0= UDR0;        //Byte auf jeden Fall abholen, sonst
               Endlosinterrupt
96
97         if(rbuffcnt_0 < RBUFFLEN)                  // kein Zeichen in einem
               vollen Ringpuffer überschreiben
98                 rbuff_0[(rbuffpos_0+rbuffcnt_0++) % RBUFFLEN] = udr_data_0;
99         // welche Position? Gelesene Zeichenpos + Anzahl Zeichen MODULO
               Pufferlänge
100        // (von 0 wieder anfangen, wenn Ende erreicht)
101 }
102
103 // Nächstes zu lesendes Zeichen aus Ringpuffer zurückgeben
104 unsigned char ser_getc_0 (void)
105 {
106        unsigned char c;
107
108        //uart_puts_0("Waiting 0");
109        while(!rbuffcnt_0);              // Warte bis ein Zeichen vorhanden
               ist
110        //uart_puts_0("complete 0!");
111        cli();  // Interruptbehandlung kurz aussetzen. Ab jetzt muß es
               schnell gehen (wenig Befehle),
112        //damit Zeichen, die ab jetzt eintreffen nicht verloren gehen.
113        rbuffcnt_0--;                                  // anschl. ein
               Zeichen weniger zum ausgeben
114        c = rbuff_0 [rbuffpos_0++];      // Zeichen holen, was nach dem
               bereits gelesenen liegt
115        if (rbuffpos_0 >= RBUFFLEN)  rbuffpos_0 = 0;
116        // wenn hinterstes Zeichen (rechts im Puffer) gelesen wurde, dann
               wieder vorne anfangen
117
118        sei();  // Interruptbehandlung wieder aktivieren
119
120        return (c);      // Zeichen zurückgeben
121 }
122
```

```
123 // Ein Zeichen senden
124 void uart_putc_0(unsigned char c)
125 {
126     while(!(UCSR0A & (1 << UDRE0)));     // warte, bis UDR bereit
127
128     UDR0= c;              // sende Zeichen
129 }
130
131 // Einen String senden
132 void uart_puts_0(char *s)
133 {
134           while (*s != '\0') { // send all chars except \0 (end of string)
135       uart_putc_0(*s);
136       s++; // increment pointer
137           }
138 }
139
140 // USART initialisieren
141 void uart_ini_0 ()
142 {
143         sei();  // Interruptbehandlung aktivieren
144
145         // Baudrate wählen
146         UBRR0H=(uint8_t)(UART_BAUD_CALC(UART_BAUD_RATE,F_CPU)>>8);
147         UBRR0L=(uint8_t)UART_BAUD_CALC(UART_BAUD_RATE,F_CPU);
148
149         UCSR0B |= (1 << TXEN0);                    // UART TX (senden)
              einschalten
150         UCSR0B |= (1 << RXEN0 );                   // UART RX (empfangen)
              einschalten
151         UCSR0B |= (1 << RXCIE0 );
152         UCSR0C  |= (1<<UCSZ00)|(1<<UCSZ00);              // Asynchron, 8N1
153         //UCSR1A |= (1<<U2X1);
154 }
155
156 //Returns the Buffer Count from USART 0
157 unsigned char Getrbuffcnt_0(void)
158 {
159         return rbuffcnt_0;
160 }
```

SPI_Master.h

```
1  /*
2   * SPI_Master.h
3   *
4   * Created: 28.09.2011 12:59:39
5   *  Author: Niki
6   */
7
8
9  #ifndef SPI_MASTER_H_
10 #define SPI_MASTER_H_
11 #include <avr/io.h>
12 #define DDR_SPI DDRB
13 #define DD_MOSI DDB5
14 #define DD_MISO DDB6
15 #define DD_SCK  DDB7
16 #define DD_SPISS DDB4
```

```
17
18 //Chip Select für den Slave (Active Low)
19 #define DDR_SS DDRB
20 #define DD_SS  DDB3
21
22 //Initialisierung für den Master
23 void SPI_MasterInit(void);
24
25 //Übertragen Daten
26 void SPI_MasterTransmit(char data);
27
28 //Empfange Daten
29 char SPI_MasterReceive(void);
30
31
32 #endif /* SPI_MASTER_H_ */
```

SPI_Master.c

```
1  /*
2   * SPI_Master.c
3   *
4   * Created: 28.09.2011 12:42:14
5   *  Author: Niki
6   */
7
8  #include "SPI_Master.h"
9
10 #define SPCR     _SFR_IO8(0x2C)
11 #define SPIE     7
12 #define SPE              6
13 #define DORD     5
14 #define MSTR     4
15 #define CPOL     3
16 #define CPHA     2
17 #define SPR1     1
18 #define SPR0     0
19
20 #define SPDR     _SFR_IO8(0x2E)
21
22 #define SPSR     _SFR_IO8(0x2D)
23 #define SPIF     7
24 #define WCOL     6
25 #define SPI2X    0
26
27
28 //Initialisiert den Master bei einer SPI Verbindung
29 //Setzt die Portrichtigungen richtig
30 //Tragt die richtigen Werte in das SPI Control Register ein
31 void SPI_MasterInit(void)
32 {
33         DDR_SPI |= (1<<DD_MOSI) | (1<<DD_SCK) | (1<<DD_SPISS);  //Mosi, SCK
              und SS vom SPI als Ausgang schalten, damit es funktioniert
34         DDR_SPI &= ~(1<<DD_MISO); //Miso als Eingang
35
36         DDR_SS |= (1<<DD_SS); //SS für den Slave als Ausgang schalten
37
38         SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR1); //SPI Enablen, Master, 1/64
               Clk
```

```
39 }
40
41 //Übertragen Daten
42 void SPI_MasterTransmit(char data)
43 {
44         SPDR = data;                                    // Übertragung
               starten
45         while(!(SPSR & (1<<SPIF)));                     // Warten bis fertig
46 }
47
48 //Empfange Daten
49 char SPI_MasterReceive(void)
50 {
51         while(!(SPSR & (1<<SPIF)));                     // warte bis Empfang
               komplett
52         return SPDR;                                    // empfangenes Byte
              zurueckgeben
53 }
```

ENC28J60.h

```
1  /*
2   * ENC28J60.h
3   *
4   * Created: 16.02.2012 12:31:49
5   *  Author: Niki
6   */
7
8
9  #ifndef ENC28J60_H_
10 #define ENC28J60_H_
11
12 //ControlRegister
13 //XXYY
14 //XX ... Bank
15 //YY ... Address
16 #define ERDPTL   0x0000
17 #define ERDPTH   0x0001
18 #define EWRPTL   0x0002
19 #define EWRPTH   0x0003
20 #define ETXSTL   0x0004
21 #define ETXSTH   0x0005
22 #define ETXNDL   0x0006
23 #define ETXNDH   0x0007
24 #define ERXSTL   0x0008
25 #define ERXSTH   0x0009
26 #define ERXNDL   0x000A
27 #define ERXNDH   0x000B
28 #define ERXRDPTL         0x000C
29 #define ERXRDPTH         0x000D
30 #define ERXWRPTL         0x000E
31 #define ERXWRPTH         0x000F
32 #define EDMASTL 0x0010
33 #define EDMASTH 0x0011
34 #define EDMANDL 0x0012
35 #define EDMANDH 0x0013
36 #define EDMADSTL         0x0014
37 #define EDMADSTH         0x0015
38 #define EDMACSL 0x0016
```

```
39 #define EDMACSH 0x0017
40 #define EIE     0x001B
41 #define EIR     0x001C
42 #define ESTAT   0x001D
43 #define ECON2   0x001E
44 #define ECON1   0x001F
45 #define EHT0    0x0100
46 #define EHT1    0x0101
47 #define EHT2    0x0102
48 #define EHT3    0x0103
49 #define EHT4    0x0104
50 #define EHT5    0x0105
51 #define EHT6    0x0106
52 #define EHT7    0x0107
53 #define EPMM0   0x0108
54 #define EPMM1   0x0109
55 #define EPMM2   0x010A
56 #define EPMM3   0x010B
57 #define EPMM4   0x010C
58 #define EPMM5   0x010D
59 #define EPMM6   0x010E
60 #define EPMM7   0x010F
61 #define EPMCSL  0x0110
62 #define EPMCSH  0x0111
63 #define EPMOL   0x0114
64 #define EPMOH   0x0115
65 #define ERXFCON 0x0118
66 #define EPKTCNT 0x0119
67 #define MACON1  0x0200
68 #define MACON3  0x0202
69 #define MACON4  0x0203
70 #define MABBIPG 0x0204
71 #define MAIPGL  0x0206
72 #define MAIPGH  0x0207
73 #define MACLCON1        0x0208
74 #define MACLCON2        0x0209
75 #define MAMXFLL 0x020A
76 #define MAMXFLH 0x020B
77 #define MICMD   0x0212
78 #define MIREGADR        0x0214
79 #define MIWRL   0x0216
80 #define MIWRH   0x0217
81 #define MIRDL   0x0218
82 #define MIRDH   0x0219
83 #define MAADR5  0x0300
84 #define MAADR6  0x0301
85 #define MAADR3  0x0302
86 #define MAADR4  0x0303
87 #define MAADR1  0x0304
88 #define MAADR2  0x0305
89 #define EBSTSD  0x0306
90 #define EBSTCON 0x0307
91 #define EBSTCSL 0x0308
92 #define EBSTCSH 0x0309
93 #define MISTAT  0x030A
94 #define EREVID  0x0312
95 #define ECOCON  0x0315
96 #define EFLOCON 0x0317
97 #define EPAUSL  0x0318
```

```
98  #define EPAUSH  0x0319
99
100
101 //Instruction Opcode
102 #define ReadControlRegister             0x00
103 #define ReadBufferMemory                0x20
104 #define WriteControlRegister    0x40
105 #define WriteBufferMemory               0x60
106 #define BitFieldSet                             0x80
107 #define BitFieldClear                   0xA0
108 #define SystemResetCommand              0xE0
109
110 //BufferSize Receive
111 #define ReceiveBufferStart      0x0000
112 #define ReceiveBufferEnd        0x1AFF
113
114 //Buffer Transmit
115 //Brauchen nicht viele Bytes
116 #define TransmitBufferStart 0x1B00
117
118 //MAC Adresse
119 #define MAC_ADDRESS 0x001eec06fbe6
120
121 //IP Adresse
122 //#define IP_ADDRESS 0xC0A81432//192.168.20.50
123
124 #include <avr/io.h>
125
126 //Wechselt in eine andere Speicherbank
127 //Param 1: Neue Bank
128 void ChangeBank(char NewBank);
129
130 //Funktion liest ein Control Register
131 //Param 1: Register das zu lesen ist
132 //Param 2: Unterscheidung zwischen ETH und MAC Register (0 = ETH, <> 0 MAC
        oder MII)
133 //Return: Wert des Registers
134 char ReadCR(uint16_t Register, char flag);
135
136 //Funktion schreibt ein Control Register
137 //Param 1: Register da zu schreiben ist
138 //Param 2: Wert der geschrieben werden soll
139 void WriteCR(uint16_t Register, char Value);
140
141 //Paket einlesen
142 void ReadPacket(void);
143
144 //Kontrolliert ob die Bank gewechselt werden muss
145 //Param 1: Register das benutzt werden soll
146 //Return 0: no Change 1: must be change
147 char BankChangeNecessary(int16_t UsedRegister);
148
149 //Funktion überprüft ob der Ethernet Controller schon bereit ist
150 //Return 0: ready 1: not ready
151 char CheckReady();
152
153 //Funktion gibt die Anzahl der empfangenen Pakete zurück
154 //@return: Anzahl der Pakete (EPKTCNT Register)
155 char PaketsReceived(void);
```

```
156
157 //Funktion liest die aktuelle Speicherbank des ENC28J60 aus
158 //Sollte nur im Testmodus aufgerufen werden
159 void ReadBank(void);
160
161 //Funktion setzt ein Bit Field
162 //Param 1: betreffendes Register
163 //Param 2: Bitfeldmuster
164 void BitFS(uint16_t Register, char BitField);
165
166 //Funktion löscht ein Bit Field
167 //Param 1: betreffendes Register
168 //Param 2: Bitfeldmuster
169 void BitFC(uint16_t Register, char BitField);
170
171 //Funktion liest den Buffer Memory aus
172 //Autoinc ist aktiviert
173 //Return Value: Pointer auf den Speicher vom ausgelesenen Datenpaket
174 // 0 Fehler
175 // sonst free nicht vergessen
176 char * ReadBM(void);
177
178 //Funktion gibt den Wert binär aus
179 //Sollte nur zum testen verwendet werden
180 //Param 1: char Wert der auszugeben ist
181 void AusgabeReturnValue(char returnValue);
182
183 //Funktion gibt ein empfangenes Datenpaket aus
184 //Sollte nur zum testen verwendet werden
185 //Param 1: Buffer des Pakets
186 void AusgabeDatenPaket(char *buffer);
187
188 //Funktion überprüft ob das Paket ein Broadcast oder direkt adressiert ist
189 //Return: 0 wenn es nicht zutrifft 1 wenn Broadcast 2 wenn direkt
190 char CheckBroadcastDirect(void);
191
192 //Funktion verwirft ein Paket
193 void ThrowAway(void);
194
195 //Funktion überprüft ob es ein UDP oder ARP Paket ist
196 //Return 0 was anderes, 1 ARP, 2 UDP
197 char CheckUDPARP(void);
198
199 //Funktion sendet ein Daten Paket
200 void Transmit(char *daten, char AnzahlBytes);
201
202 //Funktion generiert ein ARP Antwort Paket
203 void SendARPBack(void);
204
205 //Funktion sendet ein Test Ethernet Paket
206 void SendTest(void);
207
208 //Funktion gibt den Transmit Statusvektor aus
209 //Param 1: Adresse des Speichers, der ausgegeben werden soll
210 void AusgabeTransmitStatusVector(uint16_t Pos);
211
212 //Ausgabe Datenpaket
213 //Param 1: Pointer auf den Ende der Datenübertragung
214 //Param 2: Anzahl der zu übertragenen Zeichen
```

```
215 void AusgabeTransmitPaket(uint16_t EndBereich,char AnzahlBytes);
216
217 //Funktion kontrolliert ob Übertragung erfolgreich war
218 //Param 1: Adresse des Speichers für Status Vektor
219 //Return 0 wenn erfolgreich, sonst error
220 char TransmitStatusVectorCheck(uint16_t Pos);
221
222 //Funktion überprüft UDP Broadcast und generiert gegebenenfalls eine
    Antwort
223 void CheckUDPBroadcast(void);
224
225 //Funktion addiert zwei uint_16 + addiert den Überlauf
226 //return berechneter Wert
227 uint16_t AddTwoValue(uint16_t value1, uint16_t value2);
228
229 //Funktion sendet ein UDP Paket
230 //Param 1 MAC Adresse an die gesendet werden soll
231 //Param 2 IP Adresse an die gesendet werden soll
232 //Param 3 Daten die gesendet werden soll
233 //Param 4 Anzahl der Daten
234 //Param 5 Port
235 void SendUDP(char* MAC_desti, char* IP_desti, char *bufferdaten, uint16_t
    countDaten, uint16_t Port_desti);
236
237 //Funktion liest ein UDP Paket aus und sendet die Daten an die serielle
    Schnittstelle weiter
238 void ReadUDP(void);
239
240 #endif /* ENC28J60_H_ */
```

ENC28J60.c

```
1  /*
2   * ENC28J60.c
3   *
4   * Created: 16.02.2012 13:09:46
5   *  Author: Niki
6   */
7
8  #include "ENC28J60.h"
9
10
11 #ifndef DEBUG
12         #define DEBUG 1
13 #endif
14
15 //Variablen die die aktuelle Start Position des Read Zeigers speichern
16 char ActualReadPositionL;
17 char ActualReadPositionH;
18
19 //IP Adresse
20 //192.168.20.50
21 char IP[] = {50,20,168,192};
22
23 //IP Adresse der letzten Verbindung
24 char LastIP[] = {0,0,0,0};
25
26 //MAC Adresse der letzten Verbindung
27 char LastMAC[] = {0,0,0,0,0,0};
```

```
28
29 //Port
30 uint16_t LastPort;
31
32
33 //Funktion liest die aktuelle Speicherbank des ENC28J60 aus
34 //Sollte nur im Testmodus aufgerufen werden
35 void ReadBank(void)
36 {
37         char testread;
38         char transmitSPIValue;
39         char returnBuffer;
40
41         for(testread = 0x00; testread < 0x20; testread ++)
42         {
43                 transmitSPIValue = ReadControlRegister | testread;
44                 PORTB &= ~(1<<PINB3);
45                 SPI_MasterTransmit(transmitSPIValue);
46                 SPI_MasterTransmit(0);//Dummy Übertragung
47                 returnBuffer= SPI_MasterReceive();
48
49                 AusgabeReturnValue(returnBuffer);
50
51                 PORTB |= (1<<PINB3);
52                 uart_puts_1("\n\r");
53         }
54 }
55
56 //Funktion setzt ein Bit Field
57 //Param 1: betreffendes Register
58 //Param 2: Bitfeldmuster
59 void BitFS(uint16_t Register, char BitField)
60 {
61         if(BankChangeNecessary(Register)==1)
62         {
63                 ChangeBank(((Register>>8) & 0x03));
64         }
65
66         PORTB &= ~(1<<PINB3);
67         SPI_MasterTransmit(BitFieldSet|(Register & 0x1F));
68         SPI_MasterTransmit(BitField);
69         PORTB |= (1<<PINB3);
70
71 }
72
73 //Funktion löscht ein bit Field
74 //Param 1: betreffendes Register
75 //Param 2: Bitfeldmuster
76 void BitFC(uint16_t Register, char BitField)
77 {
78         if(BankChangeNecessary(Register)==1)
79         {
80                 ChangeBank(((Register>>8) & 0x03));
81         }
82
83         PORTB &= ~(1<<PINB3);
84         SPI_MasterTransmit(BitFieldClear|(Register & 0x1F));
85         SPI_MasterTransmit(BitField);
86         PORTB |= (1<<PINB3);
```

```
87
88 }
89
90 //Funktion liest den Buffer Memory aus
91 //Autoinc ist aktiviert
92 //Return Value: Pointer auf den Speicher vom ausgelesenen Datenpaket
93 // 0 Fehler
94 // sonst free nicht vergessen
95 char * ReadBM(void)
96 {
97         char FixedValues[6]; //2 Next Packet Pointer + 4 Receive Status
              Vector
98         uint16_t CountBytes;
99         char *buffer;
100        uint16_t i;
101
102        PORTB &= ~(1<<PINB3);
103        SPI_MasterTransmit(ReadBufferMemory|0x1A);
104        for(i=0;i<6;i++)
105        {
106                SPI_MasterTransmit(0);//Dummy Übertragung
107                FixedValues[i] = SPI_MasterReceive();
108        }
109        //Restliche Anzahl von Datenpaketen bestimmen
110        CountBytes = (FixedValues[3]<<8) | FixedValues[2];
111
112        buffer = malloc((CountBytes+6) * sizeof(char));//die ENJ28C60 Werte
              sollen auch übergeben werden
113
114        if(buffer != 0)
115        {
116                //FixedValues kopieren
117                for(i=0;i<6;i++)
118                {
119                        buffer[i] = FixedValues[i];
120                }
121
122                //restlichen Pakte auslesen
123                for(i=6;i<CountBytes+6;i++)
124                {
125                        SPI_MasterTransmit(0);//Dummy Übertragung
126                        buffer[i] = SPI_MasterReceive();
127                }
128        }
129
130        PORTB |= (1<<PINB3);
131
132        //ERXRDPT erhöhen auf das nächste Paket
133        //damit der Speicher benutzt werden kann
134        WriteCR(ERXRDPTL, FixedValues[0]);
135        WriteCR(ERXRDPTH, FixedValues[1]);
136
137        //ERDPT auf das nächste Paket setzen
138        WriteCR(ERDPTL, FixedValues[0]);
139        WriteCR(ERDPTH, FixedValues[1]);
140
141        //Aktuellen Zeiger speichern
142        ActualReadPositionL = FixedValues[0];
143        ActualReadPositionH = FixedValues[1];
```

```
144
145         //ECON2.PKTDEC bit auf 1 setzen
146         BitFS(ECON2,0x40);
147
148         return buffer;
149 }
150
151 //Funktion liest ein Control Register
152 //Param 1: Register das zu lesen ist
153 //Param 2: Unterscheidung zwischen ETH und MAC Register (0 = ETH, <> 0 MAC
        oder MII)
154 //Return: Wert des Registers
155 char ReadCR(uint16_t Register, char flag)
156 {
157         char ReturnValue;
158
159         if(BankChangeNecessary(Register)==1)
160         {
161                 ChangeBank(((Register>>8) & 0x03));
162         }
163
164         //Aktuelles EconRegister auslesen
165         PORTB &= ~(1<<PINB3);
166         SPI_MasterTransmit(ReadControlRegister|(Register & 0x1F));
167         if(flag = 0)
168         {
169                 SPI_MasterTransmit(0);//Dummy Übertragung
170                 ReturnValue= SPI_MasterReceive();
171         }
172         else
173         {
174                 SPI_MasterTransmit(0);//Dummy Übertragung
175                 SPI_MasterTransmit(0);//Dummy Übertragung
176                 ReturnValue= SPI_MasterReceive();
177         }
178
179         PORTB |= (1<<PINB3);
180
181         return ReturnValue;
182 }
183
184 //Funktion schreibt ein Control Register
185 //Param 1: Register da zu schreiben ist
186 //Param 2: Wert der geschrieben werden soll
187 void WriteCR(uint16_t Register, char Value)
188 {
189         if(BankChangeNecessary(Register)==1)
190         {
191                 ChangeBank(((Register>>8) & 0x03));
192         }
193
194         PORTB &= ~(1<<PINB3);
195         SPI_MasterTransmit(WriteControlRegister|(Register & 0x1F));
196         SPI_MasterTransmit(Value);
197         PORTB |= (1<<PINB3);
198 }
199
200 //Initialiserung ENC28J60
201 void InitENC()
```

```
202 {
203         //char Transmit;
204         //char tmp;
205 
206         //Receive Buffer initialisieren
207         //ERXST und ERXND
208         WriteCR(ERXSTL,(char)(ReceiveBufferStart&0xFF));
209 
210         //ERDPT auf Startwert setzen
211         WriteCR(ERDPTL,(char)(ReceiveBufferStart&0xFF));
212         WriteCR(ERXSTH,(char) ((ReceiveBufferStart>>8)&0xFF));
213 
214         //ERDPT auf Startwert setzen
215         WriteCR(ERDPTH,(char) ((ReceiveBufferStart>>8)&0xFF));
216         WriteCR(ERXNDL, (char) (ReceiveBufferEnd&0xFF));
217 
218         WriteCR(ERXNDH,   (char)((ReceiveBufferEnd>>8)&0xFF));
219 
220         //Aktuellen Zeiger für ReadPosition speichern
221         ActualReadPositionL = (char)(ReceiveBufferStart&0xFF);
222         ActualReadPositionH = (char) ((ReceiveBufferStart>>8)&0xFF);
223 
224         //Mac Init
225         WriteCR(MACON1,   0x01);//MARXEN = 1, Rest ist 0
226         WriteCR(MACON3,   0xF0);//1111 0000 Padcfg, txcrcen, fuldpx, frmlnen
227         WriteCR(MACON4,   0x40);//defer
228 
229         //MAMXFLL und MAMXFLH bleiben auf den Default Wert
230 
231         WriteCR(MABBIPG,   0x12);//12h Half Duplex
232 
233         WriteCR(MAIPGL,   0x12);//Most applications will program this
                register with 12h.
234 
235         WriteCR(MAIPGH,   0x0C);//If half duplex is used, the Non-Back-to-
                Back Inter-Packet Gap register high byte, MAIPGH, should be
                programmed. Most applications will program this register to 0Ch.
236 
237         //If Half-Duplex mode is used, program the
238         //Retransmission and Collision Window registers,
239         //MACLCON1 and MACLCON2. Most applications
240         //will not need to change the default Reset values.
241         //If the network is spread over exceptionally long
242         //cables, the default value of MACLCON2 may
243         //need to be increased.
244 
245         //MAC Adresse
246         WriteCR(MAADR1, (char) (MAC_ADDRESS & 0xFF));
247         WriteCR(MAADR2, (char) ((MAC_ADDRESS>>8) & 0xFF));
248         WriteCR(MAADR3, (char) ((MAC_ADDRESS>>16) & 0xFF));
249         WriteCR(MAADR4, (char) ((MAC_ADDRESS>>24) & 0xFF));
250         WriteCR(MAADR5, (char) ((MAC_ADDRESS>>32) & 0xFF));
251         WriteCR(MAADR6, (char) ((MAC_ADDRESS>>40) & 0xFF));
252 
253         //Interrupt für Receive Pakets einschalten
254         BitFS(EIE,0xC0);
255 
256         //Receiving Pakets einschalten
257         BitFS(ECON1,0x04);
```

```
258
259         //Filter richtig stellen
260         WriteCR(ERXFCON,0x00);
261
262         //Letzen Port init
263         LastPort = 0;
264
265         #ifdef DEBUG
266                 uart_puts_1("Init Ready\n\r");
267         #endif
268 }
269
270 //Wechselt in eine andere Speicherbank
271 //Param 1: Neue Bank
272 void ChangeBank(char NewBank)
273 {
274         char NewEcon;
275
276         #ifdef DEBUG
277                 uart_puts_1("Change Bank\n\r");
278         #endif
279
280         //Aktuelles EconRegister auslesen
281         NewEcon= ReadCR(ECON1,0);
282
283         //Neues Econ1 ermitteln
284         NewEcon = ((NewEcon & 0xFC) | NewBank);
285
286         //Econ1 speichern
287         WriteCR(ECON1, NewEcon);
288 }
289
290 //Kontrolliert ob die Bank gewechselt werden muss
291 //Param 1: Register das benutzt werden soll
292 //Return 0: no Change 1: must be change
293 char BankChangeNecessary(int16_t UsedRegister)
294 {
295         char ActualEcon1;
296
297         if(UsedRegister == EIE || UsedRegister == EIR || UsedRegister ==
                ESTAT || UsedRegister == ECON1 || UsedRegister == ECON2)
298         {
299                 return 0;
300         }
301
302         ActualEcon1 = ReadCR(ECON1,0);
303
304         if(((UsedRegister>>8) & 0x03) != (ActualEcon1&0x03))
305         {
306                 return 1;
307         }
308         else
309         {
310                 return 0;
311         }
312
313 }
314
315
```

```
316 //Funktion überprüft ob der Ethernet Controller schon bereit ist
317 //Return 0: ready 1: not ready
318 char CheckReady()
319 {
320         char ESTATRegister;
321 
322         ESTATRegister = ReadCR(ESTAT,0);
323 
324         if(ESTATRegister & 0x01 == 1)
325         {
326                 #ifdef DEBUG
327                         uart_puts_1("ENC Ready\n\r");
328                 #endif
329                 return 0;
330         }
331         else
332         {
333                 #ifdef DEBUG
334                         uart_puts_1("ENC not Ready\n\r");
335                 #endif
336                 return 1;
337         }
338 
339 }
340 
341 //Funktion gibt die Anzahl der empfangenen Pakete zurück
342 //@return: Anzahl der Pakete (EPKTCNT Register)
343 char PaketsReceived(void)
344 {
345         char EPKTCNTRegister;
346 
347         EPKTCNTRegister = ReadCR(EPKTCNT,0);
348 
349         return EPKTCNTRegister;
350 }
351 
352 //Paket einlesen
353 void ReadPacket(void)
354 {
355         char *buffer;
356 
357         #ifdef DEBUG
358                 uart_puts_1("Einlesen des Paketes\n\r");
359         #endif
360 
361         //Paket einlesen
362         buffer = ReadBM();
363 
364         if (buffer == 0)
365         {
366                 //Fehler
367                 #ifdef DEBUG
368                         uart_puts_1("Nicht genug Speicher vorhanden\n\r");
369                 #endif
370                 return;
371         }
372         else
373         {
374                 //Paket konnte erfolgreich eingelesen werden
```

```
375                 #ifdef DEBUG
376                         uart_puts_1("Ausgabe Datenpaket:\n\r");
377                         AusgabeDatenPaket(buffer);
378                         uart_puts_1("\n\r");
379                 #endif
380         }
381
382         //Pointer auf das Datenpaket löschen
383         free(buffer);
384
385         return;
386 }
387
388
389 //Funktion gibt den Wert binär aus
390 //Sollte nur zum testen verwendet werden
391 //Param 1: char Wert der auszugeben ist
392 void AusgabeReturnValue(char returnValue)
393 {
394         char i;
395
396         for(i=7;i<255;i--)
397         {
398                 if(((returnValue>>i)&0x01)==0x01)
399                 {
400                         uart_putc_1('1');
401                 }
402                 else
403                 {
404                         uart_putc_1('0');
405                 }
406         }
407 }
408
409 //Funktion gibt ein empfangenes Datenpaket aus
410 //Sollte nur zum testen verwendet werden
411 //Param 1: Buffer des Pakets
412 void AusgabeDatenPaket(char *buffer)
413 {
414         uint16_t CountBytes;
415         uint16_t i;
416
417         CountBytes = ((buffer[3]<<8) | buffer[2])+6;
418
419         for(i=0;i<CountBytes;i++)
420         {
421                 AusgabeReturnValue(buffer[i]);
422         }
423 }
424
425 //Funktion überprüft ob das Paket ein Broadcast oder direkt adressiert ist
426 //Return: 0 wenn es nicht zutrifft 1 wenn Broadcast 2 wenn direkt
427 char CheckBroadcastDirect(void)
428 {
429         char FixedValues[12]; //2 Next Packet Pointer + 4 Receive Status
                Vector + 6 MAC Destination Adresse
430         char i;
431
432         PORTB &= ~(1<<PINB3);
```

```
433             SPI_MasterTransmit(ReadBufferMemory|0x1A);
434             for(i=0;i<12;i++)
435             {
436                     SPI_MasterTransmit(0);//Dummy Übertragung
437                     FixedValues[i] = SPI_MasterReceive();
438             }
439
440             PORTB |= (1<<PINB3);
441
442             //ERDPT wieder zurücksetzen
443             WriteCR(ERDPTL, ActualReadPositionL);
444             WriteCR(ERDPTH, ActualReadPositionH);
445
446             if((FixedValues[6]&0x01) == 0x01)
447             {
448                     //Broadcast
449                     return 1;
450             }
451
452             if(FixedValues[6]==((MAC_ADDRESS>>40) & 0xFF) && FixedValues[7]==((
                    MAC_ADDRESS>>32) & 0xFF) && FixedValues[8]==((MAC_ADDRESS>>24) &
                     0xFF) && FixedValues[9]==((MAC_ADDRESS>>16) & 0xFF) &&
                    FixedValues[10]==((MAC_ADDRESS>>8) & 0xFF) && FixedValues[11]==(
                    MAC_ADDRESS & 0xFF) )
453             {
454                     //An an das Modul adressiert
455                     return 2;
456             }
457
458             return 0;
459     }
460
461     //Funktion verwirft ein Paket
462     void ThrowAway(void)
463     {
464             char FixedValues[2]; //2 Next Packet Pointer
465             uint16_t i;
466
467             PORTB &= ~(1<<PINB3);
468             SPI_MasterTransmit(ReadBufferMemory|0x1A);
469             for(i=0;i<2;i++)
470             {
471                     SPI_MasterTransmit(0);//Dummy Übertragung
472                     FixedValues[i] = SPI_MasterReceive();
473             }
474
475             PORTB |= (1<<PINB3);
476
477             //ERXRDPT erhöhen auf das nächste Paket
478             //damit der Speicher benutzt werden kann
479             WriteCR(ERXRDPTL, FixedValues[0]);
480             WriteCR(ERXRDPTH, FixedValues[1]);
481
482             //ERDPT auf das nächste Paket setzen
483             WriteCR(ERDPTL, FixedValues[0]);
484             WriteCR(ERDPTH, FixedValues[1]);
485
486             //Aktuellen Zeiger speichern
487             ActualReadPositionL = FixedValues[0];
```

```
488         ActualReadPositionH = FixedValues[1];
489
490         //ECON2.PKTDEC bit auf 1 setzen
491         BitFS(ECON2,0x40);
492
493
494         #ifdef DEBUG
495                 uart_puts_1("Paket verwerfen\n\r");
496         #endif
497 }
498
499 //Funktion überprüft ob es ein UDP oder ARP Paket ist
500 //Return 0 was anderes, 1 ARP, 2 UDP
501 char CheckUDPARP(void)
502 {
503         char FixedValues[30]; //2 Next Packet Pointer + 4 Receive Status
                Vector + 6 MAC Destination Adresse + 6 Source Adresse + 2 Typ +
                 10 IPv4 Protokoll
504         char i;
505
506         PORTB &= ~(1<<PINB3);
507         SPI_MasterTransmit(ReadBufferMemory|0x1A);
508         for(i=0;i<20;i++)
509         {
510                 SPI_MasterTransmit(0);//Dummy Übertragung
511                 FixedValues[i] = SPI_MasterReceive();
512         }
513
514         PORTB |= (1<<PINB3);
515
516         //ERDPT wieder zurücksetzen
517         WriteCR(ERDPTL, ActualReadPositionL);
518         WriteCR(ERDPTH, ActualReadPositionH);
519
520         if(FixedValues[18] == 0x08 &&  FixedValues[19] == 0x06 )
521         {
522                 //ARP
523                 return 1;
524         }
525
526         if(FixedValues[18] != 0x08 &&  FixedValues[19] != 0x00 )
527         {
528                 //Nicht IPv4
529                 return 0;
530         }
531
532         if(FixedValues[29] != 17 )
533         {
534                 //UDP
535                 return 2;
536         }
537
538         //was anderes
539         #ifdef DEBUG
540                 uart_puts_1("unkown\n\r");
541         #endif
542
543         return 0;
544 }
```

```
545
546 //Funktion generiert ein ARP Antwort Paket
547 //Falls die gesuchte IP die des Modules ist
548 void SendARPBack(void)
549 {
550         char *buffer;
551         char *ARPPaket;
552         char lengthARP;
553         char i;
554
555         #ifdef DEBUG
556                 uart_puts_1("Einlesen des Paketes\n\r");
557         #endif
558
559         //Paket einlesen
560         buffer = ReadBM();
561
562         if (buffer == 0)
563         {
564                 //Fehler
565                 #ifdef DEBUG
566                         uart_puts_1("Nicht genug Speicher vorhanden\n\r");
567                 #endif
568                 return;
569         }
570
571         //Kontrollieren ob die gesuchte IP die des Modules ist
572         if(buffer[44] != IP[3] || buffer[45] !=IP[2] || buffer[46] !=IP[1]
                || buffer[47] !=IP[0])
573         {
574                 //nicht für uns
575                 //Pointer auf das Datenpaket löschen
576                 free(buffer);
577
578                 #ifdef DEBUG
579                         uart_puts_1("ARP nicht für uns\n\r");
580                 #endif
581
582                 return;
583         }
584         //ARP Paket erzeugen
585         lengthARP = 42 +1; //ARP Paket + Control
586
587         ARPPaket = malloc(lengthARP * sizeof(char));
588
589         if(ARPPaket == 0)
590         {
591                 //Fehler
592                 #ifdef DEBUG
593                         uart_puts_1("Nicht genug Speicher vorhanden\n\r");
594                 #endif
595                 return;
596         }
597
598         ARPPaket[0] = 0x0E; //ControlByte
599
600         //DestinationAddress füllen
601         //Source MAC Addresse vom Buffer
602         for(i=1;i<7;i++)
```

```
603         {
604                 ARPPaket[i] = buffer[i+11];//Next Packet Pointer + Receive
                        Status Vector + destination MAC
605         }
606
607         //Source MAC füllen
608         ARPPaket[7] = (char) ((MAC_ADDRESS>>40) & 0xFF);
609         ARPPaket[8] = (char) ((MAC_ADDRESS>>32) & 0xFF);
610         ARPPaket[9] = (char) ((MAC_ADDRESS>>24) & 0xFF);
611         ARPPaket[10] = (char) ((MAC_ADDRESS>>16) & 0xFF);
612         ARPPaket[11] = (char) ((MAC_ADDRESS>>8) & 0xFF);
613         ARPPaket[12] = (char) (MAC_ADDRESS & 0xFF);
614
615         //Type == ARP
616         ARPPaket[13] = 0x08;
617         ARPPaket[14] = 0x06;
618
619         //DatenARP
620         //HardwareAdresstyp
621         ARPPaket[15] = 0x00;
622         ARPPaket[16] = 0x01;
623
624         //Protkolladresstyp IPv4
625         ARPPaket[17] = 0x08;
626         ARPPaket[18] = 0x00;
627
628         //Hardwareadressgröße
629         ARPPaket[19] = 0x06;
630
631         //Protokolladressgröße
632         ARPPaket[20] = 0x04;
633
634         //Operation
635         ARPPaket[21] = 0x00;
636         ARPPaket[22] = 0x02;
637
638         //Quell MAC
639         ARPPaket[23] = (char) ((MAC_ADDRESS>>40) & 0xFF);
640         ARPPaket[24] = (char) ((MAC_ADDRESS>>32) & 0xFF);
641         ARPPaket[25] = (char) ((MAC_ADDRESS>>24) & 0xFF);
642         ARPPaket[26] = (char) ((MAC_ADDRESS>>16) & 0xFF);
643         ARPPaket[27] = (char) ((MAC_ADDRESS>>8) & 0xFF);
644         ARPPaket[28] = (char) (MAC_ADDRESS & 0xFF);
645
646         //Quell IP
647         ARPPaket[29] = IP[3];
648         ARPPaket[30] = IP[2];
649         ARPPaket[31] = IP[1];
650         ARPPaket[32] = IP[0];
651
652         //Ziel MAC
653         ARPPaket[33] = ARPPaket[1];
654         ARPPaket[34] = ARPPaket[2];
655         ARPPaket[35] = ARPPaket[3];
656         ARPPaket[36] = ARPPaket[4];
657         ARPPaket[37] = ARPPaket[5];
658         ARPPaket[38] = ARPPaket[6];
659
660         //Ziel IP
```

```
661             ARPPaket[39] = buffer[34]; //29+6-1
662             ARPPaket[40] = buffer[35];
663             ARPPaket[41] = buffer[36];
664             ARPPaket[42] = buffer[37];
665
666             //Für das senden übergeben
667             Transmit(ARPPaket,lengthARP);
668
669             free(ARPPaket);
670
671             //Pointer auf das Datenpaket löschen
672             free(buffer);
673
674             return;
675     }
676
677     //Funktion sendet ein Daten Paket
678     void Transmit(char *daten, char AnzahlBytes)
679     {
680             char i;
681             uint16_t EndPlace;
682
683             //EWRPTL auf den Anfangswert setzen
684             WriteCR(EWRPTL,(char) (TransmitBufferStart & 0xFF));
685             WriteCR(EWRPTH,(char) ((TransmitBufferStart>>8) & 0xFF));
686
687             //Neuer Endpointer finden
688             EndPlace = TransmitBufferStart + AnzahlBytes-1;
689
690             //Start und Ende setzen
691             WriteCR(ETXSTL,(char) (TransmitBufferStart & 0xFF));
692             WriteCR(ETXSTH,(char) ((TransmitBufferStart>>8) & 0xFF));
693
694             WriteCR(ETXNDL,(char) (EndPlace & 0xFF));
695             WriteCR(ETXNDH,(char) ((EndPlace>>8) & 0xFF));
696
697             PORTB &= ~(1<<PINB3);
698             SPI_MasterTransmit(WriteBufferMemory|0x1A);
699
700             for(i=0;i<AnzahlBytes;i++)
701             {
702                     SPI_MasterTransmit(daten[i]);
703             }
704
705             PORTB |= (1<<PINB3);
706
707             #ifdef DEBUG
708                     //AusgabeTransmitPaket(EndPlace,AnzahlBytes);
709             #endif
710
711             //ECON1.TXRTS. bit auf 1 setzen
712             BitFS(ECON1,0x08);
713
714             #ifdef DEBUG
715                     uart_puts_1("Start Sending\n\r");
716             #endif
717
718             //Solange warten, bis das Feld wieder gelöscht ist
719             while((ReadCR(ECON1,0) & 0x08) == 0x08)
```

```
720         {
721                 ;
722         }
723
724         #ifdef DEBUG
725                 //Fertig mit der Übertragung
726                 if((ReadCR(ESTAT,0) & 0x02) == 0x02 && (
                        TransmitStatusVectorCheck(EndPlace+1)== 1))
727                 {
728                         //Fehler
729                         uart_puts_1("Transmit Error\n\r");
730                         AusgabeTransmitStatusVector(EndPlace+1);
731                 }
732                 else
733                 {
734                         //Kein fehler
735                         uart_puts_1("Transmit erfolgreich\n\r");
736                         //AusgabeTransmitStatusVector(EndPlace+1);
737                 }
738         #endif
739         }
740
741 //Funktion kontrolliert ob Übertragung erfolgreich war
742 //Param 1: Adresse des Speichers für Status Vektor
743 //Return 0 wenn erfolgreich, sonst error
744 char TransmitStatusVectorCheck(uint16_t Pos)
745 {
746         char buffer[7];
747         char i;
748
749         //Read Register auf auslese Bereich
750         WriteCR(ERDPTL,(char) (Pos & 0xFF));
751         WriteCR(ERDPTH,(char) ((Pos>>8) & 0xFF));
752
753         //7 Bytes auslesen
754         PORTB &= ~(1<<PINB3);
755         SPI_MasterTransmit(ReadBufferMemory|0x1A);
756         for(i=0;i<7;i++)
757         {
758                 SPI_MasterTransmit(0);//Dummy Übertragung
759                 buffer[i] = SPI_MasterReceive();
760         }
761         PORTB |= (1<<PINB3);
762
763         //Read Register wieder zurücksetzen
764         //ERDPT
765         WriteCR(ERDPTL,ActualReadPositionL);
766         WriteCR(ERDPTH,ActualReadPositionH);
767
768         if((buffer[2] & 0x80) == 0x80)
769         {
770                 return 0;
771         }
772
773         return 1;
774 }
775
776 //Funktion gibt den Transmit Statusvektor aus
777 //Param 1: Adresse des Speichers, der ausgegeben werden soll
```

```
778 void AusgabeTransmitStatusVector(uint16_t Pos)
779 {
780         char VectorValue;
781         char i;
782 
783         #ifdef DEBUG
784                 uart_puts_1("Ausgabe Vektor:\n\r");
785         #endif
786 
787         //Read Register auf auslese Bereich
788         WriteCR(ERDPTL,(char) (Pos & 0xFF));
789         WriteCR(ERDPTH,(char) ((Pos>>8) & 0xFF));
790 
791         //7 Bytes auslesen
792         PORTB &= ~(1<<PINB3);
793         SPI_MasterTransmit(ReadBufferMemory|0x1A);
794         for(i=0;i<7;i++)
795         {
796                 SPI_MasterTransmit(0);//Dummy Übertragung
797                 VectorValue = SPI_MasterReceive();
798                 AusgabeReturnValue(VectorValue);
799         }
800         PORTB |= (1<<PINB3);
801 
802         #ifdef DEBUG
803                 uart_puts_1("\n\r");
804         #endif
805 
806         //Read Register wieder zurücksetzen
807         //ERDPT
808         WriteCR(ERDPTL,ActualReadPositionL);
809         WriteCR(ERDPTH,ActualReadPositionH);
810 }
811 
812 //Ausgabe Datenpaket
813 //Param 1: Pointer auf den Ende der Datenübertragung
814 //Param 2: Anzahl der zu übertragenen Zeichen
815 void AusgabeTransmitPaket(uint16_t EndBereich,char AnzahlBytes)
816 {
817         char VectorValue;
818         char i;
819 
820         #ifdef DEBUG
821                 char Buff[100];
822                 uart_puts_1("Ausgabe SendeDaten:\n\r");
823                 sprintf(Buff,"Start: %d Ende: %d \r\n",TransmitBufferStart,
                        EndBereich);
824                 uart_puts_1(Buff);
825         #endif
826 
827         //Read Register auf auslese Bereich
828         WriteCR(ERDPTL,(char) (TransmitBufferStart & 0xFF));
829         WriteCR(ERDPTH,(char) ((TransmitBufferStart>>8) & 0xFF));
830 
831         //7 Bytes auslesen
832         PORTB &= ~(1<<PINB3);
833         SPI_MasterTransmit(ReadBufferMemory|0x1A);
834         for(i=0;i<AnzahlBytes;i++)
835         {
```

```
836                     SPI_MasterTransmit(0);//Dummy Übertragung
837                     VectorValue = SPI_MasterReceive();
838                     AusgabeReturnValue(VectorValue);
839             }
840             PORTB |= (1<<PINB3);
841
842             #ifdef DEBUG
843                     uart_puts_1("\n\r");
844             #endif
845
846             //Read Register wieder zurücksetzen
847             //ERDPT
848             WriteCR(ERDPTL, ActualReadPositionL);
849             WriteCR(ERDPTH, ActualReadPositionH);
850     }
851
852     //Funktion sendet ein Test Ethernet Paket
853     void SendTest(void)
854     {
855             char *buffer;
856             char i;
857
858             #ifdef DEBUG
859                     uart_puts_1("Senden eines Testpaketes\n\r");
860             #endif
861
862
863             buffer = malloc(23*sizeof(char));
864
865             if(buffer == 0)
866             {
867                     //Fehler
868                     #ifdef DEBUG
869                             uart_puts_1("Nicht genug Speicher vorhanden\n\r");
870                     #endif
871                     return;
872             }
873
874             buffer[0] = 0x0F; //ControlByte
875
876             for(i = 0; i< 6;i++)
877             {
878                     buffer[i+1] = i+2;
879             }
880
881             //Source MAC füllen
882             buffer[7] = (char) ((MAC_ADDRESS>>40) & 0xFF);
883             buffer[8] = (char) ((MAC_ADDRESS>>32) & 0xFF);
884             buffer[9] = (char) ((MAC_ADDRESS>>24) & 0xFF);
885             buffer[10] = (char) ((MAC_ADDRESS>>16) & 0xFF);
886             buffer[11] = (char) ((MAC_ADDRESS>>8) & 0xFF);
887             buffer[12] = (char) (MAC_ADDRESS & 0xFF);
888
889             //Länge
890             buffer[13] = 0;
891             buffer[14] = 8;
892
893             //Daten
894             for(i = 0; i< 8;i++)
```

```
895         {
896                 buffer[i+14] = 'A'+i;
897         }
898
899         //Für das senden übergeben
900         Transmit(buffer,23);
901
902         //Pointer auf das Datenpaket löschen
903         free(buffer);
904
905         return;
906 }
907
908 //Funktion überprüft UDP Broadcast und generiert gegebenenfalls eine
        Antwort
909 void CheckUDPBroadcast(void)
910 {
911         char *buffer;
912         char countCheck;
913         char i;
914         char MAC_desti[6];
915         char IP_desti[4];
916         char AntwortDaten[22];
917         uint16_t Port_desti;
918
919         #ifdef DEBUG
920                 uart_puts_1("Einlesen des Paketes\n\r");
921         #endif
922
923         //Paket einlesen
924         buffer = ReadBM();
925
926         if (buffer == 0)
927         {
928                 //Fehler
929                 #ifdef DEBUG
930                         uart_puts_1("Nicht genug Speicher vorhanden\n\r");
931                 #endif
932                 return;
933         }
934
935         countCheck = 20 + (buffer[20] & 0x0F)*4+8;//14 Ethernet Frame +
                Länge im IP Header + UDP Header
936
937
938         //Kontrollieren ob der Text "Tell me!" übertragen wurde
939         //Von Windows werden keine Options verwendet
940         if(buffer[countCheck] != 'T' || buffer[countCheck+1] !='e' ||
                buffer[countCheck+2] !='l' || buffer[countCheck+3] !='l' ||
                buffer[countCheck+4] !=' ' || buffer[countCheck+5] !='m'||
                buffer[countCheck+6] !='e'|| buffer[countCheck+7] !='!')
941         {
942                 //nicht für uns
943                 //Pointer auf das Datenpaket löschen
944                 free(buffer);
945
946                 #ifdef DEBUG
947                         uart_puts_1("UDP Broadcast nicht für uns\n\r");
948                 #endif
```

```
949
950             return;
951         }
952
953         #ifdef DEBUG
954             uart_puts_1("UDP_Broadcast_Anforderung\n\r");
955         #endif
956
957         //Ziel IP
958         IP_desti[3] = buffer[32];
959         IP_desti[2] = buffer[33];
960         IP_desti[1] = buffer[34];
961         IP_desti[0] = buffer[35];
962
963         //MAC Destination
964         for(i=0;i<6;i++)
965         {
966             MAC_desti[i] = buffer[i+12];//Next Packet Pointer + Receive
                    Status Vector + destination MAC
967         }
968
969         //Port
970         Port_desti = ((uint16_t)buffer[42]<<8) + ((uint16_t)buffer[43]);
971
972         //Antwort Daten
973         //IP Modul
974         AntwortDaten[0] = IP[3];
975         AntwortDaten[1] = IP[2];
976         AntwortDaten[2] = IP[1];
977         AntwortDaten[3] = IP[0];
978
979         //MAC Modul
980         AntwortDaten[4] = (char) ((MAC_ADDRESS>>40) & 0xFF);
981         AntwortDaten[5] = (char) ((MAC_ADDRESS>>32) & 0xFF);
982         AntwortDaten[6] = (char) ((MAC_ADDRESS>>24) & 0xFF);
983         AntwortDaten[7] = (char) ((MAC_ADDRESS>>16) & 0xFF);
984         AntwortDaten[8] = (char) ((MAC_ADDRESS>>8) & 0xFF);
985         AntwortDaten[9] = (char) (MAC_ADDRESS & 0xFF);
986
987         //IP Senders letzten Befehles
988         AntwortDaten[10] = LastIP[3];
989         AntwortDaten[11] = LastIP[2];
990         AntwortDaten[12] = LastIP[1];
991         AntwortDaten[13] = LastIP[0];
992
993         //MAC Senders letzten Befehles
994         AntwortDaten[14] = LastMAC[5];
995         AntwortDaten[15] = LastMAC[4];
996         AntwortDaten[16] = LastMAC[3];
997         AntwortDaten[17] = LastMAC[2];
998         AntwortDaten[18] = LastMAC[1];
999         AntwortDaten[19] = LastMAC[0];
1000
1001        //Ziel-Port
1002        AntwortDaten[20] = (char) ((LastPort>>8)&0xFF);
1003        AntwortDaten[21] = (char) (LastPort&0xFF);
1004
1005        //Pointer auf das Datenpaket löschen
1006        free(buffer);
```

```
1007
1008            //UDP AnforderungsPaket senden
1009            SendUDP(MAC_desti, IP_desti, AntwortDaten,22,Port_desti);
1010
1011            return;
1012    }
1013
1014    //Funktion addiert zwei uint_16 + addiert den Überlauf
1015    //return berechneter Wert
1016    uint16_t AddTwoValue(uint16_t value1, uint16_t value2)
1017    {
1018            uint32_t result;
1019            uint16_t returnvalue;
1020
1021            result = (uint32_t) value1 + (uint32_t) value2;
1022
1023            returnvalue = (uint16_t) ((result>>16) & 0xFF) + (uint16_t) (result
                    & 0xFFFF);
1024
1025            return returnvalue;
1026    }
1027
1028    //Funktion sendet ein UDP Paket
1029    //Param 1 MAC Adresse an die gesendet werden soll
1030    //Param 2 IP Adresse an die gesendet werden soll
1031    //Param 3 Daten die gesendet werden soll
1032    //Param 4 Anzahl der Daten
1033    //Param 5 Port
1034    void SendUDP(char* MAC_desti, char* IP_desti, char *bufferdaten, uint16_t
           countDaten, uint16_t Port_desti)
1035    {
1036            uint16_t countBytes;
1037            char *buffer;
1038            uint16_t lengthIP;
1039            uint16_t lengthUDP;
1040            uint16_t CheckSum;
1041            int16_t i;
1042
1043            //Größe des Paketes bestimmen
1044            countBytes = 1+14+20+8+countDaten; //Control Byte + Ethernet Frame
                    + IP Paket + UDP +countDaten
1045
1046            buffer = malloc(countBytes*sizeof(char));
1047
1048            if(buffer == 0)
1049            {
1050                    //Fehler
1051                    #ifdef DEBUG
1052                            uart_puts_1("Nicht genug Speicher vorhanden\n\r");
1053                    #endif
1054                    return;
1055            }
1056
1057            buffer[0] = 0x0F; //ControlByte
1058
1059            //Ethernet Frame
1060            //DestinationAddress füllen
1061            buffer[1]=MAC_desti[0];
1062            buffer[2]=MAC_desti[1];
```

```
1063            buffer[3]=MAC_desti[2];
1064            buffer[4]=MAC_desti[3];
1065            buffer[5]=MAC_desti[4];
1066            buffer[6]=MAC_desti[5];
1067
1068            //Source MAC füllen
1069            buffer[7] = (char) ((MAC_ADDRESS>>40) & 0xFF);
1070            buffer[8] = (char) ((MAC_ADDRESS>>32) & 0xFF);
1071            buffer[9] = (char) ((MAC_ADDRESS>>24) & 0xFF);
1072            buffer[10] = (char) ((MAC_ADDRESS>>16) & 0xFF);
1073            buffer[11] = (char) ((MAC_ADDRESS>>8) & 0xFF);
1074            buffer[12] = (char) (MAC_ADDRESS & 0xFF);
1075
1076            //Type
1077            buffer[13] = 0x08;
1078            buffer[14] = 0x00;
1079
1080            //IP
1081            //Version + Header
1082            buffer[15] = 0x45;
1083
1084            //Type of Service
1085            buffer[16] = 0x00;
1086
1087            //Total Length
1088            lengthIP = 20 + countDaten + 8;//IP Header + Anzahl Daten + UDP
                    Header
1089            buffer[17] = (char) ((lengthIP>>8) & 0xFF);
1090            buffer[18] = (char) (lengthIP & 0xFF);
1091
1092            //Identification
1093            buffer[19] = 0x12;
1094            buffer[20] = 0x34;
1095
1096            //Flags + Fragment Offset
1097            buffer[21] = 0x40;
1098            buffer[22] = 0x00;
1099
1100            //Time to Live
1101            buffer[23] = 0x80;
1102
1103            //Protocol
1104            buffer[24] = 0x11; //17 UDP
1105
1106            //Source Address
1107            buffer[27] = IP[3];
1108            buffer[28] = IP[2];
1109            buffer[29] = IP[1];
1110            buffer[30] = IP[0];
1111
1112            //Destination Address
1113            buffer[31] = IP_desti[3];
1114            buffer[32] = IP_desti[2];
1115            buffer[33] = IP_desti[1];
1116            buffer[34] = IP_desti[0];
1117
1118            //Header Checksum am Schluss
1119            CheckSum = AddTwoValue(((uint16_t) buffer[15]<<8)+(uint16_t) buffer
                    [16],((uint16_t) buffer[17]<<8)+(uint16_t) buffer[18]);
```

```
1120            CheckSum = AddTwoValue(((uint16_t) buffer[19]<<8)+(uint16_t) buffer
                    [20],CheckSum);
1121            CheckSum = AddTwoValue(((uint16_t) buffer[21]<<8)+(uint16_t) buffer
                    [22],CheckSum);
1122            CheckSum = AddTwoValue(((uint16_t) buffer[23]<<8)+(uint16_t) buffer
                    [24],CheckSum);
1123            CheckSum = AddTwoValue(((uint16_t) buffer[27]<<8)+(uint16_t) buffer
                    [28],CheckSum);
1124            CheckSum = AddTwoValue(((uint16_t) buffer[29]<<8)+(uint16_t) buffer
                    [30],CheckSum);
1125            CheckSum = AddTwoValue(((uint16_t) buffer[31]<<8)+(uint16_t) buffer
                    [32],CheckSum);
1126            CheckSum = AddTwoValue(((uint16_t) buffer[33]<<8)+(uint16_t) buffer
                    [34],CheckSum);
1127            CheckSum = ~CheckSum;
1128
1129            buffer[25] =(char) ((CheckSum>>8) & 0xFF);
1130            buffer[26] =(char) (CheckSum & 0xFF);
1131
1132            //UDP
1133            //QuellPort
1134            buffer[35] = (char) ((Port_desti>>8) & 0xFF);
1135            buffer[36] = (char) (Port_desti & 0xFF);
1136
1137            //ZielPort
1138            buffer[37] = (char) ((Port_desti>>8) & 0xFF);
1139            buffer[38] = (char) (Port_desti & 0xFF);
1140
1141            //Längefeld UDP
1142            lengthUDP = 8 +countDaten; //UDP Header + Daten
1143            buffer[39] = (char) ((lengthUDP>>8) & 0xFF);
1144            buffer[40] = (char) (lengthUDP & 0xFF);
1145
1146            //Prüfsummer
1147            buffer[41] = 0x00;
1148            buffer[42] = 0x00;
1149
1150            //Daten
1151            for(i=0;i<countDaten;i++)
1152            {
1153                    buffer[43+i] = bufferdaten[i];
1154            }
1155
1156            //Für das senden übergeben
1157            Transmit(buffer,countBytes);
1158
1159            //Buffer wieder freigeben
1160            free(buffer);
1161    }
1162
1163    //Funktion liest ein UDP Paket aus und sendet die Daten an die serielle
        Schnittstelle weiter
1164    void ReadUDP(void)
1165    {
1166            char *buffer;
1167            char i;
1168            char countDataPosition;
1169            uint16_t countDaten;
1170
```

```
1171            #ifdef DEBUG
1172                    uart_puts_1("Einlesen des Paketes\n\r");
1173            #endif
1174
1175            //Paket einlesen
1176            buffer = ReadBM();
1177
1178            if (buffer == 0)
1179            {
1180                    //Fehler
1181                    #ifdef DEBUG
1182                            uart_puts_1("Nicht genug Speicher vorhanden\n\r");
1183                    #endif
1184                    return;
1185            }
1186
1187            //MAC und IP speichern
1188            LastIP[3] = buffer[32];
1189            LastIP[2] = buffer[33];
1190            LastIP[1] = buffer[34];
1191            LastIP[0] = buffer[35];
1192
1193            for(i=0;i<6;i++)
1194            {
1195                    LastMAC[5-i] = buffer[i+12];//Next Packet Pointer + Receive
                            Status Vector + destination MAC
1196            }
1197
1198
1199
1200            //Daten an serielle Schnistelle senden
1201            //Anzahl der Daten ermitteln
1202            countDataPosition = 20 + (buffer[20] & 0x0F)*4+8;//14 Ethernet
                    Frame + Länge im IP Header + UDP Header
1203
1204            countDaten = (buffer[countDataPosition-4]<<8) + buffer[
                    countDataPosition-3]-8;//Header abziehen
1205
1206            //Port Speichern
1207            LastPort =  ((uint16_t)buffer[42]<<8) + ((uint16_t)buffer[43]);
1208
1209            //Daten an serielle Schnittstelle senden
1210            for(i = 0; i< countDaten;i++)
1211            {
1212                    uart_putc_1('A'+buffer[countDataPosition+i]);
1213                    uart_putc_0(buffer[countDataPosition+i]);
1214            }
1215
1216
1217            return;
1218    }
1219
1220    //Funktion generiert ein Daten UDP Paket
1221    //@Param1: ausgelesener Code
1222    //@Param2: ausgelesene Anzahl der Parameter
1223    //@Param3: TransactionID
1224    //@Param4: ParameterFeld
1225    void CreateUDPData(uint32_t CodeResponseWort, unsigned short countParam,
            uint16_t transID, char *arrayParam)
```

```
1226 {
1227
1228         char *buffer;
1229         uint16_t countBytes;
1230         char i;
1231
1232         #ifdef DEBUG
1233                 uart_puts_1("Versenden Daten Paket\n\r");
1234         #endif
1235
1236         countBytes = 8+countParam; //Header + Daten
1237
1238         //Paket einlesen
1239         buffer = malloc(sizeof(char) *countBytes );
1240
1241         if (buffer == 0)
1242         {
1243                 //Fehler
1244                 #ifdef DEBUG
1245                         uart_puts_1("Nicht genug Speicher vorhanden\n\r");
1246                 #endif
1247                 return;
1248         }
1249
1250         buffer[0]= (char) (transID & 0xff);
1251         buffer[1]= (char) ((transID>>8) & 0xff);
1252
1253         buffer[2] = (char) (CodeResponseWort & 0xff);
1254         buffer[3] = (char) ((CodeResponseWort>>8) & 0xff);
1255         buffer[4] = (char) ((CodeResponseWort>>16) & 0xff);
1256         buffer[5] = (char) ((CodeResponseWort>>24) & 0xff);
1257
1258         buffer[6]= (char) (countParam & 0xff);
1259         buffer[7]= (char) ((countParam>>8) & 0xff);
1260
1261
1262         for(i=0;i<countParam;i++)
1263         {
1264                 buffer[8+i]= arrayParam[i];
1265         }
1266
1267         SendUDP(LastMAC, LastIP, buffer, countBytes, LastPort);
1268
1269         return;
1270 }
```

Atmega324a_Ethernet.h

```
1  /*
2   * Atmega324a_Ethernet.h
3   *
4   * Created: 13.02.2012 17:04:24
5   *  Author: Niki
6   */
7
8
9  #ifndef ATMEGA324A_ETHERNET_H_
10 #define ATMEGA324A_ETHERNET_H_
11
```

```
12 #include <avr/io.h>
13 #include <stdio.h>
14 #include <util/delay.h>
15 #include <avr/interrupt.h>
16 #include "SPI_Master.h"
17 #include "uart.h"
18 #include "ENC28J60.h"
19
20
21 //ReadContainerUSART
22 //Funktion liest einen Container ein
23 //@Param1: ausgelesener Code
24 //@Param2: ausgelesene Anzahl der Parameter
25 //@Param3: TransactionID
26 // return : Parameterfeld (free nicht vergessen)
27 char * ReadContainerUSART(uint32_t *CodeResponseWort, unsigned short *
       countParam, uint16_t *transID);
28
29 #endif /* ATMEGA324A_ETHERNET_H_ */
```

Atmega324a_Ethernet.c

```
1  /*
2   * Atmega324a_Ethernet.c
3   *
4   * Created: 13.02.2012 16:56:18
5   *  Author: Niki
6   */
7
8  #ifndef DEBUG
9          #define DEBUG 1
10 #endif
11
12 #ifndef F_CPU
13         #define F_CPU 12000000UL
14 #endif
15
16 #define UART_BAUD_RATE 19200L
17 #define UART_BAUD_CALC(UART_BAUD_RATE,F_CPU)  ((F_CPU)/((UART_BAUD_RATE)*16L
       )-1)
18
19 #include "Atmega324a_Ethernet.h"
20
21 volatile char PacketReceived;
22
23 //Interrupt
24 //Receive Paket from the Ethernet
25 ISR(INT2_vect)
26 {
27         //Clear Interrupt Flag
28         BitFC(EIR,0x40);
29         PacketReceived=1;
30
31         #ifdef DEBUG
32                 uart_puts_1("Interrupt\n\r");
33         #endif
34 }
35
36 //Initisalisierung Controller
```

```
37 void Init(void)
38 {
39         //PortA als Ausgang
40         DDRA = 0xFF;
41
42         //PORTD Ausgänge, werden automatisch vom USART gesetzt
43         DDRD = 0xFF;
44
45         //PORTB alles Ausgänge SPI_Master_Init setzt es richtig
46         DDRB = 0xFF;
47
48
49         //JTAG deaktivieren
50         MCUCR |= (1<<JTD);
51         MCUCR |= (1<<JTD);
52
53         sei();  // Interruptbehandlung aktivieren
54
55         //Serielle Schnittstellen aktivieren
56         uart_ini_1();
57         uart_ini_0();
58
59         //SPI initialisieren
60         SPI_MasterInit();
61
62         //Chip Select auf 1 setzen
63         PORTB |= (1<<PINB3);
64
65         //Reset vom ENC28J60
66         DDRB |= (1<<PINB1);
67         PORTB |= (1<<PINB1);
68
69         //PB2 als Eingang für Interrupt
70         DDRB &= ~(1<<PINB2);
71
72         //Interrupts aktivieren INT2
73         EICRA = 0x20; //XX10 000
74         EIMSK = 0x04;
75
76 }
77
78 int main(void)
79 {
80     char transmitSPIValue;
81         char returnValue;
82         uint32_t CodeResponseWort;
83         unsigned short countParam;
84         uint16_t transID;
85         char *arrayParam;
86         char Buff[100];
87         uint32_t turn;
88
89         turn =0;
90
91         arrayParam=0;
92         transmitSPIValue=0;
93         Init();
94
95         //warten bis den ENC bereit ist
```

```
96          do
97          {
98          } while (CheckReady() != 0);
99
100         InitENC();
101
102         //1 Sekunde warten, sonst wird der Empfangsinterrupt schon
                ausgelöst
103         _delay_ms(1000);
104         PacketReceived=0;
105
106         #ifdef DEBUG
107                 uart_puts_1("Hello Ethernet V1.84\n\r");
108                 //ser_getc_1();
109                 uart_puts_1("Run\n\r");
110         #endif
111
112         while(1)
113     {
114                 if(turn ==( UINT16_MAX * 64))
115                 {
116                         #ifdef DEBUG
117                                 uart_puts_1("Beep\n\r");
118                         #endif
119                         turn=0;
120                 }
121                 else
122                 {
123                         turn++;
124                 }
125
126                 //TODO:: Please write your application code
127                 if(PacketReceived==1)
128                 {
129                         do
130                         {
131                                 #ifdef DEBUG
132                                         uart_puts_1("Ethernet Paket
                                            empfangen\n\r");
133                                 #endif
134
135                                 //Broadcast oder direkt adressiert
                                    herausfinden
136                                 if(CheckBroadcastDirect() == 0)
137                                 {
138                                         //Paket verwerfen
139                                         ThrowAway();
140                                 }
141                                 else
142                                 {
143                                         //UDP oder ARP
144                                         #ifdef DEBUG
145                                                 uart_puts_1("UDP oder ARP\n
                                                    \r");
146                                         #endif
147
148                                         returnValue = CheckUDPARP();
149
150                                         switch(returnValue)
```

```
151                                     {
152                                             case 0:
153                                                     //was anderes
154                                                     //Paket verwerfen
155                                                     ThrowAway();
156                                                     break;
157                                             case 1:
158                                                     //ARP
159                                                     #ifdef DEBUG
160                                                             uart_puts_1
                                                                ("ARP\n\
                                                                r");
161                                                     #endif
162                                                     SendARPBack();
163                                                     break;
164                                             case 2:
165                                                     //UDP
166                                                     #ifdef DEBUG
167                                                             uart_puts_1
                                                                ("UDP\n\
                                                                r");
168                                                     #endif
169                                                     //Broadcast oder
                                                            direkt
170                                                     if(
                                                            CheckBroadcastDirect
                                                            ()==1)
171                                                     {
172                                                             //Broadcast
173                                                             #ifdef
                                                                    DEBUG
174                                                                     uart_puts_1
                                                                        (
                                                                        "
                                                                        Broadcast
                                                                        \
                                                                        n
                                                                        \
                                                                        r
                                                                        "
                                                                        )
                                                                        ;

175                                                                     CheckUDPBroadcas
                                                                        ()
                                                                        ;

176                                                             #endif
177                                                     }
178                                                     else
179                                                     {
180                                                             //Direkt
181                                                             #ifdef
                                                                    DEBUG
182                                                                     uart_puts_1
                                                                        (
                                                                        "
                                                                        Direkt
                                                                        \
```

```
                                                                    n
                                                                    \
                                                                    r
                                                                    "
                                                                    )
                                                                    ;
183                                                                 ReadUDP
                                                                        ()
                                                                        ;
184                                                                     #endif
185                                                             }
186                                                             break;
187                                                     }
188                                             }
189                                             //Es können wieder Interrupts empfangen
                                                    werden
190                                             BitFC(EIR,0x40);
191                                             PacketReceived=0;
192                                     }while(PaketsReceived() > 0);
193                     }
194
195
196                     //Kontrollieren ob wir was senden müssen
197                     if(Getrbuffcnt_0()>7)
198                     {
199                             #ifdef DEBUG
200                                     uart_puts_1("Daten zum senden bereit\n\r");
201                             #endif
202
203                             arrayParam = ReadContainerUSART(&CodeResponseWort,
                                    &countParam, &transID);
204
205                             //UDP Paket generieren und versenden
206                             CreateUDPData(CodeResponseWort,countParam,transID,
                                    arrayParam);
207
208                             if(arrayParam != 0)
209                             {
210                                     free(arrayParam);
211                             }
212                     }
213
214     }
215 }
216
217 //ReadContainerUSART
218 //Funktion liest einen Container ein
219 //Param1: ausgelesener Code
220 //Param2: ausgelesene Anzahl der Parameter
221 //Param3: TransactionID
222 // return : Parameterfeld (free nicht vergessen)
223 char *ReadContainerUSART(uint32_t *CodeResponseWort, unsigned short *
        countParam, uint16_t *transID)
224 {
225         char buffer[8];
226         char * arrayParam;
227         unsigned short i;
```

```
228
229
230         arrayParam =0;
231         //USART Implementation ATmega
232         //Wenn 8 Zeichen im Buffer sind, lese sie aus
233         i = 0;
234
235         for(i=0;i<8;i++)
236         {
237                 buffer[i]= ser_getc_0();
238         }
239
240         //Buffer auswerten
241         *transID = (uint16_t) (buffer[1]<<8) + (uint16_t)buffer[0];
242
243         *countParam=(uint16_t)(buffer[7]<<8) + (uint16_t) buffer[6];
244
245
246         *CodeResponseWort = ((uint32_t) buffer[5] <<24) +((uint32_t) buffer
                [4] <<16) +((uint32_t)buffer[3] <<8) + (uint32_t) buffer[2];
247
248         //Parameter einlesen
249         if((*countParam)>0)
250         {
251
252                 #ifdef DEBUG
253                         uart_puts_1("Daten einlesen\n\r");
254                 #endif
255
256                 arrayParam = malloc(sizeof(char)*(*countParam));
257
258                 if (arrayParam==0)
259                 {
260                         //Kein PLatz mehr
261                         #ifdef DEBUG
262                                 uart_puts_1("Kein Platz USART\r\n");
263                         #endif
264                         return 0;
265                 }
266
267                 for(i=0;i<(*countParam);i++)
268                 {
269                         arrayParam[i]=ser_getc_0();
270                 }
271                 //Free nicht vergessen
272
273         }
274         return arrayParam;
275 }
```

H VNC Firmware

VNC_SPI.h

```
1  /*
2  ** VNC_SPI.h
3  **
4  ** C Header File
5  **
6  ** Part of solution SPI in project VNC_SPI
7  */
8
9  #ifndef _VNC_SPI_H_
10 #define _VNC_SPI_H_
11
12 #include "vos.h"
13 #include "Protokoll.h"
14 #include "USBHost.h"
15 #include "Host.h"
16 #include "PTP.h"
17
18
19 /* FTDI:SHF Header Files */
20 /* FTDI:EHF */
21
22 /* FTDI:SDC Driver Constants */
23 #define SIZEOF_FIRMWARE_TASK_MEMORY 0x1000
24
25 #define NUMBER_OF_DEVICES                    4
26
27 #define VOS_DEV_SPI_SLAVE                    0
28
29 #define VOS_DEV_USB_HOST_1                   1
30 #define VOS_DEV_USB_HOST_2                   2
31
32
33 /* FTDI:EDC */
34
35 /* FTDI:SXH Externs */
36 /* FTDI:EXH */
37
38 #endif /* _VNC_SPI_H_ */
```

VNC_SPI.c

```
1  /*
2  ** Filename: VNC_SPI.c
3  **
4  ** C Source File
5  **
6  ** Part of solution SPI in project VNC_SPI
7  **
8  ** Comments:
9  */
10 #include "VNC_SPI.h"
11
12 void firmware();
13 /* FTDI:STP Thread Prototypes */
14 vos_tcb_t *tcbFIRMWARE;
```

```
15
16 void main(void)
17 {
18         /* Main code to be added here */
19
20         // USB Host configuration context
21         usbhost_context_t usb_ctx;
22
23         /* FTDI:SKI Kernel Initialisation */
24         vos_init(10, VOS_TICK_INTERVAL, NUMBER_OF_DEVICES);
25         vos_set_clock_frequency(VOS_48MHZ_CLOCK_FREQUENCY);
26         vos_set_idle_thread_tcb_size(512);
27         /* FTDI:EKI */
28
29         /* FTDI:SDI Driver Initialisation */
30         /* FTDI:EDI */
31         // use a max of 4 USB devices
32         usb_ctx.if_count = 4;
33         usb_ctx.ep_count = 8;
34         usb_ctx.xfer_count = 2;
35         usb_ctx.iso_xfer_count = 2;
36         usbhost_init( VOS_DEV_USB_HOST_2,VOS_DEV_USB_HOST_1, &usb_ctx);
37
38
39         /* FTDI:SCT Thread Creation */
40         tcbFIRMWARE = vos_create_thread_ex(20, SIZEOF_FIRMWARE_TASK_MEMORY,
               firmware, "Application", 0);
41         /* FTDI:ECT */
42
43         spislave_init(VOS_DEV_SPI_SLAVE,NULL); //64 Byte Buffer
44
45         vos_start_scheduler();
46
47         while(1)
48         ;
49 }
50
51 /* Application Threads */
52
53 void firmware()
54 {
55
56         // test buffer
57         char buf[64];
58         char returnbuf[64];
59         unsigned short num_read;
60         unsigned short num_written;
61         common_ioctl_cb_t spi_iocb;
62         char returnvalue;
63         char x;
64         VOS_HANDLE hSPISlave;
65
66         uint32 CodeResponseWort;
67         unsigned short countParam;
68         char* arrayParam;
69
70         //USB Host 1
71         SetUSBHost1(NULL);
72         SetDeviceHandle1(NULL);
```

```
73          SetInterruptEP1(NULL);
74          SetBulkInEP1( NULL);
75          SetBulkEPOut1(NULL);
76
77          //USB Host 2
78          SetUSBHost2(NULL);
79          SetDeviceHandle2(NULL);
80          SetInterruptEP2(NULL);
81          SetBulkInEP2( NULL);
82          SetBulkEPOut2(NULL);
83
84          x = 0;
85          returnvalue = 0;
86          countParam = 0;
87          CodeResponseWort= 0;
88          arrayParam=0;
89          //Gerät öffnen
90          SetSPIHandle(vos_dev_open(VOS_DEV_SPI_SLAVE));
91
92          hSPISlave = GetSPIHandle();
93
94          SetUSBHost1(vos_dev_open(VOS_DEV_USB_HOST_1));
95          SetUSBHost2(vos_dev_open(VOS_DEV_USB_HOST_2));
96
97          SetPTPTransactionID(1);
98          do
99          {
100                 //Anzahl der eingelesenen Bytes herausfinden
101                 spi_iocb.ioctl_code = VOS_IOCTL_COMMON_GET_RX_QUEUE_STATUS;
102                 vos_dev_ioctl(hSPISlave, &spi_iocb);
103                 num_written = spi_iocb.get.queue_stat;
104
105                 memset(buf,0,64*sizeof(char));
106                 memset(returnbuf,14,64*sizeof(char));
107
108                 // limit to 64 bytes per transaction
109                 if (num_written > 64)
110                         num_written = 64;
111
112                 //Mindestens 8 Byte müssen übertragen sein (2 Bytes
                        TransactionID, 4 Bytes Command, 2 Bytes Länge)
113                 if (num_written > 7)
114                 {
115
116                         //Read Container aufrufen
117                         arrayParam = ReadContainer(&CodeResponseWort,&
                            countParam,arrayParam);
118
119                         LogicUnit(CodeResponseWort,      countParam,
                            arrayParam);
120
121                         //free von den Parametern machen
122                         if( arrayParam != 0)
123                         {
124                                 free(arrayParam);
125                         }
126                 }
127                 else
128                 {
```

```
129                             //Nop
130                             buf[0] = 0;
131                     }
132             }
133             while(1);
134     }
```

PTP.h

```
1   /*
2           PTP Implementation
3   */
4
5   #include "VNC_SPI.h"
6   #include "Host.h"
7
8   //USB PTP Container
9   #define USB_PTP_CONTAINER_UNDEFINED       0x0000
10  #define USB_PTP_CONTAINER_COMMANDBLOCK    0x0001
11  #define USB_PTP_CONTAINER_DATABLOCK               0x0002
12  #define USB_PTP_CONTAINER_RESPONSEBLOCK 0x0003
13  #define USB_PTP_CONTAINER_EVENTBLOCK      0x0004
14
15  //PTP OPCodes
16  #define PTP_OPEN_SESSION                                    0x1002
17  #define PTP_CLOSE_SESSION                                   0x1003
18  #define PTP_OPCODE_SESSIONOPEN_1                  0x9114
19  #define PTP_OPCODE_SESSIONOPEN_2                  0x9115
20  #define PTP_OPCODE_SHOOT                                    0x910F
21  #define PTP_OPCODE_TV_Value                                 0x9110
22  #define PTP_OPCODE_AV_Value                                 0x9110
23  #define PTP_OPCODE_BELICHTUNG                     0x9110
24  #define PTP_OPCODE_LIVEVIEW                                 0x9110
25  #define PTP_OPCODE_FOCUS                                    0x9155
26
27  //SendDataPTP
28  //Funktion generiert die zu sendeten Daten und verschickt diese
29  //return länge des gesendeten wenn alles ok, sonst einen Wert kleiner der
        zu sendenen Bytes
30  char SendDataPTP(VOS_HANDLE hUsbHost, usbhost_ep_handle ep,unsigned short
        OperationCode, unsigned int PTPTransactionID, char CountParamter,
        unsigned int Parameter1, unsigned int Parameter2, unsigned int
        Parameter3, unsigned short mode);
31
32  //ReadDataPTP
33  //Funktion empfängt die Daten
34  //return länge wenn erfolgreich, sonst 0,1,2,3,4
35  char ReadDataPTP(VOS_HANDLE hUsbHost, usbhost_ep_handle ep, char length);
36
37  //GetPTPTransactionID
38  //Funktion liefert die verwendete TransactionID zurück
39  //Wird der einfachheit für beide USB Hosts verwendet
40  unsigned short GetPTPTransactionID(void);
41
42  //SetPTPTransactionID
43  //Funktion setzt die verwendete TransactionID zurück
44  //Wird der einfachheit für beide USB Hosts verwendet
45  void SetPTPTransactionID(unsigned short);
46
```

```
47 //IncreasePTPTransactionID
48 //Funktion erhöht die verwendete TransactionID zurück
49 //Wird der einfachheit für beide USB Hosts verwendet
50 void IncreasePTPTransactionID(void);
```

PTP.c

```
1  #include "PTP.h"
2
3  unsigned short PTPTransactionID;
4
5  //SendDataPTP
6  //Funktion generiert die zu sendeten Daten und verschickt diese
7  //return länge des gesendeten wenn alles ok, sonst einen Wert kleiner der
       zu sendenen Bytes
8  char SendDataPTP(VOS_HANDLE hUsbHost, usbhost_ep_handle ep,unsigned short
       OperationCode, unsigned int PTPTransactionID, char CountParamter,
       unsigned int Parameter1, unsigned int Parameter2, unsigned int
       Parameter3, unsigned short mode)
9  {
10         unsigned char i;
11         char lengthContainer;
12         unsigned char transmit[32];
13         usbhost_xfer_t xfer;
14         vos_semaphore_t s;
15         unsigned char status;
16
17         status = 0;
18
19         memset(transmit,0,32);
20
21         //Länge des Containers
22         lengthContainer = 12 + CountParamter * 4;
23
24         //Senden der einzelnen Container Elemente in little Endian Format
               vorbereiten
25         for (i=0;i<24; i++)
26         {
27                 switch(i)
28                 {
29                         //ContainerLength
30                         case 0:
31                         case 1:
32                         case 2:
33                         case 3:
34                                 transmit[i] = (lengthContainer>>(i*8)) & 0
                                       xff;
35                                 break;
36                         //ContainerType
37                         case 4:
38                         case 5:
39                                 transmit[i] = (mode>>((i-4)*8)) & 0xff;
40                                 break;
41                         //Code
42                         case 6:
43                         case 7:
44                                 transmit[i] = (OperationCode>>((i-6)*8)) &
                                       0xff;
45                                 break;
```

```
46                          //TransactionID
47                          case 8:
48                          case 9:
49                          case 10:
50                          case 11:
51                                  transmit[i] = (PTPTransactionID>>((i-8)*8))
                                        & 0xff;
52                                  break;
53                          //Parameter1
54                          case 12:
55                          case 13:
56                          case 14:
57                          case 15:
58                                  transmit[i] = (Parameter1>>((i-12)*8)) & 0
                                        xff;
59                                  break;
60                          //Parameter2
61                          case 16:
62                          case 17:
63                          case 18:
64                          case 19:
65                                  transmit[i] = (Parameter2>>((i-16)*8)) & 0
                                        xff;
66                                  break;
67                          //Parameter3
68                          case 20:
69                          case 21:
70                          case 22:
71                          case 23:
72                                  transmit[i] = (Parameter3>>((i-20)*8)) & 0
                                        xff;
73                                  break;
74                  }
75          }
76
77          memset(&xfer, 0, sizeof(usbhost_xfer_t));
78
79          vos_init_semaphore(&s, 0);
80          xfer.buf = transmit;
81          xfer.len = lengthContainer;
82          xfer.ep = ep;
83          xfer.s = &s;
84          xfer.cond_code = USBHOST_CC_NOTACCESSED;
85          xfer.flags = USBHOST_XFER_FLAG_START_BULK_ENDPOINT_LIST;
86
87          status = vos_dev_write(hUsbHost, (unsigned char *)&xfer, sizeof(
                usbhost_xfer_t), NULL);
88          if (status != USBHOST_OK)
89          {
90                  return 0;
91          }
92          status = xfer.cond_code;
93
94          if (status != USBHOST_CC_NOERROR)
95          {
96                  if (status == USBHOST_CC_STALL)
97                  {
98                          return 1;
99                  }
```

```
100                     else
101                     {
102                             return 2;
103                     }
104             }
105             return xfer.len;
106
107     }
108
109     //ReadDataPTP
110     //Funktion empfängt die Daten
111     //return länge wenn erfolgreich, sonst 0,1,2,3,4
112     char ReadDataPTP(VOS_HANDLE hUsbHost, usbhost_ep_handle ep, char length)
113     {
114             usbhost_ioctl_cb_ep_info_t epInfo;
115             usbhost_ioctl_cb_t host_ioctl_cb;
116             usbhost_xfer_t xfer;
117
118             //Max 100 Zeichen empfange
119             unsigned char buf[100];
120             vos_semaphore_t s;
121             unsigned char status;
122
123             unsigned char bulkInEp = USBHOST_XFER_FLAG_START_BULK_ENDPOINT_LIST
                    ;
124
125             status = 0;
126
127             host_ioctl_cb.ioctl_code =
                    VOS_IOCTL_USBHOST_DEVICE_GET_ENDPOINT_INFO;
128             host_ioctl_cb.handle.ep = ep;
129             host_ioctl_cb.get = &epInfo;
130
131             vos_dev_ioctl(hUsbHost, &host_ioctl_cb);
132
133             memset(buf, 0, 100*sizeof(unsigned char));
134             memset(&xfer, 0, sizeof(usbhost_xfer_t));
135             vos_init_semaphore(&s, 0);
136
137             xfer.buf = buf;
138             xfer.len = length;
139             xfer.ep = ep;
140             xfer.s = &s;
141             xfer.cond_code = USBHOST_CC_NOTACCESSED;
142             xfer.flags = USBHOST_XFER_FLAG_START_BULK_ENDPOINT_LIST ;
143
144             status = vos_dev_read(hUsbHost, (unsigned char *) &xfer, sizeof(
                    usbhost_xfer_t), NULL);
145             if (status != USBHOST_OK)
146             {
147                     return 0;
148             }
149
150             status = xfer.cond_code;
151             if (status != USBHOST_CC_NOERROR)
152             {
153                     if (status == USBHOST_CC_STALL)
154                     {
155                             return 1;
```

```
156                     }
157                     else
158                     {
159                             return 2;
160                     }
161             }
162
163             if(xfer.len!=length)
164             {
165                     return 3;
166             }
167             else
168             {
169                     if(buf[6] != 1 || buf[7] != 0x20)
170                     {
171                             return 4;
172                     }
173                     else
174                     {
175                             return length;
176                     }
177             }
178
179             return xfer.len;
180     }
181
182     //GetPTPTransactionID
183     //Funktion liefert die verwendete TransactionID zurück
184     //Wird der einfachheit für beide USB Hosts verwendet
185     unsigned short GetPTPTransactionID(void)
186     {
187             return PTPTransactionID;
188     }
189
190     //SetPTPTransactionID
191     //Funktion setzt die verwendete TransactionID zurück
192     //Wird der einfachheit für beide USB Hosts verwendet
193     void SetPTPTransactionID(unsigned short value)
194     {
195             PTPTransactionID = value;
196     }
197
198     //IncreasePTPTransactionID
199     //Funktion erhöht die verwendete TransactionID zurück
200     //Wird der einfachheit für beide USB Hosts verwendet
201     void IncreasePTPTransactionID(void)
202     {
203             PTPTransactionID++;
204     }
```

Protokoll.h

```
1   #ifndef PROTOKOLL_H_
2   #define PROTOKOLL_H_
3
4   #include <stdio.h>
5   #include <stdlib.h>
6   #include "SPISlave.h"
7   #include "string.h"
```

```
 8 #include "Commands.h"
 9 #include "VNC_SPI.h"
10 #include "Host.h"
11
12 //GetSPIHandle
13 //Funktion liefert die gesendete SPIHandle zurück
14 //@Return: TransactionID
15 VOS_HANDLE GetSPIHandle();
16
17 //SetSPIHandle
18 //Funktion setzt die gesendete TransactionID zurück
19 //@Param1: hHandle
20 void SetSPIHandle(VOS_HANDLE hHandle);
21
22
23 //GetTransactionID
24 //Funktion liefert die gesendete TransactionID zurück
25 //@Return: TransactionID
26 unsigned short GetTransactionID();
27
28 //SetTransactionID
29 //Funktion setzt die gesendete TransactionID zurück
30 //@Param1: TransactionID
31 void SetTransactionID(unsigned short transID);
32
33 //GetTransactionIDRead
34 //Funktion liefert die gesendete TransactionIDRead zurück
35 //@Return: TransactionID
36 unsigned short GetTransactionIDRead();
37
38 //SetTransactionIDRead
39 //Funktion setzt die gesendete TransactionIDRead zurück
40 //@Param1: TransactionID
41 void SetTransactionIDRead(unsigned short transID);
42
43 //CreateContainer
44 //Funktion die den zu übertragenden Container generiert
45 //Es wird keine dynamische Speicherverwaltung verwendet,
46 //damit auch größere Daten transferiert werden können
47 // @ Param1: Zu Übertragender Code
48 // @ Param2: Anzahl der Parameter
49 // @ Param3: Parameter Array
50 // @ Param4: Ab wann Parameter übertragen werden
51 // @ Param5: Bis wohin Parameter übertragen werden
52 // @ Param6: Container mit maximaler Größe
53 // @ Param7: TransactionID die zu übertragen ist
54 void CreateContainer(uint32 CodeWort, unsigned short countParam, char *
       arrayParam, unsigned short start, unsigned short stop, char * Container,
        unsigned short TransID);
55
56 //SendContainer
57 //Funktion sendet den Container
58 //Abhängig von der Transportebene
59 //Container wird hier definiert
60 // @ Param1: Zu Übertragender Code
61 // @ Param2: Anzahl der Parameter
62 // @ Param3: Parameter Array
63 // @ Param4: zusendente TransaktionsID
64 // @ Return Value: 0 success, sonst <>  0
```

```
65 char SendContainer(uint32 CodeWort, unsigned short countParam, char *
       arrayParam, unsigned short TransID);
66
67
68 //ReadContainer
69 //Funktion liest einen Container ein
70 //@Param1: ausgelesener Code
71 //@Param2: ausgelesene Anzahl der Parameter
72 //@Param3: Parameterfeld (free nicht vergessen)
73 //Return ParameterArray
74 char* ReadContainer(uint32 *CodeWort, unsigned short *countParam, char *
       arrayParam);
75
76 //LogicUnit
77 //Funktion managed die Kommunikation
78 // @Param1: Comment oder Response Code
79 // @Param2: Anzahl der Parameter
80 // @Param3: Parameter Array
81 void LogicUnit(uint32 CommandResponseCode, unsigned short countParam, char
       * arrayParam);
82
83 //CheckCommand
84 //Funktion überprüft den Command Code ob er unterstützt wird und ob die
       Parameter stimmen
85 // @Param1: Command Code
86 // @Param2: Anzahl der Parameter
87 //Return Command.h
88 unsigned short CheckCommand(uint32 CommandResponseCode, unsigned short
       countParam);
89
90 #endif /* PROTOKOLL_H_ */
```

Protokoll.c

```
1  /*
2   * Protokoll.c
3   *
4   * Created: 01.10.2011 16:49:36
5   *  Author: Niki
6   */
7
8  #include "Protokoll.h"
9
10 unsigned short TransactionID; //ID für die Transaktion Senden
11 unsigned short TransactionIDRead; //ID für TransaktionID lesen
12
13 VOS_HANDLE hSPISlave;
14
15  //GetSPIHandle
16 //Funktion liefert die gesendete SPIHandle zurück
17 //@Return: TransactionID
18 VOS_HANDLE GetSPIHandle()
19 {
20         return    hSPISlave;
21 }
22
23 //SetSPIHandle
24 //Funktion setzt die gesendete TransactionID zurück
25 //@Param1: hHandle
```

```
26 void SetSPIHandle(VOS_HANDLE hHandle)
27 {
28         hSPISlave = hHandle;
29 }
30
31
32  //GetTransactionID
33 //Funktion liefert die gesendete TransactionID zurück
34 //@Return: TransactionID
35 unsigned short GetTransactionID()
36 {
37         return    TransactionID;
38 }
39
40 //SetTransactionID
41 //Funktion setzt die gesendete TransactionID zurück
42 //@Param1: TransactionID
43 void SetTransactionID(unsigned short transID)
44 {
45         TransactionID = transID;
46 }
47
48 //GetTransactionIDRead
49 //Funktion liefert die gesendete TransactionIDRead zurück
50 //@Return: TransactionID
51 unsigned short GetTransactionIDRead()
52 {
53         return    TransactionIDRead;
54 }
55
56
57 //SetTransactionIDRead
58 //Funktion setzt die gesendete TransactionIDRead zurück
59 //@Param1: TransactionID
60 void SetTransactionIDRead(unsigned short transID)
61 {
62         TransactionIDRead = transID;
63 }
64
65 //CreateContainer
66 //Funktion die den zu übertragenden Container generiert
67 //Es wird keine dynamische Speicherverwaltung verwendet,
68 //damit auch größere Daten transferiert werden können
69 // @ Param1: Zu Übertragender Code
70 // @ Param2: Anzahl der Parameter
71 // @ Param3: Parameter Array
72 // @ Param4: Ab wann Parameter übertragen werden (inklusive) (0 wenn alles
      in einem Container übertragen werden kann)
73 // @ Param5: Bis wohin Parameter übertragen werden (inklusive)
74 // @ Param6: Container mit maximaler Größe
75 // @ Param7: TransactionID die zu übertragen ist
76 void CreateContainer(uint32 CodeWort, unsigned short countParam, char *
     arrayParam, unsigned short start, unsigned short stop, char * Container,
      unsigned short TransID)
77 {
78         uint32    i;
79         uint32    lengthContainer; //Länge des Containers in Byte
80
81         if(start == 0) //Keine Teilung des Containers notwendig
```

```
82          {
83                  //Länge des Containers bestimmen
84                  lengthContainer = 2 /*TransactionID*/ + 4 /*Code*/ + 2 /*
                        AnzahlParameter*/ + countParam;
85
86                  for (i=0;i<lengthContainer;i++)
87                  {
88                          switch(i)
89                          {
90                                  case 0:
91                                  case 1:
92                                          //TransaktionID
93                                          Container[i] = (char) (TransID>> i
                                                *8) & 0xFF;
94                                          break;
95                                  case 2:
96                                  case 3:
97                                  case 4:
98                                  case 5:
99                                          //Code
100                                         Container[i] = (char) (CodeWort>> (
                                                i-2)*8) & 0xFF;
101                                         break;
102                                 case 6:
103                                 case 7:
104                                         //Anzahl Parameter
105                                         Container[i] = (char) (countParam>>
                                                (i-6)*8) & 0xFF;
106                                         break;
107                                 default:
108                                         //Parameter
109                                         Container[i] = *(arrayParam+(i-8));
110                                         break;
111                         }
112                 }
113         }
114         else //Nur die Parameter übertragen
115         {
116                 //Länge des Containers bestimmen
117                 lengthContainer = stop - start + 1;
118                 for (i=0;i<stop-start +1 ;i++)
119                 {
120                         switch(i)
121                         {
122                                 default:
123                                         //Parameter
124                                         Container[i] = *(arrayParam+start+i
                                                );
125                                         break;
126                         }
127                 }
128         }
129 }
130
131 //SendContainer
132 //Funktion sendet den Container
133 //Abhängig von der Transportebene
134 //Container wird hier definiert
135 // @ Param1: Zu Übertragender Code
```

```
136 // @ Param2: Anzahl der Parameter
137 // @ Param3: Parameter Array
138 // @ Param4: zusendente TransaktionsID
139 // @ Return Value: 0 success, sonst <> 0
140 char SendContainer(uint32 CodeWort, unsigned short countParam, char *
        arrayParam, unsigned short TransID)
141 {
142         unsigned short ParameterLeft;
143         char i;
144         unsigned short countTransmitParameter;
145         char returnvalue;
146
147         //SPI Buffer
148         char buffer[64];
149         unsigned short start;
150         unsigned short stop;
151         unsigned short j;
152         unsigned short num_written;
153         unsigned short num_read;
154
155         start = 0;
156         stop = 0;
157         i = 0;
158
159         for(i=0;i<64;i++)
160         {
161                 buffer[i] = 0;
162         }
163
164         //maximale Anzahl der Parameter die übertragen sind
165         //64-8 = 56
166         //Wenn mehr sind, müssen mehrere Container gesendet werden
167
168         ParameterLeft = countParam;
169
170         if(ParameterLeft>56)
171         {
172                 countTransmitParameter = 56;
173         }
174         else
175         {
176                 countTransmitParameter = ParameterLeft;
177         }
178
179         do
180         {
181                 //Container generieren
182                 CreateContainer(CodeWort,countTransmitParameter,arrayParam,
                        start,stop,buffer,TransID);
183
184                 //Abhängig von der Transportebene Start
185                 //
                        ------------------------------------------------------------------------

186
187                 //SPI von VNC
188
189                 if (countTransmitParameter < 56 && start == 0)
190                 {
```

```
191                     //erster durchlauf
192                     num_read = 8 + countTransmitParameter;
193             }
194             else
195             {
196                     //Mehr Paramter werden in der aktuellen Version
                            nicht unterstützt
197                     return 3;
198             }
199
200
201             returnvalue = vos_dev_write(hSPISlave, (char *) buffer,
                    num_read, &num_written);
202
203             if (returnvalue != SPISLAVE_OK)
204             {
205                     return 1;
206             }
207             else
208             {
209                     returnvalue = SPISLAVE_OK;
210             }
211
212             //Bluetooth
213
214             //Ethernet
215
216             //
                    ------------------------------------------------------------------------

217             //Abhängig von der Transportebene Ende
218
219             if(start == 0)
220             {
221                     //Erster Durchlauf
222                     if(ParameterLeft>56)
223                     {
224                             start = 56;
225                             ParameterLeft-=56;
226                     }
227                     else
228                     {
229                             ParameterLeft=0;
230                     }
231             }
232             else
233             {
234                     //Xter Durchlauf
235                     if(ParameterLeft>64)
236                     {
237                             start = start + 64;
238                             ParameterLeft-=64;
239                     }
240                     else
241                     {
242                             start = start + 64;
243                             ParameterLeft=0;
244                     }
245             }
```

```
246
247                 //Stop setzen
248                 if(ParameterLeft>64)
249                 {
250                         stop = start+64;
251                         countTransmitParameter = 64;
252                 }
253                 else
254                 {
255                         stop = start + ParameterLeft;
256                         countTransmitParameter = ParameterLeft;
257                 }
258         }while(ParameterLeft>0);
259
260
261         return 0;
262 }
263
264
265 //ReadContainer
266 //Funktion liest einen Container ein
267 //array Param wird als Parameter nicht bentötigt,
268 //IDE kann nicht mit der Übergabe umgehen
269 //@Param1: ausgelesener Code
270 //@Param2: ausgelesene Anzahl der Parameter
271 //@Param3: Parameterfeld (free nicht vergessen)
272 //return ParamArray
273 char* ReadContainer(uint32 *CodeResponseWort, unsigned short *countParam,
      char* arrayParam)
274 {
275         char buffer[8]; //Empfangsbuffer, abhängig von der Transportebene
276         char *array;
277         unsigned short num_read;
278         unsigned short num_written;
279         unsigned short transid;
280         unsigned short countP;
281         unsigned int code;
282         common_ioctl_cb_t spi_iocb;
283
284         //SPI Implementation VNC
285         //Die ersten 8 Zeichen einlesen
286         vos_dev_read(hSPISlave, (char *) buffer, 8, &num_read);
287
288
289         //Buffer auswerten
290         //Der Umweg über die Zusatzvariablen musste gewählt werden
291         //weil das sonst nicht funktioniert hat (IDE Problem)
292         transid = buffer[1]<<8 | buffer[0];
293         countP =buffer[7]<<8 | buffer[6];
294         code = buffer[5] <<24|buffer[4] <<16|buffer[3] <<8 | buffer[2];
295         *countParam = countP;
296         *CodeResponseWort = code;
297         SetTransactionIDRead(transid);
298         array = 0;
299
300
301         //Mehr als 56 Parameter werden hier nicht unterstützt
302         if(countP>56)
303         {
```

```
304                     return 0;
305             }
306             else
307             {
308                     //Parameter einlesen
309                     if(countP>0)
310                     {
311                             array  = malloc(sizeof(char)*(*countParam));
312
313                             if (array==0)
314                             {
315                                     //Kein Platz mehr
316                                     return array;
317                             }
318                             else
319                             {
320                                     //SPI Implementation VNC
321
322                                     //warten bis alle Parameter übertragen sind
323                                     do
324                                     {
325                                             spi_iocb.ioctl_code =
                                                VOS_IOCTL_COMMON_GET_RX_QUEUE_STATUS
                                                ;
326                                             vos_dev_ioctl(hSPISlave, &spi_iocb)
                                                ;
327                                             num_written = spi_iocb.get.
                                                queue_stat;
328                                     }while(num_written < countP);
329
330                                     //Die Parameter einlesen
331                                     vos_dev_read(hSPISlave, (char *) array, (*
                                            countParam), &num_read);
332                                     return array;
333                             }
334                     }
335             }
336
337             return 0;
338     }
339
340     //LogicUnit
341     //Funktion managed die Kommunikation
342     // @Param1: Comment oder Response Code
343     // @Param2: Anzahl der Parameter
344     // @Param3: Parameter Array
345     void LogicUnit(uint32 CommandResponseCode, unsigned short countParam, char
            * arrayParam)
346     {
347             char buffer[10];
348             char check;
349             char i;
350             char tmp;
351             unsigned short returnValue;
352             VOS_HANDLE hUsbHost;
353             usbhost_device_handle *ifDev;
354             usbhost_ep_handle hEP;
355             usbhost_ep_handle hEPIn;
356             usbhost_ep_handle hEPOut;
```

```
357
358         hEP = NULL;
359         hEPIn = NULL;
360         hEPOut = NULL;
361
362         //Container auswerten
363         returnValue = CheckCommand(CommandResponseCode,countParam);
364
365         ifDev = NULL;
366
367         if(returnValue != RESPONSE_OK)
368         {
369                 //Fehler
370                 //ErrorMessage erzeugen
371                 if(arrayParam != 0)
372                 {
373                         free(arrayParam);
374                 }
375                 else
376                 {
377                         buffer[0] = 0;
378                 }
379                 CommandResponseCode = 0xFF000000 | returnValue;
380
381                 SendContainer(CommandResponseCode,0,buffer,
                        GetTransactionIDRead());
382         }
383         else
384         {
385                 //Command auswerten
386                 check = (char) (CommandResponseCode & 0xFF);
387                 switch(check)
388                 {
389                         case COMMAND_PTP_SESSION_OPEN:
390                                 if(UpdateConnection(CommandResponseCode)>1)
391                                 {
392                                         //Fehler
393                                         CommandResponseCode = 0xFF000000 |
                                            RESPONSE_ERROR;
394                                         SendContainer(CommandResponseCode
                                            ,0,buffer,GetTransactionIDRead()
                                            );
395                                 }
396                                 else
397                                 {
398                                         //Passt
399
400                                         //Für welchen USB Host ist es
401                                         check = (char) (CommandResponseCode
                                            >>8 & 0xFF);
402                                         if(check == 0x00)
403                                         {
404                                                 hUsbHost = GetUSBHost1();
405                                                 hEPIn = GetBulkInEP1();
406                                                 hEPOut = GetBulkOutEP1();
407                                         }
408                                         else
409                                         {
410                                                 hUsbHost = GetUSBHost2();
```

```
411                     hEPIn = GetBulkInEP2();
412                     hEPOut = GetBulkOutEP2();
413                 }
414                 //PTP Session starten
415                 if(SendDataPTP(hUsbHost,hEPOut,
                        PTP_OPEN_SESSION,
                        GetPTPTransactionID(), 1,1,0,0,
                        USB_PTP_CONTAINER_COMMANDBLOCK)
                        != 0x010)
416                 {
417                     //Fehler
418                     CommandResponseCode = 0
                            xFF000000 |
                            RESPONSE_PTP_ERROR;
419                     SendContainer(
                            CommandResponseCode,0,
                            buffer,
                            GetTransactionIDRead());
420                     break;
421                 }
422                 else
423                 {
424                     //Daten Antwort empfangen
425                     if(ReadDataPTP(hUsbHost,
                            hEPIn, 12) != 12)
426                     {
427                         //Fehler
428                         CommandResponseCode
                                = 0xFF000000 |
                                RESPONSE_PTP_ERROR
                                ;
429                         SendContainer(
                                CommandResponseCode
                                ,0,buffer,
                                GetTransactionIDRead
                                ());
430                         break;
431                     }
432                 }
433
434                 //Befehl kann gesendet werden
435                 if(SendDataPTP(hUsbHost,hEPOut,
                        PTP_OPCODE_SESSIONOPEN_1,
                        GetPTPTransactionID(), 1,1,0,0,
                        USB_PTP_CONTAINER_COMMANDBLOCK)
                        != 0x010)
436                 {
437                     //Fehler
438                     CommandResponseCode = 0
                            xFF000000 |
                            RESPONSE_PTP_ERROR;
439                     SendContainer(
                            CommandResponseCode,0,
                            buffer,
                            GetTransactionIDRead());
440                     break;
441                 }
442                 else
443                 {
```

```
444                     //Daten Antwort empfangen
445                     if(ReadDataPTP(hUsbHost,
                            hEPIn, 12) != 12)
446                     {
447                             //Fehler
448                             CommandResponseCode
                                    = 0xFF000000 |
                                RESPONSE_PTP_ERROR
                                ;
449                             SendContainer(
                                CommandResponseCode
                                ,0,buffer,
                                GetTransactionIDRead
                                ());
450                             break;
451                     }
452             }
453             IncreasePTPTransactionID();
454
455             //Befehl kann gesendet werden
456             if(SendDataPTP(hUsbHost,hEPOut,
                    PTP_OPCODE_SESSIONOPEN_2,
                    GetPTPTransactionID(), 1,1,0,0,
                    USB_PTP_CONTAINER_COMMANDBLOCK)
                    != 0x010)
457             {
458                     //Fehler
459                     CommandResponseCode = 0
                            xFF000000 |
                            RESPONSE_PTP_ERROR;
460                     SendContainer(
                            CommandResponseCode,0,
                            buffer,
                            GetTransactionIDRead());
461                     break;
462             }
463             else
464             {
465                     //Daten Antwort empfangen
466                     if(ReadDataPTP(hUsbHost,
                            hEPIn, 12) != 12)
467                     {
468                             //Fehler
469                             CommandResponseCode
                                    = 0xFF000000 |
                                RESPONSE_PTP_ERROR
                                ;
470                             SendContainer(
                                CommandResponseCode
                                ,0,buffer,
                                GetTransactionIDRead
                                ());
471                             break;
472                     }
473             }
474             IncreasePTPTransactionID();
475
476             //Response mit OK zurück schicken
```

```
477                                     CommandResponseCode = 0xFF000000 |
                                            RESPONSE_OK   ;
478                                     SendContainer(CommandResponseCode
                                            ,0,buffer,GetTransactionIDRead()
                                            );
479                             }
480                             break;
481                     case COMMAND_PTP_SESSION_CLOSE:
482                             if(UpdateConnection(CommandResponseCode)>1)
483                             {
484                                     //Fehler
485                                     CommandResponseCode = 0xFF000000 |
                                            RESPONSE_ERROR;
486                                     SendContainer(CommandResponseCode
                                            ,0,buffer,GetTransactionIDRead()
                                            );
487                             }
488                             else
489                             {
490                                     //Passt
491
492                                     //Für welchen USB Host ist es
493                                     check = (char) (CommandResponseCode
                                            >>8 & 0xFF);
494                                     if(check == 0x00)
495                                     {
496                                             hUsbHost = GetUSBHost1();
497                                             hEPIn = GetBulkInEP1();
498                                             hEPOut = GetBulkOutEP1();
499                                     }
500                                     else
501                                     {
502                                             hUsbHost = GetUSBHost2();
503                                             hEPIn = GetBulkInEP2();
504                                             hEPOut = GetBulkOutEP2();
505                                     }
506
507                                     //Befehl kann gesendet werden
508                                     if(SendDataPTP(hUsbHost,hEPOut,
                                            PTP_OPCODE_SESSIONOPEN_2,
                                            GetPTPTransactionID(), 1,0,0,0,
                                            USB_PTP_CONTAINER_COMMANDBLOCK)
                                            != 0x010)
509                                     {
510                                             //Fehler
511                                             CommandResponseCode = 0
                                                    xFF000000 |
                                                    RESPONSE_PTP_ERROR;
512                                             SendContainer(
                                                    CommandResponseCode,0,
                                                    buffer,
                                                    GetTransactionIDRead());
513                                             break;
514                                     }
515                                     else
516                                     {
517                                             //Daten Antwort empfangen
518                                             if(ReadDataPTP(hUsbHost,
                                                    hEPIn, 12) != 12)
```

```
519                     {
520                         //Fehler
521                         CommandResponseCode = 0xFF000000 | RESPONSE_PTP_ERROR;
522                         SendContainer(CommandResponseCode,0,buffer,GetTransactionIDRead());
523                         break;
524                     }
525                 }
526                 IncreasePTPTransactionID();
527
528                 //Befehl kann gesendet werden
529                 if(SendDataPTP(hUsbHost,hEPOut,PTP_OPCODE_SESSIONOPEN_1,GetPTPTransactionID(),1,0,0,0,USB_PTP_CONTAINER_COMMANDBLOCK) != 0x010)
530                 {
531                     //Fehler
532                     CommandResponseCode = 0xFF000000 | RESPONSE_PTP_ERROR;
533                     SendContainer(CommandResponseCode,0,buffer,GetTransactionIDRead());
534                     break;
535                 }
536                 else
537                 {
538                     //Daten Antwort empfangen
539                     if(ReadDataPTP(hUsbHost,hEPIn,12) != 12)
540                     {
541                         //Fehler
542                         CommandResponseCode = 0xFF000000 | RESPONSE_PTP_ERROR;
543                         SendContainer(CommandResponseCode,0,buffer,GetTransactionIDRead());
544                         break;
545                     }
546                 }
547                 IncreasePTPTransactionID();
548
549                 //PTP Session starten
550                 if(SendDataPTP(hUsbHost,hEPOut,PTP_CLOSE_SESSION,GetPTPTransactionID(),1,1,0,0,USB_PTP_CONTAINER_COMMANDBLOCK)
```

```
                                        != 0x010)
551                                     {
552                                             //Fehler
553                                             CommandResponseCode = 0
                                                    xFF000000 |
                                                    RESPONSE_PTP_ERROR;
554                                             SendContainer(
                                                    CommandResponseCode,0,
                                                    buffer,
                                                    GetTransactionIDRead());
555                                             break;
556                                     }
557                                     else
558                                     {
559                                             //Daten Antwort empfangen
560                                             if(ReadDataPTP(hUsbHost,
                                                    hEPIn, 12) != 12)
561                                             {
562                                                     //Fehler
563                                                     CommandResponseCode
                                                            = 0xFF000000 |
                                                            RESPONSE_PTP_ERROR
                                                            ;
564                                                     SendContainer(
                                                            CommandResponseCode
                                                            ,0,buffer,
                                                            GetTransactionIDRead
                                                            ());
565                                                     break;
566                                             }
567                                     }
568
569                                     //Response mit OK zurück schicken
570                                     CommandResponseCode = 0xFF000000 |
                                            RESPONSE_OK   ;
571                                     SendContainer(CommandResponseCode
                                            ,0,buffer,GetTransactionIDRead()
                                            );
572                             }
573                             break;
574                     case COMMAND_PTP_SHOOT:
575                             if(UpdateConnection(CommandResponseCode)>1)
576                             {
577                                     //Fehler
578                                     CommandResponseCode = 0xFF000000 |
                                            RESPONSE_ERROR;
579                                     SendContainer(CommandResponseCode
                                            ,0,buffer,GetTransactionIDRead()
                                            );
580                             }
581                             else
582                             {
583                                     //Passt
584
585                                     //Für welchen USB Host ist es
586                                     check = (char) (CommandResponseCode
                                            >>8 & 0xFF);
587                                     if(check == 0x00)
588                                     {
```

```
589                                     hUsbHost = GetUSBHost1();
590                                     hEPIn = GetBulkInEP1();
591                                     hEPOut = GetBulkOutEP1();
592                             }
593                             else
594                             {
595                                     hUsbHost = GetUSBHost2();
596                                     hEPIn = GetBulkInEP2();
597                                     hEPOut = GetBulkOutEP2();
598                             }
599
600                             //Befehl kann gesendet werden
601                             if(SendDataPTP(hUsbHost,hEPOut,
                                    PTP_OPCODE_SHOOT,
                                    GetPTPTransactionID(), 0,0,0,0,
                                    USB_PTP_CONTAINER_COMMANDBLOCK)
                                    != 0x00C)
602                             {
603                                     //Fehler
604                                     CommandResponseCode = 0
                                            xFF000000 |
                                            RESPONSE_PTP_ERROR;
605                                     SendContainer(
                                            CommandResponseCode,0,
                                            buffer,
                                            GetTransactionIDRead());
606                                     break;
607                             }
608                             else
609                             {
610                                     //Daten Antwort empfangen
611                                     if(ReadDataPTP(hUsbHost,
                                            hEPIn, 0x10) != 0x10)
612                                     {
613                                             //Fehler
614                                             CommandResponseCode
                                                    = 0xFF000000 |
                                                    RESPONSE_PTP_ERROR
                                                    ;
615                                             SendContainer(
                                                    CommandResponseCode
                                                    ,0,buffer,
                                                    GetTransactionIDRead
                                                    ());
616                                             break;
617                                     }
618                             }
619                             IncreasePTPTransactionID();
620
621                             //Response mit OK zurück schicken
622                             CommandResponseCode = 0xFF000000 |
                                    RESPONSE_OK   ;
623                             SendContainer(CommandResponseCode
                                    ,0,buffer,GetTransactionIDRead()
                                    );
624                     }
625                     break;
626             case COMMAND_PTP_CHANGE_TV:
627                     if(UpdateConnection(CommandResponseCode)>1)
```

```
628                                     {
629                                             //Fehler
630                                             CommandResponseCode = 0xFF000000 |
                                                    RESPONSE_ERROR;
631                                             SendContainer(CommandResponseCode
                                                    ,0,buffer,GetTransactionIDRead()
                                                    );
632                                     }
633                                     else
634                                     {
635                                             //Passt
636                                             //Für welchen USB Host ist es
637                                             check = (char) (CommandResponseCode
                                                    >>8 & 0xFF);
638                                             if(check == 0x00)
639                                             {
640                                                     hUsbHost = GetUSBHost1();
641                                                     hEPIn = GetBulkInEP1();
642                                                     hEPOut = GetBulkOutEP1();
643                                             }
644                                             else
645                                             {
646                                                     hUsbHost = GetUSBHost2();
647                                                     hEPIn = GetBulkInEP2();
648                                                     hEPOut = GetBulkOutEP2();
649                                             }
650
651                                             //Befehl kann gesendet werden
652                                             if(SendDataPTP(hUsbHost,hEPOut,
                                                    PTP_OPCODE_TV_Value,
                                                    GetPTPTransactionID(), 0,0,0,0,
                                                    USB_PTP_CONTAINER_COMMANDBLOCK)
                                                    != 0x00C)
653                                             {
654                                                     //Fehler
655                                                     CommandResponseCode = 0
                                                            xFF000000 |
                                                            RESPONSE_PTP_ERROR;
656                                                     SendContainer(
                                                            CommandResponseCode,0,
                                                            buffer,
                                                            GetTransactionIDRead());
657                                                     break;
658                                             }
659                                             else
660                                             {
661
662                                                     if(SendDataPTP(hUsbHost,
                                                            hEPOut,
                                                            PTP_OPCODE_TV_Value,
                                                            GetPTPTransactionID(),
                                                            3,0x000C,0xD102 , (
                                                            unsigned int) arrayParam
                                                            [0] ,
                                                            USB_PTP_CONTAINER_DATABLOCK
                                                            ) != 0x18)
663                                                     {
664                                                             //Fehler
```

```
665                                                     CommandResponseCode
                                                            = 0xFF000000 |
                                                        RESPONSE_PTP_ERROR
                                                        ;
666                                                     SendContainer(
                                                        CommandResponseCode
                                                        ,0,buffer,
                                                        GetTransactionIDRead
                                                        ());
667                                                     break;
668                                             }
669                                             else
670                                             {
671                                                     //Daten Antwort
                                                        empfangen
672                                                     if(ReadDataPTP(
                                                        hUsbHost, hEPIn,
                                                         0x0C) != 0x0C)
673                                                     {
674                                                             //Fehler
675                                                             CommandResponseCode
                                                                = 0
                                                                xFF000000
                                                                |
                                                                RESPONSE_PTP_ERROR
                                                                ;
676                                                             SendContainer
                                                                (
                                                                CommandResponseCode
                                                                ,0,
                                                                buffer,
                                                                GetTransactionIDRead
                                                                ());
677                                                             break;
678                                                     }
679                                             }
680                                     }
681                                     IncreasePTPTransactionID();
682
683                                     //Response mit OK zurück schicken
684                                     CommandResponseCode = 0xFF000000 |
                                            RESPONSE_OK   ;
685                                     SendContainer(CommandResponseCode
                                            ,0,buffer,GetTransactionIDRead()
                                            );
686                             }
687                             break;
688                     case COMMAND_PTP_CHANGE_AV:
689                             if(UpdateConnection(CommandResponseCode)>1)
690                             {
691                                     //Fehler
692                                     CommandResponseCode = 0xFF000000 |
                                            RESPONSE_ERROR;
693                                     SendContainer(CommandResponseCode
                                            ,0,buffer,GetTransactionIDRead()
                                            );
694                             }
695                             else
696                             {
```

```
697                                     //Passt
698
699                                     //Für welchen USB Host ist es
700                                     check = (char) (CommandResponseCode
                                            >>8 & 0xFF);
701                                     if(check == 0x00)
702                                     {
703                                             hUsbHost = GetUSBHost1();
704                                             hEPIn = GetBulkInEP1();
705                                             hEPOut = GetBulkOutEP1();
706                                     }
707                                     else
708                                     {
709                                             hUsbHost = GetUSBHost2();
710                                             hEPIn = GetBulkInEP2();
711                                             hEPOut = GetBulkOutEP2();
712                                     }
713
714                                     //Befehl kann gesendet werden
715                                     if(SendDataPTP(hUsbHost,hEPOut,
                                            PTP_OPCODE_AV_Value,
                                            GetPTPTransactionID(), 0,0,0,0,
                                            USB_PTP_CONTAINER_COMMANDBLOCK)
                                            != 0x00C)
716                                     {
717                                             //Fehler
718                                             CommandResponseCode = 0
                                                    xFF000000 |
                                                    RESPONSE_PTP_ERROR;
719                                             SendContainer(
                                                    CommandResponseCode,0,
                                                    buffer,
                                                    GetTransactionIDRead());
720                                             break;
721                                     }
722                                     else
723                                     {
724
725                                             if(SendDataPTP(hUsbHost,
                                                    hEPOut,
                                                    PTP_OPCODE_AV_Value,
                                                    GetPTPTransactionID(),
                                                    3,0x000C,0xD101 , (
                                                    unsigned int) arrayParam
                                                    [0] ,
                                                    USB_PTP_CONTAINER_DATABLOCK
                                                    ) != 0x18)
726                                             {
727                                                     //Fehler
728                                                     CommandResponseCode
                                                            = 0xFF000000 |
                                                            RESPONSE_PTP_ERROR
                                                            ;
729                                                     SendContainer(
                                                            CommandResponseCode
                                                            ,0,buffer,
                                                            GetTransactionIDRead
                                                            ());
730                                                     break;
```

```
731                                             }
732                                             else
733                                             {
734                                                     //Daten Antwort
                                                            empfangen
735                                                     if(ReadDataPTP(
                                                            hUsbHost, hEPIn,
                                                             0x0C) != 0x0C)
736                                                     {
737                                                             //Fehler
738                                                             CommandResponseCode
                                                                    = 0
                                                                xFF000000
                                                                 |
                                                                RESPONSE_PTP_ERROR
                                                                ;
739                                                             SendContainer
                                                                (
                                                                CommandResponseCode
                                                                ,0,
                                                                buffer,
                                                                GetTransactionIDRead
                                                                ());
740                                                             break;
741                                                     }
742                                             }
743                                     }
744                                     IncreasePTPTransactionID();
745
746                                     //Response mit OK zurück schicken
747                                     CommandResponseCode = 0xFF000000 |
                                            RESPONSE_OK   ;
748                                     SendContainer(CommandResponseCode
                                            ,0,buffer,GetTransactionIDRead()
                                            );
749                             }
750                             break;
751                     case COMMAND_PTP_CHANGE_BELICHTUNG:
752                             if(UpdateConnection(CommandResponseCode)>1)
753                             {
754                                     //Fehler
755                                     CommandResponseCode = 0xFF000000 |
                                            RESPONSE_ERROR;
756                                     SendContainer(CommandResponseCode
                                            ,0,buffer,GetTransactionIDRead()
                                            );
757                             }
758                             else
759                             {
760                                     //Passt
761
762                                     //Für welchen USB Host ist es
763                                     check = (char) (CommandResponseCode
                                            >>8 & 0xFF);
764                                     if(check == 0x00)
765                                     {
766                                             hUsbHost = GetUSBHost1();
767                                             hEPIn = GetBulkInEP1();
768                                             hEPOut = GetBulkOutEP1();
```

```
769                                     }
770                                     else
771                                     {
772                                             hUsbHost = GetUSBHost2();
773                                             hEPIn = GetBulkInEP2();
774                                             hEPOut = GetBulkOutEP2();
775                                     }
776
777                                     //Befehl kann gesendet werden
778                                     if(SendDataPTP(hUsbHost,hEPOut,
                                            PTP_OPCODE_BELICHTUNG,
                                            GetPTPTransactionID(), 0,0,0,0,
                                            USB_PTP_CONTAINER_COMMANDBLOCK)
                                            != 0x00C)
779                                     {
780                                             //Fehler
781                                             CommandResponseCode = 0
                                                    xFF000000 |
                                                    RESPONSE_PTP_ERROR;
782                                             SendContainer(
                                                    CommandResponseCode,0,
                                                    buffer,
                                                    GetTransactionIDRead());
783                                             break;
784                                     }
785                                     else
786                                     {
787
788                                             if(SendDataPTP(hUsbHost,
                                                    hEPOut,
                                                    PTP_OPCODE_BELICHTUNG,
                                                    GetPTPTransactionID(),
                                                    3,0x000C,0xD104 , (
                                                    unsigned int) arrayParam
                                                    [0] ,
                                                    USB_PTP_CONTAINER_DATABLOCK
                                                    ) != 0x18)
789                                             {
790                                                     //Fehler
791                                                     CommandResponseCode
                                                            = 0xFF000000 |
                                                            RESPONSE_PTP_ERROR
                                                            ;
792                                                     SendContainer(
                                                            CommandResponseCode
                                                            ,0,buffer,
                                                            GetTransactionIDRead
                                                            ());
793                                                     break;
794                                             }
795                                             else
796                                             {
797                                                     //Daten Antwort
                                                            empfangen
798                                                     if(ReadDataPTP(
                                                            hUsbHost, hEPIn,
                                                             0x0C) != 0x0C)
799                                                     {
800                                                             //Fehler
```

```
801                                                                 CommandResponseCode = 0xFF000000 | RESPONSE_PTP_ERROR;
802                                                                 SendContainer(CommandResponseCode,0,buffer,GetTransactionIDRead());
803                                                                 break;
804                                                             }
805                                                         }
806                                                     }
807                                                     IncreasePTPTransactionID();
808
809                                                     //Response mit OK zurück schicken
810                                                     CommandResponseCode = 0xFF000000 | RESPONSE_OK;
811                                                     SendContainer(CommandResponseCode,0,buffer,GetTransactionIDRead());
812                                             }
813                                             break;
814                                     case COMMAND_PTP_LIVE_VIEW_ON:
815                                             if(UpdateConnection(CommandResponseCode)>1)
816                                             {
817                                                     //Fehler
818                                                     CommandResponseCode = 0xFF000000 | RESPONSE_ERROR;
819                                                     SendContainer(CommandResponseCode,0,buffer,GetTransactionIDRead());
820                                             }
821                                             else
822                                             {
823                                                     //Passt
824
825                                                     //Für welchen USB Host ist es
826                                                     check = (char) (CommandResponseCode>>8 & 0xFF);
827                                                     if(check == 0x00)
828                                                     {
829                                                             hUsbHost = GetUSBHost1();
830                                                             hEPIn = GetBulkInEP1();
831                                                             hEPOut = GetBulkOutEP1();
832                                                     }
833                                                     else
834                                                     {
835                                                             hUsbHost = GetUSBHost2();
836                                                             hEPIn = GetBulkInEP2();
837                                                             hEPOut = GetBulkOutEP2();
838                                                     }
839
840                                                     //Befehl kann gesendet werden
```

```
841                 if(SendDataPTP(hUsbHost,hEPOut,
                        PTP_OPCODE_LIVEVIEW,
                        GetPTPTransactionID(), 0,0,0,0,
                        USB_PTP_CONTAINER_COMMANDBLOCK)
                        != 0x00C)
842                 {
843                     //Fehler
844                     CommandResponseCode = 0
                            xFF000000 |
                            RESPONSE_PTP_ERROR;
845                     SendContainer(
                            CommandResponseCode,0,
                            buffer,
                            GetTransactionIDRead());
846                     break;
847                 }
848                 else
849                 {
850
851                     if(SendDataPTP(hUsbHost,
                            hEPOut,
                            PTP_OPCODE_LIVEVIEW,
                            GetPTPTransactionID(),
                            3,0x000C,0xD1B0 , 2 ,
                            USB_PTP_CONTAINER_DATABLOCK
                            ) != 0x18)
852                     {
853                         //Fehler
854                         CommandResponseCode
                                = 0xFF000000 |
                                RESPONSE_PTP_ERROR
                                ;
855                         SendContainer(
                                CommandResponseCode
                                ,0,buffer,
                                GetTransactionIDRead
                                ());
856                         break;
857                     }
858                     else
859                     {
860                         //Daten Antwort
                                empfangen
861                         if(ReadDataPTP(
                                hUsbHost, hEPIn,
                                0x0C) != 0x0C)
862                         {
863                             //Fehler
864                             CommandResponseCode
                                    = 0
                                    xFF000000
                                    |
                                    RESPONSE_PTP_ERROR
                                    ;
865                             SendContainer
                                    (
                                    CommandResponseCode
                                    ,0,
                                    buffer,
```

```
                                                                GetTransactionIDRead
                                                                ());
866                                                         break;
867                                                 }
868                                         }
869                                 }
870                                 IncreasePTPTransactionID();
871
872                                 //Response mit OK zurück schicken
873                                 CommandResponseCode = 0xFF000000 |
                                        RESPONSE_OK   ;
874                                 SendContainer(CommandResponseCode
                                        ,0,buffer,GetTransactionIDRead()
                                        );
875                         }
876                         break;
877                 case COMMAND_PTP_LIVE_VIEW_OFF:
878                         if(UpdateConnection(CommandResponseCode)>1)
879                         {
880                                 //Fehler
881                                 CommandResponseCode = 0xFF000000 |
                                        RESPONSE_ERROR;
882                                 SendContainer(CommandResponseCode
                                        ,0,buffer,GetTransactionIDRead()
                                        );
883                         }
884                         else
885                         {
886                                 //Passt
887
888                                 //Für welchen USB Host ist es
889                                 check = (char) (CommandResponseCode
                                        >>8 & 0xFF);
890                                 if(check == 0x00)
891                                 {
892                                         hUsbHost = GetUSBHost1();
893                                         hEPIn = GetBulkInEP1();
894                                         hEPOut = GetBulkOutEP1();
895                                 }
896                                 else
897                                 {
898                                         hUsbHost = GetUSBHost2();
899                                         hEPIn = GetBulkInEP2();
900                                         hEPOut = GetBulkOutEP2();
901                                 }
902
903                                 //Befehl kann gesendet werden
904                                 if(SendDataPTP(hUsbHost,hEPOut,
                                        PTP_OPCODE_LIVEVIEW,
                                        GetPTPTransactionID(), 0,0,0,0,
                                        USB_PTP_CONTAINER_COMMANDBLOCK)
                                        != 0x00C)
905                                 {
906                                         //Fehler
907                                         CommandResponseCode = 0
                                                xFF000000 |
                                                RESPONSE_PTP_ERROR;
908                                         SendContainer(
                                                CommandResponseCode,0,
```

```
                    buffer,
                    GetTransactionIDRead());
909                 break;
910             }
911             else
912             {
913
914                 if(SendDataPTP(hUsbHost,
                        hEPOut,
                        PTP_OPCODE_LIVEVIEW,
                        GetPTPTransactionID(),
                        3,0x000C,0xD1B0 , 0 ,
                        USB_PTP_CONTAINER_DATABLOCK
                        ) != 0x18)
915                 {
916                     //Fehler
917                     CommandResponseCode
                            = 0xFF000000 |
                            RESPONSE_PTP_ERROR
                            ;
918                     SendContainer(
                            CommandResponseCode
                            ,0,buffer,
                            GetTransactionIDRead
                            ());
919                     break;
920                 }
921                 else
922                 {
923                     //Daten Antwort
                            empfangen
924                     if(ReadDataPTP(
                            hUsbHost, hEPIn,
                             0x0C) != 0x0C)
925                     {
926                         //Fehler
927                         CommandResponseCode
                                = 0
                                xFF000000
                                |
                                RESPONSE_PTP_ERROR
                                ;
928                         SendContainer
                                (
                                CommandResponseCode
                                ,0,
                                buffer,
                                GetTransactionIDRead
                                ());
929                         break;
930                     }
931                 }
932             }
933             IncreasePTPTransactionID();
934
935             //Response mit OK zurück schicken
936             CommandResponseCode = 0xFF000000 |
                    RESPONSE_OK   ;
```

```
937                                     SendContainer(CommandResponseCode
                                            ,0,buffer,GetTransactionIDRead()
                                            );
938                             }
939                             break;
940                     case COMMAND_PTP_FOKUS:
941                             if(UpdateConnection(CommandResponseCode)>1)
942                             {
943                                     //Fehler
944                                     CommandResponseCode = 0xFF000000 |
                                            RESPONSE_ERROR;
945                                     SendContainer(CommandResponseCode
                                            ,0,buffer,GetTransactionIDRead()
                                            );
946                             }
947                             else
948                             {
949                                     //Passt
950
951                                     //Für welchen USB Host ist es
952                                     check = (char) (CommandResponseCode
                                            >>8 & 0xFF);
953                                     if(check == 0x00)
954                                     {
955                                             hUsbHost = GetUSBHost1();
956                                             hEPIn = GetBulkInEP1();
957                                             hEPOut = GetBulkOutEP1();
958                                     }
959                                     else
960                                     {
961                                             hUsbHost = GetUSBHost2();
962                                             hEPIn = GetBulkInEP2();
963                                             hEPOut = GetBulkOutEP2();
964                                     }
965
966                                     //Befehl kann gesendet werden
967                                     if(SendDataPTP(hUsbHost,hEPOut,
                                            PTP_OPCODE_FOCUS,
                                            GetPTPTransactionID(), 1,(
                                            unsigned int) arrayParam[1] << 8
                                              | arrayParam[0],0,0,
                                            USB_PTP_CONTAINER_COMMANDBLOCK)
                                            != 0x010)
968                                     {
969                                             //Fehler
970                                             CommandResponseCode = 0
                                                    xFF000000 |
                                                    RESPONSE_PTP_ERROR;
971                                             SendContainer(
                                                    CommandResponseCode,0,
                                                    buffer,
                                                    GetTransactionIDRead());
972                                             break;
973                                     }
974                                     else
975                                     {
976                                             //Daten Antwort empfangen
977                                             if(ReadDataPTP(hUsbHost,
                                                    hEPIn, 0x0C) != 0x0C)
```

```
978                                             {
979                                                     //Fehler
980                                                     CommandResponseCode
                                                            = 0xFF000000 |
                                                        RESPONSE_PTP_ERROR
                                                        ;
981                                                     SendContainer(
                                                        CommandResponseCode
                                                        ,0,buffer,
                                                        GetTransactionIDRead
                                                        ());
982                                                     break;
983                                             }
984                                     }
985                                     IncreasePTPTransactionID();
986
987                                     //Response mit OK zurück schicken
988                                     CommandResponseCode = 0xFF000000 |
                                            RESPONSE_OK   ;
989                                     SendContainer(CommandResponseCode
                                            ,0,buffer,GetTransactionIDRead()
                                            );
990                             }
991                             break;
992                     case COMMAND_PTP_GETCONNECTION:
993                             //Für welchen USB Host ist es
994                             check = (char) (CommandResponseCode>>8 & 0
                                    xFF);
995
996                             buffer[0]=0;
997
998                             for(i=0;i<2;i++)
999                             {
1000
1001                                    if(i == 0)
1002                                    {
1003                                            hUsbHost = GetUSBHost1();
1004                                    }
1005                                    else
1006                                    {
1007                                            hUsbHost = GetUSBHost2();
1008                                    }
1009                                    //Kontrolle ob etwas angesteckt ist
1010                                    tmp = GetConnectionState(hUsbHost);
1011
1012                                    if(tmp != 0)
1013                                    {
1014                                            //Es ist etwas angesteckt
1015                                            //Kontrolle ob es ein PTP
                                                    Gerät ist
1016                                            ifDev = GetPTPDevice(
                                                    hUsbHost, ifDev);
1017
1018                                            if(ifDev == NULL)
1019                                            {
1020                                                    //Kein PTP Gerät
1021                                                    tmp = 1;
1022                                            }
1023                                            else
```

```
1024                                                    {
1025                                                            tmp = 3;
1026                                                    }
1027                                            }
1028                                            else
1029                                            {
1030                                                    //Es ist nichts angesteckt
1031                                                    tmp = 0;
1032                                            }
1033
1034                                            buffer[0] |=  (tmp<<2*i);
1035                                    }
1036
1037                                    //Datenphase mit Ergebnis zurückschicken
1038                                    CommandResponseCode = 0x00FFFFFF & tmp;
1039                                    SendContainer(CommandResponseCode,1,buffer,
                                            GetTransactionIDRead());
1040
1041                                    //Response mit OK zurück schicken
1042                                    CommandResponseCode = 0xFF000000 |
                                            RESPONSE_OK   ;
1043                                    SendContainer(CommandResponseCode,0,buffer,
                                            GetTransactionIDRead());
1044
1045                                    break;
1046                            default:
1047                                    return;
1048                    }
1049            }
1050
1051    }
1052
1053    //CheckCommand
1054    //Funktion überprüft den Command Code ob er unterstützt wird und ob die
            Parameter stimmen
1055    //Muss für jedes Device angepasst werden
1056    // @Param1: Command Code
1057    // @Param2: Anzahl der Parameter
1058    //Return Command.h
1059    unsigned short CheckCommand(uint32 CommandResponseCode, unsigned short
            countParam)
1060    {
1061            char check;
1062
1063            check = (char) (CommandResponseCode>>24 & 0xFF);
1064
1065            if(check == 0 || check == 0xFF)
1066            {
1067                    //ResponseCode wird nicht unterstützt
1068                    return RESPONSE_NOT_SUPPORTED_PHASE;
1069            }
1070            else
1071            {
1072                    //Es wird nur die Klasse 5 unterstützt
1073                    check = (char) (CommandResponseCode>>16 & 0xFF);
1074                    if(check != 0x05)
1075                    {
1076                            return RESPONSE_NOT_SUPPORTED_CLASS;
1077                    }
```

```
1078                    else
1079                    {
1080                            //Es wird nur USB Host 0 und 1 unterstützt
1081                            check = (char) (CommandResponseCode>>8 & 0xFF);
1082                            if(check > 0x01)
1083                            {
1084                                    return RESPONSE_NOT_SUPPORTED_ADCLASS;
1085                            }
1086                            else
1087                            {
1088                                    //Anzahl der Parameter überprüfen
1089                                    //CommandCode checken
1090                                    check = (char) (CommandResponseCode & 0xFF)
                                            ;
1091                                    switch(check)
1092                                    {
1093                                            case COMMAND_PTP_SESSION_OPEN:
1094                                            case COMMAND_PTP_SESSION_CLOSE:
1095                                            case COMMAND_PTP_SHOOT:
1096                                            case COMMAND_PTP_GETCONNECTION:
1097                                            case COMMAND_PTP_LIVE_VIEW_ON:
1098                                            case COMMAND_PTP_LIVE_VIEW_OFF:
1099                                                    if(countParam!=0)
1100                                                    {
1101                                                            //Anzahl stimmt
                                                                    nicht
1102                                                            return
                                                                    RESPONSE_NOT_SUPPORTED_PARAME
                                                                    ;
1103                                                    }
1104                                                    break;
1105                                            case COMMAND_PTP_CHANGE_TV:
1106                                            case COMMAND_PTP_CHANGE_AV:
1107                                            case COMMAND_PTP_CHANGE_BELICHTUNG:
1108                                                    if(countParam!=1)
1109                                                    {
1110                                                            //Anzahl stimmt
                                                                    nicht
1111                                                            return
                                                                    RESPONSE_NOT_SUPPORTED_PARAME
                                                                    ;
1112                                                    }
1113                                                    break;
1114                                            case COMMAND_PTP_FOKUS:
1115                                                    if(countParam!=2)
1116                                                    {
1117                                                            //Anzahl stimmt
                                                                    nicht
1118                                                            return
                                                                    RESPONSE_NOT_SUPPORTED_PARAME
                                                                    ;
1119                                                    }
1120                                                    break;
1121                                            default:
1122                                                    return
                                                            RESPONSE_NOT_SUPPORTED;
1123                                    }
1124                            }
1125                    }
```

```
1126            }
1127            return RESPONSE_OK;
1128    }
```

Host.h

```
1   //Zugriff Daten auf den USB Host des VNC
2
3   #ifndef Host_H_
4   #define Host_H_
5
6   #include "vos.h"
7   #include "USBHost.h"
8   #include "USB.h"
9
10
11  //Funktion liefert den ConnectionState zurück
12  /* Return Values
13   1 wenn PORT_STATE_ENUMERATED sonst 0
14  */
15  unsigned char GetConnectionState(VOS_HANDLE hUsbHost);
16
17  //Funktion gibt den Handle für PTP Interface zurück
18  //Rückgabe NULL wenn keines vorhanden ist
19  usbhost_device_handle GetPTPDevice(VOS_HANDLE hUsbHost,
        usbhost_device_handle *ifDev);
20
21  //Funktion gibt den DeviceHandle vom USB Host 1 zurück
22  usbhost_device_handle *GetDeviceHandle1();
23
24  //Funktion gibt den DeviceHandle vom USB Host 2 zurück
25  usbhost_device_handle *GetDeviceHandle2();
26
27  //Funktion setzt den DeviceHandle vom USB Host 1
28  void SetDeviceHandle1(usbhost_device_handle *ifDev);
29
30  //Funktion setzt den DeviceHandle vom USB Host 2
31  void SetDeviceHandle2(usbhost_device_handle *ifDev);
32
33  //Return Handle from USBHost1
34  VOS_HANDLE GetUSBHost1();
35
36  //Return Handle from USBHost1
37  VOS_HANDLE GetUSBHost2();
38
39  //Set Handle from USBHost1
40  void SetUSBHost1(VOS_HANDLE handle);
41
42  //Set Handle from USBHost2
43  void SetUSBHost2(VOS_HANDLE handle);
44
45  //Set Interrupt EP USBHost1
46  void SetInterruptEP1(usbhost_ep_handle hInterruptEpIn);
47
48  //Set Interrupt EP USBHost2
49  void SetInterruptEP2(usbhost_ep_handle hInterruptEpIn);
50
51  //Get Interrupt EP USBHost 1
52  usbhost_ep_handle GetInterruptEP1(void);
```

```
53
54 //Get Interrupt EP USBHost 2
55 usbhost_ep_handle GetInterruptEP2(void);
56
57 //Set BulkIn EP USBHost1
58 void SetBulkInEP1(usbhost_ep_handle hBulkInEpIn);
59
60 //Set BulkIn EP USBHost2
61 void SetBulkInEP2(usbhost_ep_handle hBulkInEpIn);
62
63 //Get BulkIn EP USBHost 1
64 usbhost_ep_handle GetBulkInEP1(void);
65
66 //Get BulkIn EP USBHost 2
67 usbhost_ep_handle GetBulkInEP2(void);
68
69 //Set Bulk EP USBHost1
70 void SetBulkEPOut1(usbhost_ep_handle hBulkEpOut);
71 //Set Bulk EP USBHost2
72 void SetBulkEPOut2(usbhost_ep_handle hBulkEpOut);
73
74 //Get Bulk EP USBHost 1
75 usbhost_ep_handle GetBulkOutEP1(void);
76
77 //Get Bulk EP USBHost 2
78 usbhost_ep_handle GetBulkOutEP2(void);
79
80 //Funktion gibt den Interrupt IN Endpunkt zurück
81 //Übergabe ist Null wenn kein Intterupt IN Endpunkt gefunden wurde
82 usbhost_ep_handle GetInterruptInEP(VOS_HANDLE hUsbHost,
       usbhost_device_handle *ifDev, usbhost_ep_handle hEp);
83
84 //Funktion gibt den BULK IN Endpunkt zurück
85 //Übergabe ist Null wenn kein BULK IN Endpunkt gefunden wurde
86 usbhost_ep_handle GetBulkInEP(VOS_HANDLE hUsbHost, usbhost_device_handle *
       ifDev, usbhost_ep_handle hEp);
87
88 //Funktion gibt den BULK OUT Endpunkt zurück
89 //Übergabe ist Null wenn kein BULK OUT Endpunkt gefunden wurde
90 usbhost_ep_handle GetBulkOutEP(VOS_HANDLE hUsbHost, usbhost_device_handle *
       ifDev, usbhost_ep_handle hEp);
91
92 //Funktion erneuert eine Verbindung
93 //Param1: Handle zum USBHost
94 //Param2: Number USB Host (0 = USB Host 1, 1 = USB Host 2)
95 //Return 0 success 1 any Error
96 char RenewConnection(VOS_HANDLE hUsbHost, char USBHost);
97
98 //UpdateConnection
99 //Param1: CommandCode der Anweisung für die Festellung welcher USBHost
100 //return        0: Success
101 //                      1: Daten sind schon vorhanden
102 //                      2: No PTP
103 //                      4: Nichts angesteckt
104 //                      8: Error Renew
105 //                      16: Verlorener Fall
106 char UpdateConnection(uint32 CommandResponseCode);
107
108 #endif /* Host_H_ */
```

Host.c

```
1  #include "Host.h"
2
3
4  //USB Host 1
5  VOS_HANDLE hUsbHost1;
6  usbhost_ep_handle hInterruptEpIn1;
7  usbhost_ep_handle hBulkEpIn1;
8  usbhost_ep_handle hBulkEpOut1;
9  usbhost_device_handle *ifDev1;
10
11 //USB Host 2
12 VOS_HANDLE hUsbHost2;
13 usbhost_ep_handle hInterruptEpIn2;
14 usbhost_ep_handle hBulkEpIn2;
15 usbhost_ep_handle hBulkEpOut2;
16 usbhost_device_handle *ifDev2;
17
18 //Funktion liefert den ConnectionState zurück
19 /* Return Values
20  1 wenn PORT_STATE_ENUMERATED sonst 0
21 */
22 unsigned char GetConnectionState(VOS_HANDLE hUsbHost)
23 {
24                 unsigned char status;
25                 usbhost_ioctl_cb_t usbhost_iocb;
26
27                 do
28                 {
29                         vos_delay_msecs(1);
30                         usbhost_iocb.ioctl_code =
                               VOS_IOCTL_USBHOST_GET_CONNECT_STATE;
31                         usbhost_iocb.get = &status;
32                         vos_dev_ioctl(hUsbHost, &usbhost_iocb);
33                 } while(status == PORT_STATE_CONNECTED && status !=
                       PORT_STATE_DISCONNECTED);
34
35
36                 if(status == PORT_STATE_ENUMERATED)
37                 {
38                         return 1;
39                 }
40                 else
41                 {
42                         return 0;
43                 }
44 }
45
46 //Funktion gibt den Handle für PTP Interface zurück
47 //Rückgabe NULL wenn keines vorhanden ist
48 usbhost_device_handle GetPTPDevice(VOS_HANDLE hUsbHost,
       usbhost_device_handle *ifDev)
49 {
50         usbhost_ioctl_cb_t usbhost_iocb; // ioctl block
51         usbhost_ioctl_cb_class_t hc_iocb_class;
52
53         //Interface mit PTP Klasse
54         hc_iocb_class.dev_class = USB_CLASS_IMAGE;
```

```
55          hc_iocb_class.dev_subclass = USB_SUBCLASS_IMAGE_STILLIMAGE;
56          hc_iocb_class.dev_protocol = USB_PROTOCOL_IMAGE_PIMA;
57
58          usbhost_iocb.ioctl_code =
                VOS_IOCTL_USBHOST_DEVICE_FIND_HANDLE_BY_CLASS;
59          usbhost_iocb.handle.dif = NULL;
60          usbhost_iocb.set = &hc_iocb_class;
61          usbhost_iocb.get = &ifDev;
62          vos_dev_ioctl(hUsbHost, &usbhost_iocb);
63
64          return ifDev;
65  }
66
67  //Funktion gibt den DeviceHandle vom USB Host 1 zurück
68  usbhost_device_handle * GetDeviceHandle1()
69  {
70          return ifDev1;
71  }
72
73  //Funktion gibt den DeviceHandle vom USB Host 2 zurück
74  usbhost_device_handle * GetDeviceHandle2()
75  {
76          return ifDev2;
77  }
78
79  //Funktion setzt den DeviceHandle vom USB Host 1
80  void SetDeviceHandle1(usbhost_device_handle *ifDev)
81  {
82          ifDev1 = ifDev;
83  }
84
85  //Funktion setzt den DeviceHandle vom USB Host 2
86  void SetDeviceHandle2(usbhost_device_handle *ifDev)
87  {
88          ifDev2 = ifDev;
89  }
90
91  //Return Handle from USBHost1
92  VOS_HANDLE GetUSBHost1()
93  {
94          return hUsbHost1;
95  }
96
97  //Return Handle from USBHost1
98  VOS_HANDLE GetUSBHost2()
99  {
100         return hUsbHost2;
101 }
102
103
104 //Set Handle from USBHost1
105 void SetUSBHost1(VOS_HANDLE handle)
106 {
107         hUsbHost1 = handle;
108 }
109
110 //Set Handle from USBHost2
111 void SetUSBHost2(VOS_HANDLE handle)
112 {
```

```
113         hUsbHost2 = handle;
114 }
115
116
117 //Set Interrupt EP USBHost1
118 void SetInterruptEP1(usbhost_ep_handle hInterruptEpIn)
119 {
120         hInterruptEpIn1 = hInterruptEpIn;
121 }
122
123 //Set Interrupt EP USBHost2
124 void SetInterruptEP2(usbhost_ep_handle hInterruptEpIn)
125 {
126         hInterruptEpIn2 = hInterruptEpIn;
127 }
128
129
130 //Get Interrupt EP USBHost 1
131 usbhost_ep_handle GetInterruptEP1(void)
132 {
133         return hInterruptEpIn1;
134 }
135
136 //Get Interrupt EP USBHost 2
137 usbhost_ep_handle GetInterruptEP2(void)
138 {
139         return hInterruptEpIn2;
140 }
141
142
143 //Set BulkIn EP USBHost1
144 void SetBulkInEP1(usbhost_ep_handle hBulkInEpIn)
145 {
146         hBulkEpIn1 = hBulkInEpIn;
147 }
148
149 //Set BulkIn EP USBHost2
150 void SetBulkInEP2(usbhost_ep_handle hBulkInEpIn)
151 {
152         hBulkEpIn2 = hBulkInEpIn;
153 }
154
155
156 //Get BulkIn EP USBHost 1
157 usbhost_ep_handle GetBulkInEP1(void)
158 {
159         return hBulkEpIn1;
160 }
161
162 //Get BulkIn EP USBHost 2
163 usbhost_ep_handle GetBulkInEP2(void)
164 {
165         return hBulkEpIn2;
166 }
167
168 //Set Interrupt EP USBHost1
169 void SetBulkEPOut1(usbhost_ep_handle hBulkEpOut)
170 {
171         hBulkEpOut1 = hBulkEpOut;
```

```
172 }
173
174 //Set Bulk EP USBHost2
175 void SetBulkEPOut2(usbhost_ep_handle hBulkEpOut)
176 {
177         hBulkEpOut2 = hBulkEpOut;
178 }
179
180
181 //Get Bulk EP USBHost 1
182 usbhost_ep_handle GetBulkOutEP1(void)
183 {
184         return hBulkEpOut1;
185 }
186
187 //Get Bulk EP USBHost 2
188 usbhost_ep_handle GetBulkOutEP2(void)
189 {
190         return hBulkEpOut2;
191 }
192
193 //Funktion gibt den Interrupt IN Endpunkt zurück
194 //Übergabe ist Null wenn kein Intterupt IN Endpunkt gefunden wurde
195 usbhost_ep_handle GetInterruptInEP(VOS_HANDLE hUsbHost,
        usbhost_device_handle *ifDev, usbhost_ep_handle hEp)
196 {
197         usbhost_ioctl_cb_t host_ioctl_cb; // ioctl block
198
199         host_ioctl_cb.ioctl_code =
            VOS_IOCTL_USBHOST_DEVICE_GET_INT_IN_ENDPOINT_HANDLE;
200         host_ioctl_cb.handle.dif = ifDev;
201         host_ioctl_cb.get = &hEp;
202         vos_dev_ioctl(hUsbHost, &host_ioctl_cb);
203
204         return hEp;
205 }
206
207 //Funktion gibt den BULK IN Endpunkt zurück
208 //Übergabe ist Null wenn kein BULK IN Endpunkt gefunden wurde
209 usbhost_ep_handle GetBulkInEP(VOS_HANDLE hUsbHost, usbhost_device_handle *
        ifDev, usbhost_ep_handle hEp)
210 {
211         usbhost_ioctl_cb_t host_ioctl_cb; // ioctl block
212
213         host_ioctl_cb.ioctl_code =
            VOS_IOCTL_USBHOST_DEVICE_GET_BULK_IN_ENDPOINT_HANDLE;
214         host_ioctl_cb.handle.dif = ifDev;
215         host_ioctl_cb.get = &hEp;
216         vos_dev_ioctl(hUsbHost, &host_ioctl_cb);
217
218         return hEp;
219 }
220
221 //Funktion gibt den BULK OUT Endpunkt zurück
222 //Übergabe ist Null wenn kein BULK OUT Endpunkt gefunden wurde
223 usbhost_ep_handle GetBulkOutEP(VOS_HANDLE hUsbHost, usbhost_device_handle *
        ifDev, usbhost_ep_handle hEp)
224 {
225         usbhost_ioctl_cb_t host_ioctl_cb; // ioctl block
```

```
226
227         host_ioctl_cb.ioctl_code =
                VOS_IOCTL_USBHOST_DEVICE_GET_BULK_OUT_ENDPOINT_HANDLE;
228         host_ioctl_cb.handle.dif = ifDev;
229         host_ioctl_cb.get = &hEp;
230         vos_dev_ioctl(hUsbHost, &host_ioctl_cb);
231
232         return hEp;
233 }
234
235 //Funktion erneuert eine Verbindung
236 //Param1: Handle zum USBHost
237 //Param2: Number USB Host (0 = USB Host 1, 1 = USB Host 2)
238 //Return 0 success 1 any Error
239 char RenewConnection(VOS_HANDLE hUsbHost, char USBHost)
240 {
241         usbhost_device_handle *ifDev;
242         usbhost_ep_handle hEP;
243
244         ifDev = NULL;
245         hEP = NULL;
246
247         if(USBHost == 0)
248         {
249                 //USB Host 1
250                 //DeviceHandle
251                 ifDev = GetPTPDevice(hUsbHost,ifDev);
252
253                 if(ifDev == NULL)
254                 {
255                         return 1;
256                 }
257                 else
258                 {
259                         SetDeviceHandle1(ifDev);
260                 }
261
262                 hEP = GetInterruptInEP(hUsbHost,ifDev,hEP);
263
264                 if(hEP == NULL)
265                 {
266                         return 1;
267                 }
268                 else
269                 {
270                         SetInterruptEP1(hEP);
271                 }
272
273                 hEP = GetBulkInEP(hUsbHost,ifDev,hEP);
274
275                 if(hEP == NULL)
276                 {
277                         return 1;
278                 }
279                 else
280                 {
281                         SetBulkInEP1(hEP);
282                 }
283
```

```
284                 hEP = GetBulkOutEP(hUsbHost,ifDev,hEP);
285
286                 if(hEP == NULL)
287                 {
288                         return 1;
289                 }
290                 else
291                 {
292                         SetBulkEPOut1(hEP);
293                 }
294
295         }
296         else
297         {
298                 //USBHost2
299                 ifDev = GetPTPDevice(hUsbHost,ifDev);
300
301                 if(ifDev == NULL)
302                 {
303                         return 1;
304                 }
305                 else
306                 {
307                         SetDeviceHandle2(ifDev);
308                 }
309
310                 hEP = GetInterruptInEP(hUsbHost,ifDev,hEP);
311
312                 if(hEP == NULL)
313                 {
314                         return 1;
315                 }
316                 else
317                 {
318                         SetInterruptEP2(hEP);
319                 }
320
321                 hEP = GetBulkInEP(hUsbHost,ifDev,hEP);
322
323                 if(hEP == NULL)
324                 {
325                         return 1;
326                 }
327                 else
328                 {
329                         SetBulkInEP2(hEP);
330                 }
331
332                 hEP = GetBulkOutEP(hUsbHost,ifDev,hEP);
333
334                 if(hEP == NULL)
335                 {
336                         return 1;
337                 }
338                 else
339                 {
340                         SetBulkEPOut2(hEP);
341                 }
342         }
```

```
343         return 0;
344 }
345
346 //UpdateConnection
347 //Param1: CommandCode der Anweisung für die Festellung welcher USBHost
348 //return        0: Success
349 //                      1: Daten sind schon vorhanden
350 //                      2: No PTP
351 //                      4: Nichts angesteckt
352 //                      8: Error Renew
353 //                      16: Verlorener Fall
354 char UpdateConnection(uint32 CommandResponseCode)
355 {
356         usbhost_device_handle *ifDev;
357         usbhost_ep_handle hEP;
358         VOS_HANDLE hUsbHost;
359         char check;
360         unsigned char tmp;
361
362         ifDev = NULL;
363         hEP = NULL;
364         hUsbHost = NULL;
365
366         check = (char) (CommandResponseCode>>8 & 0xFF);
367         if(check == 0x00)
368         {
369                 //Host 1
370                 hUsbHost = GetUSBHost1();
371         }
372         else
373         {
374                 //Host 2
375                 hUsbHost = GetUSBHost2();
376         }
377
378         //Kontrollieren ob etwas angesteckt ist
379         tmp = GetConnectionState(hUsbHost);
380
381         if(tmp != 0)
382         {
383                 //Es ist etwas angesteckt
384                 //Kontrollieren ob es vorher schon angesteckt war
385
386                 //Es ist etwas angesteckt
387                 //Kontrolle ob es ein PTP Gerät ist
388                 ifDev = GetPTPDevice(hUsbHost, ifDev);
389
390                 if(ifDev == NULL)
391                 {
392                         //Kein PTP Gerät
393                         if(check == 0x00)
394                         {
395                                 //Host 1
396                                 SetDeviceHandle1(NULL);
397                                 SetInterruptEP1(NULL);
398                                 SetBulkInEP1( NULL);
399                                 SetBulkEPOut1(NULL);
400                         }
401                         else
```

```
402                         {
403                                 //Host 2
404                                 SetDeviceHandle1(NULL);
405                                 SetInterruptEP1(NULL);
406                                 SetBulkInEP1( NULL);
407                                 SetBulkEPOut1(NULL);
408                         }
409                         return 2;
410                 }
411                 else
412                 {
413                         if(check == 0x00)
414                         {
415                                 //USB Host 1
416                                 hEP = GetInterruptEP1();
417
418                                 if(hEP == NULL)
419                                 {
420                                         //Vorher war nix da
421                                         if(RenewConnection(hUsbHost,0) !=
                                                0)
422                                         {
423                                                 return 8;
424                                         }
425                                         else
426                                         {
427                                                 return 0;
428                                         }
429                                 }
430                                 else
431                                 {
432                                         //Daten sind schon gefüllt
433                                         return 1;
434                                 }
435                         }
436                         else
437                         {
438                                 //USB Host 2
439                                 hEP = GetInterruptEP2();
440
441                                 if(hEP == NULL)
442                                 {
443                                         //Vorher war nix da
444                                         if(RenewConnection(hUsbHost,1) !=
                                                0)
445                                         {
446                                                 return 8;
447                                         }
448                                         else
449                                         {
450                                                 return 0;
451                                         }
452                                 }
453                                 else
454                                 {
455                                         //Daten sind schon gefüllt
456                                         return 1;
457                                 }
458                         }
```

```
459                     }
460             }
461             else
462             {
463                     //Es ist nichts angesteckt
464                     if(check == 0x00)
465                     {
466                             //Host 1
467                             SetDeviceHandle1(NULL);
468                             SetInterruptEP1(NULL);
469                             SetBulkInEP1( NULL);
470                             SetBulkEPOut1(NULL);
471                     }
472                     else
473                     {
474                             //Host 2
475                             SetDeviceHandle1(NULL);
476                             SetInterruptEP1(NULL);
477                             SetBulkInEP1( NULL);
478                             SetBulkEPOut1(NULL);
479                     }
480                     return 4;
481             }
482
483             return 16;
484 }
```

Commands.h

```
1  /*
2   * Commands.h
3   *
4   * Created: 07.10.2011 16:39:51
5   *  Author: Niki
6   */
7
8
9  #ifndef COMMANDS_H_
10 #define COMMANDS_H_
11
12 //Commands PTP
13 #define COMMAND_PTP_SESSION_OPEN                    0x0001
14 #define COMMAND_PTP_SESSION_CLOSE                   0x0002
15 #define COMMAND_PTP_SHOOT                                   0
       x0003
16 #define COMMAND_PTP_CHANGE_TV                       0x0004
17 #define COMMAND_PTP_CHANGE_AV                       0x0005
18 #define COMMAND_PTP_CHANGE_BELICHTUNG          0x0006
19 #define COMMAND_PTP_LIVE_VIEW_ON                    0x0007
20 #define COMMAND_PTP_FOKUS                                   0
       x0008
21 #define COMMAND_PTP_GETCONNECTION                   0x0009
22 #define COMMAND_PTP_LIVE_VIEW_OFF                   0x000A
23
24 //Rückgabe der Funktion Check Command
25 #define RESPONSE_OK
                0x0000
26 #define RESPONSE_WRONG_PARAMETER                    0x0001
27 #define RESPONSE_NOT_SUPPORTED_CLASS           0x0002
```

```
28 #define RESPONSE_NOT_SUPPORTED_ADCLASS                  0x0003
29 #define RESPONSE_NOT_SUPPORTED_PARAMETERCOUNT    0x0004
30 #define RESPONSE_NOT_SUPPORTED                                  0x0005
31 #define RESPONSE_NOT_SUPPORTED_PHASE                    0x0006
32 #define RESPONSE_ERROR                                                  0
       x0007
33 #define RESPONSE_PTP_ERROR                                              0
       x0008
34
35 #endif /* COMMANDS_H_ */
```

I Android Klasse VI

AndroidManifest.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3        package="MasterThesis.CameraControl"
4        android:versionCode="1"
5        android:versionName="1.0"
6          >
7      <uses-sdk android:minSdkVersion="9" />
8          <uses-permission android:name="android.permission.BLUETOOTH" />
9          <uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/
               >
10
11     <application android:icon="@drawable/app_icon" android:label="@string/
           app_name">
12
13         <activity android:name="TabMain"
14                           android:label="@string/app_name" android:theme="
                                 @android:style/Theme.NoTitleBar"
                                 android:screenOrientation="portrait">
15             <intent-filter>
16                 <action android:name="android.intent.action.MAIN" />
17                 <category android:name="android.intent.category.LAUNCHER" /
                       >
18             </intent-filter></activity>
19         <activity android:name="Cam1" android:screenOrientation="portrait">
               </activity>
20         <activity android:name="Cam2" android:screenOrientation="portrait">
               </activity>
21         <activity android:name="Settings" android:screenOrientation="
               portrait"></activity>
22         <activity android:name=".CameraControl"></activity>
23         <activity android:name=".DeviceListActivity"
24                   android:label="@string/select_device"
25                   android:theme="@android:style/Theme.Dialog"
26                   android:configChanges="orientation|keyboardHidden" />
27     </application>
28 </manifest>
```

BluetoothChatService.java

```
1  /*
2   * Copyright (C) 2009 The Android Open Source Project
3   *
4   * Licensed under the Apache License, Version 2.0 (the "License");
5   * you may not use this file except in compliance with the License.
6   * You may obtain a copy of the License at
7   *
8   *      http://www.apache.org/licenses/LICENSE-2.0
9   *
10  * Unless required by applicable law or agreed to in writing, software
11  * distributed under the License is distributed on an "AS IS" BASIS,
12  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13  * See the License for the specific language governing permissions and
14  * limitations under the License.
15  */
16
```

```
17 package MasterThesis.CameraControl;
18
19 import java.io.IOException;
20 import java.io.InputStream;
21 import java.io.OutputStream;
22 import java.util.UUID;
23
24 import android.bluetooth.BluetoothAdapter;
25 import android.bluetooth.BluetoothDevice;
26 import android.bluetooth.BluetoothServerSocket;
27 import android.bluetooth.BluetoothSocket;
28 import android.content.Context;
29 import android.os.Bundle;
30 import android.os.Handler;
31 import android.os.Message;
32 import android.util.Log;
33
34 /**
35  * This class does all the work for setting up and managing Bluetooth
36  * connections with other devices. It has a thread that listens for
37  * incoming connections, a thread for connecting with a device, and a
38  * thread for performing data transmissions when connected.
39  */
40 public class BluetoothChatService {
41     // Debugging
42     private static final String TAG = "BluetoothChatService";
43     private static final boolean D = true;
44
45     // Name for the SDP record when creating server socket
46     private static final String NAME = "BluetoothChat";
47
48     // Unique UUID for this application
49     private static final UUID MY_UUID = UUID.fromString("
           00001101-0000-1000-8000-00805F9B34FB");
50
51     // Member fields
52     private final BluetoothAdapter mAdapter;
53     private final Handler mHandler;
54     private AcceptThread mAcceptThread;
55     private ConnectThread mConnectThread;
56     private ConnectedThread mConnectedThread;
57     private int mState;
58
59     // Constants that indicate the current connection state
60     public static final int STATE_NONE = 0;       // we're doing nothing
61     public static final int STATE_LISTEN = 1;     // now listening for
           incoming connections
62     public static final int STATE_CONNECTING = 2; // now initiating an
           outgoing connection
63     public static final int STATE_CONNECTED = 3;  // now connected to a
           remote device
64
65     /**
66      * Constructor. Prepares a new BluetoothChat session.
67      * @param context  The UI Activity Context
68      * @param handler  A Handler to send messages back to the UI Activity
69      */
70     public BluetoothChatService(Context context, Handler handler) {
71         mAdapter = BluetoothAdapter.getDefaultAdapter();
```

```
72          mState = STATE_NONE;
73          mHandler = handler;
74      }
75
76      /**
77       * Set the current state of the chat connection
78       * @param state  An integer defining the current connection state
79       */
80      private synchronized void setState(int state) {
81          if (D) Log.d(TAG, "setState() " + mState + " -> " + state);
82          mState = state;
83
84          // Give the new state to the Handler so the UI Activity can update
85          mHandler.obtainMessage(TabMain.MESSAGE_STATE_CHANGE, state, -1).
                sendToTarget();
86      }
87
88      /**
89       * Return the current connection state. */
90      public synchronized int getState() {
91          return mState;
92      }
93
94      /**
95       * Start the chat service. Specifically start AcceptThread to begin a
96       * session in listening (server) mode. Called by the Activity onResume
             () */
97      public synchronized void start() {
98          if (D) Log.d(TAG, "start");
99
100         // Cancel any thread attempting to make a connection
101         if (mConnectThread != null) {mConnectThread.cancel();
                mConnectThread = null;}
102
103         // Cancel any thread currently running a connection
104         if (mConnectedThread != null) {mConnectedThread.cancel();
                mConnectedThread = null;}
105
106         // Start the thread to listen on a BluetoothServerSocket
107         if (mAcceptThread == null) {
108             mAcceptThread = new AcceptThread();
109             mAcceptThread.start();
110         }
111         setState(STATE_LISTEN);
112     }
113
114     /**
115      * Start the ConnectThread to initiate a connection to a remote device.
116      * @param device  The BluetoothDevice to connect
117      */
118     public synchronized void connect(BluetoothDevice device) {
119         if (D) Log.d(TAG, "connect to: " + device);
120
121         // Cancel any thread attempting to make a connection
122         if (mState == STATE_CONNECTING) {
123             if (mConnectThread != null) {mConnectThread.cancel();
                    mConnectThread = null;}
124         }
125
```

```
126         // Cancel any thread currently running a connection
127         if (mConnectedThread != null) {mConnectedThread.cancel();
                mConnectedThread = null;}
128
129         // Start the thread to connect with the given device
130         mConnectThread = new ConnectThread(device);
131         mConnectThread.start();
132         setState(STATE_CONNECTING);
133     }
134
135     /**
136      * Start the ConnectedThread to begin managing a Bluetooth connection
137      * @param socket  The BluetoothSocket on which the connection was made
138      * @param device  The BluetoothDevice that has been connected
139      */
140     public synchronized void connected(BluetoothSocket socket,
            BluetoothDevice device) {
141         if (D) Log.d(TAG, "connected");
142
143         // Cancel the thread that completed the connection
144         if (mConnectThread != null) {mConnectThread.cancel();
                mConnectThread = null;}
145
146         // Cancel any thread currently running a connection
147         if (mConnectedThread != null) {mConnectedThread.cancel();
                mConnectedThread = null;}
148
149         // Cancel the accept thread because we only want to connect to one
                device
150         if (mAcceptThread != null) {mAcceptThread.cancel(); mAcceptThread =
                null;}
151
152         // Start the thread to manage the connection and perform
                transmissions
153         mConnectedThread = new ConnectedThread(socket);
154         mConnectedThread.start();
155
156         // Send the name of the connected device back to the UI Activity
157         Message msg = mHandler.obtainMessage(TabMain.MESSAGE_DEVICE_NAME);
158         Bundle bundle = new Bundle();
159         bundle.putString(TabMain.DEVICE_NAME, device.getName());
160         msg.setData(bundle);
161         mHandler.sendMessage(msg);
162
163         setState(STATE_CONNECTED);
164     }
165
166     /**
167      * Stop all threads
168      */
169     public synchronized void stop() {
170         if (D) Log.d(TAG, "stop");
171         if (mConnectThread != null) {mConnectThread.cancel();
                mConnectThread = null;}
172         if (mConnectedThread != null) {mConnectedThread.cancel();
                mConnectedThread = null;}
173         if (mAcceptThread != null) {mAcceptThread.cancel(); mAcceptThread =
                null;}
174         setState(STATE_NONE);
```

```
175     }
176
177     /**
178      * Write to the ConnectedThread in an unsynchronized manner
179      * @param out The bytes to write
180      * @see ConnectedThread#write(byte[])
181      */
182     public void write(byte[] out) {
183         // Create temporary object
184         ConnectedThread r;
185         // Synchronize a copy of the ConnectedThread
186         synchronized (this) {
187             if (mState != STATE_CONNECTED) return;
188             r = mConnectedThread;
189         }
190         // Perform the write unsynchronized
191         r.write(out);
192     }
193
194     /**
195      * Indicate that the connection attempt failed and notify the UI
             Activity.
196      */
197     private void connectionFailed() {
198         setState(STATE_LISTEN);
199
200         // Send a failure message back to the Activity
201         Message msg = mHandler.obtainMessage(TabMain.MESSAGE_TOAST);
202         Bundle bundle = new Bundle();
203         bundle.putString(TabMain.TOAST, "Unable to connect to Device!");
204         msg.setData(bundle);
205         mHandler.sendMessage(msg);
206     }
207
208     /**
209      * Indicate that the connection was lost and notify the UI Activity.
210      */
211     private void connectionLost() {
212         setState(STATE_LISTEN);
213
214         // Send a failure message back to the Activity
215         Message msg = mHandler.obtainMessage(TabMain.MESSAGE_TOAST);
216         Bundle bundle = new Bundle();
217         bundle.putString(TabMain.TOAST, "Connection Lost!");
218         msg.setData(bundle);
219         mHandler.sendMessage(msg);
220     }
221
222     /**
223      * This thread runs while listening for incoming connections. It
             behaves
224      * like a server-side client. It runs until a connection is accepted
225      * (or until cancelled).
226      */
227     private class AcceptThread extends Thread {
228         // The local server socket
229         private final BluetoothServerSocket mmServerSocket;
230
231         public AcceptThread() {
```

```
232             BluetoothServerSocket tmp = null;
233
234             // Create a new listening server socket
235             try {
236                 tmp = mAdapter.listenUsingRfcommWithServiceRecord(NAME,
                        MY_UUID);
237             } catch (IOException e) {
238                 Log.e(TAG, "listen() failed", e);
239             }
240             mmServerSocket = tmp;
241         }
242
243         public void run() {
244             if (D) Log.d(TAG, "BEGIN mAcceptThread" + this);
245             setName("AcceptThread");
246             BluetoothSocket socket = null;
247
248             // Listen to the server socket if we're not connected
249             while (mState != STATE_CONNECTED) {
250                 try {
251                     // This is a blocking call and will only return on a
252                     // successful connection or an exception
253                     socket = mmServerSocket.accept();
254                 } catch (IOException e) {
255                     Log.e(TAG, "accept() failed", e);
256                     break;
257                 }
258
259                 // If a connection was accepted
260                 if (socket != null) {
261                     synchronized (BluetoothChatService.this) {
262                         switch (mState) {
263                         case STATE_LISTEN:
264                         case STATE_CONNECTING:
265                             // Situation normal. Start the connected thread
                                .
266                             connected(socket, socket.getRemoteDevice());
267                             break;
268                         case STATE_NONE:
269                         case STATE_CONNECTED:
270                             // Either not ready or already connected.
                                Terminate new socket.
271                             try {
272                                 socket.close();
273                             } catch (IOException e) {
274                                 Log.e(TAG, "Could not close unwanted socket
                                    ", e);
275                             }
276                             break;
277                         }
278                     }
279                 }
280             }
281             if (D) Log.i(TAG, "END mAcceptThread");
282         }
283
284         public void cancel() {
285             if (D) Log.d(TAG, "cancel " + this);
286             try {
```

```
287                 mmServerSocket.close();
288             } catch (IOException e) {
289                 Log.e(TAG, "close() of server failed", e);
290             }
291         }
292     }
293
294
295     /**
296      * This thread runs while attempting to make an outgoing connection
297      * with a device. It runs straight through; the connection either
298      * succeeds or fails.
299      */
300     private class ConnectThread extends Thread {
301         private final BluetoothSocket mmSocket;
302         private final BluetoothDevice mmDevice;
303
304         public ConnectThread(BluetoothDevice device) {
305             mmDevice = device;
306             BluetoothSocket tmp = null;
307
308             // Get a BluetoothSocket for a connection with the
309             // given BluetoothDevice
310             try {
311                 tmp = device.createRfcommSocketToServiceRecord(MY_UUID);
312             } catch (IOException e) {
313                 Log.e(TAG, "create() failed", e);
314             }
315             mmSocket = tmp;
316         }
317
318         public void run() {
319             Log.i(TAG, "BEGIN mConnectThread");
320             setName("ConnectThread");
321
322             // Always cancel discovery because it will slow down a
                    connection
323             mAdapter.cancelDiscovery();
324
325             // Make a connection to the BluetoothSocket
326             try {
327                 // This is a blocking call and will only return on a
328                 // successful connection or an exception
329                 mmSocket.connect();
330             } catch (IOException e) {
331                 connectionFailed();
332                 // Close the socket
333                 try {
334                     mmSocket.close();
335                 } catch (IOException e2) {
336                     Log.e(TAG, "unable to close() socket during connection
                            failure", e2);
337                 }
338                 // Start the service over to restart listening mode
339                 BluetoothChatService.this.start();
340                 return;
341             }
342
343             // Reset the ConnectThread because we're done
```

```
344             synchronized (BluetoothChatService.this) {
345                 mConnectThread = null;
346             }
347 
348             // Start the connected thread
349             connected(mmSocket, mmDevice);
350         }
351 
352         public void cancel() {
353             try {
354                 mmSocket.close();
355             } catch (IOException e) {
356                 Log.e(TAG, "close() of connect socket failed", e);
357             }
358         }
359     }
360 
361     /**
362      * This thread runs during a connection with a remote device.
363      * It handles all incoming and outgoing transmissions.
364      */
365     private class ConnectedThread extends Thread {
366         private final BluetoothSocket mmSocket;
367         private final InputStream mmInStream;
368         private final OutputStream mmOutStream;
369 
370         public ConnectedThread(BluetoothSocket socket) {
371             Log.d(TAG, "create ConnectedThread");
372             mmSocket = socket;
373             InputStream tmpIn = null;
374             OutputStream tmpOut = null;
375 
376             // Get the BluetoothSocket input and output streams
377             try {
378                 tmpIn = socket.getInputStream();
379                 tmpOut = socket.getOutputStream();
380             } catch (IOException e) {
381                 Log.e(TAG, "temp sockets not created", e);
382             }
383 
384             mmInStream = tmpIn;
385             mmOutStream = tmpOut;
386         }
387 
388         public void run() {
389             Log.i(TAG, "BEGIN mConnectedThread");
390             byte[] buffer = new byte[1024];
391             int bytes;
392 
393             // Keep listening to the InputStream while connected
394             while (true) {
395                 try {
396                     // Read from the InputStream
397                     bytes = mmInStream.read(buffer);
398 
399                     // Send the obtained bytes to the UI Activity
400                     mHandler.obtainMessage(TabMain.MESSAGE_READ, bytes, -1,
                            buffer)
401                             .sendToTarget();
```

```
402                 } catch (IOException e) {
403                     Log.e(TAG, "disconnected", e);
404                     connectionLost();
405                     break;
406                 }
407             }
408         }
409 
410         /**
411          * Write to the connected OutStream.
412          * @param buffer  The bytes to write
413          */
414         public void write(byte[] buffer) {
415             try {
416                 mmOutStream.write(buffer);
417 
418                 // Share the sent message back to the UI Activity
419                 mHandler.obtainMessage(TabMain.MESSAGE_WRITE, -1, -1,
                        buffer)
420                         .sendToTarget();
421             } catch (IOException e) {
422                 Log.e(TAG, "Exception during write", e);
423             }
424         }
425 
426         public void cancel() {
427             try {
428                 mmSocket.close();
429             } catch (IOException e) {
430                 Log.e(TAG, "close() of connect socket failed", e);
431             }
432         }
433     }
434 }
```

CameraControl.java

```
1  package MasterThesis.CameraControl;
2 
3 
4 
5  import java.util.LinkedList;
6  import java.util.Set;
7 
8  import android.app.Activity;
9  import android.bluetooth.BluetoothAdapter;
10 import android.bluetooth.BluetoothDevice;
11 import android.content.BroadcastReceiver;
12 import android.content.Context;
13 import android.content.Intent;
14 import android.content.IntentFilter;
15 import android.os.Bundle;
16 import android.os.Handler;
17 import android.os.Message;
18 import android.text.format.Time;
19 import android.util.Log;
20 import android.view.View;
21 import android.widget.ArrayAdapter;
22 import android.widget.ListView;
```

```
23 import android.widget.Spinner;
24 import android.widget.Toast;
25
26 public class CameraControl extends Activity {
27          LinkedList<String> li;
28          ArrayAdapter<String>  adapterList;
29          static final int REQUEST_ENABLE_BT = 1001;
30
31     //Broadcast Message Strings
32     public static final String BROADCAST_SEND_MESSAGE = "
           BROADCAST_SEND_MESSAGE";
33
34     //Broadcast Extra Strings
35     public static final String BROADCAST_EXTRA_SEND_MESSAGE = "SEND_MESSAGE
           ";
36
37     //Broadcast Extra Strings
38     public static final String BROADCAST_EXTRA_SEND_MESSAGE_CODE = "
           SEND_MESSAGE_CODE";
39     public static final String BROADCAST_EXTRA_SEND_MESSAGE_COUNT_PARAMETER
           = "SEND_MESSAGE_COUNT_PARAMETER";
40     public static final String BROADCAST_EXTRA_SEND_MESSAGE_PARAMETER = "
           SEND_MESSAGE_PARAMETER";
41
42          private int cameranumber;
43
44          //Warten auf Nachrichten vom TabMain
45          private int OpenCode=0;
46          private char OpenState=0;
47
48
49          //TVValues aus der EDSDK
50          public static char[] TVValues = {0x0C/*Bulb*/, 0x10/*30"*/, 0x13/*
               25"*/, 0x14/*20"*/, 0x15/*20"(1/3)*/, 0x18/*15"*/, 0x1B/*13"*/,
               0x1C/*10"*/, 0x1D/*10"(1/3)*/, 0x20/*8"*/, 0x23/*6"(1/3)*/, 0x24
               /*6"*/, 0x25/*5"*/, 0x28/*4"*/, 0x2B/*3"2*/, 0x2C/*3"*/, 0x2D/*
               2"5*/, 0x30/*2"*/, 0x33/*1"6*/, 0x34/*1"5*/, 0x35/*1"3*/, 0x38/*
               1"*/, 0x3B/*0"8*/, 0x3C/*0"7*/, 0x3D/*0"6*/, 0x40/*0"5*/, 0x43/*
               0"4*/, 0x44/*0"3*/, 0x45/*0"3(1/3)*/, 0x48/*1/4*/, 0x4B/*1/5*/,
               0x4C/*1/6*/, 0x4D/*1/6(1/3)*/, 0x50/*1/8*/, 0x53/*1/10(1/3)*/, 0
               x54/*1/10*/, 0x55/*1/13*/, 0x58/*1/15*/, 0x5B/*1/20(1/3)*/, 0x5C
               /*1/20*/, 0x5D/*1/25*/, 0x60/*1/30*/, 0x63/*1/40*/, 0x64/*1/45*/
               , 0x65/*1/50*/, 0x68/*1/60*/, 0x6B/*1/80*/, 0x6C/*1/90*/, 0x6D/*
               1/100*/, 0x70/*1/125*/, 0x73/*1/160*/, 0x74/*1/180*/, 0x75/*
               1/200*/, 0x78/*1/250*/, 0x7B/*1/320*/, 0x7C/*1/350*/, 0x7D/*
               1/400*/, 0x80/*1/500*/, 0x83/*1/640*/, 0x84/*1/750*/, 0x85/*
               1/800*/, 0x88/*1/1000*/, 0x8B/*1/1250*/, 0x8C/*1/1500*/, 0x8D/*
               1/1600*/, 0x90/*1/2000*/, 0x93/*1/2500*/, 0x94/*1/3000*/, 0x95/*
               1/3200*/, 0x98/*1/4000*/, 0x9B/*1/5000*/, 0x9C/*1/6000*/, 0x9D/*
               1/6400*/, 0xA0/*1/8000*/};
51
52          //AVValues aus der EDSK
53          public static char[] AVValues = { 0x08/*1*/, 0x0B/*1.1*/, 0x0C/*1.2
               */, 0x0D/*1.2(1/3)*/, 0x10/*1.4*/, 0x13/*1.6*/, 0x14/*1.8*/, 0
               x15/*1.8(1/3)*/, 0x18/*2*/, 0x1B/*2.2*/, 0x1C/*2.5*/, 0x1D/*
               2.5(1/3)*/, 0x20/*2.8*/, 0x23/*3.2*/, 0x24/*3.5*/, 0x25/*
               3.5(1/3)*/, 0x28/*4*/, 0x2B/*4.5*/, 0x2C/*4.5*/, 0x2D/*5.0*/, 0
               x30/*5.6*/, 0x33/*6.3*/, 0x34/*6.7*/, 0x35/*7.1*/, 0x38/*8*/, 0
               x3B/*9*/, 0x3C/*9.5*/, 0x3D/*10*/, 0x40/*11*/, 0x43/*13(1/3)*/,
```

```
        0x44/*13*/, 0x45/*14*/, 0x48/*16*/, 0x4B/*18*/, 0x4C/*19*/, 0x4D
        /*20*/, 0x50/*22*/, 0x53/*25*/, 0x54/*27*/, 0x55/*29*/, 0x58/*32
        */, 0x5B/*36*/, 0x5C/*38*/, 0x5D/*40*/, 0x60/*45*/, 0x63/*51*/,
        0x64/*54*/, 0x65/*57*/, 0x68/*64*/, 0x6B/*72*/, 0x6C/*76*/, 0x6D
        /*80*/, 0x70/*91*/};
54
55      //Belichtungswerte
56      public static char[] BelValues = { 0x18 /*  +3  */,  0x15 /*  +2
        2/3  */,  0x14 /*  +2 1/2  */,  0x13 /*  +2 1/3  */,  0x10 /*
        +2  */,  0x0D  /* +1 2/3  */,  0x0C  /* +1 1/2  */,  0x0B /*  +1
         1/3  */,  0x08 /*  +1  */,  0x05 /*  +2/3  */,  0x04 /*  +1/2
        */,  0x03 /*  +1/3  */,  0x00 /*  0  */,  0xFD  /* -1/3  */,  0
        xFC  /* -1/2  */,  0xFB  /* -2/3  */,  0xF8  /* -1  */,  0xF5
        /* -1 1/3  */,  0xF4  /* -1 1/2  */,  0xF3  /* -1 2/3  */,  0xF0
          /* -2  */,  0xED  /* -2 1/3  */,  0xEC  /* -2 1/2  */,  0xEB
        /* -2 2/3  */,  0xE8  /* -3  */};
57
58      //Focuswerte
59      public static char[] FocusValues = { 0x01 /*Near 1*/, 0x02 /*Near 2
        */, 0x03 /*Near 3*/, 0x8001 /*Far 1*/, 0x8002 /*Far 2*/, 0x8003
        /*Far 3*/};
60
61      //Fills the Parameter Array
62      //Count is the amount of parameters in bytes
63      //MaxCount = 2 (actual
64      public void FillsParameterList(byte [] array, char count, char
            Parameter1)
65      {
66                      switch(count)
67                      {
68
69                              case 2:
70                                      array[1] = (byte) ( Parameter1 >>
                                            8);
71                              case 1:
72                                      array[0] = (byte) Parameter1;
73                              default:
74                                      return;
75                      }
76      }
77
78      //Funktion überprüft das Codewort in der Response Phase
79      //0 alles passt
80      //1 Fehler
81      private char CheckAnswer(int code, Context context)
82      {
83              String ValueReturn;
84      Time time = new Time();
85
86      ValueReturn="";
87      time.setToNow();
88
89              switch(code & 0xFFFFFF)
90              {
91              case Commands.RESPONSE_OK:
92                      li.addFirst(time.format("%d.%m.%Y␣%H:%M:%S") + "␣"
                            + "RESPONSE_OK");
93                      adapterList.notifyDataSetChanged();
94                      return 0;
```

```
95                  case Commands.RESPONSE_WRONG_PARAMETER:
96                          ValueReturn ="RESPONSE_WRONG_PARAMETER";
97                          break;
98                  case Commands.RESPONSE_NOT_SUPPORTED_CLASS:
99                          ValueReturn ="RESPONSE_NOT_SUPPORTED_CLASS";
100                         break;
101                 case Commands.RESPONSE_NOT_SUPPORTED_ADCLASS:
102                         ValueReturn ="RESPONSE_NOT_SUPPORTED_ADCLASS";
103                         break;
104                 case Commands.RESPONSE_NOT_SUPPORTED_PARAMETERCOUNT:
105                         ValueReturn ="RESPONSE_NOT_SUPPORTED_PARAMETERCOUNT
                                "";
106                         break;
107                 case Commands.RESPONSE_NOT_SUPPORTED:
108                         ValueReturn ="RESPONSE_NOT_SUPPORTED";
109                         break;
110                 case Commands.RESPONSE_NOT_SUPPORTED_PHASE:
111                         ValueReturn ="RESPONSE_NOT_SUPPORTED_PHASE";
112                         break;
113                 case Commands.RESPONSE_ERROR:
114                         ValueReturn ="RESPONSE_ERROR";
115                         break;
116                 case Commands.RESPONSE_PTP_ERROR:
117                         ValueReturn ="RESPONSE_PTP_ERROR";
118                         break;
119                 case Commands.RESPONSE_DATA:
120                         ValueReturn ="RESPONSE_PTP_ERROR";
121                         break;
122                 }
123                 li.addFirst(time.format("%d.%m.%Y_%H:%M:%S") + "_" +
                        ValueReturn);
124                 adapterList.notifyDataSetChanged();
125                 return 1;
126         }
127
128         private final BroadcastReceiver mReceiver = new BroadcastReceiver()
                {
129             public void onReceive(Context context, Intent intent) {
130                 String action = intent.getAction();
131
132                 if(OpenState != 0 && action == TabMain.
                        BROADCAST_RECEIVED_CONTAINER)
133                 {
134                         //Werte auslesen
135                         int code;
136                         char parametercount;
137                         char TransactionIDReturn;
138                         code= intent.getIntExtra(TabMain.
                            BROADCAST_RECEIVED_CONTAINER_CODE, 0);
139
140                         parametercount = intent.getCharExtra(TabMain.
                            BROADCAST_RECEIVED_CONTAINER_NUMPARAMETERS, (
                            char) 0);
141                         TransactionIDReturn = intent.getCharExtra(TabMain.
                            BROADCAST_RECEIVED_CONTAINER_TRANSACTIONID, (
                            char) 0);
142                         byte[] parameter = intent.getByteArrayExtra(TabMain
                            .BROADCAST_RECEIVED_CONTAINER_PARAMETER);
143
```

```
144                     CheckAnswer(code,context);
145
146                     switch(OpenCode)
147                     {
148                     case Commands.COMMAND_PTP_SESSION_OPEN:
149
150                                     //Status
151                             switch(OpenState)
152                             {
153                             case 1:
154                                     OpenState = 0;
155                                     OpenCode = 0;
156                                     break;
157                             }
158
159                             //Werte auslesen
160                             break;
161
162                     case Commands.COMMAND_PTP_SESSION_CLOSE:
163
164                             //Status
165                     switch(OpenState)
166                     {
167                     case 1:
168                                     OpenState = 0;
169                                     OpenCode = 0;
170                                     break;
171                     }
172
173                     //Werte auslesen
174                     break;
175
176                     case Commands.COMMAND_PTP_SHOOT:
177
178                                     //Status
179                             switch(OpenState)
180                             {
181                             case 1:
182                                             OpenState = 0;
183                                             OpenCode = 0;
184                                             break;
185                             }
186
187                     //Werte auslesen
188                     break;
189
190                     case Commands.COMMAND_PTP_CHANGE_TV:
191
192                             //Status
193                     switch(OpenState)
194                     {
195                     case 1:
196                                     OpenState = 0;
197                                     OpenCode = 0;
198                                     break;
199                     }
200
201                             //Werte auslesen
202                             break;
```

```
203
204                         case Commands.COMMAND_PTP_CHANGE_BELICHTUNG:
205
206                                 //Status
207                         switch(OpenState)
208                         {
209                         case 1:
210                                         OpenState = 0;
211                                         OpenCode = 0;
212                                         break;
213                         }
214
215                                 //Werte auslesen
216                                 break;
217
218                         case Commands.COMMAND_PTP_CHANGE_AV:
219
220                                 //Status
221                         switch(OpenState)
222                         {
223                         case 1:
224                                         OpenState = 0;
225                                         OpenCode = 0;
226                                         break;
227                         }
228
229                                 //Werte auslesen
230                                 break;
231
232                         case Commands.COMMAND_PTP_FOKUS:
233
234                                 //Status
235                         switch(OpenState)
236                         {
237                         case 1:
238                                         OpenState = 0;
239                                         OpenCode = 0;
240                                         break;
241                         }
242
243                                 //Werte auslesen
244                                 break;
245
246                         case Commands.COMMAND_PTP_LIVE_VIEW_ON:
247
248                                 //Status
249                         switch(OpenState)
250                         {
251                         case 1:
252                                         OpenState = 0;
253                                         OpenCode = 0;
254                                         break;
255                         }
256
257                                 //Werte auslesen
258                                 break;
259
260                         case Commands.COMMAND_PTP_LIVE_VIEW_OFF:
261
```

```
262                             //Status
263                         switch(OpenState)
264                         {
265                         case 1:
266                                         OpenState = 0;
267                                         OpenCode = 0;
268                                         break;
269                         }
270
271                                 //Werte auslesen
272                                 break;
273
274                         }
275                 }
276
277             }
278         };
279
280     /** Called when the activity is first created. */
281     @Override
282     public void onCreate(Bundle savedInstanceState) {
283         super.onCreate(savedInstanceState);
284         setContentView(R.layout.main);
285
286         cameranumber = 0;
287
288         IntentFilter filter = new IntentFilter(TabMain.
                BROADCAST_RECEIVED_CONTAINER);
289         registerReceiver(mReceiver, filter); // Don't forget to unregister
                during onDestroy
290
291
292         Spinner spinner = (Spinner) findViewById(R.id.spinnerTV);
293         ArrayAdapter<CharSequence> adapter = ArrayAdapter.
                createFromResource(
294                 this, R.array.TV_SpinnerArray, android.R.layout.
                        simple_spinner_item);
295         adapter.setDropDownViewResource(android.R.layout.
                simple_spinner_dropdown_item);
296         spinner.setAdapter(adapter);
297
298         spinner = (Spinner) findViewById(R.id.spinnerAV);
299         adapter = ArrayAdapter.createFromResource(
300                 this, R.array.AV_SpinnerArray, android.R.layout.
                        simple_spinner_item);
301         adapter.setDropDownViewResource(android.R.layout.
                simple_spinner_dropdown_item);
302         spinner.setAdapter(adapter);
303
304         spinner = (Spinner) findViewById(R.id.spinnerBel);
305         adapter = ArrayAdapter.createFromResource(
306                 this, R.array.Bel_SpinnerArray, android.R.layout.
                        simple_spinner_item);
307         adapter.setDropDownViewResource(android.R.layout.
                simple_spinner_dropdown_item);
308         spinner.setAdapter(adapter);
309
310         spinner = (Spinner) findViewById(R.id.spinnerFocus);
311         adapter = ArrayAdapter.createFromResource(
```

```
312                 this, R.array.Focus_SpinnerArray, android.R.layout.
                        simple_spinner_item);
313         adapter.setDropDownViewResource(android.R.layout.
                simple_spinner_dropdown_item);
314         spinner.setAdapter(adapter);
315
316
317         ListView list = (ListView)findViewById(R.id.listViewAusgabe);
318
319         li = new LinkedList<String>();
320         adapterList = new ArrayAdapter<String>(this,android.R.layout.
                simple_list_item_1 , li);
321         list.setAdapter(adapterList);
322
323         Intent intent = getIntent();
324         cameranumber = intent.getIntExtra("Cam", 0);
325
326     }
327
328
329     @Override
330     public void onDestroy() {
331         super.onDestroy();
332
333         unregisterReceiver(mReceiver);
334     }
335
336     public void PressButton(View view)
337     {
338         Context context = getApplicationContext();
339         Spinner spinner;
340         Toast toast;
341         Intent intent;
342
343         //Code der übertragen werden solll
344         int code;
345
346         //Anzahl der Parameter
347         char parametercount;
348
349         //Array mit den Parametern
350         byte[] parameter = new byte[20];
351                 int position;
352
353         Time time = new Time();
354
355         time.setToNow();
356
357         switch(view.getId())
358         {
359         case R.id.ButtonTVValue:
360                 spinner = (Spinner) findViewById(R.id.spinnerTV);
361                 li.addFirst(time.format("%d.%m.%Y␣%H:%M:%S") + "␣" +
                        getString(R.string.TV_Value));
362
363                 position = spinner.getSelectedItemPosition();
364                 code = 0x22050000 | ((byte)cameranumber-1 << 8) | (byte)
                        Commands.COMMAND_PTP_CHANGE_TV ; // DeviceID Klasse AD
                        Klasse Code
```

```
365                 parametercount = 1;
366
367                 //Parameterliste erzeugen
368                 FillsParameterList(parameter,(char) parametercount,
                        TVValues[position]);
369
370                 //Intent auslösen
371                 //Daten zuvor bereitstellen
372                 intent =new Intent();
373             intent.setAction(CameraControl.BROADCAST_SEND_MESSAGE);
374             intent.putExtra(CameraControl.BROADCAST_EXTRA_SEND_MESSAGE_CODE
                    ,code);
375             intent.putExtra(CameraControl.
                    BROADCAST_EXTRA_SEND_MESSAGE_COUNT_PARAMETER,parametercount)
                    ;
376             intent.putExtra(CameraControl.
                    BROADCAST_EXTRA_SEND_MESSAGE_PARAMETER,parameter);
377             sendBroadcast(intent);
378
379                 //Warten auf Rückgabe setzen
380                 OpenCode = Commands.COMMAND_PTP_CHANGE_TV;
381                 OpenState=1;
382                 break;
383         case R.id.ButtonAVValue:
384                 li.addFirst(time.format("%d.%m.%Y␣%H:%M:%S") + "␣" +
                        getString(R.string.AV_Value));
385                 spinner = (Spinner) findViewById(R.id.spinnerAV);
386
387                 position = spinner.getSelectedItemPosition();
388                 code = 0x22050000 | ((byte)cameranumber-1 << 8) | (byte)
                        Commands.COMMAND_PTP_CHANGE_AV ; // DeviceID Klasse AD
                        Klasse Code
389                 parametercount = 1;
390
391                 //Parameterliste erzeugen
392                 FillsParameterList(parameter,(char) parametercount,
                        AVValues[position]);
393
394                 //Intent auslösen
395                 //Daten zuvor bereitstellen
396                 intent =new Intent();
397             intent.setAction(CameraControl.BROADCAST_SEND_MESSAGE);
398             intent.putExtra(CameraControl.BROADCAST_EXTRA_SEND_MESSAGE_CODE
                    ,code);
399             intent.putExtra(CameraControl.
                    BROADCAST_EXTRA_SEND_MESSAGE_COUNT_PARAMETER,parametercount)
                    ;
400             intent.putExtra(CameraControl.
                    BROADCAST_EXTRA_SEND_MESSAGE_PARAMETER,parameter);
401             sendBroadcast(intent);
402
403                 //Warten auf Rückgabe setzen
404                 OpenCode = Commands.COMMAND_PTP_CHANGE_AV ;
405                 OpenState=1;
406
407                 break;
408         case R.id.ButtonBelValue:
409                 li.addFirst(time.format("%d.%m.%Y␣%H:%M:%S") + "␣" +
                        getString(R.string.Bel_Value));
```

```
410                 spinner = (Spinner) findViewById(R.id.spinnerBel);
411
412
413                 position = spinner.getSelectedItemPosition();
414                 code = 0x22050000 | ((byte)cameranumber-1 << 8) | (byte)
                        Commands.COMMAND_PTP_CHANGE_BELICHTUNG ; // DeviceID
                        Klasse AD Klasse Code
415                 parametercount = 1;
416
417                 //Parameterliste erzeugen
418                 FillsParameterList(parameter,(char) parametercount,
                        BelValues[position]);
419
420                 //Intent auslösen
421                 //Daten zuvor bereitstellen
422                 intent =new Intent();
423             intent.setAction(CameraControl.BROADCAST_SEND_MESSAGE);
424             intent.putExtra(CameraControl.BROADCAST_EXTRA_SEND_MESSAGE_CODE
                    ,code);
425             intent.putExtra(CameraControl.
                    BROADCAST_EXTRA_SEND_MESSAGE_COUNT_PARAMETER,parametercount)
                    ;
426             intent.putExtra(CameraControl.
                    BROADCAST_EXTRA_SEND_MESSAGE_PARAMETER,parameter);
427             sendBroadcast(intent);
428
429                 //Warten auf Rückgabe setzen
430                 OpenCode = Commands.COMMAND_PTP_CHANGE_BELICHTUNG ;
431                 OpenState=1;
432
433                 break;
434         case R.id.ButtonSessionOpen:
435                 li.addFirst(time.format("%d.%m.%Y␣%H:%M:%S") + "␣" +
                        getString(R.string.SessionOpen_Value));
436                 code = 0x22050000 | ((byte)cameranumber-1 << 8) | (byte)
                        Commands.COMMAND_PTP_SESSION_OPEN ; // DeviceID Klasse
                        AD Klasse Code
437                 parametercount = 0;
438
439                 //Parameterliste erzeugen
440                 FillsParameterList(parameter,(char) parametercount, (char)
                        0);
441
442                 //Intent auslösen
443                 //Daten zuvor bereitstellen
444                 intent =new Intent();
445             intent.setAction(CameraControl.BROADCAST_SEND_MESSAGE);
446             intent.putExtra(CameraControl.BROADCAST_EXTRA_SEND_MESSAGE_CODE
                    ,code);
447             intent.putExtra(CameraControl.
                    BROADCAST_EXTRA_SEND_MESSAGE_COUNT_PARAMETER,parametercount)
                    ;
448             intent.putExtra(CameraControl.
                    BROADCAST_EXTRA_SEND_MESSAGE_PARAMETER,parameter);
449             sendBroadcast(intent);
450
451                 //Warten auf Rückgabe setzen
452                 OpenCode = Commands.COMMAND_PTP_SESSION_OPEN ;
453                 OpenState=1;
```

```
454
455                 break;
456         case R.id.ButtonSessionClose:
457                 li.addFirst(time.format("%d.%m.%Y %H:%M:%S") + " " +
                        getString(R.string.SessionClose_Value));
458                 code = 0x22050000 | ((byte)cameranumber-1 << 8) | (byte)
                        Commands.COMMAND_PTP_SESSION_CLOSE; // DeviceID Klasse
                        AD Klasse Code
459                 parametercount = 0;
460
461                 //Parameterliste erzeugen
462                 FillsParameterList(parameter,(char) parametercount, (char)
                        0);
463
464                 //Intent auslösen
465                 //Daten zuvor bereitstellen
466                 intent =new Intent();
467             intent.setAction(CameraControl.BROADCAST_SEND_MESSAGE);
468             intent.putExtra(CameraControl.BROADCAST_EXTRA_SEND_MESSAGE_CODE
                    ,code);
469             intent.putExtra(CameraControl.
                    BROADCAST_EXTRA_SEND_MESSAGE_COUNT_PARAMETER,parametercount)
                    ;
470             intent.putExtra(CameraControl.
                    BROADCAST_EXTRA_SEND_MESSAGE_PARAMETER,parameter);
471             sendBroadcast(intent);
472
473                 //Warten auf Rückgabe setzen
474                 OpenCode = Commands.COMMAND_PTP_SESSION_CLOSE ;
475                 OpenState=1;
476
477                 break;
478         case R.id.TakeFoto:
479                 li.addFirst(time.format("%d.%m.%Y %H:%M:%S") + " " +
                        getString(R.string.TakeFoto_Value));
480                 code = 0x22050000 | ((byte)cameranumber-1 << 8) | (byte)
                        Commands.COMMAND_PTP_SHOOT ; // DeviceID Klasse AD
                        Klasse Code
481                 parametercount = 0;
482
483                 //Parameterliste erzeugen
484                 FillsParameterList(parameter,(char) parametercount, (char)
                        0);
485
486                 //Intent auslösen
487                 //Daten zuvor bereitstellen
488                 intent =new Intent();
489             intent.setAction(CameraControl.BROADCAST_SEND_MESSAGE);
490             intent.putExtra(CameraControl.BROADCAST_EXTRA_SEND_MESSAGE_CODE
                    ,code);
491             intent.putExtra(CameraControl.
                    BROADCAST_EXTRA_SEND_MESSAGE_COUNT_PARAMETER,parametercount)
                    ;
492             intent.putExtra(CameraControl.
                    BROADCAST_EXTRA_SEND_MESSAGE_PARAMETER,parameter);
493             sendBroadcast(intent);
494
495                 //Warten auf Rückgabe setzen
496                 OpenCode = Commands.COMMAND_PTP_SHOOT ;
```

```
497                 OpenState=1;
498
499                 break;
500             case R.id.LiveView_On:
501                 li.addFirst(time.format("%d.%m.%Y %H:%M:%S") + " " +
                        getString(R.string.LiveViewOn_Value));
502                 code = 0x22050000 | ((byte)cameranumber-1 << 8) | (byte)
                        Commands.COMMAND_PTP_LIVE_VIEW_ON ; // DeviceID Klasse
                        AD Klasse Code
503                 parametercount = 0;
504
505                 //Parameterliste erzeugen
506                 FillsParameterList(parameter,(char) parametercount, (char)
                        0);
507
508                 //Intent auslösen
509                 //Daten zuvor bereitstellen
510                 intent =new Intent();
511             intent.setAction(CameraControl.BROADCAST_SEND_MESSAGE);
512             intent.putExtra(CameraControl.BROADCAST_EXTRA_SEND_MESSAGE_CODE
                    ,code);
513             intent.putExtra(CameraControl.
                    BROADCAST_EXTRA_SEND_MESSAGE_COUNT_PARAMETER,parametercount)
                    ;
514             intent.putExtra(CameraControl.
                    BROADCAST_EXTRA_SEND_MESSAGE_PARAMETER,parameter);
515             sendBroadcast(intent);
516
517                 //Warten auf Rückgabe setzen
518                 OpenCode = Commands.COMMAND_PTP_LIVE_VIEW_ON ;
519                 OpenState=1;
520                 break;
521             case R.id.LiveView_Off:
522                 li.addFirst(time.format("%d.%m.%Y %H:%M:%S") + " " +
                        getString(R.string.LiveViewOff_Value));
523
524                 code = 0x22050000 | ((byte)cameranumber-1 << 8) | (byte)
                        Commands.COMMAND_PTP_LIVE_VIEW_OFF ; // DeviceID Klasse
                        AD Klasse Code
525                 parametercount = 0;
526
527                 //Parameterliste erzeugen
528                 FillsParameterList(parameter,(char) parametercount, (char)
                        0);
529
530                 //Intent auslösen
531                 //Daten zuvor bereitstellen
532                 intent =new Intent();
533             intent.setAction(CameraControl.BROADCAST_SEND_MESSAGE);
534             intent.putExtra(CameraControl.BROADCAST_EXTRA_SEND_MESSAGE_CODE
                    ,code);
535             intent.putExtra(CameraControl.
                    BROADCAST_EXTRA_SEND_MESSAGE_COUNT_PARAMETER,parametercount)
                    ;
536             intent.putExtra(CameraControl.
                    BROADCAST_EXTRA_SEND_MESSAGE_PARAMETER,parameter);
537             sendBroadcast(intent);
538
539                 //Warten auf Rückgabe setzen
```

```
540                 OpenCode = Commands.COMMAND_PTP_LIVE_VIEW_OFF ;
541                 OpenState=1;
542
543                 break;
544         case R.id.ButtonFocusValue:
545                 li.addFirst(time.format("%d.%m.%Y %H:%M:%S") + " " +
                        getString(R.string.Focus_Value));
546
547                 spinner = (Spinner) findViewById(R.id.spinnerFocus);
548
549                 position = spinner.getSelectedItemPosition();
550                 code = 0x22050000 | ((byte)cameranumber-1 << 8) | (byte)
                        Commands.COMMAND_PTP_FOKUS; // DeviceID Klasse AD Klasse
                         Code
551                 parametercount = 2;
552
553                 //Parameterliste erzeugen
554                 FillsParameterList(parameter,(char) parametercount,
                        FocusValues[position]);
555
556                 //Intent auslösen
557                 //Daten zuvor bereitstellen
558                 intent =new Intent();
559             intent.setAction(CameraControl.BROADCAST_SEND_MESSAGE);
560             intent.putExtra(CameraControl.BROADCAST_EXTRA_SEND_MESSAGE_CODE
                    ,code);
561             intent.putExtra(CameraControl.
                    BROADCAST_EXTRA_SEND_MESSAGE_COUNT_PARAMETER,parametercount)
                    ;
562             intent.putExtra(CameraControl.
                    BROADCAST_EXTRA_SEND_MESSAGE_PARAMETER,parameter);
563             sendBroadcast(intent);
564
565                 //Warten auf Rückgabe setzen
566                 OpenCode = Commands.COMMAND_PTP_FOKUS ;
567                 OpenState=1;
568
569                 break;
570         default:
571                 li.addFirst(time.format("%d.%m.%Y %H:%M:%S") + " " +
                        getString(R.string.unknown_Button));
572                 break;
573         }
574
575         adapterList.notifyDataSetChanged();
576     }
577
578 }
```

Commands.java

```
1  package MasterThesis.CameraControl;
2
3  public class Commands {
4
5          //Codes für das Protokoll
6          //Rückgabe der Funktion Check Command
7          //Klasse V
8          public final static char COMMAND_PTP_SESSION_OPEN        = 0x0001;
```

```
9           public final static char COMMAND_PTP_SESSION_CLOSE = 0x0002;
10          public final static char COMMAND_PTP_SHOOT = 0x0003;
11          public final static char COMMAND_PTP_CHANGE_TV = 0x0004;
12          public final static char COMMAND_PTP_CHANGE_AV = 0x0005;
13          public final static char COMMAND_PTP_CHANGE_BELICHTUNG = 0x0006;
14          public final static char COMMAND_PTP_LIVE_VIEW_ON = 0x0007;
15          public final static char COMMAND_PTP_FOKUS = 0x0008;
16          public final static char COMMAND_PTP_GETCONNECTION = 0x0009;
17          public final static char COMMAND_PTP_LIVE_VIEW_OFF = 0x000A;
18
19          //Rückgabe der Funktion Check Command
20          //Klasse I
21          public final static char COMMAND_PIN_READ = 0x0001;
22          public final static char COMMAND_PIN_SET = 0x0002;
23          public final static char COMMAND_PIN_STORE = 0x0003;
24          public final static char COMMAND_PIN_STATE = 0x0004;
25          public final static char COMMAND_GETCLASS = 0x0005;
26          public final static char COMMAND_SETOUTPUT = 0x0006;
27
28          //Rückgabe der Funktion Check Command
29          public final static char RESPONSE_OK = 0x0000;
30          public final static char RESPONSE_WRONG_PARAMETER = 0x0001;
31          public final static char RESPONSE_NOT_SUPPORTED_CLASS = 0x0002;
32          public final static char RESPONSE_NOT_SUPPORTED_ADCLASS = 0x0003;
33          public final static char RESPONSE_NOT_SUPPORTED_PARAMETERCOUNT = 0
                x0004;
34          public final static char RESPONSE_NOT_SUPPORTED = 0x0005;
35          public final static char RESPONSE_NOT_SUPPORTED_PHASE   = 0x0006;
36          public final static char RESPONSE_ERROR = 0x0007;
37          public final static char RESPONSE_PTP_ERROR = 0x0008;
38          public final static char RESPONSE_DATA = 0x0009;
39
40          //Klassen interne Definitionen
41          public final static char INTERNAL_COMMAND_TIMELAPS = 0x1000;
42          public final static char INTERNAL_COMMAND_3D = 0x1001;
43          public final static char INTERNAL_COMMAND_3DHDR = 0x1002;
44
45  }
```

DeviceListActivity.java

```
1   /*
2    * Copyright (C) 2009 The Android Open Source Project
3    *
4    * Licensed under the Apache License, Version 2.0 (the "License");
5    * you may not use this file except in compliance with the License.
6    * You may obtain a copy of the License at
7    *
8    *      http://www.apache.org/licenses/LICENSE-2.0
9    *
10   * Unless required by applicable law or agreed to in writing, software
11   * distributed under the License is distributed on an "AS IS" BASIS,
12   * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13   * See the License for the specific language governing permissions and
14   * limitations under the License.
15   */
16
17  package MasterThesis.CameraControl;
18
```

```
19 import java.util.Set;
20
21 import android.app.Activity;
22 import android.bluetooth.BluetoothAdapter;
23 import android.bluetooth.BluetoothDevice;
24 import android.content.BroadcastReceiver;
25 import android.content.Context;
26 import android.content.Intent;
27 import android.content.IntentFilter;
28 import android.os.Bundle;
29 import android.util.Log;
30 import android.view.View;
31 import android.view.Window;
32 import android.view.View.OnClickListener;
33 import android.widget.AdapterView;
34 import android.widget.ArrayAdapter;
35 import android.widget.Button;
36 import android.widget.ListView;
37 import android.widget.TextView;
38 import android.widget.Toast;
39 import android.widget.AdapterView.OnItemClickListener;
40
41 /**
42  * This Activity appears as a dialog. It lists any paired devices and
43  * devices detected in the area after discovery. When a device is chosen
44  * by the user, the MAC address of the device is sent back to the parent
45  * Activity in the result Intent.
46  */
47 public class DeviceListActivity extends Activity {
48     // Debugging
49     private static final String TAG = "DeviceListActivity";
50     private static final boolean D = true;
51         private static final int REQUEST_ENABLE_BT = 2121;
52
53     // Return Intent extra
54     public static String EXTRA_DEVICE_ADDRESS = "device_address";
55
56     // Member fields
57     private BluetoothAdapter mBtAdapter;
58     private ArrayAdapter<String> mPairedDevicesArrayAdapter;
59     private ArrayAdapter<String> mNewDevicesArrayAdapter;
60
61     @Override
62     protected void onCreate(Bundle savedInstanceState) {
63         Context context = getApplicationContext();
64         Toast toast;
65         super.onCreate(savedInstanceState);
66
67         // Setup the window
68         requestWindowFeature(Window.FEATURE_INDETERMINATE_PROGRESS);
69         setContentView(R.layout.device_list);
70
71         // Set result CANCELED incase the user backs out
72         setResult(Activity.RESULT_CANCELED);
73
74         // Initialize the button to perform device discovery
75         Button scanButton = (Button) findViewById(R.id.button_scan);
76         scanButton.setOnClickListener(new OnClickListener() {
77             public void onClick(View v) {
```

```
78                  doDiscovery();
79                  v.setVisibility(View.GONE);
80              }
81          });
82
83          // Initialize array adapters. One for already paired devices and
84          // one for newly discovered devices
85          mPairedDevicesArrayAdapter = new ArrayAdapter<String>(this, R.
                layout.device_name);
86          mNewDevicesArrayAdapter = new ArrayAdapter<String>(this, R.layout.
                device_name);
87
88          // Find and set up the ListView for paired devices
89          ListView pairedListView = (ListView) findViewById(R.id.
                paired_devices);
90          pairedListView.setAdapter(mPairedDevicesArrayAdapter);
91          pairedListView.setOnItemClickListener(mDeviceClickListener);
92
93          // Find and set up the ListView for newly discovered devices
94          ListView newDevicesListView = (ListView) findViewById(R.id.
                new_devices);
95          newDevicesListView.setAdapter(mNewDevicesArrayAdapter);
96          newDevicesListView.setOnItemClickListener(mDeviceClickListener);
97
98          // Register for broadcasts when a device is discovered
99          IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND
                );
100         this.registerReceiver(mReceiver, filter);
101
102         // Register for broadcasts when discovery has finished
103         filter = new IntentFilter(BluetoothAdapter.
                ACTION_DISCOVERY_FINISHED);
104         this.registerReceiver(mReceiver, filter);
105
106         // Get the local Bluetooth adapter
107         mBtAdapter = BluetoothAdapter.getDefaultAdapter();
108
109         //Kontrollen ob Bluetooth vorhanden ist
110         if(mBtAdapter != null)
111         {
112
113                 //kontrollieren ob Bluetooth eingeschaltet ist
114                 if(mBtAdapter.isEnabled())
115                 {
116                 // Get a set of currently paired devices
117                 Set<BluetoothDevice> pairedDevices = mBtAdapter.
                        getBondedDevices();
118
119                 // If there are paired devices, add each one to the
                        ArrayAdapter
120                 if (pairedDevices.size() > 0) {
121                     findViewById(R.id.title_paired_devices).setVisibility(
                            View.VISIBLE);
122                     for (BluetoothDevice device : pairedDevices) {
123                         mPairedDevicesArrayAdapter.add(device.getName() + "
                                \n" + device.getAddress());
124                     }
125                 } else {
```

```
126                     String noDevices = getResources().getText(R.string.
                            none_paired).toString();
127                     mPairedDevicesArrayAdapter.add(noDevices);
128                 }
129                 }
130                 else
131                 {
132                     Intent enableBtIntent = new Intent(BluetoothAdapter.
                            ACTION_REQUEST_ENABLE);
133                     startActivityForResult(enableBtIntent,
                            REQUEST_ENABLE_BT);
134                     finish();
135                 }
136
137         }
138         else
139         {
140                 toast = Toast.makeText(context,getResources().getText(R.
                        string.No_Bluetooth_Exist).toString(), Toast.LENGTH_LONG
                         );
141                 toast.show();
142         }
143     }
144
145     @Override
146     protected void onDestroy() {
147         super.onDestroy();
148
149         // Make sure we're not doing discovery anymore
150         if (mBtAdapter != null) {
151             mBtAdapter.cancelDiscovery();
152         }
153
154         // Unregister broadcast listeners
155         this.unregisterReceiver(mReceiver);
156     }
157
158     /**
159      * Start device discover with the BluetoothAdapter
160      */
161     private void doDiscovery() {
162         if (D) Log.d(TAG, "doDiscovery()");
163
164         // Indicate scanning in the title
165         setProgressBarIndeterminateVisibility(true);
166         setTitle(R.string.scanning);
167
168         // Turn on sub-title for new devices
169         findViewById(R.id.title_new_devices).setVisibility(View.VISIBLE);
170
171         // If we're already discovering, stop it
172         if (mBtAdapter.isDiscovering()) {
173             mBtAdapter.cancelDiscovery();
174         }
175
176         // Request discover from BluetoothAdapter
177         mBtAdapter.startDiscovery();
178     }
179
```

```
180     // The on-click listener for all devices in the ListViews
181     private OnItemClickListener mDeviceClickListener = new
            OnItemClickListener() {
182         public void onItemClick(AdapterView<?> av, View v, int arg2, long
                arg3) {
183
184                 //Cancel discovery because it's costly and we're about to
                        connect
185             mBtAdapter.cancelDiscovery();
186             String address;
187
188             // Get the device MAC address, which is the last 17 chars in
                    the View
189             String info = ((TextView) v).getText().toString();
190             if(info == getResources().getText(R.string.none_found).toString
                    ())
191             {
192                 address = null;
193             }
194             else
195             {
196                 address = info.substring(info.length() - 17);
197                 }
198
199             // Create the result Intent and include the MAC address
200             Intent intent = new Intent();
201             intent.putExtra(EXTRA_DEVICE_ADDRESS, address);
202
203             // Set result and finish this Activity
204             setResult(Activity.RESULT_OK, intent);
205             finish();
206         }
207     };
208
209     // The BroadcastReceiver that listens for discovered devices and
210     // changes the title when discovery is finished
211     private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
212         @Override
213         public void onReceive(Context context, Intent intent) {
214             String action = intent.getAction();
215
216             // When discovery finds a device
217             if (BluetoothDevice.ACTION_FOUND.equals(action)) {
218                 // Get the BluetoothDevice object from the Intent
219                 BluetoothDevice device = intent.getParcelableExtra(
                        BluetoothDevice.EXTRA_DEVICE);
220                 // If it's already paired, skip it, because it's been
                        listed already
221                 if (device.getBondState() != BluetoothDevice.BOND_BONDED) {
222                     mNewDevicesArrayAdapter.add(device.getName() + "\n" +
                            device.getAddress());
223                 }
224             // When discovery is finished, change the Activity title
225             } else if (BluetoothAdapter.ACTION_DISCOVERY_FINISHED.equals(
                    action)) {
226                 setProgressBarIndeterminateVisibility(false);
227                 setTitle(R.string.select_device);
228                 if (mNewDevicesArrayAdapter.getCount() == 0) {
```

```
229                     String noDevices = getResources().getText(R.string.
                            none_found).toString();
230                     mNewDevicesArrayAdapter.add(noDevices);
231                 }
232             }
233         }
234     };
235
236 }
```

Settings.java

```
1  package MasterThesis.CameraControl;
2
3  import android.app.Activity;
4  import android.content.BroadcastReceiver;
5  import android.content.Context;
6  import android.content.Intent;
7  import android.content.IntentFilter;
8  import android.content.res.Configuration;
9  import android.os.Bundle;
10 import android.os.Handler;
11 import android.util.Log;
12 import android.view.View;
13 import android.widget.Button;
14 import android.widget.CheckBox;
15 import android.widget.RadioButton;
16 import android.widget.TextView;
17 import android.widget.Toast;
18
19 public class Settings extends Activity {
20
21         //Warten auf Nachrichten vom TabMain
22         private int OpenCode=0;
23         private char OpenState=0;
24
25
26         private Handler mHandler3D = new Handler();
27         private Handler mHandlerTimeLaps = new Handler();
28         private Handler mHandler3DHDR = new Handler();
29
30
31         //Code der übertragen werden solll
32         int code;
33
34         //Anzahl der Parameter
35         char parametercount;
36
37         //Array mit den Parametern
38         byte[] parameter = new byte[20];
39
40         //StateMachineValues;
41         byte StateTimeLaps = 0;
42         byte State3D = 0;
43         byte State3DHDR = 0;
44
45         //StateMachineHelpValues
46         byte State3DStartPosition = 0;
47         byte State3DShootSet = 0;
```

```
48          byte Host3DHDR = 0;
49
50
51          /** Called when the activity is first created. */
52          @Override
53          public void onCreate(Bundle savedInstanceState) {
54              super.onCreate(savedInstanceState);
55              setContentView(R.layout.settings);
56
57              registerReceiver(mReceiver, filter); // Don't forget to
                    unregister during onDestroy
58          }
59
60          // Register the BroadcastReceiver
61          IntentFilter filter = new IntentFilter(TabMain.
                BROADCAST_RECEIVED_CONTAINER);
62
63          //Funktion überprüft das Codewort in der Response Phase
64          //0 alles passt
65          //1 Fehler
66          private char CheckAnswer(int code, Context context)
67          {
68                  switch(code & 0xFFFFFF)
69                  {
70                  case Commands.RESPONSE_OK:
71                          return 0;
72                  case Commands.RESPONSE_WRONG_PARAMETER:
73                          Toast.makeText(context, "RESPONSE_WRONG_PARAMETER!"
                                , Toast.LENGTH_LONG).show();
74                          break;
75                  case Commands.RESPONSE_NOT_SUPPORTED_CLASS:
76                          Toast.makeText(context, "
                                RESPONSE_NOT_SUPPORTED_CLASS!", Toast.
                                LENGTH_LONG).show();
77                          break;
78                  case Commands.RESPONSE_NOT_SUPPORTED_ADCLASS:
79                          Toast.makeText(context, "
                                RESPONSE_NOT_SUPPORTED_ADCLASS!", Toast.
                                LENGTH_LONG).show();
80                          break;
81                  case Commands.RESPONSE_NOT_SUPPORTED_PARAMETERCOUNT:
82                          Toast.makeText(context, "
                                RESPONSE_NOT_SUPPORTED_PARAMETERCOUNT!", Toast.
                                LENGTH_LONG).show();
83                          break;
84                  case Commands.RESPONSE_NOT_SUPPORTED:
85                          Toast.makeText(context, "RESPONSE_NOT_SUPPORTED!",
                                Toast.LENGTH_LONG).show();
86                          break;
87                  case Commands.RESPONSE_NOT_SUPPORTED_PHASE:
88                          Toast.makeText(context, "
                                RESPONSE_NOT_SUPPORTED_PHASE!", Toast.
                                LENGTH_LONG).show();
89                          break;
90                  case Commands.RESPONSE_ERROR:
91                          Toast.makeText(context, "RESPONSE_ERROR!", Toast.
                                LENGTH_LONG).show();
92                          break;
93                  case Commands.RESPONSE_PTP_ERROR:
```

```
94                      Toast.makeText(context, "RESPONSE_PTP_ERROR!",
                            Toast.LENGTH_LONG).show();
95                      break;
96              case Commands.RESPONSE_DATA:
97                      Toast.makeText(context, "RESPONSE_DATA!", Toast.
                            LENGTH_LONG).show();
98                      break;
99              }
100
101             return 1;
102     }
103
104     private void SetCheckBoxState(byte InOut, byte State)
105     {
106             //InOut CheckBox array
107             CheckBox checkInOut[] = new CheckBox[8];
108
109             checkInOut[0]=  (CheckBox) findViewById(R.id.InOut0);
110             checkInOut[1]=  (CheckBox) findViewById(R.id.InOut1);
111             checkInOut[2]=  (CheckBox) findViewById(R.id.InOut2);
112             checkInOut[3]=  (CheckBox) findViewById(R.id.InOut3);
113             checkInOut[4]=  (CheckBox) findViewById(R.id.InOut4);
114             checkInOut[5]=  (CheckBox) findViewById(R.id.InOut5);
115             checkInOut[6]=  (CheckBox) findViewById(R.id.InOut6);
116             checkInOut[7]=  (CheckBox) findViewById(R.id.InOut7);
117
118             //Befüllen
119             for(int i = 0; i<8;i++)
120             {
121                     if(((InOut>>i)&0x01) == 0x01)
122                     {
123                             checkInOut[i].setChecked(true);
124                     }
125                     else
126                     {
127                             checkInOut[i].setChecked(false);
128                     }
129             }
130
131             //InOut CheckBox array
132             CheckBox checkState[] = new CheckBox[8];
133
134             checkState[0]=  (CheckBox) findViewById(R.id.State0);
135             checkState[1]=  (CheckBox) findViewById(R.id.State1);
136             checkState[2]=  (CheckBox) findViewById(R.id.State2);
137             checkState[3]=  (CheckBox) findViewById(R.id.State3);
138             checkState[4]=  (CheckBox) findViewById(R.id.State4);
139             checkState[5]=  (CheckBox) findViewById(R.id.State5);
140             checkState[6]=  (CheckBox) findViewById(R.id.State6);
141             checkState[7]=  (CheckBox) findViewById(R.id.State7);
142
143             //Befüllen
144             for(int i = 0; i<8;i++)
145             {
146                     if(((State>>i)&0x01) == 0x01)
147                     {
148                             checkState[i].setChecked(true);
149                     }
150                     else
```

```
151                         {
152                                 checkState[i].setChecked(false);
153                         }
154                 }
155
156         }
157
158         private final BroadcastReceiver mReceiver = new BroadcastReceiver()
                {
159             public void onReceive(Context context, Intent intent) {
160                 String action = intent.getAction();
161
162                 if(OpenState != 0 && action == TabMain.
                        BROADCAST_RECEIVED_CONTAINER)
163                 {
164                         //Werte auslesen
165                         int code;
166                         char parametercount;
167                         char TransactionIDReturn;
168                         code= intent.getIntExtra(TabMain.
                            BROADCAST_RECEIVED_CONTAINER_CODE, 0);
169
170                         parametercount = intent.getCharExtra(TabMain.
                            BROADCAST_RECEIVED_CONTAINER_NUMPARAMETERS, (
                            char) 0);
171                         TransactionIDReturn = intent.getCharExtra(TabMain.
                            BROADCAST_RECEIVED_CONTAINER_TRANSACTIONID, (
                            char) 0);
172                         byte[] parameter = intent.getByteArrayExtra(TabMain
                            .BROADCAST_RECEIVED_CONTAINER_PARAMETER);
173
174                         switch(OpenCode)
175                         {
176                         case Commands.INTERNAL_COMMAND_3D:
177                                 if(CheckAnswer(code,context)!=0)
178                                 {
179                                         //Fehler
180                                         StopTimer3DFunction();
181                                 }
182                                 else
183                                 {
184                                         if(State3D==0)
185                                         {
186                                                 State3D=1;
187                                         }
188                                         else if(State3D ==1)
189                                         {
190                                                 State3D=2;
191                                         }
192                                 }
193
194                                 break;
195
196                         case Commands.INTERNAL_COMMAND_3DHDR:
197                                 if(CheckAnswer(code,context)!=0)
198                                 {
199                                         //Fehler
200                                         StopTimer3DHDRFunction();
201                                 }
```

```
202
203                     break;
204                 case Commands.INTERNAL_COMMAND_TIMELAPS:
205                     if(CheckAnswer(code,context)!=0)
206                     {
207                         //Fehler
208                         StopTimerTimeLap();
209                     }
210                     else
211                     {
212                         if(StateTimeLaps==0)
213                         {
214                             StateTimeLaps=1;
215                         }
216                     }
217                     break;
218
219                 case Commands.COMMAND_PIN_STATE:
220
221                         //Status
222                     switch(OpenState)
223                     {
224                     case 1:
225                         //Wenn keine Datenphase zurückkommt
                                , Fehler ausgeben
226                         if((code & 0xFF000000) != 0
                            x00000000)
227                         {
228                             Toast.makeText(context, "
                                Aspected Dataphase",
                                Toast.LENGTH_LONG).show
                                ();
229                             OpenState = 0;
230                             OpenCode = 0;
231                             break;
232                         }
233
234                         //Erwartung: Datenphase mit 2 Bytes
235                         if(parametercount != 2)
236                         {
237                             Toast.makeText(context, "
                                Wrong Parameter Count
                                returned!", Toast.
                                LENGTH_LONG).show();
238                             OpenState = 0;
239                             OpenCode = 0;
240                             break;
241                         }
242                         else
243                         {
244                             SetCheckBoxState(parameter
                                [0],parameter[1]);
245                         }
246
247                         //Daten sind ok - befüllen der
                            checkboxen
248
249                         OpenState = 2;
250
```

```
251                                         break;
252                                 case 2:
253                                         //Response Phase
254                                         //Wenn keine Response zurückkommt,
                                                Fehler ausgeben
255                                         if((code & 0xFF000000) != 0
                                                xFF000000)
256                                         {
257                                                 Toast.makeText(context, "
                                                        Aspected Responsephase",
                                                         Toast.LENGTH_LONG).show
                                                        ();
258                                                 OpenState = 0;
259                                                 OpenCode = 0;
260                                                 break;
261                                         }
262
263                                         //Überprüfen der Antwort
264                                         if(CheckAnswer(code,context)==0)
265                                         {
266                                                 Toast.makeText(context, "
                                                        COMMAND_PIN_STATE finish
                                                        !", Toast.LENGTH_LONG).
                                                        show();
267                                                 //Daten setzen
268                                         }
269
270                                         OpenState = 0;
271                                         OpenCode = 0;
272                                         break;
273                                 }
274
275                                 //Werte auslesen
276                                 break;
277
278                         case Commands.COMMAND_GETCLASS:
279
280                                 //Status
281                         switch(OpenState)
282                         {
283                         case 1:
284                                 //Wenn keine Datenphase zurückkommt, Fehler
                                        ausgeben
285                                 if((code & 0xFF000000) != 0x00000000)
286                                 {
287                                         Toast.makeText(context, "Aspected
                                                Dataphase", Toast.LENGTH_LONG).
                                                show();
288                                         OpenState = 0;
289                                         OpenCode = 0;
290                                         break;
291                                 }
292
293                                 //Erwartung: Datenphase mit 1 Bytes
294                                 if(parametercount != 1)
295                                 {
296                                         Toast.makeText(context, "Wrong
                                                Parameter Count returned!",
                                                Toast.LENGTH_LONG).show();
```

```
297                                     OpenState = 0;
298                                     OpenCode = 0;
299                                     break;
300                             }
301                             else
302                             {
303                                     String StringConnection;
304                                     StringConnection = "";
305
306                                     switch(parameter[0])
307                                     {
308                                     case 0:
309                                             StringConnection = "Klasse 
                                                4 E";
310                                             break;
311                                     case 1:
312                                             StringConnection = "Klasse 
                                                4 B";
313                                             break;
314                                     case 2:
315                                             StringConnection = "Klasse 
                                                5 E";
316                                             break;
317                                     case 3:
318                                             StringConnection = "Klasse 
                                                5 B";
319                                             break;
320                                     default:
321                                             StringConnection = "unkown"
                                                ;
322                                             break;
323                                     }
324
325                                     TextView textview = (TextView)
                                            findViewById(R.id.GetClassBox);
326                                     textview.setText(StringConnection);
327                             }
328
329                             //Daten sind ok
330                             OpenState = 2;
331                                     break;
332                     case 2:
333                             //Response Phase
334                             //Wenn keine Response zurückkommt, Fehler
                                ausgeben
335                             if((code & 0xFF000000) != 0xFF000000)
336                             {
337                                     Toast.makeText(context, "Aspected 
                                            Responsephase", Toast.
                                            LENGTH_LONG).show();
338                                     OpenState = 0;
339                                     OpenCode = 0;
340                                     break;
341                             }
342
343                             //Überprüfen der Antwort
344                             if(CheckAnswer(code,context)==0)
345                             {
```

```
346                                     Toast.makeText(context, "
                                            COMMAND_GETCLASS finish!", Toast
                                            .LENGTH_LONG).show();
347                                     //Daten setzen
348                             }
349
350                                     OpenState = 0;
351                                     OpenCode = 0;
352                             break;
353                     }
354                             break;
355                     case Commands.COMMAND_PTP_GETCONNECTION:
356
357                             //Status
358                     switch(OpenState)
359                     {
360                     case 1:
361                             //Wenn keine Datenphase zurückkommt, Fehler
                                    ausgeben
362                             if((code & 0xFF000000) != 0x00000000)
363                             {
364                                     Toast.makeText(context, "Aspected
                                            Dataphase", Toast.LENGTH_LONG).
                                            show();
365                                     OpenState = 0;
366                                     OpenCode = 0;
367                                     break;
368                             }
369
370                             //Erwartung: Datenphase mit 1 Bytes
371                             if(parametercount != 1)
372                             {
373                                     Toast.makeText(context, "Wrong
                                            Parameter Count returned!",
                                            Toast.LENGTH_LONG).show();
374                                     OpenState = 0;
375                                     OpenCode = 0;
376                                     break;
377                             }
378                             else
379                             {
380                                     String StringConnection;
381                                     StringConnection = "";
382                                     if((parameter[0]&0x03) == 0x00)
383                                             StringConnection = "Host 1
                                                    leer";
384                                     else if((parameter[0]&0x03) == 0x01
                                            )
385                                             StringConnection = "Host 1
                                                    angesteckt";
386                                     else if((parameter[0]&0x03) == 0x03
                                            )
387                                             StringConnection = "Host 1
                                                    PTP";
388
389                                     if((parameter[0]&0x0C) == 0x00)
390                                             StringConnection =
                                                    StringConnection + "
                                                    Host 2 leer";
```

```
391                                     else if((parameter[0]&0x0C) == 0x04
                                            )
392                                         StringConnection =
                                                StringConnection + "
                                                Host 2 angesteckt";
393                                     else if((parameter[0]&0x0C) == 0x0C
                                            )
394                                         StringConnection =
                                                StringConnection + "
                                                Host 2 PTP";
395
396                                     TextView textview = (TextView)
                                            findViewById(R.id.
                                            GetConnectionBox);
397                                     textview.setText(StringConnection);
398
399                                 }
400
401                                 //Daten sind ok
402                                 OpenState = 2;
403                                     break;
404                             case 2:
405                                 //Response Phase
406                                 //Wenn keine Response zurückkommt, Fehler
                                        ausgeben
407                                 if((code & 0xFF000000) != 0xFF000000)
408                                 {
409                                     Toast.makeText(context, "Aspected
                                            Responsephase", Toast.
                                            LENGTH_LONG).show();
410                                     OpenState = 0;
411                                     OpenCode = 0;
412                                     break;
413                                 }
414
415                                 //Überprüfen der Antwort
416                                 if(CheckAnswer(code,context)==0)
417                                 {
418                                     Toast.makeText(context, "
                                            COMMAND_GETCLASS finish!", Toast
                                            .LENGTH_LONG).show();
419                                     //Daten setzen
420                                 }
421
422                                     OpenState = 0;
423                                     OpenCode = 0;
424                                 break;
425                             }
426
427                             //Werte auslesen
428                             break;
429                             }
430                     }
431
432                 }
433             };
434
435     @Override
436     public void onDestroy() {
```

```
437             super.onDestroy();
438
439             unregisterReceiver(mReceiver);
440         }
441
442             //3D Scanner
443         public void SetTimer3D(View view)
444         {
445             mHandler3D.removeCallbacks(mUpdateTimeTask3D);
446             mHandler3D.postDelayed(mUpdateTimeTask3D, 2000);
447
448             Button button = (Button) findViewById(R.id.Stop3D);
449             DisableAllExcept(button);
450
451             //Rückgabe kontrollieren
452             OpenCode = Commands.INTERNAL_COMMAND_3D ;
453             OpenState=1;
454
455
456             //Anfangswert setzen
457             State3D = 0;
458             State3DShootSet=0;
459             State3DStartPosition=0;
460         }
461
462             //3D HDR
463         public void SetTimer3DHDR(View view)
464         {
465             mHandler3D.removeCallbacks(mUpdateTimeTask3DHDR);
466             mHandler3D.postDelayed(mUpdateTimeTask3DHDR, 2000);
467
468             Button button = (Button) findViewById(R.id.Stop3DHDR);
469             DisableAllExcept(button);
470
471             //Rückgabe kontrollieren
472             OpenCode = Commands.INTERNAL_COMMAND_3DHDR ;
473             OpenState=1;
474
475             //Anfangswert setzen
476             State3DHDR = 0;
477             Host3DHDR =0;
478         }
479
480         public void StopTimer3DHDR(View view)
481         {
482             StopTimer3DHDRFunction();
483         }
484
485         public void StopTimer3DHDRFunction()
486         {
487             mHandler3DHDR.removeCallbacks(mUpdateTimeTask3DHDR);
488             Toast.makeText(getApplicationContext(), "Stop", Toast.LENGTH_SHORT)
                    .show();
489
490             OpenCode = 0 ;
491             OpenState=0;
492
493             ActivateALL();
494         }
```

```
495
496     private Runnable mUpdateTimeTask3DHDR = new Runnable() {
497             public void run() {
498
499                     StateMachine3DHDR();
500
501                    mHandler3DHDR.postDelayed(this, 1000 );
502
503             }
504         };
505
506         private void StateMachine3DHDR()
507         {
508
509             switch(State3DHDR)
510             {
511             case 0:
512                     //Session Open
513                     SessionOpen((byte) Host3DHDR);
514
515                     if(Host3DHDR==1)
516                     {
517                             Host3DHDR=0;
518                             State3DHDR++;
519                     }
520                     else
521                     {
522                             Host3DHDR=1;
523                     }
524                     break;
525             case 1:
526                     //-2 BV einstellen
527                     ChangeBel((byte) Host3DHDR,(char) 0xF0);
528
529                     if(Host3DHDR==1)
530                     {
531                             Host3DHDR=0;
532                             State3DHDR++;
533                     }
534                     else
535                     {
536                             Host3DHDR=1;
537                     }
538                     break;
539             case 2:
540                     //Foto
541                     Shoot((byte) Host3DHDR);
542
543                     if(Host3DHDR==1)
544                     {
545                             Host3DHDR=0;
546                             State3DHDR++;
547                     }
548                     else
549                     {
550                             Host3DHDR=1;
551                     }
552
553                     break;
```

```
554             case 3:
555                     //-1 BV einstellen
556                     ChangeBel((byte) Host3DHDR,(char) 0xF8);
557
558                     if(Host3DHDR==1)
559                     {
560                             Host3DHDR=0;
561                             State3DHDR++;
562                     }
563                     else
564                     {
565                             Host3DHDR=1;
566                     }
567                     break;
568             case 4:
569                     //Foto
570                     Shoot((byte) Host3DHDR);
571
572                     if(Host3DHDR==1)
573                     {
574                             Host3DHDR=0;
575                             State3DHDR++;
576                     }
577                     else
578                     {
579                             Host3DHDR=1;
580                     }
581
582                     break;
583             case 5:
584                     //0 BV einstellen
585                     ChangeBel((byte) Host3DHDR,(char) 0x00);
586
587                     if(Host3DHDR==1)
588                     {
589                             Host3DHDR=0;
590                             State3DHDR++;
591                     }
592                     else
593                     {
594                             Host3DHDR=1;
595                     }
596                     break;
597             case 6:
598                     //Foto
599                     Shoot((byte) Host3DHDR);
600
601                     if(Host3DHDR==1)
602                     {
603                             Host3DHDR=0;
604                             State3DHDR++;
605                     }
606                     else
607                     {
608                             Host3DHDR=1;
609                     }
610
611                     break;
612             case 7:
```

```
613                     //+1 BV einstellen
614                     ChangeBel((byte) Host3DHDR,(char) 0x08);
615
616                     if(Host3DHDR==1)
617                     {
618                             Host3DHDR=0;
619                             State3DHDR++;
620                     }
621                     else
622                     {
623                             Host3DHDR=1;
624                     }
625                     break;
626             case 8:
627                     //Foto
628                     Shoot((byte) Host3DHDR);
629
630                     if(Host3DHDR==1)
631                     {
632                             Host3DHDR=0;
633                             State3DHDR++;
634                     }
635                     else
636                     {
637                             Host3DHDR=1;
638                     }
639
640                     break;
641             case 9:
642                     //+2 BV einstellen
643                     ChangeBel((byte) Host3DHDR,(char) 0x10);
644
645                     if(Host3DHDR==1)
646                     {
647                             Host3DHDR=0;
648                             State3DHDR++;
649                     }
650                     else
651                     {
652                             Host3DHDR=1;
653                     }
654                     break;
655             case 10:
656                     //Foto
657                     Shoot((byte) Host3DHDR);
658
659                     if(Host3DHDR==1)
660                     {
661                             Host3DHDR=0;
662                             State3DHDR++;
663                     }
664                     else
665                     {
666                             Host3DHDR=1;
667                     }
668
669                     break;
670             case 11:
671                     //Session Close
```

```
672                     SessionClose((byte) Host3DHDR);
673
674                     if(Host3DHDR==1)
675                     {
676                             Host3DHDR=0;
677                             State3DHDR++;
678                     }
679                     else
680                     {
681                             Host3DHDR=1;
682                     }
683                     break;
684             }
685
686         }
687
688     public void StopTimer3D(View view)
689     {
690         StopTimer3DFunction();
691     }
692
693     public void StopTimer3DFunction()
694     {
695         mHandler3D.removeCallbacks(mUpdateTimeTask3D);
696         Toast.makeText(getApplicationContext(), "Stop", Toast.LENGTH_SHORT)
                .show();
697
698         OpenCode = 0 ;
699         OpenState=0;
700
701         ActivateALL();
702     }
703
704     private Runnable mUpdateTimeTask3D = new Runnable() {
705             public void run() {
706
707                     //Rest verringern
708                 TextView textview = (TextView) findViewById(R.id.Edit3D);
709                 int RestPicture = Integer.parseInt(textview.getText().
                        toString())  -1;
710
711                 if(RestPicture==-1)
712                 {
713                     if(State3D < 4)
714                     {
715                             State3D=4;
716                             StateMachine3D();
717                             mHandler3D.postDelayed(this, 1000 );
718                     }
719                     else if(State3D == 4)
720                     {
721                             State3D=5;
722                             StateMachine3D();
723                             StopTimer3DFunction();
724                     }
725                 }
726                 else
727                 {
728                     StateMachine3D();
```

```
729                     //Session open, Live View und Fokus richtig stellen
                            sollen die Fotos nicht verringern
730                     if(State3D==3 && State3DShootSet==0)
731                     {
732                             textview.setText(String.valueOf(RestPicture));
733                     }
734
735                         mHandler3D.postDelayed(this, 1000 );
736                 }
737             }
738         };
739
740     //ReadState auslesen
741     public void ReadState(View view)
742     {
743
744         //Warten auf Rückgabe setzen
745         OpenCode = Commands.COMMAND_PIN_STATE ;
746         OpenState=1;
747
748                 code = 0x22010000 |  Commands.COMMAND_PIN_STATE ; //
                        DeviceID Klasse AD Klasse Code
749                 parametercount = 0;
750
751                 //
752                 byte test[] = new byte[20];
753
754                 //Intent auslösen
755                 //Daten zuvor bereitstellen
756                 Intent intent = new Intent();
757         intent.setAction(CameraControl.BROADCAST_SEND_MESSAGE);
758         intent.putExtra(CameraControl.BROADCAST_EXTRA_SEND_MESSAGE_CODE,
                code);
759         intent.putExtra(CameraControl.
               BROADCAST_EXTRA_SEND_MESSAGE_COUNT_PARAMETER, parametercount);
760         intent.putExtra(CameraControl.
               BROADCAST_EXTRA_SEND_MESSAGE_PARAMETER, test);
761         sendBroadcast(intent);
762
763     }
764
765     //Klasse vom Modul auslesen
766     public void GetClass(View view)
767     {
768
769         //Warten auf Rückgabe setzen
770         OpenCode = Commands.COMMAND_GETCLASS;
771         OpenState=1;
772
773                 code = 0x22010000 |  Commands.COMMAND_GETCLASS ; //
                        DeviceID Klasse AD Klasse Code
774                 parametercount = 0;
775
776                 //
777                 byte test[] = new byte[20];
778
779                 //Intent auslösen
780                 //Daten zuvor bereitstellen
781                 Intent intent = new Intent();
```

```
782         intent.setAction(CameraControl.BROADCAST_SEND_MESSAGE);
783         intent.putExtra(CameraControl.BROADCAST_EXTRA_SEND_MESSAGE_CODE,
                code);
784         intent.putExtra(CameraControl.
                BROADCAST_EXTRA_SEND_MESSAGE_COUNT_PARAMETER, parametercount);
785         intent.putExtra(CameraControl.
                BROADCAST_EXTRA_SEND_MESSAGE_PARAMETER, test);
786         sendBroadcast(intent);
787
788     }
789
790     //ReadState auslesen
791     public void GetConnection(View view)
792     {
793
794         //Warten auf Rückgabe setzen
795         OpenCode = Commands.COMMAND_PTP_GETCONNECTION ;
796         OpenState=1;
797
798                 code = 0x22050000 |  Commands.COMMAND_PTP_GETCONNECTION ;
                        // DeviceID Klasse AD Klasse Code
799                 parametercount = 0;
800
801                 //
802                 byte test[] = new byte[20];
803
804                 //Intent auslösen
805                 //Daten zuvor bereitstellen
806                 Intent intent = new Intent();
807         intent.setAction(CameraControl.BROADCAST_SEND_MESSAGE);
808         intent.putExtra(CameraControl.BROADCAST_EXTRA_SEND_MESSAGE_CODE,
                code);
809         intent.putExtra(CameraControl.
                BROADCAST_EXTRA_SEND_MESSAGE_COUNT_PARAMETER, parametercount);
810         intent.putExtra(CameraControl.
                BROADCAST_EXTRA_SEND_MESSAGE_PARAMETER, test);
811         sendBroadcast(intent);
812
813     }
814
815     //TimeLaps
816     public void StartTimerTimeLap(View view)
817     {
818         mHandlerTimeLaps.removeCallbacks(mUpdateTimeTaskTimeLaps);
819         mHandlerTimeLaps.postDelayed(mUpdateTimeTaskTimeLaps, 2000);
820
821         Button button = (Button) findViewById(R.id.StopTimelap);
822         DisableAllExcept(button);
823
824         //Rückgabe kontrollieren
825         OpenCode = Commands.INTERNAL_COMMAND_TIMELAPS ;
826         OpenState=1;
827
828         //Anfangswert setzen
829         StateTimeLaps = 0;
830     }
831
832     public void StopTimerTimeL(View view)
833     {
```

```
834             StopTimerTimeLap();
835         }
836
837         public void StopTimerTimeLap()
838         {
839             mHandlerTimeLaps.removeCallbacks(mUpdateTimeTaskTimeLaps);
840             Toast.makeText(getApplicationContext(), "Stop", Toast.LENGTH_SHORT)
                    .show();
841
842             OpenCode = 0 ;
843             OpenState=0;
844
845             ActivateALL();
846         }
847
848             private void StateMachine3D()
849             {
850                     byte Host;
851
852               //Welcher Host
853           RadioButton radiob =  (RadioButton) findViewById(R.id.radio03D);
854
855                 if(radiob.isChecked()==true)
856                 {
857                         Host = 0;
858                 }
859                 else
860                 {
861                         Host = 1;
862                 }
863
864                 switch(State3D)
865                 {
866                 case 0:
867                         //Session Open
868                         SessionOpen(Host);
869                         break;
870                 case 1:
871                         //Live View an
872                         LiveViewOn(Host);
873                         break;
874                 case 2:
875                         //Ausgangsposition einstellen
876                         ChangeFocus(Host, (char) 0x03);
877                         State3DStartPosition++;
878
879                         //Kontrolle ob schon die richtige Position erreicht
                                wurde
880                         if(State3DStartPosition==20)
881                         {
882                                 //Aufnahmen beginnen
883                                 State3D=3;
884                         }
885
886                         break;
887                 case 3:
888                         //Aufnehmen
889                         if(State3DShootSet==0)
890                         {
```

```
891                             //Foto schießen
892                             Shoot(Host);
893                             State3DShootSet=1;
894                     }
895                     else
896                     {
897                             //Eine Einstellung weiter drehen
898                             ChangeFocus(Host, (char) 0x8002);
899                             State3DShootSet=0;
900                     }
901
902                     break;
903             case 4:
904                     //Live View aus
905                     LiveViewOff(Host);
906                     break;
907             case 5:
908                     //Session Close
909                     SessionClose(Host);
910                     break;
911             }
912
913         }
914
915         private void StateMachineTimeLaps()
916         {
917                 byte Host;
918
919           //Welcher Host
920       RadioButton radiob =  (RadioButton) findViewById(R.id.radio0TimeLap);
921
922             if(radiob.isChecked()==true)
923             {
924                     Host = 0;
925             }
926             else
927             {
928                     Host = 1;
929             }
930
931             switch(StateTimeLaps)
932             {
933             case 0:
934                     //Session Open
935                     SessionOpen(Host);
936                     break;
937             case 1:
938                     //Foto schießen
939                     Shoot(Host);
940                     break;
941             case 2:
942                     //Session Close
943                     SessionClose(Host);
944                     break;
945
946             }
947
948         }
949
```

```
950     private Runnable mUpdateTimeTaskTimeLaps = new Runnable() {
951             public void run() {
952
953                     Toast.makeText(getApplicationContext(), "timer
                            Zeitraffer" , Toast.LENGTH_SHORT).show();
954
955                     //Rest verringern
956                 TextView textview = (TextView) findViewById(R.id.
                        EditTimelapsAmount);
957                 int RestPicture = Integer.parseInt(textview.getText().
                        toString())  -1;
958
959                 if(RestPicture==-1)
960                 {
961                     if(StateTimeLaps==1)
962                     {
963                             StateTimeLaps=2;
964                             StateMachineTimeLaps();
965                     }
966                     StopTimerTimeLap();
967                 }
968                 else
969                 {
970                     StateMachineTimeLaps();
971                     //Session open soll die Fotos nicht verringern
972                     if(StateTimeLaps!=0)
973                     {
974                             textview.setText(String.valueOf(RestPicture));
975                     }
976
977                         textview = (TextView) findViewById(R.id.
                            EditTimeLapsTime);
978
979                         int timeValue = 1000 *  Integer.parseInt(textview.
                            getText().toString());
980
981                         if(timeValue <0)
982                             timeValue = -1*timeValue;
983
984                         if(timeValue ==0)
985                         {
986                             timeValue =5000;
987                             textview.setText("5");
988                         }
989
990                         mHandlerTimeLaps.postDelayed(this, timeValue);
991                 }
992
993             }
994         };
995
996     private void ActivateALL()
997     {
998         //Rest aktivieren
999         Button button = (Button) findViewById(R.id.Start3D);
1000        button.setEnabled(true);
1001
1002        button = (Button) findViewById(R.id.Start3DHDR);
1003        button.setEnabled(true);
```

```
1004
1005            button = (Button) findViewById(R.id.StartTimelap);
1006            button.setEnabled(true);
1007
1008            button = (Button) findViewById(R.id.ReadState);
1009            button.setEnabled(true);
1010
1011            button = (Button) findViewById(R.id.GetConnection);
1012            button.setEnabled(true);
1013
1014            button = (Button) findViewById(R.id.GetClass);
1015            button.setEnabled(true);
1016
1017            //Stop auf False setzen
1018            button = (Button) findViewById(R.id.Stop3D);
1019            button.setEnabled(false);
1020
1021            button = (Button) findViewById(R.id.Stop3DHDR);
1022            button.setEnabled(false);
1023
1024            button = (Button) findViewById(R.id.StopTimelap);
1025            button.setEnabled(false);
1026
1027            //Edit Boxen aktivieren
1028            TextView textview = (TextView) findViewById(R.id.Edit3D);
1029            textview.setEnabled(true);
1030
1031            textview = (TextView) findViewById(R.id.EditTimelapsAmount);
1032            textview.setEnabled(true);
1033
1034            textview = (TextView) findViewById(R.id.EditTimeLapsTime);
1035            textview.setEnabled(true);
1036
1037            //Radio Boxen aktivieren
1038            RadioButton radiobutton = (RadioButton) findViewById(R.id.radio03D)
                    ;
1039            radiobutton.setEnabled(true);
1040
1041            radiobutton = (RadioButton) findViewById(R.id.radio13D);
1042            radiobutton.setEnabled(true);
1043
1044            radiobutton = (RadioButton) findViewById(R.id.radio0TimeLap);
1045            radiobutton.setEnabled(true);
1046
1047            radiobutton = (RadioButton) findViewById(R.id.radio1TimeLap);
1048            radiobutton.setEnabled(true);
1049        }
1050
1051        //Diabled alle bis auf einen
1052        private void DisableAllExcept(Button Me)
1053        {
1054            Button button = (Button) findViewById(R.id.Start3D);
1055            if(Me == button)
1056            {
1057                    button.setEnabled(true);
1058            }
1059            else
1060            {
1061                    button.setEnabled(false);
```

```
1062            }
1063
1064
1065            button = (Button) findViewById(R.id.Stop3D);
1066            if(Me == button)
1067            {
1068                    button.setEnabled(true);
1069            }
1070            else
1071            {
1072                    button.setEnabled(false);
1073            }
1074
1075            button = (Button) findViewById(R.id.Start3DHDR);
1076            if(Me == button)
1077            {
1078                    button.setEnabled(true);
1079            }
1080            else
1081            {
1082                    button.setEnabled(false);
1083            }
1084
1085
1086            button = (Button) findViewById(R.id.Stop3DHDR);
1087            if(Me == button)
1088            {
1089                    button.setEnabled(true);
1090            }
1091            else
1092            {
1093                    button.setEnabled(false);
1094            }
1095
1096
1097            button = (Button) findViewById(R.id.GetClass);
1098            if(Me == button)
1099            {
1100                    button.setEnabled(true);
1101            }
1102            else
1103            {
1104                    button.setEnabled(false);
1105            }
1106
1107            button = (Button) findViewById(R.id.StartTimelap);
1108            if(Me == button)
1109            {
1110                    button.setEnabled(true);
1111            }
1112            else
1113            {
1114                    button.setEnabled(false);
1115            }
1116
1117            button = (Button) findViewById(R.id.StopTimelap);
1118            if(Me == button)
1119            {
1120                    button.setEnabled(true);
```

```
1121            }
1122            else
1123            {
1124                    button.setEnabled(false);
1125            }
1126
1127            button = (Button) findViewById(R.id.GetConnection);
1128            if(Me == button)
1129            {
1130                    button.setEnabled(true);
1131            }
1132            else
1133            {
1134                    button.setEnabled(false);
1135            }
1136
1137            button = (Button) findViewById(R.id.ReadState);
1138            if(Me == button)
1139            {
1140                    button.setEnabled(true);
1141            }
1142            else
1143            {
1144                    button.setEnabled(false);
1145            }
1146
1147            //Edit Boxen disablen
1148            TextView textview = (TextView) findViewById(R.id.Edit3D);
1149            textview.setEnabled(false);
1150
1151            textview = (TextView) findViewById(R.id.EditTimelapsAmount);
1152            textview.setEnabled(false);
1153
1154            textview = (TextView) findViewById(R.id.EditTimeLapsTime);
1155            textview.setEnabled(false);
1156
1157            //Radio Button disablen
1158            RadioButton radiobutton = (RadioButton) findViewById(R.id.radio03D)
                    ;
1159            radiobutton.setEnabled(false);
1160
1161            radiobutton = (RadioButton) findViewById(R.id.radio13D);
1162            radiobutton.setEnabled(false);
1163
1164            radiobutton = (RadioButton) findViewById(R.id.radio0TimeLap);
1165            radiobutton.setEnabled(false);
1166
1167            radiobutton = (RadioButton) findViewById(R.id.radio1TimeLap);
1168            radiobutton.setEnabled(false);
1169
1170    }
1171
1172    void SessionOpen(byte Host)
1173    {
1174                    code = 0x22050000 | (Host << 8) | (byte) Commands.
                            COMMAND_PTP_SESSION_OPEN ; // DeviceID Klasse AD Klasse
                            Code
1175                    parametercount = 0;
1176
```

```
1177                //Parameterliste erzeugen
1178                FillsParameterList(parameter,(char) parametercount, (char)
                        0);
1179
1180                //Intent auslösen
1181                //Daten zuvor bereitstellen
1182                Intent intent =new Intent();
1183            intent.setAction(CameraControl.BROADCAST_SEND_MESSAGE);
1184            intent.putExtra(CameraControl.BROADCAST_EXTRA_SEND_MESSAGE_CODE,
                    code);
1185            intent.putExtra(CameraControl.
                    BROADCAST_EXTRA_SEND_MESSAGE_COUNT_PARAMETER,parametercount);
1186            intent.putExtra(CameraControl.
                    BROADCAST_EXTRA_SEND_MESSAGE_PARAMETER,parameter);
1187            sendBroadcast(intent);
1188        }
1189
1190        void SessionClose(byte Host)
1191        {
1192                code = 0x22050000 | (Host << 8) | (byte) Commands.
                        COMMAND_PTP_SESSION_CLOSE; // DeviceID Klasse AD Klasse
                        Code
1193                parametercount = 0;
1194
1195                //Parameterliste erzeugen
1196                FillsParameterList(parameter,(char) parametercount, (char)
                        0);
1197
1198                //Intent auslösen
1199                //Daten zuvor bereitstellen
1200                Intent intent =new Intent();
1201            intent.setAction(CameraControl.BROADCAST_SEND_MESSAGE);
1202            intent.putExtra(CameraControl.BROADCAST_EXTRA_SEND_MESSAGE_CODE,
                    code);
1203            intent.putExtra(CameraControl.
                    BROADCAST_EXTRA_SEND_MESSAGE_COUNT_PARAMETER,parametercount);
1204            intent.putExtra(CameraControl.
                    BROADCAST_EXTRA_SEND_MESSAGE_PARAMETER,parameter);
1205            sendBroadcast(intent);
1206        }
1207
1208        void Shoot(byte Host)
1209        {
1210                code = 0x22050000 | (Host << 8) | (byte) Commands.
                        COMMAND_PTP_SHOOT ; // DeviceID Klasse AD Klasse Code
1211                parametercount = 0;
1212
1213                //Parameterliste erzeugen
1214                FillsParameterList(parameter,(char) parametercount, (char)
                        0);
1215
1216                //Intent auslösen
1217                //Daten zuvor bereitstellen
1218                Intent intent =new Intent();
1219            intent.setAction(CameraControl.BROADCAST_SEND_MESSAGE);
1220            intent.putExtra(CameraControl.BROADCAST_EXTRA_SEND_MESSAGE_CODE,
                    code);
1221            intent.putExtra(CameraControl.
                    BROADCAST_EXTRA_SEND_MESSAGE_COUNT_PARAMETER,parametercount);
```

```
1222            intent.putExtra(CameraControl.
                    BROADCAST_EXTRA_SEND_MESSAGE_PARAMETER, parameter);
1223            sendBroadcast(intent);
1224        }
1225
1226        void ChangeFocus(byte Host, char Value)
1227        {
1228                    code = 0x22050000 |  (Host << 8)| (byte) Commands.
                            COMMAND_PTP_FOKUS; // DeviceID Klasse AD Klasse Code
1229                    parametercount = 2;
1230
1231                    //Parameterliste erzeugen
1232                    FillsParameterList(parameter,(char) parametercount,Value);
1233
1234                    //Intent auslösen
1235                    //Daten zuvor bereitstellen
1236                    Intent intent =new Intent();
1237            intent.setAction(CameraControl.BROADCAST_SEND_MESSAGE);
1238            intent.putExtra(CameraControl.BROADCAST_EXTRA_SEND_MESSAGE_CODE,
                    code);
1239            intent.putExtra(CameraControl.
                    BROADCAST_EXTRA_SEND_MESSAGE_COUNT_PARAMETER, parametercount);
1240            intent.putExtra(CameraControl.
                    BROADCAST_EXTRA_SEND_MESSAGE_PARAMETER, parameter);
1241            sendBroadcast(intent);
1242        }
1243
1244        void ChangeBel(byte Host, char Value)
1245        {
1246                    code = 0x22050000 |  (Host << 8)| (byte) Commands.
                            COMMAND_PTP_CHANGE_BELICHTUNG; // DeviceID Klasse AD
                            Klasse Code
1247                    parametercount = 1;
1248
1249                    //Parameterliste erzeugen
1250                    FillsParameterList(parameter,(char) parametercount,Value);
1251
1252                    //Intent auslösen
1253                    //Daten zuvor bereitstellen
1254                    Intent intent =new Intent();
1255            intent.setAction(CameraControl.BROADCAST_SEND_MESSAGE);
1256            intent.putExtra(CameraControl.BROADCAST_EXTRA_SEND_MESSAGE_CODE,
                    code);
1257            intent.putExtra(CameraControl.
                    BROADCAST_EXTRA_SEND_MESSAGE_COUNT_PARAMETER, parametercount);
1258            intent.putExtra(CameraControl.
                    BROADCAST_EXTRA_SEND_MESSAGE_PARAMETER, parameter);
1259            sendBroadcast(intent);
1260        }
1261
1262        void LiveViewOn(byte Host)
1263        {
1264                    code = 0x22050000 | (Host << 8) | (byte) Commands.
                            COMMAND_PTP_LIVE_VIEW_ON ; // DeviceID Klasse AD Klasse
                            Code
1265                    parametercount = 0;
1266
1267                    //Parameterliste erzeugen
```

```
1268                FillsParameterList(parameter,(char) parametercount, (char)
                        0);
1269
1270                //Intent auslösen
1271                //Daten zuvor bereitstellen
1272                Intent intent =new Intent();
1273        intent.setAction(CameraControl.BROADCAST_SEND_MESSAGE);
1274        intent.putExtra(CameraControl.BROADCAST_EXTRA_SEND_MESSAGE_CODE,
                code);
1275        intent.putExtra(CameraControl.
               BROADCAST_EXTRA_SEND_MESSAGE_COUNT_PARAMETER,parametercount);
1276        intent.putExtra(CameraControl.
               BROADCAST_EXTRA_SEND_MESSAGE_PARAMETER,parameter);
1277        sendBroadcast(intent);
1278    }
1279
1280    void LiveViewOff(byte Host)
1281    {
1282                code = 0x22050000 | (Host << 8) | (byte) Commands.
                       COMMAND_PTP_LIVE_VIEW_OFF ; // DeviceID Klasse AD Klasse
                        Code
1283                parametercount = 0;
1284
1285                //Parameterliste erzeugen
1286                FillsParameterList(parameter,(char) parametercount, (char)
                        0);
1287
1288                //Intent auslösen
1289                //Daten zuvor bereitstellen
1290                Intent intent =new Intent();
1291        intent.setAction(CameraControl.BROADCAST_SEND_MESSAGE);
1292        intent.putExtra(CameraControl.BROADCAST_EXTRA_SEND_MESSAGE_CODE,
                code);
1293        intent.putExtra(CameraControl.
               BROADCAST_EXTRA_SEND_MESSAGE_COUNT_PARAMETER,parametercount);
1294        intent.putExtra(CameraControl.
               BROADCAST_EXTRA_SEND_MESSAGE_PARAMETER,parameter);
1295        sendBroadcast(intent);
1296    }
1297
1298        //Fills the Parameter Array
1299        //Count is the amount of parameters in bytes
1300        //MaxCount = 2 (actual
1301        public void FillsParameterList(byte [] array, char count, char
                Parameter1)
1302        {
1303                        switch(count)
1304                        {
1305
1306                                case 2:
1307                                        array[1] = (byte) ( Parameter1 >>
                                                8);
1308                                case 1:
1309                                        array[0] = (byte) Parameter1;
1310                                default:
1311                                        return;
1312                        }
1313        }
1314
```

```
1315 }
```

TabMain.java

```
1  package MasterThesis.CameraControl;
2
3
4  import android.app.Activity;
5  import android.app.TabActivity;
6  import android.bluetooth.BluetoothAdapter;
7  import android.bluetooth.BluetoothDevice;
8  import android.content.BroadcastReceiver;
9  import android.content.Context;
10 import android.content.Intent;
11 import android.content.IntentFilter;
12 import android.content.res.Configuration;
13 import android.content.res.Resources;
14 import android.os.Bundle;
15 import android.os.Handler;
16 import android.os.Message;
17 import android.util.Log;
18 import android.view.Menu;
19 import android.view.MenuInflater;
20 import android.view.MenuItem;
21 import android.widget.TabHost;
22 import android.widget.Toast;
23
24 public class TabMain extends TabActivity {
25
26     // Debugging
27     private static final String TAG = "Camera Control";
28     private static final boolean D = true;
29
30     // Message types sent from the BluetoothChatService Handler
31     public static final int MESSAGE_STATE_CHANGE = 1;
32     public static final int MESSAGE_READ = 2;
33     public static final int MESSAGE_WRITE = 3;
34     public static final int MESSAGE_DEVICE_NAME = 4;
35     public static final int MESSAGE_TOAST = 5;
36
37     // Key names received from the BluetoothChatService Handler
38     public static final String DEVICE_NAME = "device_name";
39     public static final String TOAST = "toast";
40
41     //Broadcast Message ReceiveContainer
42     public static final String BROADCAST_RECEIVED_CONTAINER = "
           RECEIVED_CONTAINER";
43     public static final String BROADCAST_RECEIVED_CONTAINER_TRANSACTIONID =
           "RECEIVED_CONTAINER_TRANSACTIONID";
44     public static final String BROADCAST_RECEIVED_CONTAINER_CODE = "
           RECEIVED_CONTAINER_CODE";
45     public static final String BROADCAST_RECEIVED_CONTAINER_NUMPARAMETERS =
           "RECEIVED_CONTAINER_NUMPARAMETERS";
46     public static final String BROADCAST_RECEIVED_CONTAINER_PARAMETER = "
           RECEIVED_CONTAINER_PARAMETER";
47
48     private String deviceconnected;
49
50     // Local Bluetooth adapter
```

```
51      private BluetoothAdapter mBluetoothAdapter = null;
52
53      // Member object for the chat services
54      private BluetoothChatService mChatService = null;
55
56      // String buffer for outgoing messages
57      private StringBuffer mOutStringBuffer;
58
59      // Intent request codes
60      private static final int REQUEST_CONNECT_DEVICE = 1;
61          private static final int REQUEST_ENABLE_BT = 0;
62
63          //TransactionID Send
64          private static char TransactionIDSend = 0;
65
66          //TransactionID Send
67          private static byte[] ReceiveBuffer = new byte[1024];
68          private static char BufferElements = 0;
69          private static byte Final = 0;
70          private static char NumParameters = 0;
71
72          //Read Container
73          private static char NumParametersRead = 0;
74          private static char TransactionIDRead = 0;
75          private static int ResponseCode = 0;
76
77
78          /* Function empty the Buffer*/
79          public void EmptyBuffer()
80          {
81
82                  do
83                  {
84                          //Element löschen
85                          ReceiveBuffer[BufferElements-1]= 0;
86                          //Elemente verringern
87                          BufferElements--;
88                  }while(BufferElements>0);
89
90          }
91
92          /** Called when the activity is first created. */
93          @Override
94          public void onCreate(Bundle savedInstanceState) {
95              super.onCreate(savedInstanceState);
96
97              setContentView(R.layout.tablayout);
98
99              Resources res = getResources(); // Resource object to get
                    Drawables
100             TabHost tabHost = getTabHost();  // The activity TabHost
101             TabHost.TabSpec spec;  // Reusable TabSpec for each tab
102             Intent intent;  // Reusable Intent for each tab
103
104             // Create an Intent to launch an Activity for the tab (to be
                    reused)
105             intent = new Intent().setClass(this, CameraControl.class);
106             intent.putExtra("Cam", 1);
107             // Initialize a TabSpec for each tab and add it to the TabHost
```

```
108             spec = tabHost.newTabSpec("Cam1").setIndicator(getString(R.
                    string.Tab_Cam1),
109                             res.getDrawable(R.drawable.ic_tab_camera_one)
                                    )
110                     .setContent(intent);
111             tabHost.addTab(spec);
112
113             // Do the same for the other tabs
114
115             intent = new Intent().setClass(this, CameraControl.class);
116             intent.putExtra("Cam", 2);
117             spec = tabHost.newTabSpec("Cam2").setIndicator(getString(R.
                    string.Tab_Cam2),
118                             res.getDrawable(R.drawable.ic_tab_camera_two)
                                    )
119                     .setContent(intent);
120             tabHost.addTab(spec);
121
122             intent = new Intent().setClass(this, Settings.class);
123             spec = tabHost.newTabSpec("Settings").setIndicator(getString(R.
                    string.Tab_Setting),
124                             res.getDrawable(R.drawable.ic_tab_setting))
125                     .setContent(intent);
126             tabHost.addTab(spec);
127             tabHost.setCurrentTab(0);
128
129
130             deviceconnected = "";
131
132             // Get local Bluetooth adapter
133         mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
134
135         // If the adapter is null, then Bluetooth is not supported
136         if (mBluetoothAdapter == null) {
137             Toast.makeText(this, "Bluetooth is not available", Toast.
                    LENGTH_LONG).show();
138             finish();
139             return;
140         }
141
142         registerReceiver(mReceiver, filter); // Don't forget to unregister
                during onDestroy
143
144         }
145
146         // Register the BroadcastReceiver
147         IntentFilter filter = new IntentFilter(CameraControl.
                BROADCAST_SEND_MESSAGE);
148
149         private final BroadcastReceiver mReceiver = new BroadcastReceiver()
                {
150             public void onReceive(Context context, Intent intent) {
151                 Toast ToastMessage;
152                 String action = intent.getAction();
153
154                 if(mChatService == null)
155                 {
156                         ToastMessage = Toast.makeText(context,getString(R.
                                string.services_not_started) , Toast.LENGTH_LONG
```

```
                        );
157                     ToastMessage.show();
158             }
159             else if(mChatService.getState() != BluetoothChatService.
                    STATE_CONNECTED )
160             {
161                     ToastMessage = Toast.makeText(context,getString(R.
                            string.not_connected) , Toast.LENGTH_LONG);
162                     ToastMessage.show();
163             }
164             else
165             {
166                     TransactionIDSend++;
167                     //Nachricht kann gesendet werden
168                     if(action == CameraControl.BROADCAST_SEND_MESSAGE)
169                     {
170                             int code;
171                             char count;
172                             char parametercount;
173                             byte[] paramter = new byte[10];
174                             int cameranumber;
175
176                             code = 0;
177                             String messageToSend = intent.
                                    getStringExtra(CameraControl.
                                    BROADCAST_EXTRA_SEND_MESSAGE);
178
179                             code = intent.getIntExtra(CameraControl.
                                    BROADCAST_EXTRA_SEND_MESSAGE_CODE, 0);
180                             parametercount = intent.getCharExtra(
                                    CameraControl.
                                    BROADCAST_EXTRA_SEND_MESSAGE_COUNT_PARAMETER
                                    , (char) 0);
181                             paramter=intent.getByteArrayExtra(
                                    CameraControl.
                                    BROADCAST_EXTRA_SEND_MESSAGE_PARAMETER);
182
183                             SendContainer(code,(char) parametercount,
                                    paramter,TransactionIDSend);
184
185                     }
186
187             }
188
189         }
190     };
191
192     @Override
193     public synchronized void onResume() {
194         super.onResume();
195         if(D) Log.e(TAG, "+ ON RESUME +");
196
197         // Performing this check in onResume() covers the case in which BT
                was
198         // not enabled during onStart(), so we were paused to enable it...
199         // onResume() will be called when ACTION_REQUEST_ENABLE activity
                returns.
200         if (mChatService != null) {
```

```
201             // Only if the state is STATE_NONE, do we know that we haven't
                    started already
202             if (mChatService.getState() == BluetoothChatService.STATE_NONE)
                    {
203               // Start the Bluetooth chat services
204               mChatService.start();
205             }
206         }
207     }
208
209     @Override
210     public void onDestroy() {
211         super.onDestroy();
212         // Stop the Bluetooth chat services
213         if (mChatService != null) mChatService.stop();
214         if(D) Log.e(TAG, "--- ON DESTROY ---");
215
216         unregisterReceiver(mReceiver);
217     }
218
219     @Override
220     public void onStart() {
221         super.onStart();
222         if(D) Log.e(TAG, "++ ON START ++");
223
224         // If BT is not on, request that it be enabled.
225         // setupChat() will then be called during onActivityResult
226         if (!mBluetoothAdapter.isEnabled()) {
227             Intent enableIntent = new Intent(BluetoothAdapter.
                        ACTION_REQUEST_ENABLE);
228             startActivityForResult(enableIntent, REQUEST_ENABLE_BT);
229         // Otherwise, setup the chat session
230         } else {
231             if (mChatService == null) setupChat();
232         }
233     }
234
235         private void setupChat() {
236             Log.d(TAG, "setupChat()");
237
238             // Initialize the BluetoothChatService to perform bluetooth
                        connections
239             mChatService = new BluetoothChatService(this, mHandler);
240
241             // Initialize the buffer for outgoing messages
242             mOutStringBuffer = new StringBuffer("");
243         }
244
245
246     @Override
247     public boolean onCreateOptionsMenu(Menu menu) {
248         MenuInflater inflater = getMenuInflater();
249         inflater.inflate(R.menu.option_menu, menu);
250         return true;
251     }
252
253     @Override
254     public boolean onOptionsItemSelected(MenuItem item) {
255         switch (item.getItemId()) {
```

```
256         case R.id.scan:
257             // Launch the DeviceListActivity to see devices and do scan
258             Intent serverIntent = new Intent(this, DeviceListActivity.class
                    );
259             startActivityForResult(serverIntent, REQUEST_CONNECT_DEVICE);
260             return true;
261         case R.id.discoverable:
262             // Ensure this device is discoverable by others
263             ensureDiscoverable();
264             return true;
265         }
266         return false;
267     }
268 
269 
270     // The Handler that gets information back from the BluetoothChatService
271     private final Handler mHandler = new Handler() {
272         @Override
273         public void handleMessage(Message msg) {
274             switch (msg.what) {
275             case MESSAGE_STATE_CHANGE:
276                 if(D) Log.i(TAG, "MESSAGE_STATE_CHANGE: " + msg.arg1);
277                 switch (msg.arg1) {
278                 case BluetoothChatService.STATE_CONNECTED:
279 
280                     break;
281                 case BluetoothChatService.STATE_CONNECTING:
282 
283                     break;
284                 case BluetoothChatService.STATE_LISTEN:
285                 case BluetoothChatService.STATE_NONE:
286 
287                     break;
288                 }
289                 break;
290             case MESSAGE_WRITE:
291                 //byte[] writeBuf = (byte[]) msg.obj;
292                 // construct a string from the buffer
293                 //String writeMessage = new String(writeBuf);
294 
295 
296                 break;
297             case MESSAGE_READ:
298                 byte[] readBuf = (byte[]) msg.obj;
299                 byte[] returnvalue = new byte[1];
300                 returnvalue[0] = (byte) 'o';
301                 // construct a string fromfff the valid bytes in the buffer
302                 //String readMessage = new String(readBuf,0,readBuf.
                        toString().length());
303 
304                 //ReceiveContainer(readBuf);
305 
306                 ReceiveBuffer[BufferElements]= readBuf[0];
307                 BufferElements++;
308 
309 
310 
311 
```

```
312                     //Erst warten bis mindestens 8 Elemente vorhanden
                            sind
313                     if(BufferElements + NumParametersRead >= 8)
314                     {
315                             //Anzahl Parameter einlesen
316                             NumParametersRead = (char) (((char)
                                    ReceiveBuffer[7]<<8) + (char)
                                    ReceiveBuffer[6]);
317                             TransactionIDRead = (char) (((char)
                                    ReceiveBuffer[1]<<8) + (char)
                                    ReceiveBuffer[0]);
318                             ResponseCode = (int) (((int) ReceiveBuffer
                                    [3]<<8)+((int) ReceiveBuffer[4]<<16)+((
                                    int) ReceiveBuffer[5]<<24)+((int)
                                    ReceiveBuffer[2]));
319
320                             //Wenn alles eingelesen wurde, beginnt die
                                    Verarbeitung
321                             //Muss noch einmal aufgerufen werden, damit
                                     die Bedingung nicht schon beim ersten
                                    Aufruf erfüllt ist
322                             if(BufferElements == 8+NumParametersRead)
323                             {
324                     Toast.makeText(getApplicationContext(), "
                            Datenpaket empfangen! Anzahl empfangener
                            Elemente: " + String.valueOf((int)
                            BufferElements) + "  gerade empfangen: " +String
                            .valueOf((int) ReceiveBuffer[BufferElements-1]),
                             Toast.LENGTH_SHORT).show();
325
326                     byte[] parameter = new byte[NumParametersRead];
327
328                     for(int i=0; i< NumParametersRead;i++)
329                     {
330                             parameter[i] = ReceiveBuffer[8+i];
331                     }
332
333                     //Intent auslösen
334                     Intent intent=new Intent();
335                     intent.setAction(TabMain.
                            BROADCAST_RECEIVED_CONTAINER);
336                     intent.putExtra(TabMain.
                            BROADCAST_RECEIVED_CONTAINER_TRANSACTIONID,
                            TransactionIDRead);
337                     intent.putExtra(TabMain.
                            BROADCAST_RECEIVED_CONTAINER_CODE, ResponseCode)
                            ;
338                     intent.putExtra(TabMain.
                            BROADCAST_RECEIVED_CONTAINER_NUMPARAMETERS,
                            NumParametersRead);
339                     intent.putExtra(TabMain.
                            BROADCAST_RECEIVED_CONTAINER_PARAMETER,
                            parameter);
340                     sendBroadcast(intent);
341                     //Buffer löschen
342                     EmptyBuffer();
343                             }
344                     }
345
```

```
346                 //o zurücksenden
347                 mChatService.write(returnvalue);
348
349                 break;
350             case MESSAGE_DEVICE_NAME:
351                 // save the connected device's name
352                 Toast.makeText(getApplicationContext(), "Connected to "
353                                 + msg.getData().getString(DEVICE_NAME),
                                    Toast.LENGTH_SHORT).show();
354                 break;
355             case MESSAGE_TOAST:
356                 Toast.makeText(getApplicationContext(), msg.getData().
                        getString(TOAST),
357                                 Toast.LENGTH_SHORT).show();
358             }
359         }
360     };
361
362     private void ensureDiscoverable() {
363         if(D) Log.d(TAG, "ensure discoverable");
364         if (mBluetoothAdapter.getScanMode() !=
365             BluetoothAdapter.SCAN_MODE_CONNECTABLE_DISCOVERABLE) {
366             Intent discoverableIntent = new Intent(BluetoothAdapter.
                        ACTION_REQUEST_DISCOVERABLE);
367             discoverableIntent.putExtra(BluetoothAdapter.
                        EXTRA_DISCOVERABLE_DURATION, 300);
368             startActivity(discoverableIntent);
369         }
370     }
371
372     /**
373      * Sends a message.
374      * @param message  A string of text to send.
375      */
376     private void sendMessage(byte[] message,int lengthContainer) {
377         // Check that we're actually connected before trying anything
378         if (mChatService.getState() != BluetoothChatService.STATE_CONNECTED
                ) {
379             Toast.makeText(this, R.string.not_connected, Toast.LENGTH_SHORT
                    ).show();
380             return;
381         }
382
383         // Check that there's actually something to send
384                 //if (message.length() > 0) {  XXX Delete not used
385         // Get the message bytes and tell the BluetoothChatService to write
386         int i;
387
388         byte[] send = new byte[lengthContainer];
389
390         for(i=0;i<lengthContainer;i++)
391         {
392                 send[i] = message[i];
393         }
394
395         mChatService.write(send);
396
397         // Reset out string buffer to zero and clear the edit text field
398         mOutStringBuffer.setLength(0);
```

```
399
400         //}  XXX Delete not used
401     }
402
403     public void onActivityResult(int requestCode, int resultCode, Intent
            data) {
404         if(D) Log.d(TAG, "onActivityResult " + resultCode);
405         switch (requestCode) {
406         case REQUEST_CONNECT_DEVICE:
407             // When DeviceListActivity returns with a device to connect
408             if (resultCode == Activity.RESULT_OK) {
409                 // Get the device MAC address
410                 String address = data.getExtras()
411                                      .getString(DeviceListActivity.
                                          EXTRA_DEVICE_ADDRESS);
412                 if(address != null)
413                 {
414                         deviceconnected = address;
415
416                         // Get the BluetoothDevice object
417                         BluetoothDevice device = mBluetoothAdapter.
                                getRemoteDevice(address);
418                         // Attempt to connect to the device
419                         mChatService.connect(device);
420                 }
421
422             }
423
424             break;
425         case REQUEST_ENABLE_BT:
426             // When the request to enable Bluetooth returns
427             if (resultCode == Activity.RESULT_OK) {
428                 // Bluetooth is now enabled, so set up a chat session
429                 setupChat();
430             } else {
431                 // User did not enable Bluetooth or an error occured
432                 Log.d(TAG, "BT not enabled");
433                 Toast.makeText(this, R.string.bt_not_enabled_leaving, Toast
                        .LENGTH_SHORT).show();
434                 finish();
435             }
436             break;
437         }
438     }
439
440         public String GetConnectedDevice(){
441                 return deviceconnected;
442         }
443
444         //CreateContainer
445         //Funktion die den zu übertragenden Container generiert
446         //Es wird keine dynamische Speicherverwaltung verwendet,
447         //damit auch größere Daten transferiert werden können
448         // @ Param1: Zu Übertragender Code
449         // @ Param2: Anzahl der Parameter
450         // @ Param3: Parameter Array
451         // @ Param4: Ab wann Parameter übertragen werden (inklusive) (0
                wenn alles in einem Container übertragen werden kann)
452         // @ Param5: Bis wohin Parameter übertragen werden (inklusive)
```

```
453         // @Param6: Container mit maximaler Größe
454         // @Param7: TransactionID die zu übertragen ist
455         private int CreateContainer(int  CodeWort, char countParam, byte
                arrayParam[], char start, char stop, byte Container[], char
                TransID)
456         {
457                 int       i;
458                 int       lengthContainer; //Länge des Containers in Byte
459
460                 if(start == 0) //Keine Teilung des Containers notwendig
461                 {
462                         //Länge des Containers bestimmen
463                         lengthContainer = 2 /*TransactionID*/ + 4 /*Code*/
                            + 2 /*AnzahlParameter*/ + countParam;
464
465                         for (i=0;i<lengthContainer;i++)
466                         {
467                                 switch(i)
468                                 {
469                                         case 0:
470                                         case 1:
471                                                 //TransaktionID
472                                                 Container[i] = (byte) (
                                                    TransID >> i*8) ;
473                                                 break;
474                                         case 2:
475                                         case 3:
476                                         case 4:
477                                         case 5:
478                                                 //Code
479                                                 Container[i] = (byte) (
                                                    CodeWort>> (i-2)*8);
480                                                 break;
481                                         case 6:
482                                         case 7:
483                                                 //Anzahl Parameter
484                                                 Container[i] = (byte) (
                                                    countParam>> (i-6)*8);
485                                                 break;
486                                         default:
487                                                 //Parameter
488                                                 Container[i] = arrayParam[i
                                                    -8];
489                                                 break;
490                                 }
491                         }
492                 }
493                 else //Nur die Parameter übertragen
494                 {
495                         //Länge des Containers bestimmen
496                         lengthContainer = stop - start + 1;
497                         for (i=0;i<stop-start +1 ;i++)
498                         {
499                                 switch(i)
500                                 {
501                                         default:
502                                                 //Parameter
503                                                 Container[i] = arrayParam[
                                                    start+i];
```

```
504                                                     break;
505                                             }
506                                     }
507                             }
508
509                             return  lengthContainer;
510             }
511
512             //SendContainer
513             //Funktion sendet den Container
514             //Abhängig von der Transportebene
515             //Container wird hier definiert
516             // @ Param1: Zu Übertragender Code
517             // @ Param2: Anzahl der Parameter
518             // @ Param3: Parameter Array
519             // @ Param4: zusendente TransaktionsID
520             // @ Return Value: 0 success, sonst <>  0
521             public char SendContainer(int  CodeWort, char countParam, byte
                    arrayParam[], char TransID)
522             {
523                             char ParameterLeft;
524                             char i;
525                             char countTransmitParameter;
526
527                             byte buffer[] = new byte[128];
528                             char start;
529                             char stop;
530                             char j;
531                             int lengthContainer;
532
533                             start = 0;
534                             stop = 0;
535                             i = 0;
536
537                             for(i=0;i<128;i++)
538                             {
539                                     buffer[i] = 0;
540                             }
541
542                             //maximale Anzahl der Parameter die übertragen sind
543                             //64-8 = 56
544                             //Wenn mehr sind, müssen mehrere Container gesendet werden
545
546                             ParameterLeft = countParam;
547
548                             if(ParameterLeft>56)
549                             {
550                                     countTransmitParameter = 56;
551                             }
552                             else
553                             {
554                                     countTransmitParameter = ParameterLeft;
555                             }
556
557                             do
558                             {
559                                     //Container generieren
560                                     lengthContainer = CreateContainer( CodeWort,
                                            countTransmitParameter,arrayParam,start,stop,
```

```
                    buffer,TransID);
561
562                 //CreateContainer
563                 //CreateContainer(CodeWort,countTransmitParameter,
                        arrayParam,start,stop,buffer,TransID);
564
565                 //Abhängig von der Transportebene Start
566                 //
                        ------------------------------------------------------------------------
567
568                 //Bluetooth
569                 sendMessage(buffer,lengthContainer);
570
571                 //
                        ------------------------------------------------------------------------
572                 //Abhängig von der Transportebene Ende
573
574                 if(start == 0)
575                 {
576                         //Erster Durchlauf
577                         if(ParameterLeft>56)
578                         {
579                                 start = 56;
580                                 ParameterLeft-=56;
581                         }
582                         else
583                         {
584                                 ParameterLeft=0;
585                         }
586                 }
587                 else
588                 {
589                         //Xter Durchlauf
590                         if(ParameterLeft>64)
591                         {
592                                 start = (char) (start + 64);
593                                 ParameterLeft-=64;
594                         }
595                         else
596                         {
597                                 start = (char) (start + 64);
598                                 ParameterLeft=0;
599                         }
600                 }
601
602                 //Stop setzen
603                 if(ParameterLeft>64)
604                 {
605                         stop = (char) (start+64);
606                         countTransmitParameter = 64;
607                 }
608                 else
609                 {
610                         stop = (char) (start + ParameterLeft);
611                         countTransmitParameter = ParameterLeft;
612                 }
613         }while(ParameterLeft>0);
```

```
614
615
616                     return 0;
617             }
618
619
620 }
```

Strings.xml - Deutsch

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3      <string name="app_name">Camera Control</string>
4      <string name="TV_Value">TV Ã€ndern</string>
5      <string name="TV_Spinner">TV auswÃ€hlen</string>
6      <string-array name="TV_SpinnerArray">
7                     <item>Bulb</item>
8                     <item>30\"</item>
9  ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<item>25\"</item>
10                    <item>20\"</item>
11 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<item>20\"(1/3)</item>
12                    <item>15\"</item>
13 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<item>13\"</item>
14                    <item>10\"</item>
15 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<item>10\"(1/3)</item>
16                    <item>8\"</item>
17 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<item>6\"(1/3)</item>
18                    <item>6\"</item>
19 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<item>5\"</item>
20                    <item>4\"</item>
21 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<item>3\"2</item>
22                    <item>3\"</item>
23 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<item>2\"5</item>
24                    <item>2\"</item>
25 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<item>1\"6</item>
26                    <item>1\"5</item>
27 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<item>1\"3</item>
28                    <item>1\"</item>
29 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<item>0\"8</item>
30                    <item>0\"7</item>
31 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<item>0\"6</item>
32                    <item>0\"5</item>
33 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<item>0\"4</item>
34                    <item>0\"3</item>
35 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<item>0\"3(1/3)</item>
36                    <item>1/4</item>
37                    <item>1/5</item>
38                    <item>1/6</item>
39                    <item>1/6(1/3)</item>
40                    <item>1/8</item>
41                    <item>1/10(1/3)</item>
42                    <item>1/10</item>
43                    <item>1/13</item>
44                    <item>1/15</item>
45                    <item>1/20(1/3)</item>
46                    <item>1/20</item>
47                    <item>1/25</item>
48                    <item>1/30</item>
49                    <item>1/40</item>
```

```
50                  <item>1/45</item>
51                  <item>1/50</item>
52                  <item>1/60</item>
53                  <item>1/80</item>
54                  <item>1/90</item>
55                  <item>1/100</item>
56                  <item>1/125</item>
57                  <item>1/160</item>
58                  <item>1/180</item>
59                  <item>1/200</item>
60                  <item>1/250</item>
61                  <item>1/320</item>
62                  <item>1/350</item>
63                  <item>1/400</item>
64                  <item>1/500</item>
65                  <item>1/640</item>
66                  <item>1/750</item>
67                  <item>1/800</item>
68                  <item>1/1000</item>
69                  <item>1/1250</item>
70                  <item>1/1500</item>
71                  <item>1/1600</item>
72                  <item>1/2000</item>
73                  <item>1/2500</item>
74                  <item>1/3000</item>
75                  <item>1/3200</item>
76                  <item>1/4000</item>
77                  <item>1/5000</item>
78                  <item>1/6000</item>
79                  <item>1/6400</item>
80                  <item>1/8000</item>
81      </string-array>
82      <string name="TV_Button">TV gedrÃŒckt</string>
83      <string name="AV_Value">AV Ã€ndern</string>
84      <string name="AV_Spinner">AV auswÃ€hlen</string>
85      <string-array name="AV_SpinnerArray">
86                  <item>1</item>
87                  <item>1.1</item>
88                  <item>1.2</item>
89                  <item>1.2(1/3)</item>
90                  <item>1.4</item>
91                  <item>1.6</item>
92                  <item>1.8</item>
93                  <item>1.8(1/3)</item>
94                  <item>2</item>
95                  <item>2.2</item>
96                  <item>2.5</item>
97                  <item>2.5(1/3)</item>
98                  <item>2.8</item>
99                  <item>3.2</item>
100                 <item>3.5</item>
101                 <item>3.5(1/3)</item>
102                 <item>4</item>
103                 <item>4.5</item>
104                 <item>4.5</item>
105                 <item>5.0</item>
106                 <item>5.6</item>
107                 <item>6.3</item>
108                 <item>6.7</item>
```

```
109                 <item>7.1</item>
110                 <item>8</item>
111                 <item>9</item>
112                 <item>9.5</item>
113                 <item>10</item>
114                 <item>11</item>
115                 <item>13(1/3)</item>
116                 <item>13</item>
117                 <item>14</item>
118                 <item>16</item>
119                 <item>18</item>
120                 <item>19</item>
121                 <item>20</item>
122                 <item>22</item>
123                 <item>25</item>
124                 <item>27</item>
125                 <item>29</item>
126                 <item>32</item>
127                 <item>36</item>
128                 <item>38</item>
129                 <item>40</item>
130                 <item>45</item>
131                 <item>51</item>
132                 <item>54</item>
133                 <item>57</item>
134                 <item>64</item>
135                 <item>72</item>
136                 <item>76</item>
137                 <item>80</item>
138                 <item>91</item>
139     </string-array>
140     <string name="AV_Button">AV gedrÃŒckt</string>
141     <string name="Bel_Value">Bel Ã€ndern</string>
142     <string name="Bel_Spinner">Bel auswÃ€hlen</string>
143     <string-array name="Bel_SpinnerArray">
144                 <item>+3</item>
145                 <item>+2 2/3</item>
146                 <item>+2 1/2</item>
147                 <item>+2 1/3</item>
148                 <item>+2</item>
149                 <item>+1 2/3</item>
150                 <item>+1 1/2</item>
151                 <item>+1 1/3</item>
152                 <item>+1</item>
153                 <item>+2/3</item>
154                 <item>+1/2</item>
155                 <item>+1/3</item>
156                 <item>0</item>
157                 <item>-1/3</item>
158                 <item>-1/2</item>
159                 <item>-2/3</item>
160                 <item>-1</item>
161                 <item>-1 1/3</item>
162                 <item>-1 1/2</item>
163                 <item>-1 2/3</item>
164                 <item>-2</item>
165                 <item>-2 1/3</item>
166                 <item>-2 1/2</item>
167                 <item>-2 2/3</item>
```

```
168                 <item>-3</item>
169     </string-array>
170     <string name="Bel_Button">Bel gedrÃŒckt</string>
171     <string name="unknown_Button">Button nicht gefunden</string>
172     <string name="SessionOpen_Button">Session Ã¶ffnen</string>
173     <string name="SessionOpen_Value">Session wird geÃ¶ffnet</string>
174     <string name="SessionClose_Button">Session beenden</string>
175     <string name="SessionClose_Value">Session wird geschlossen</string>
176     <string name="TakeFoto_Button">Foto aufnehmen</string>
177     <string name="TakeFoto_Value">Foto aufnehmen</string>
178     <string name="LiveViewOn_Button">LiveView an</string>
179     <string name="LiveViewOn_Value">LiveView an</string>
180     <string name="LiveViewOff_Button">LiveView aus</string>
181     <string name="LiveViewOff_Value">LiveView aus</string>
182     <string name="Focus_Spinner">Fokus auswÃ€hlen</string>
183     <string name="Focus_Value">Fokus Ã€ndern</string>
184     <string-array name="Focus_SpinnerArray">
185         <item>Near 1</item>
186         <item>Near 2</item>
187         <item>Near 3</item>
188         <item>Far 1</item>
189         <item>Far 2</item>
190         <item>Far 3</item>
191     </string-array>
192     <string name="Focus_Button">Fokus gedrÃŒckt</string>
193     <string name="Tab_Cam1">Kamera Host 1</string>
194     <string name="Tab_Cam2">Kamera Host 2</string>
195     <string name="Tab_Setting">Steuerung</string>
196     <string name="No_Bluetooth_Exist">Kein Bluetooth am GerÃ€t vorhanden!</
        string>
197     <string name="Bluetooth_Exist">Bluetooth am GerÃ€t vorhanden!</string>
198     <string name="Bluetooth_State_enable">Bluetooth ist aktiviert!</string>
199     <string name="Bluetooth_State_disable">Bluetooth ist deaktiviert!</
        string>
200     <string name="SetBluetooth">Bluetooth testen!</string>
201     <string name="connect">verbinden</string>
202     <string name="discoverable">GerÃ€t sichtbar machen</string>
203     <string name="enable">Bluetooth einschalten</string>
204     <string name="title_paired_devices">Paired GerÃ€te</string>
205     <string name="title_other_devices">gefundene GerÃ€te</string>
206     <string name="button_scan">GerÃ€te suchen</string>
207     <string name="none_paired">Kein GerÃ€t gepaired!</string>
208     <string name="none_found">Kein GerÃ€t gefunden!</string>
209     <string name="select_device">GerÃ€t auswÃ€hlen!</string>
210     <string name="scanning">Suchen!</string>
211     <string name="bt_not_enabled_leaving">Bluetooth ist nicht eingeschaltet
        !</string>
212     <string name="not_connected">Nicht verbunden!</string>
213     <string name="services_not_started">Service wurde nicht gestartet!</
        string>
214     <string name="Times3D">Anzahl Fotos:  </string>
215     <string name="Start3D">Start</string>
216     <string name="Stop3D">Stop</string>
217     <string name="Timelaps">Zeitraffer</string>
218     <string name="StartTimelaps">Start</string>
219     <string name="StopTimelaps">Stop</string>
220     <string name="TimesTimelapsAmount">Anzahl der Aufnahmen:</string>
```

```
221     <string name="TimesTimelapsTime">Intervall (s):</string>
222     <string name="ReadState">Port A Status:</string>
223     <string name="StringReadState">DDRA:</string>
224     <string name="StringReadStateActual">PINA:</string>
225     <string name="GetConnection">USB Host Status:</string>
226     <string name="GetClass">Klasse vom Modul herausfinden</string>
227     <string name="Start3DHDR">Start 3D HDR</string>
228     <string name="Stop3DHDR">Stop 3D HDR</string>
229 </resources>
```

Strings.xml - Englisch

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3      <string name="app_name">Camera Control</string>
4      <string name="TV_Value">change TV</string>
5      <string name="TV_Spinner">Select TV</string>
6      <string-array name="TV_SpinnerArray">
7                  <item>Bulb</item>
8                  <item>30\"</item>
9  ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<item>25\"</item>
10                 <item>20\"</item>
11 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<item>20\"(1/3)</item>
12                 <item>15\"</item>
13 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<item>13\"</item>
14                 <item>10\"</item>
15 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<item>10\"(1/3)</item>
16                 <item>8\"</item>
17 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<item>6\"(1/3)</item>
18                 <item>6\"</item>
19 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<item>5\"</item>
20                 <item>4\"</item>
21 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<item>3\"2</item>
22                 <item>3\"</item>
23 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<item>2\"5</item>
24                 <item>2\"</item>
25 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<item>1\"6</item>
26                 <item>1\"5</item>
27 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<item>1\"3</item>
28                 <item>1\"</item>
29 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<item>0\"8</item>
30                 <item>0\"7</item>
31 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<item>0\"6</item>
32                 <item>0\"5</item>
33 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<item>0\"4</item>
34                 <item>0\"3</item>
35 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣<item>0\"3(1/3)</item>
36                 <item>1/4</item>
37                 <item>1/5</item>
38                 <item>1/6</item>
39                 <item>1/6(1/3)</item>
40                 <item>1/8</item>
41                 <item>1/10(1/3)</item>
42                 <item>1/10</item>
43                 <item>1/13</item>
44                 <item>1/15</item>
45                 <item>1/20(1/3)</item>
46                 <item>1/20</item>
47                 <item>1/25</item>
```

```
48                  <item>1/30</item>
49                  <item>1/40</item>
50                  <item>1/45</item>
51                  <item>1/50</item>
52                  <item>1/60</item>
53                  <item>1/80</item>
54                  <item>1/90</item>
55                  <item>1/100</item>
56                  <item>1/125</item>
57                  <item>1/160</item>
58                  <item>1/180</item>
59                  <item>1/200</item>
60                  <item>1/250</item>
61                  <item>1/320</item>
62                  <item>1/350</item>
63                  <item>1/400</item>
64                  <item>1/500</item>
65                  <item>1/640</item>
66                  <item>1/750</item>
67                  <item>1/800</item>
68                  <item>1/1000</item>
69                  <item>1/1250</item>
70                  <item>1/1500</item>
71                  <item>1/1600</item>
72                  <item>1/2000</item>
73                  <item>1/2500</item>
74                  <item>1/3000</item>
75                  <item>1/3200</item>
76                  <item>1/4000</item>
77                  <item>1/5000</item>
78                  <item>1/6000</item>
79                  <item>1/6400</item>
80                  <item>1/8000</item>
81      </string-array>
82      <string name="TV_Button">TV pressed</string>
83      <string name="AV_Value">AV change</string>
84      <string name="AV_Spinner">Select AV</string>
85      <string-array name="AV_SpinnerArray">
86                  <item>1</item>
87                  <item>1.1</item>
88                  <item>1.2</item>
89                  <item>1.2(1/3)</item>
90                  <item>1.4</item>
91                  <item>1.6</item>
92                  <item>1.8</item>
93                  <item>1.8(1/3)</item>
94                  <item>2</item>
95                  <item>2.2</item>
96                  <item>2.5</item>
97                  <item>2.5(1/3)</item>
98                  <item>2.8</item>
99                  <item>3.2</item>
100                 <item>3.5</item>
101                 <item>3.5(1/3)</item>
102                 <item>4</item>
103                 <item>4.5</item>
104                 <item>4.5</item>
105                 <item>5.0</item>
106                 <item>5.6</item>
```

```
107                 <item>6.3</item>
108                 <item>6.7</item>
109                 <item>7.1</item>
110                 <item>8</item>
111                 <item>9</item>
112                 <item>9.5</item>
113                 <item>10</item>
114                 <item>11</item>
115                 <item>13(1/3)</item>
116                 <item>13</item>
117                 <item>14</item>
118                 <item>16</item>
119                 <item>18</item>
120                 <item>19</item>
121                 <item>20</item>
122                 <item>22</item>
123                 <item>25</item>
124                 <item>27</item>
125                 <item>29</item>
126                 <item>32</item>
127                 <item>36</item>
128                 <item>38</item>
129                 <item>40</item>
130                 <item>45</item>
131                 <item>51</item>
132                 <item>54</item>
133                 <item>57</item>
134                 <item>64</item>
135                 <item>72</item>
136                 <item>76</item>
137                 <item>80</item>
138                 <item>91</item>
139     </string-array>
140     <string name="AV_Button">AV pressed</string>
141     <string name="Bel_Value">Change Exp</string>
142     <string name="Bel_Spinner">Select Exp</string>
143     <string-array name="Bel_SpinnerArray">
144                 <item>+3</item>
145                 <item>+2 2/3</item>
146                 <item>+2 1/2</item>
147                 <item>+2 1/3</item>
148                 <item>+2</item>
149                 <item>+1 2/3</item>
150                 <item>+1 1/2</item>
151                 <item>+1 1/3</item>
152                 <item>+1</item>
153                 <item>+2/3</item>
154                 <item>+1/2</item>
155                 <item>+1/3</item>
156                 <item>0</item>
157                 <item>-1/3</item>
158                 <item>-1/2</item>
159                 <item>-2/3</item>
160                 <item>-1</item>
161                 <item>-1 1/3</item>
162                 <item>-1 1/2</item>
163                 <item>-1 2/3</item>
164                 <item>-2</item>
165                 <item>-2 1/3</item>
```

```
166                 <item>-2 1/2</item>
167                 <item>-2 2/3</item>
168                 <item>-3</item>
169     </string-array>
170     <string name="Bel_Button">Exp pressed</string>
171     <string name="unknown_Button">Button not found</string>
172     <string name="SessionOpen_Button">Session open</string>
173     <string name="SessionOpen_Value">Opening Session</string>
174     <string name="SessionClose_Button">Session close</string>
175     <string name="SessionClose_Value">Closing Session</string>
176     <string name="TakeFoto_Button">Shoot Picture</string>
177     <string name="TakeFoto_Value">Shoot Picture</string>
178     <string name="LiveViewOn_Button">LiveView on</string>
179     <string name="LiveViewOn_Value">LiveView on</string>
180     <string name="LiveViewOff_Button">LiveView off</string>
181     <string name="LiveViewOff_Value">LiveView off</string>
182     <string name="Focus_Spinner">Select Focus</string>
183     <string name="Focus_Value">Change Focus</string>
184     <string-array name="Focus_SpinnerArray">
185         <item>Near 1</item>
186         <item>Near 2</item>
187         <item>Near 3</item>
188         <item>Far 1</item>
189         <item>Far 2</item>
190         <item>Far 3</item>
191     </string-array>
192     <string name="Focus_Button">Focus pressed</string>
193     <string name="Tab_Cam1">Camera Host 1</string>
194     <string name="Tab_Cam2">Camera Host 2</string>
195     <string name="Tab_Setting">Settings</string>
196     <string name="No_Bluetooth_Exist">No Bluetooth available!</string>
197     <string name="Bluetooth_Exist">Bluetooth available!</string>
198     <string name="Bluetooth_State_enable">Bluetooth activated!</string>
199     <string name="Bluetooth_State_disable">Bluetooth disabled!</string>
200     <string name="SetBluetooth">Bluetooth Test!</string>
201     <string name="connect">Connect</string>
202     <string name="discoverable">Discover Device</string>
203     <string name="enable">Bluetooth on</string>
204     <string name="title_paired_devices">Paired Devices</string>
205     <string name="title_other_devices">Devices found</string>
206     <string name="button_scan">Device Search</string>
207     <string name="none_paired">No paired device!</string>
208     <string name="none_found">No device found!</string>
209     <string name="select_device">Select device!</string>
210     <string name="scanning">Search!</string>
211     <string name="bt_not_enabled_leaving">Bluetooth isn't␣turned␣on'!</
        string>
212     <string name="not_connected">Not connected!</string>
213     <string name="services_not_started">Service not started!</string>
214     <string name="Times3D">Amount Pictures: </string>
215     <string name="Start3D">Start</string>
216     <string name="Stop3D">Stop</string>
217     <string name="Timelaps">Time Laps</string>
218     <string name="StartTimelaps">Start</string>
219     <string name="StopTimelaps">Stop</string>
220     <string name="TimesTimelapsAmount">Amount Pictures:</string>
221     <string name="TimesTimelapsTime">Interval (s):</string>
222     <string name="ReadState">Port A Status:</string>
223     <string name="StringReadState">DDRA:</string>
```

```
224     <string name="StringReadStateActual">PINA:</string>
225     <string name="GetConnection">USB Host Status:</string>
226     <string name="GetClass">Find Class from Module</string>
227     <string name="Start3DHDR">Start 3D HDR</string>
228     <string name="Stop3DHDR">Stop 3D HDR</string>
229 </resources>
```

J ShutterSpeedTester.exe

Main.xaml

```
1  ï»¿<Window x:Class="ShutterSpeedTester.Main"
2      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4      Title="Shutter Speed Test" Height="305" Width="658" Closed="
           Window_Closed" Icon="/ShutterSpeedTester;component/Kamera.ico">
5      <Grid Background="BurlyWood">
6          <Button Height="23" HorizontalAlignment="Left" Margin="12,12,0,0"
               Name="buttonConfiguration" VerticalAlignment="Top" Width="116"
               Click="buttonConfiguration_Click">Configuration</Button>
7          <Button HorizontalAlignment="Left" Margin="12,46,0,0" Name="
               buttonLoadImages" Width="201" Click="buttonLoadImages_Click"
               Height="23" VerticalAlignment="Top">Load Images for Check</
               Button>
8
9          <ListView  Name="ListImages" Margin="12,76,15.5,38" KeyDown="
               ListImages_KeyDown">
10             <ListView.View>
11                 <GridView>
12                     <GridViewColumn DisplayMemberBinding="{Binding Path=
                           Path}" Header="Image" Width="350"/>
13                     <GridViewColumn DisplayMemberBinding="{Binding Path=
                           Status}" Header="Status" Width="100"/>
14                     <GridViewColumn DisplayMemberBinding="{Binding Path=
                           Result}" Header="Result CP" Width="150"/>
15                 </GridView>
16             </ListView.View>
17         </ListView>
18
19         <Button Height="23" HorizontalAlignment="Right" Margin="0,0,12,6"
               Name="buttonClose" VerticalAlignment="Bottom" Width="75" Click="
               buttonClose_Click">Close</Button>
20         <StackPanel Height="47" HorizontalAlignment="Right" Margin="
               0,23,12,0" Name="stackPanel1" VerticalAlignment="Top" Width="82"
               >
21             <Button Height="23" Name="buttonStart" Width="75" Click="
                   buttonStart_Click">Start</Button>
22             <Button Height="23" Name="buttonExport" Width="75" Click="
                   buttonExport_Click">Export</Button>
23         </StackPanel>
24         <Label Height="28" HorizontalAlignment="Left" Margin="133,10.52,0,0
               " Name="label1" VerticalAlignment="Top" Width="120">Config
               Picture:</Label>
25         <TextBox Height="28" Margin="218,15,104,0" Name="textBoxBitmap"
               VerticalAlignment="Top" IsReadOnly="True" Background="BurlyWood"
                BorderThickness="0">Not selected!</TextBox>
26     </Grid>
27 </Window>
```

Main.xaml.cs

```
1  ï»¿using System;
2  using System.IO;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
```

```
 6 using System.Windows;
 7 using System.Windows.Controls;
 8 using System.Windows.Data;
 9 using System.Windows.Documents;
10 using System.Windows.Input;
11 using System.Windows.Media;
12 using System.Windows.Media.Imaging;
13 using System.Windows.Shapes;
14 using System.Collections;
15
16 namespace ShutterSpeedTester
17 {
18
19     /// <summary>
20     /// Interaction logic for Main.xaml
21     /// </summary>
22     ///
23     public partial class Main : Window
24     {
25         ArrayList AControlPoints = new ArrayList();
26         ArrayList AThreshold = new ArrayList();
27         List<ImagesContainer> ListImagesContainer = new List<
               ImagesContainer>();
28         Window1 configWindow = null;
29
30
31         public Main()
32         {
33             InitializeComponent();
34             ListImages.ItemsSource = ListImagesContainer;
35         }
36
37         private void buttonClose_Click(object sender, RoutedEventArgs e)
38         {
39             this.Close();
40         }
41
42         private void buttonLoadImages_Click(object sender, RoutedEventArgs
               e)
43         {
44             Microsoft.Win32.OpenFileDialog fileDlg = new Microsoft.Win32.
                   OpenFileDialog();
45             fileDlg.Title = "Select images";
46             fileDlg.DefaultExt = ".jpg";
47             fileDlg.Filter = "Picture (.jpg)|*.jpg|All Files (*.*)|*.*";
48             fileDlg.FileOk +=
49             new System.ComponentModel.CancelEventHandler(fileDlg_FileOk);
50             fileDlg.Multiselect = true;
51             fileDlg.ShowDialog();
52         }
53
54         void fileDlg_FileOk(object sender, System.ComponentModel.
               CancelEventArgs e)
55         {
56             try
57             {
58                 foreach(string filename in (sender as Microsoft.Win32.
                       OpenFileDialog).FileNames)
59                 {
```

```
60                      ListImagesContainer.Add(new ImagesContainer(filename));
61                  }
62
63                  ListImages.Items.Refresh();
64              }
65              catch(Exception ex)
66              {
67                  MessageBox.Show("Something is going wrong! " + ex.Message,
                        "Error", MessageBoxButton.OK,MessageBoxImage.Error);
68              }
69          }
70
71          private void buttonConfiguration_Click(object sender,
                RoutedEventArgs e)
72          {
73              configWindow = new Window1(AControlPoints, AThreshold, this.
                    textBoxBitmap);
74              configWindow.ShowDialog();
75              GC.Collect();
76          }
77
78          private void buttonStart_Click(object sender, RoutedEventArgs e)
79          {
80              Color Threshold = Colors.AliceBlue;
81              //Kontrolle ob Checkpoints vorhanden sind
82              if (AControlPoints.Count == 0)
83              {
84                  MessageBox.Show("You must define Control Points first in
                        the configuration!", "Error", MessageBoxButton.OK,
                        MessageBoxImage.Information);
85                  return;
86              }
87
88              if (AThreshold.Count == 0)
89              {
90                  MessageBox.Show("You must define a threshold first in the
                        configuration!", "Error", MessageBoxButton.OK,
                        MessageBoxImage.Information);
91                  return;
92              }
93              else
94              {
95                  foreach (Threshold TR in AThreshold)
96                  {
97                      Threshold = TR.GetRGBThreshold();
98                  }
99              }
100
101             //Bilder kÃ¶nnen kontrolliert werden
102             foreach(ImagesContainer ImageContainer in ListImagesContainer)
103             {
104                 BitmapCalculation bitmapCalculation = new BitmapCalculation
                        (ImageContainer.Path);
105                 Color ImageRGB;
106
107                 ImageContainer.Result = "";
108
109                 foreach(ControlPoints CPoint in AControlPoints)
110                 {
```

```
111                     ImageRGB = bitmapCalculation.CalculateValues(CPoint.
                            GetPictureCoord().X + CPoint.GetRadiusPicture(),
                            CPoint.GetPictureCoord().Y + CPoint.GetRadiusPicture
                            (), CPoint.GetRadiusPicture());
112
113                     if (ImageRGB.R >= Threshold.R && ImageRGB.G >=
                            Threshold.G && ImageRGB.B >= Threshold.B)
114                         ImageContainer.Result = "1" + ImageContainer.Result
                                ;
115                     else
116                         ImageContainer.Result = "0" + ImageContainer.Result
                                ;
117                 }
118
119                 ImageContainer.Status = "Checked";
120             }
121
122             this.ListImages.Items.Refresh();
123
124             MessageBox.Show("Finished!", "Information", MessageBoxButton.OK
                    , MessageBoxImage.Information);
125         }
126
127         private void Window_Closed(object sender, EventArgs e)
128         {
129             if (configWindow != null)
130                 configWindow.Close();
131         }
132
133         private void ListImages_KeyDown(object sender, KeyEventArgs e)
134         {
135             if (e.Key == Key.Delete)
136             {
137                 //Selection abfragen
138                 foreach(ImagesContainer obj in this.ListImages.
                        SelectedItems)
139                 {
140                     ListImagesContainer.Remove(obj);
141                 }
142
143                 this.ListImages.Items.Refresh();
144             }
145         }
146
147         private void buttonExport_Click(object sender, RoutedEventArgs e)
148         {
149             Microsoft.Win32.SaveFileDialog fileDlg = new Microsoft.Win32.
                    SaveFileDialog();
150             fileDlg.Title = "Save␣File";
151             fileDlg.DefaultExt = ".csv";
152             fileDlg.Filter = "Excel␣(.csv)|*.csv|Text␣File␣(.txt)|*.txt";
153             fileDlg.FileOk +=
154             new System.ComponentModel.CancelEventHandler(fileSaveDlg_FileOk
                    );
155             fileDlg.ShowDialog();
156         }
157
158         void fileSaveDlg_FileOk(object sender, System.ComponentModel.
                CancelEventArgs e)
```

```
159         {
160             try
161             {
162                 foreach (string filename in (sender as Microsoft.Win32.
                        SaveFileDialog).FileNames)
163                 {
164                     StreamWriter myWriter = File.CreateText(filename);
165                     int countCheckPoints = AControlPoints.Count;
166 
167                     string header = "Pfad";
168                     string result = "";
169 
170                     for (int i = countCheckPoints; i >=1; i--)
171                     {
172                         header =  header + ";CP" + i.ToString();
173                     }
174                     myWriter.WriteLine(header);
175 
176                     foreach(ImagesContainer ImageContainer in
                            ListImagesContainer)
177                     {
178                         result = ImageContainer.Path;
179 
180                         for (int i = 1; i <= countCheckPoints; i++)
181                         {
182                             if (ImageContainer.Result.Length != 0)
183                                 result = result + ";" + ImageContainer.
                                        Result.Substring(i - 1, 1);
184                             else
185                                 result = result + ";";
186                         }
187 
188                         myWriter.WriteLine(result);
189                     }
190 
191                     myWriter.Close();
192                 }
193 
194                 ListImages.Items.Refresh();
195             }
196             catch (Exception ex)
197             {
198                 MessageBox.Show("Something␣is␣going␣wrong!␣" + ex.Message,
                        "Error", MessageBoxButton.OK, MessageBoxImage.Error);
199             }
200         }
201 
202     }
203 }
```

Window1.xaml

```
1  ï»¿<Window x:Class="ShutterSpeedTester.Window1"
2      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4      Title="Configuration" Height="520" Width="726" Closed="Window_Closed"
           Icon="/ShutterSpeedTester;component/Kamera.ico">
5      <Grid>
```

```
6           <Grid HorizontalAlignment="Left" Name="grid1" Width="146"
                Background="BurlyWood">
7               <Button Height="25" Margin="35,5,35,0" Name="ButtonImageLoad"
                    VerticalAlignment="Top" Click="BImageLoad_Click">Open Image<
                    /Button>
8               <TabControl Margin="5,36,4,37" Name="tabControl1">
9                   <TabItem Header ="Check Points" Name="tabCheckPoints"
                        IsSelected="True" GotFocus="tabCheckPoints_GotFocus">
10                      <Grid>
11                          <ToggleButton Height="25" Margin="6,15,6,0" Name="
                                toggleButtonSetCheckPoints" VerticalAlignment="
                                Top" Click="toggleButtonSetCheckPoints_Click">
                                Define Check Points</ToggleButton>
12                          <Label Height="28" Margin="16,46,63,0" Name="
                                labelRadiusCP" VerticalAlignment="Top">Radius</
                                Label>
13                          <TextBox Height="23" Margin="59,46,25,0" Name="
                                textBoxRadiusCP" VerticalAlignment="Top" KeyDown
                                ="textBoxRadiusCP_KeyDown"
                                HorizontalContentAlignment="Right" />
14                          <Label Height="28" HorizontalAlignment="Right"
                                Margin="0,46,6,0" Name="label1"
                                VerticalAlignment="Top" Width="22">px
15                          </Label>
16                          <ListView  Name="ListCheckPoints" Margin="6,75,6,0"
                                 SelectionChanged="
                                ListCheckPoints_SelectionChanged" KeyDown="
                                ListCheckPoints_KeyDown" SelectionMode="Single">
17                              <ListView.View>
18                                  <GridView>
19                                      <GridViewColumn DisplayMemberBinding="{
                                            Binding Path=Number}" Header="Nr"
                                            Width="25"/>
20                                      <GridViewColumn DisplayMemberBinding="{
                                            Binding Path=Coordinates}" Header="
                                            Coordinates"/>
21                                      <GridViewColumn DisplayMemberBinding="{
                                            Binding Path=Radius}" Header="Radius
                                            "/>
22                                  </GridView>
23                              </ListView.View>
24                          </ListView>
25                      </Grid>
26                  </TabItem>
27                  <TabItem Header="Threshold" Name="tabthreshold" IsSelected=
                        "True"  GotFocus="tabthreshold_GotFocus">
28                      <Grid>
29                          <ToggleButton   Height="25" Margin="6,15,6,0" Name=
                                "toggleButtonSetThresPoint" VerticalAlignment="
                                Top" Click="toggleButtonSetThresPoint_Click">Set
                                 Threshold</ToggleButton>
30                          <Label Height="28" Margin="16,46,63,0" Name="
                                labelRadiusThres" VerticalAlignment="Top">Radius
                                </Label>
31                          <TextBox  Height="23" Margin="59,46,25,0" Name="
                                textBoxRadiusThres" VerticalAlignment="Top"
                                KeyDown="textBoxRadiusCP_KeyDown"
                                HorizontalContentAlignment="Right" />
```

```
32                          <Label Height="28" HorizontalAlignment="Right"
                                Margin="0,46,6,0" Name="label2"
                                VerticalAlignment="Top" Width="22">px
33                          </Label>
34                          <GroupBox Header="RGB" Name="groupBoxRGB" Margin="
                                6,73,6,0" Height="56" VerticalAlignment="Top">
35                              <StackPanel Height="29" Name="stackPanelRGB"
                                    Width="102" Orientation="Horizontal">
36                                  <TextBox Height="23" Name="textBoxRGB_R"
                                        Width="32" IsEnabled="False" Background=
                                        "DarkGray" HorizontalContentAlignment="
                                        Right" />
37                                  <TextBox Height="23" Name="textBoxRGB_G"
                                        Width="32" IsEnabled="False" Background=
                                        "DarkGray" HorizontalContentAlignment="
                                        Right" />
38                                  <TextBox Height="23" Name="textBoxRGB_B"
                                        Width="32" IsEnabled="False" Background=
                                        "DarkGray" HorizontalContentAlignment="
                                        Right" />
39                              </StackPanel>
40                          </GroupBox>
41                          <GroupBox Header="Threshold" Name="
                                groupBoxRGBThreshold" Margin="6,138,6,0" Height=
                                "61.08" VerticalAlignment="Top">
42                              <StackPanel Height="29" Name="
                                    stackPanelThreshooldRGB" Width="102"
                                    Orientation="Horizontal" >
43                                  <TextBox  Height="23" Name="
                                        textBoxThresholdRGB_R" Width="32"
                                        KeyDown="textBoxThresholdRGB_R_KeyDown"
                                        HorizontalContentAlignment="Right" />
44                                  <TextBox  Height="23" Name="
                                        textBoxThresholdRGB_G" Width="32"
                                        KeyDown="textBoxThresholdRGB_R_KeyDown"
                                        HorizontalContentAlignment="Right" />
45                                  <TextBox  Height="23" Name="
                                        textBoxThresholdRGB_B" Width="32"
                                        KeyDown="textBoxThresholdRGB_R_KeyDown"
                                        HorizontalContentAlignment="Right" />
46                              </StackPanel>
47                          </GroupBox>
48                      </Grid>
49
50                  </TabItem>
51              </TabControl>
52              <Button Height="25" Name="buttonClose" Margin="35,0,35,8"
                    VerticalAlignment="Bottom" Click="buttonClose_Click">Close</
                    Button>
53
54          </Grid>
55          <Grid Margin="148,0,0,0" Name="grid2" Background="black"
                SizeChanged="TheImage_SizeChanged">
56              <Image Name="TheImage" Stretch="Uniform"  SizeChanged="
                    TheImage_SizeChanged" MouseLeftButtonDown="
                    TheImage_MouseLeftButtonDown" />
57          </Grid>
58          <Canvas Margin="148,0,0,0" Name="canvas1" >
```

```
59              <Canvas Name="canvasimage"  Height="100" Width="100" Canvas.
                    Left="0" Canvas.Top="0">
60
61              </Canvas>
62          </Canvas>
63      </Grid>
64  </Window>
```

Window1.xaml.cs

```
1   ï»¿using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Windows;
6   using System.Windows.Controls;
7   using System.Windows.Data;
8   using System.Windows.Documents;
9   using System.Windows.Input;
10  using System.Windows.Media;
11  using System.Windows.Media.Imaging;
12  using System.Windows.Navigation;
13  using System.Windows.Shapes;
14  using System.Collections;
15
16  namespace ShutterSpeedTester
17  {
18      /// <summary>
19      /// Interaction logic for Window1.xaml
20      /// </summary>
21      public partial class Window1 : Window
22      {
23          BitmapImage bi = new BitmapImage();
24
25          ArrayList AControlPoints;
26          ArrayList AHighLighting = new ArrayList();
27          ArrayList AThreshold  ;
28          TextBox textBoxBitmap;
29
30          public Window1(ArrayList AControlPoints, ArrayList AThreshold,
                TextBox textBoxBitmap)
31          {
32              InitializeComponent();
33              this.AControlPoints = AControlPoints;
34              this.AThreshold = AThreshold;
35              this.textBoxBitmap = textBoxBitmap;
36
37              this.ListCheckPoints.ItemsSource = AControlPoints;
38
39              if (this.textBoxBitmap.Text != "Not␣selected!")
40              {
41                  bi.BeginInit();
42                  bi.UriSource = new Uri(this.textBoxBitmap.Text.ToString());
43                  bi.DecodePixelWidth = 500;
44                  bi.EndInit();
45
46                  TheImage.Source = bi;
47              }
48
```

```
49              if (AThreshold.Count != 0)
50              {
51                  ShowingThreshold();
52
53                  ShowingRGB();
54
55              }
56              else
57              {
58                  this.textBoxThresholdRGB_B.Text = "0";
59                  this.textBoxThresholdRGB_G.Text = "0";
60                  this.textBoxThresholdRGB_R.Text = "0";
61              }
62          }
63
64          private void TheImage_SizeChanged(object sender,
                SizeChangedEventArgs e)
65          {
66              if (this.textBoxBitmap.Text != "Not selected!")
67                  Redraw();
68          }
69
70          private void Redraw()
71          {
72              //Content Element anpassen, damit das Bild ÃŒberdeckt wird
73              canvasimage.Width = TheImage.ActualWidth;
74              canvasimage.Height = TheImage.ActualHeight;
75              Canvas.SetLeft(canvasimage, (canvas1.ActualWidth - TheImage.
                    ActualWidth) / 2);
76              Canvas.SetTop(canvasimage, (canvas1.ActualHeight - TheImage.
                    ActualHeight) / 2);
77
78              //Aktuelle Elemente lÃ¶schen
79              canvasimage.Children.Clear();
80
81              //Kontrollpunkte der Groesee vom Bild anpassen
82
83              double x = 0;
84              double y = 0;
85              double r = 0;
86              double rScreen = 0;
87
88              foreach (ControlPoints Element in AControlPoints)
89              {
90                  r = Element.GetRadiusPicture();
91                  rScreen = r / bi.PixelWidth * TheImage.ActualWidth;
92                  x = (Element.GetPictureCoord().X / bi.PixelWidth * TheImage
                        .ActualWidth);
93                  y = (Element.GetPictureCoord().Y / bi.PixelHeight *
                        TheImage.ActualHeight);
94                  Element.SetScreenCoord(new Point(x, y));
95                  Element.SetRadiusScreen(rScreen);
96              }
97
98              foreach (ControlPoints Element in AHighLighting)
99              {
100                 r = Element.GetRadiusPicture();
101                 rScreen = r / bi.PixelWidth * TheImage.ActualWidth;
```

```
102                 x = (Element.GetPictureCoord().X / bi.PixelWidth * TheImage
                        .ActualWidth);
103                 y = (Element.GetPictureCoord().Y / bi.PixelHeight *
                        TheImage.ActualHeight);
104                 Element.SetScreenCoord(new Point(x, y));
105                 Element.SetRadiusScreen(rScreen);
106             }
107
108             foreach (Threshold Element in AThreshold)
109             {
110                 r = Element.GetRadiusPicture();
111                 rScreen = r / bi.PixelWidth * TheImage.ActualWidth;
112                 x = (Element.GetPictureCoord().X / bi.PixelWidth * TheImage
                        .ActualWidth);
113                 y = (Element.GetPictureCoord().Y / bi.PixelHeight *
                        TheImage.ActualHeight);
114                 Element.SetScreenCoord(new Point(x, y));
115                 Element.SetRadiusScreen(rScreen);
116             }
117
118             //Elemente wieder zeichnen
119             PaintControlPoints();
120
121             //Thresholds wieder zeichnen
122             PaintThreshold();
123
124             //Hightlight neu zeichnen
125             PaintHighlight();
126         }
127
128         private void TheImage_MouseLeftButtonDown(object sender,
                MouseButtonEventArgs e)
129         {
130             if (toggleButtonSetCheckPoints.IsChecked==true)
131             {
132                 double rImage = 0;
133                 try
134                 {
135                     rImage = double.Parse(this.textBoxRadiusCP.Text);
136
137                     if (rImage > Math.Min(bi.PixelHeight,bi.PixelWidth) ||
                            rImage<=0)
138                         throw new ArgumentOutOfRangeException("Radius␣is␣
                            not␣valid!");
139                 }
140                 catch (Exception ex)
141                 {
142                     MessageBox.Show("Wrong␣Radius!␣" + ex.Message,"Error",
                            MessageBoxButton.OK,MessageBoxImage.Error);
143                     return;
144                 }
145                 double rScreen = rImage/bi.PixelWidth*TheImage.ActualWidth
                        ;
146
147                 double xScreen = e.GetPosition(this.TheImage).X -rScreen;
148                 double yScreen = e.GetPosition(this.TheImage).Y -rScreen;
149
150                 double xPicture = Math.Min(bi.PixelWidth, Math.Max(0,
                        xScreen / TheImage.ActualWidth * bi.PixelWidth));
```

```
151                 double yPicture = Math.Min(bi.PixelHeight, Math.Max(0,
                        yScreen / TheImage.ActualHeight * bi.PixelHeight));
152
153                 PaintCircle(new Point(xScreen, yScreen), rScreen, Colors.
                        PeachPuff);
154
155                 AControlPoints.Add(new ControlPoints(new Point(xScreen,
                        yScreen), new Point(xPicture, yPicture), rScreen, rImage
                        ));
156
157                 this.ListCheckPoints.Items.Refresh();
158             }
159             else if (toggleButtonSetThresPoint.IsChecked == true)
160             {
161                 //Threshold erfassen
162
163                 double rImage = 0;
164                 try
165                 {
166                     rImage = double.Parse(this.textBoxRadiusThres.Text);
167
168                     if (rImage > Math.Min(bi.PixelHeight, bi.PixelWidth) ||
                            rImage <= 0)
169                         throw new ArgumentOutOfRangeException("Radius␣is␣
                            not␣valid!");
170                 }
171                 catch (Exception ex)
172                 {
173                     MessageBox.Show("Wrong␣Radius!␣" + ex.Message, "Error",
                            MessageBoxButton.OK, MessageBoxImage.Error);
174                     return;
175                 }
176
177                 double rScreen = rImage / bi.PixelWidth * TheImage.
                        ActualWidth;
178
179                 double xScreen = e.GetPosition(this.TheImage).X - rScreen;
180                 double yScreen = e.GetPosition(this.TheImage).Y - rScreen;
181
182                 double xPicture = Math.Min(bi.PixelWidth,Math.Max(0,(
                        xScreen ) / TheImage.ActualWidth * bi.PixelWidth));
183                 double yPicture = Math.Min(bi.PixelHeight,Math.Max(0,(
                        yScreen) / TheImage.ActualHeight * bi.PixelHeight));
184
185                 double RGBPositionX = Math.Min(bi.PixelWidth, Math.Max(0, (
                        xScreen + rScreen) / TheImage.ActualWidth * bi.
                        PixelWidth));
186                 double RGBPositionY = Math.Min(bi.PixelHeight, Math.Max(0,
                        (yScreen + rScreen) / TheImage.ActualHeight * bi.
                        PixelHeight));
187
188
189                 Color ColorRGB = SetRGBValue(RGBPositionX, RGBPositionY,
                        rImage);
190
191                 AThreshold.Clear();
192
193                 AThreshold.Add(new Threshold(new Point(xScreen, yScreen),
                        new Point(xPicture, yPicture), rScreen, rImage, ColorRGB
```

```
                    , Colors.Coral));
194
195                 //neuzeichnen
196                 Redraw();
197
198                 //Threshold speichern
199                 SavingThreshold();
200             }
201         }
202
203         private Color SetRGBValue(double x, double y, double r)
204         {
205             BitmapCalculation BitmapCalculation = new BitmapCalculation(
                    this.textBoxBitmap.Text.ToString());
206
207             Color ColorThreshold = new Color();
208             try
209             {
210                 ColorThreshold = BitmapCalculation.CalculateValues(x, y, r)
                        ;
211             }
212             catch(Exception ex)
213             {
214                 MessageBox.Show("Error in Calculation from RGB Value! "+ex.
                        Message, "Error",MessageBoxButton.OK,MessageBoxImage.
                        Error);
215                 return ColorThreshold;
216             }
217
218             this.textBoxRGB_R.Text = ColorThreshold.R.ToString();
219             this.textBoxRGB_G.Text = ColorThreshold.G.ToString();
220             this.textBoxRGB_B.Text = ColorThreshold.B.ToString();
221
222             return ColorThreshold;
223         }
224
225         private void PaintThreshold()
226         {
227             foreach (Threshold Element in AThreshold)
228             {
229                 PaintCircle(Element.GetScreenCoord(), Element.
                        GetRadiusScreen(), Colors.Red);
230             }
231         }
232
233         private void PaintHighlight()
234         {
235             foreach (ControlPoints Element in AHighLighting)
236             {
237                 PaintCircle(Element.GetScreenCoord(), Element.
                        GetRadiusScreen(), Colors.AliceBlue);
238             }
239         }
240
241         private void PaintControlPoints()
242         {
243             foreach(ControlPoints Element in AControlPoints)
244             {
```

```
245                 PaintCircle(Element.GetScreenCoord(),Element.
                        GetRadiusScreen(),Colors.PeachPuff);
246             }
247         }
248         private void PaintCircle(Point coord, double radius, Color colors)
249         {
250             Ellipse redRectangle = new Ellipse();
251             redRectangle.Height = radius * 2;
252             redRectangle.Width = radius * 2;
253 
254             // Create a blue and a black Brush
255             SolidColorBrush blueBrush = new SolidColorBrush();
256             blueBrush.Color = colors;
257             SolidColorBrush blackBrush = new SolidColorBrush();
258             blackBrush.Color = Colors.Black;
259 
260             // Set Ellipse's width and color
261             redRectangle.StrokeThickness = 1;
262 
263             redRectangle.Stroke = blackBrush;
264             redRectangle.MouseLeftButtonDown += new MouseButtonEventHandler
                    (TheImage_MouseLeftButtonDown);
265             // Fill rectangle with blue color
266 
267             redRectangle.Fill = blueBrush;
268 
269             redRectangle.Opacity = 0.5;
270             redRectangle.Margin = new Thickness(coord.X, coord.Y, 0, 0);
271 
272             // Add Ellipse to the Grid.
273             canvasimage.Children.Add(redRectangle);
274         }
275 
276         private void BImageLoad_Click(object sender, RoutedEventArgs e)
277         {
278             Microsoft.Win32.OpenFileDialog fileDlg = new Microsoft.Win32.
                    OpenFileDialog();
279             fileDlg.Title = "Select␣image";
280             fileDlg.DefaultExt = ".jpg";
281             fileDlg.Filter = "Picture␣(.jpg)|*.jpg|All␣Files␣(*.*)|*.*";
282             fileDlg.FileOk +=
283             new System.ComponentModel.CancelEventHandler(fileDlg_FileOk);
284             fileDlg.ShowDialog();
285         }
286 
287 
288         void fileDlg_FileOk(object sender, System.ComponentModel.
                CancelEventArgs e)
289         {
290             try
291             {
292                 string selectedFile = (sender as Microsoft.Win32.
                        OpenFileDialog).FileName;
293                 if (this.textBoxBitmap.Text == "Not␣selected!")
294                 {
295                     bi.BeginInit();
296                     bi.UriSource = new Uri(selectedFile);
297                     bi.DecodePixelWidth = 500;
298                     bi.EndInit();
```

```
299                 }
300                 else if(bi.UriSource.ToString() != new Uri(selectedFile).
                        ToString())
301                 {
302                     bi = new BitmapImage();
303                     bi.BeginInit();
304                     bi.UriSource = new Uri(selectedFile);
305                     bi.DecodePixelWidth = 500;
306                     bi.EndInit();
307                 }
308 
309                 TheImage.Source = bi;
310                 this.textBoxBitmap.Text = bi.UriSource.ToString();
311 
312             }
313             catch(Exception ex)
314             {
315                 MessageBox.Show("Something is going wrong! "+ ex.Message, "
                        Error", MessageBoxButton.OK,MessageBoxImage.Error);
316             }
317         }
318 
319         private void toggleButtonSetCheckPoints_Click(object sender,
                RoutedEventArgs e)
320         {
321             if (toggleButtonSetCheckPoints.IsChecked == true)
322             {
323                 TheImage.Cursor = Cursors.Cross;
324             }
325             else
326             {
327                 TheImage.Cursor = null;
328             }
329         }
330 
331         private void buttonClose_Click(object sender, RoutedEventArgs e)
332         {
333             this.Close();
334         }
335 
336         private void toggleButtonSetThresPoint_Click(object sender,
                RoutedEventArgs e)
337         {
338             if (toggleButtonSetThresPoint.IsChecked == true)
339             {
340                 TheImage.Cursor = Cursors.Cross;
341             }
342             else
343             {
344                 TheImage.Cursor = null;
345             }
346         }
347 
348         private void tabthreshold_GotFocus(object sender, RoutedEventArgs e
                )
349         {
350             if (toggleButtonSetCheckPoints.IsChecked == true )
351                 toggleButtonSetCheckPoints.IsChecked = false;
352         }
```

```
353
354         private void tabCheckPoints_GotFocus(object sender, RoutedEventArgs
                e)
355         {
356             if (toggleButtonSetThresPoint.IsChecked == true)
357                 toggleButtonSetThresPoint.IsChecked = false;
358         }
359
360         private void textBoxRadiusCP_KeyDown(object sender, KeyEventArgs e)
361         {
362             if (e.Key != Key.D0 && e.Key != Key.D1 && e.Key != Key.D2 && e.
                    Key != Key.D3 && e.Key != Key.D4 && e.Key != Key.D5 && e.Key
                     != Key.D6 && e.Key != Key.D7 && e.Key != Key.D8 && e.Key !=
                     Key.D9 && e.Key != Key.NumPad0 && e.Key != Key.NumPad1 && e
                    .Key != Key.NumPad2 && e.Key != Key.NumPad3 && e.Key != Key.
                    NumPad4 && e.Key != Key.NumPad5 && e.Key != Key.NumPad6 && e
                    .Key != Key.NumPad7 && e.Key != Key.NumPad8 && e.Key != Key.
                    NumPad9)
363                 e.Handled = true;
364         }
365
366         private void textBoxThresholdRGB_R_KeyDown(object sender,
                KeyEventArgs e)
367         {
368             if (e.Key != Key.D0 && e.Key != Key.D1 && e.Key != Key.D2 && e.
                    Key != Key.D3 && e.Key != Key.D4 && e.Key != Key.D5 && e.Key
                     != Key.D6 && e.Key != Key.D7 && e.Key != Key.D8 && e.Key !=
                     Key.D9 && e.Key != Key.NumPad0 && e.Key != Key.NumPad1 && e
                    .Key != Key.NumPad2 && e.Key != Key.NumPad3 && e.Key != Key.
                    NumPad4 && e.Key != Key.NumPad5 && e.Key != Key.NumPad6 && e
                    .Key != Key.NumPad7 && e.Key != Key.NumPad8 && e.Key != Key.
                    NumPad9)
369                 e.Handled = true;
370         }
371
372         private void Window_Closed(object sender, EventArgs e)
373         {
374             if (AThreshold.Count != 0)
375             {
376                 SavingThreshold();
377             }
378         }
379
380         private void SavingThreshold()
381         {
382             Color ColorValue = new Color();
383
384             if (AThreshold.Count != 0)
385             {
386                 foreach (Threshold obj in AThreshold)
387                 {
388                     ColorValue.A = 255;
389                     ColorValue.B = (byte)Math.Min(255, int.Parse(this.
                            textBoxThresholdRGB_B.Text.ToString()));
390                     ColorValue.G = (byte)Math.Min(255, int.Parse(this.
                            textBoxThresholdRGB_G.Text.ToString())); ;
391                     ColorValue.R = (byte)Math.Min(255, int.Parse(this.
                            textBoxThresholdRGB_R.Text.ToString())); ;
392
```

```
393                     this.textBoxThresholdRGB_B.Text = ColorValue.B.ToString
                            ();
394                     this.textBoxThresholdRGB_G.Text = ColorValue.G.ToString
                            ();
395                     this.textBoxThresholdRGB_R.Text = ColorValue.R.ToString
                            ();
396
397                     obj.SetRGBThreshold(ColorValue);
398                 }
399
400             }
401         }
402
403         private void ShowingThreshold()
404         {
405             Color ColorValue = new Color();
406
407             if (AThreshold.Count != 0)
408             {
409                 foreach (Threshold obj in AThreshold)
410                 {
411                     ColorValue = obj.GetRGBThreshold();
412
413                     this.textBoxThresholdRGB_B.Text = ColorValue.B.ToString
                            ();
414                     this.textBoxThresholdRGB_G.Text = ColorValue.G.ToString
                            ();
415                     this.textBoxThresholdRGB_R.Text = ColorValue.R.ToString
                            ();
416
417                 }
418
419             }
420         }
421
422         private void ShowingRGB()
423         {
424             Color ColorValue = new Color();
425
426             if (AThreshold.Count != 0)
427             {
428                 foreach (Threshold obj in AThreshold)
429                 {
430                     ColorValue = obj.GetRGBValue();
431
432                     this.textBoxRGB_B.Text = ColorValue.B.ToString();
433                     this.textBoxRGB_G.Text = ColorValue.G.ToString();
434                     this.textBoxRGB_R.Text = ColorValue.R.ToString();
435
436                     this.textBoxRadiusThres.Text = obj.GetRadiusPicture().
                            ToString();
437
438                 }
439
440             }
441         }
442
443         private void ListCheckPoints_SelectionChanged(object sender,
                SelectionChangedEventArgs e)
```

```
444         {
445             //Checkpunkt hervorheben
446 
447             ControlPoints HighLight =   (ControlPoints) this.
                    ListCheckPoints.SelectedItem;
448 
449             AHighLighting.Clear();
450 
451             //AHighLighting.Add(new ControlPoints (HighLight.GetScreenCoord
                    (),HighLight.GetPictureCoord(),HighLight.GetRadiusScreen(),
                    HighLight.GetRadiusPicture()));
452             if (HighLight!=null)
453                 AHighLighting.Add(HighLight);
454             Redraw();
455 
456         }
457 
458         private void ListCheckPoints_KeyDown(object sender, KeyEventArgs e)
459         {
460             if (e.Key == Key.Delete)
461             {
462                 ControlPoints HighLight = (ControlPoints)this.
                        ListCheckPoints.SelectedItem;
463                 AHighLighting.Remove(HighLight);
464                 AControlPoints.Remove(HighLight);
465                 this.ListCheckPoints.Items.Refresh();
466                 Redraw();
467             }
468         }
469 
470     }
471 }
```

BitmapCalculation.cs

```
1  ï»¿using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Windows.Media.Imaging;
6  using System.Windows.Media;
7 
8  namespace ShutterSpeedTester
9  {
10     class BitmapCalculation
11     {
12 
13         byte[] aBitmapPixels;
14         BitmapImage bi = new BitmapImage();
15 
16 
17         //Konstruktor
18         public BitmapCalculation(string stringPath)
19         {
20             bi.BeginInit();
21             bi.UriSource = new Uri(stringPath);
22             bi.DecodePixelWidth = 500;
23             bi.EndInit();
24 
```

```
25              aBitmapPixels = new byte[bi.PixelHeight * (bi.PixelWidth * bi.
                    Format.BitsPerPixel) / 8];
26              bi.CopyPixels(aBitmapPixels, (bi.PixelWidth * bi.Format.
                    BitsPerPixel ) / 8, 0);
27          }
28
29
30          public Color CalculateValues(double x, double y, double radius)
31          {
32              UInt32 CountElements = 0;
33              UInt32 intColorR = 0;
34              UInt32 intColorG = 0;
35              UInt32 intColorB = 0;
36
37              UInt32 row_Min, row_Max;
38              UInt32 column_Min, column_Max;
39
40              //Checking values
41              if (x < 0 || bi.PixelWidth < x)
42                  throw new ArgumentOutOfRangeException("X Value is not valid
                        !");
43
44              if (y < 0 || bi.PixelHeight < y)
45                  throw new ArgumentOutOfRangeException("Y Value is not valid
                        !");
46
47              if (radius < 0 || bi.PixelWidth < radius || bi.PixelHeight <
                    radius)
48                  throw new ArgumentOutOfRangeException("R Value is not valid
                        !");
49
50
51              //We only look at the square of interest
52              row_Min = (UInt32)Math.Round(Math.Max(0, y - radius));
53              row_Max = (UInt32)Math.Round(Math.Min(bi.PixelHeight, y +
                    radius));
54
55              column_Min = (UInt32)Math.Round(Math.Max(0, x - radius));
56              column_Max = (UInt32)Math.Round(Math.Min(bi.PixelWidth,x+radius
                    ));
57
58              x = Math.Round(x);
59              y = Math.Round(y);
60              radius = Math.Round(radius);
61
62              for (UInt32 i = row_Min; i <= row_Max; i++)
63              {
64                  for (UInt32 j = column_Min; j <= column_Max; j++)
65                  {
66                      //is the value in the circle?
67                      if (radius < Math.Sqrt((x - j) * (x - j) + (y - i) * (y
                            - i)))
68                      {
69                          //Point is in it
70                          CountElements++;
71
72                          //Sum the Color Values
73
74                          //Red
```

```
75                          intColorR = intColorR + aBitmapPixels[i * (bi.
                                PixelWidth * bi.Format.BitsPerPixel / 8) + j *
                                bi.Format.BitsPerPixel / 8 + 2];
76
77                          //Green
78                          intColorG = intColorG + aBitmapPixels[i * (bi.
                                PixelWidth * bi.Format.BitsPerPixel / 8) + j *
                                bi.Format.BitsPerPixel / 8 + 1];
79
80                          //Blue
81                          intColorB = intColorB + aBitmapPixels[i * (bi.
                                PixelWidth * bi.Format.BitsPerPixel / 8) + j *
                                bi.Format.BitsPerPixel / 8 ];
82                      }
83                  }
84              }
85
86              //Calculate the Red, Green and Blue Value
87              intColorR = Math.Min(intColorR / CountElements,255);
88              intColorG = Math.Min(intColorG / CountElements,255);
89              intColorB = Math.Min(intColorB / CountElements,255);
90
91              Color returnvalue = new Color() ;
92
93              returnvalue.A = 255;
94              returnvalue.B = (byte) intColorB;
95              returnvalue.G = (byte) intColorG;
96              returnvalue.R = (byte) intColorR;
97
98              return returnvalue;
99
100         }
101
102
103     }
104 }
```

ControlPoints.cs

```
1   ï»¿using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Windows;
6   using System.Windows.Controls;
7   using System.Windows.Data;
8   using System.Windows.Documents;
9   using System.Windows.Input;
10  using System.Windows.Media;
11  using System.Windows.Media.Imaging;
12  using System.Windows.Navigation;
13  using System.Windows.Shapes;
14
15  namespace ShutterSpeedTester
16  {
17      class ControlPoints
18      {
19          private Point pointScreenCoord;
20          private Point pointPictureCoord;
```

```
21          private double doubleRadiusScreen;
22          private double doubleRadiusPicture;
23          public string Number { get; set; }
24          public string Coordinates { get; set; }
25          public string Radius { get; set; }
26          static uint countControlPoints = 0;
27
28          public ControlPoints(Point pointScreenCoord, Point
                pointPictureCoord, double doubleRadiusScreen, double
                doubleRadiusPicture)
29          {
30              this.pointPictureCoord = pointPictureCoord;
31              this.pointScreenCoord = pointScreenCoord;
32              this.doubleRadiusPicture = doubleRadiusPicture;
33              this.doubleRadiusScreen = doubleRadiusScreen;
34              countControlPoints++;
35              this.Number = countControlPoints.ToString();
36              this.Coordinates = Math.Round(pointPictureCoord.X).ToString() +
                    ","+Math.Round(pointPictureCoord.Y).ToString();
37              this.Radius = doubleRadiusPicture.ToString();
38          }
39
40          public Point GetScreenCoord()
41          {
42              return this.pointScreenCoord;
43          }
44
45          public Point GetPictureCoord()
46          {
47              return  this.pointPictureCoord;
48          }
49
50          public double GetRadiusPicture()
51          {
52              return this.doubleRadiusPicture;
53          }
54
55          public double GetRadiusScreen()
56          {
57              return this.doubleRadiusScreen;
58          }
59
60          public void SetScreenCoord(Point newScreenCoord)
61          {
62             this.pointScreenCoord = newScreenCoord;
63          }
64
65          public void SetRadiusScreen(double doubleRadiusScreen)
66          {
67              this.doubleRadiusScreen = doubleRadiusScreen;
68          }
69
70      }
71  }
```

ImagesContainer.cs

```
1  ï»¿using System;
2  using System.Collections.Generic;
```

```
3  using System.Linq;
4  using System.Text;
5
6  namespace ShutterSpeedTester
7  {
8      class ImagesContainer
9      {
10         public string Path  {get; set;}
11         public string Status { get; set; }
12         public string Result { get; set; }
13
14         public ImagesContainer(string Path)
15         {
16             this.Path = Path;
17             this.Status = "Not Checked";
18             this.Result = "";
19         }
20     }
21 }
```

Threshold.cs

```
1  ï»¿using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Windows;
6  using System.Windows.Controls;
7  using System.Windows.Data;
8  using System.Windows.Documents;
9  using System.Windows.Input;
10 using System.Windows.Media;
11 using System.Windows.Media.Imaging;
12 using System.Windows.Navigation;
13 using System.Windows.Shapes;
14
15 namespace ShutterSpeedTester
16 {
17     class Threshold
18     {
19         private Point pointScreenCoord;
20         private Point pointPictureCoord;
21         private double doubleRadiusScreen;
22         private double doubleRadiusPicture;
23         private Color RGBValue;
24         private Color RGBThreshold;
25
26         public Threshold(Point pointScreenCoord, Point pointPictureCoord,
               double doubleRadiusScreen, double doubleRadiusPicture, Color
               RGBValue, Color RGBThreshold)
27         {
28             this.pointPictureCoord = pointPictureCoord;
29             this.pointScreenCoord = pointScreenCoord;
30             this.doubleRadiusPicture = doubleRadiusPicture;
31             this.doubleRadiusScreen = doubleRadiusScreen;
32             this.RGBThreshold = RGBThreshold;
33             this.RGBValue = RGBValue;
34         }
35
```

```
36          public Point GetScreenCoord()
37          {
38              return this.pointScreenCoord;
39          }
40
41          public Point GetPictureCoord()
42          {
43              return  this.pointPictureCoord;
44          }
45
46          public double GetRadiusPicture()
47          {
48              return this.doubleRadiusPicture;
49          }
50
51          public double GetRadiusScreen()
52          {
53              return this.doubleRadiusScreen;
54          }
55
56          public Color GetRGBThreshold()
57          {
58              return this.RGBThreshold;
59          }
60
61          public Color GetRGBValue()
62          {
63              return this.RGBValue;
64          }
65
66          public void SetRGBValue(Color RGBValue)
67          {
68              this.RGBValue = RGBValue;
69          }
70
71
72          public void SetRGBThreshold(Color RGBThreshold)
73          {
74              this.RGBThreshold = RGBThreshold;
75          }
76
77
78          public void SetScreenCoord(Point newScreenCoord)
79          {
80             this.pointScreenCoord = newScreenCoord;
81          }
82
83          public void SetRadiusScreen(double doubleRadiusScreen)
84          {
85              this.doubleRadiusScreen = doubleRadiusScreen;
86          }
87
88      }
89  }
```

K Windows Klasse VI

Window1.xaml

```
1  ï»¿<Window x:Class="WPFNetzwerk.Window1"
2      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4      Title="Klasse VI Windows" Height="600" Width="800" Closing="
           Window_Closing">
5
6      <Grid>
7          <Label Height="28" Name="labelPort" VerticalAlignment="Top"
               HorizontalAlignment="Left" Width="73" HorizontalContentAlignment
               ="Right" Margin="0,3,0,0">Used Port:</Label>
8          <TextBox Height="23" HorizontalAlignment="Left" Margin="76,5,0,0"
               Name="textBox_Port" VerticalAlignment="Top" Width="101"
               TextAlignment="Right" CharacterCasing="Lower" TextChanged="
               textBox_Port_TextChanged"/>
9          <Label Name="labelIP" HorizontalAlignment="Left" Width="73" Margin=
               "0,26,0,0" FlowDirection="LeftToRight"
               HorizontalContentAlignment="Right" Height="28" VerticalAlignment
               ="Top">IP:</Label>
10         <TextBox Height="23" Margin="76,28,0,0" Name="textBox_IP"
               VerticalAlignment="Top" TextAlignment="Right" CharacterCasing="
               Lower" HorizontalAlignment="Left" Width="101" />
11         <Button Height="49" HorizontalAlignment="Left" Margin="187,5,0,0"
               Name="buttonStart" VerticalAlignment="Top" Width="75" Click="
               buttonStart_Click">Start</Button>
12         <Separator Height="2" Margin="0,57,0,0" Name="separator1"
               VerticalAlignment="Top" />
13         <Button Height="49" HorizontalAlignment="Left" Margin="14,81,0,0"
               Name="buttonUDPBroadcast" VerticalAlignment="Top" Width="163"
               Click="buttonUDPBroadcast_Click" IsEnabled="False">UDP Broadcast
               </Button>
14         <Label IsEnabled="False" Name="labelFoundModul" HorizontalAlignment
               ="Left" Width="124" Margin="206,67,0,0" FlowDirection="
               LeftToRight" HorizontalContentAlignment="Left" Height="28"
               VerticalAlignment="Top">gefundene Module:</Label>
15         <ListBox IsEnabled="False" Margin="206,95,31,0" Name="
               listBoxFoundModul" Height="65" VerticalAlignment="Top" />
16         <Label IsEnabled="False" Name="labelZielIP" HorizontalAlignment="
               Left" Width="73" Margin="0,138,0,0" FlowDirection="LeftToRight"
               HorizontalContentAlignment="Right" Height="28" VerticalAlignment
               ="Top">Ziel IP:</Label>
17         <TextBox IsEnabled="False" Height="23" Margin="76,138,0,0" Name="
               textBox_ZielIP" VerticalAlignment="Top" TextAlignment="Right"
               CharacterCasing="Lower" HorizontalAlignment="Left" Width="101" /
               >
18         <Label IsEnabled="False" Name="labelInformation" Margin="
               476,167,0,0" FlowDirection="LeftToRight"
               HorizontalContentAlignment="Left" Height="28" VerticalAlignment=
               "Top" HorizontalAlignment="Left" Width="80">Information:</Label>
19         <ListBox IsEnabled="False" Margin="476,195,31,19" Name="
               listBoxInformation" />
20         <Button Height="29" HorizontalAlignment="Left" Margin="14,179,0,0"
               Name="buttonGetClass" VerticalAlignment="Top" Width="87"
               IsEnabled="False" Click="buttonCommandClick">Get Klasse</Button>
21         <Label IsEnabled="False" Name="labelGetClass" HorizontalAlignment="
               Left" Width="73" Margin="76,179,0,0" FlowDirection="LeftToRight"
```

```
            HorizontalContentAlignment="Right" Height="28"
            VerticalAlignment="Top">Klasse:</Label>
22          <TextBox CharacterCasing="Lower" Height="23" HorizontalAlignment="
            Left" IsEnabled="False" Margin="148,181,0,0" Name="
            textBoxGetClass" TextAlignment="Right" VerticalAlignment="Top"
            Width="69" />
23          <Button IsEnabled="False" Margin="14,223,0,0" Name="
            buttonPinSettingsRead" HorizontalAlignment="Left" Width="155"
            Height="29" VerticalAlignment="Top" Click="buttonCommandClick">
            Pin Settings auslesen</Button>
24          <Button IsEnabled="False" Margin="244,177.48,0,0" Name="
            buttonPinSettingsSave" Height="29" VerticalAlignment="Top" Click
            ="buttonCommandClick" HorizontalAlignment="Left" Width="130">Pin
             Settings speichern</Button>
25          <TabControl Margin="14,264,0,19" IsEnabled="False" Name="TabPin"
            HorizontalAlignment="Left" Width="422">
26           <TabItem Header="Pin 1">
27               <Grid Margin="0,0,0,0" Name="gridPin1" HorizontalAlignment=
                  "Left" Width="413" Height="244" VerticalAlignment="Top">
28                   <CheckBox Height="16" Name="Pin1SettingBit7" Margin="
                      110,35,0,0" VerticalAlignment="Top"
                      HorizontalAlignment="Left" Width="18"></CheckBox>
29                   <CheckBox Height="16"  Width="18" Name="Pin1SettingBit6
                      " HorizontalAlignment="Left" Margin="125,35,0,0"
                      VerticalAlignment="Top"></CheckBox>
30                   <CheckBox Height="16"  Width="18" Name="Pin1SettingBit5
                      " HorizontalAlignment="Left" Margin="140,35,0,0"
                      VerticalAlignment="Top"></CheckBox>
31                   <CheckBox Height="16"  Width="18" Name="Pin1SettingBit4
                      " HorizontalAlignment="Left" Margin="155,35,0,0"
                      VerticalAlignment="Top"></CheckBox>
32                   <CheckBox Height="16"  Width="18" Name="Pin1SettingBit3
                      " HorizontalAlignment="Left" Margin="170,35,0,0"
                      VerticalAlignment="Top"></CheckBox>
33                   <CheckBox Height="16"  Width="18" Name="Pin1SettingBit2
                      " HorizontalAlignment="Left" Margin="185,35,0,0"
                      VerticalAlignment="Top"></CheckBox>
34                   <CheckBox Height="16"  Width="18" Name="Pin1SettingBit1
                      " HorizontalAlignment="Left" Margin="200,35,0,0"
                      VerticalAlignment="Top"></CheckBox>
35                   <CheckBox Height="16" Name="Pin1SettingBit0" Margin="
                      215,35,0,0" VerticalAlignment="Top"></CheckBox>
36                   <Label Height="28" HorizontalAlignment="Left" Name="
                      labelPin1Setting" VerticalAlignment="Top" Width="106
                      "  Margin="0,28,0,0">Pin 1 Settings(7:0):</Label>
37                   <Label Height="28" HorizontalAlignment="Left" Name="
                      labelPin1AffectedPins" VerticalAlignment="Top" Width
                      ="135" Margin="0,54,0,0">Pin 1 Affected Pins(7:0):</
                      Label>
38                   <CheckBox Height="16"  Width="18" Name="
                      Pin1AffectedPinsBit7" HorizontalAlignment="Left"
                      Margin="140,61,0,0" VerticalAlignment="Top" ></
                      CheckBox>
39                   <CheckBox Height="16"  Width="18" Name="
                      Pin1AffectedPinsBit6" HorizontalAlignment="Left"
                      Margin="155,61,0,0" VerticalAlignment="Top"></
                      CheckBox>
40                   <CheckBox Height="16"  Width="18" Name="
                      Pin1AffectedPinsBit5" HorizontalAlignment="Left"
```

```
                Margin="170,61,0,0" VerticalAlignment="Top"></
                CheckBox>
41              <CheckBox Height="16"  Width="18" Name="
                Pin1AffectedPinsBit4" HorizontalAlignment="Left"
                Margin="185,61,0,0" VerticalAlignment="Top"></
                CheckBox>
42              <CheckBox Height="16"  Width="18" Name="
                Pin1AffectedPinsBit3" HorizontalAlignment="Left"
                Margin="200,61,0,0" VerticalAlignment="Top"></
                CheckBox>
43              <CheckBox Height="16" Name="Pin1AffectedPinsBit2"
                Margin="215,61,0,0" VerticalAlignment="Top"></
                CheckBox>
44              <CheckBox Height="16"  Width="18" Name="
                Pin1AffectedPinsBit1" HorizontalAlignment="Left"
                Margin="230,61,0,0" VerticalAlignment="Top"></
                CheckBox>
45              <CheckBox Height="16"  Width="18" Name="
                Pin1AffectedPinsBit0" HorizontalAlignment="left"
                Margin="245,61,0,0" VerticalAlignment="Top"></
                CheckBox>
46              <Label Height="28" HorizontalAlignment="Left" Name="
                labelPin1OutputState" VerticalAlignment="Top" Width=
                "135" Margin="0,78,0,0">Pin 1 Output State(7:0):</
                Label>
47              <CheckBox Height="16"  Width="18" Name="
                Pin1OutputStateBit7" HorizontalAlignment="Left"
                Margin="140,86,0,0" VerticalAlignment="Top" ></
                CheckBox>
48              <CheckBox Height="16"  Width="18" Name="
                Pin1OutputStateBit6" HorizontalAlignment="Left"
                Margin="155,86,0,0" VerticalAlignment="Top"></
                CheckBox>
49              <CheckBox Height="16"  Width="18" Name="
                Pin1OutputStateBit5" HorizontalAlignment="Left"
                Margin="170,86,0,0" VerticalAlignment="Top"></
                CheckBox>
50              <CheckBox Height="16"  Width="18" Name="
                Pin1OutputStateBit4" HorizontalAlignment="Left"
                Margin="185,86,0,0" VerticalAlignment="Top"></
                CheckBox>
51              <CheckBox Height="16"  Width="18" Name="
                Pin1OutputStateBit3" HorizontalAlignment="Left"
                Margin="200,86,0,0" VerticalAlignment="Top"></
                CheckBox>
52              <CheckBox Height="16" Name="Pin1OutputStateBit2" Margin
                ="215,86,0,0" VerticalAlignment="Top"></CheckBox>
53              <CheckBox Height="16"  Width="18" Name="
                Pin1OutputStateBit1" HorizontalAlignment="left"
                Margin="230,86,0,0" VerticalAlignment="Top"></
                CheckBox>
54              <CheckBox Height="16"  Width="18" Name="
                Pin1OutputStateBit0" HorizontalAlignment="left"
                Margin="245,86,0,0" VerticalAlignment="Top"></
                CheckBox>
55
56              <Label HorizontalAlignment="Left" Name="
                labelPin1TimeAState" Width="123" Margin="0,103,0,113
                ">Pin 1 Time A State:</Label>
```

```
57                  <CheckBox Width="18" Name="Pin1TimeAState"
                        HorizontalAlignment="Left" Margin="120,111,0,117"></
                        CheckBox>
58
59                  <Label HorizontalAlignment="Left" Name="labelPin1TimaA"
                         Width="123" Margin="0,0,0,88" Height="28"
                        VerticalAlignment="Bottom">Pin 1 Time A:</Label>
60                  <TextBox CharacterCasing="Lower"  Margin="96,0,170,93"
                        Name="textBoxPin1TimeA" TextAlignment="Right" Height
                        ="23" VerticalAlignment="Bottom" />
61
62                  <Label Height="28" HorizontalAlignment="Left" Name="
                        labelPin1TimeBState" VerticalAlignment="Bottom"
                        Width="123" Margin="0,0,0,63">Pin 1 Time B State:</
                        Label>
63                  <CheckBox Height="16"  Width="18" Name="Pin1TimeBState"
                         HorizontalAlignment="Left" Margin="120,0,0,67"
                        VerticalAlignment="Bottom" ></CheckBox>
64
65                  <Label Height="28" HorizontalAlignment="Left" Name="
                        labelPin1TimaB" VerticalAlignment="Bottom" Width="
                        123" Margin="0,0,0,38">Pin 1 Time B:</Label>
66                  <TextBox CharacterCasing="Lower"  Margin="96,0,170,43"
                        Name="textBoxPin1TimeB" TextAlignment="Right" Height
                        ="23" VerticalAlignment="Bottom" />
67
68
69              </Grid>
70          </TabItem>
71          <TabItem Header="Pin␣2">
72              <Grid Margin="0,0,0,0" Name="gridPin2" HorizontalAlignment=
                    "Left" Width="413" Height="244" VerticalAlignment="Top">
73                  <CheckBox Height="16"  Width="18" Name="Pin2SettingBit7
                        " HorizontalAlignment="Left" Margin="110,35,0,0"
                        VerticalAlignment="Top" ></CheckBox>
74                  <CheckBox Height="16"  Width="18" Name="Pin2SettingBit6
                        " HorizontalAlignment="Left" Margin="125,35,0,0"
                        VerticalAlignment="Top"></CheckBox>
75                  <CheckBox Height="16"  Width="18" Name="Pin2SettingBit5
                        " HorizontalAlignment="Left" Margin="140,35,0,0"
                        VerticalAlignment="Top"></CheckBox>
76                  <CheckBox Height="16"  Width="18" Name="Pin2SettingBit4
                        " HorizontalAlignment="Left" Margin="155,35,0,0"
                        VerticalAlignment="Top"></CheckBox>
77                  <CheckBox Height="16"  Width="18" Name="Pin2SettingBit3
                        " HorizontalAlignment="Left" Margin="170,35,0,0"
                        VerticalAlignment="Top"></CheckBox>
78                  <CheckBox Height="16"  Width="18" Name="Pin2SettingBit2
                        " HorizontalAlignment="Left" Margin="185,35,0,0"
                        VerticalAlignment="Top"></CheckBox>
79                  <CheckBox Height="16"  Width="18" Name="Pin2SettingBit1
                        " HorizontalAlignment="Left" Margin="200,35,0,0"
                        VerticalAlignment="Top"></CheckBox>
80                  <CheckBox Height="16" Name="Pin2SettingBit0" Margin="
                        215,35,0,0" VerticalAlignment="Top"></CheckBox>
81                  <Label Height="28" HorizontalAlignment="Left" Name="
                        labelPin2Setting" VerticalAlignment="Top" Width="106
                        "   Margin="0,28,0,0">Pin 2 Settings(7:0):</Label>
```

```
82          <Label Height="28" HorizontalAlignment="Left" Name="
                labelPin2AffectedPins" VerticalAlignment="Top" Width
                ="135" Margin="0,54,0,0">Pin 2 Affected Pins(7:0):</
                Label>
83          <CheckBox Height="16"  Width="18" Name="
                Pin2AffectedPinsBit7" HorizontalAlignment="Left"
                Margin="140,61,0,0" VerticalAlignment="Top" ></
                CheckBox>
84          <CheckBox Height="16"  Width="18" Name="
                Pin2AffectedPinsBit6" HorizontalAlignment="Left"
                Margin="155,61,0,0" VerticalAlignment="Top"></
                CheckBox>
85          <CheckBox Height="16"  Width="18" Name="
                Pin2AffectedPinsBit5" HorizontalAlignment="Left"
                Margin="170,61,0,0" VerticalAlignment="Top"></
                CheckBox>
86          <CheckBox Height="16"  Width="18" Name="
                Pin2AffectedPinsBit4" HorizontalAlignment="Left"
                Margin="185,61,0,0" VerticalAlignment="Top"></
                CheckBox>
87          <CheckBox Height="16"  Width="18" Name="
                Pin2AffectedPinsBit3" HorizontalAlignment="Left"
                Margin="200,61,0,0" VerticalAlignment="Top"></
                CheckBox>
88          <CheckBox Height="16" Name="Pin2AffectedPinsBit2"
                Margin="215,61,0,0" VerticalAlignment="Top"></
                CheckBox>
89          <CheckBox Height="16"  Width="18" Name="
                Pin2AffectedPinsBit1" HorizontalAlignment="Left"
                Margin="230,61,0,0" VerticalAlignment="Top"></
                CheckBox>
90          <CheckBox Height="16"  Width="18" Name="
                Pin2AffectedPinsBit0" HorizontalAlignment="left"
                Margin="245,61,0,0" VerticalAlignment="Top"></
                CheckBox>
91          <Label Height="28" HorizontalAlignment="Left" Name="
                labelPin2OutputState" VerticalAlignment="Top" Width=
                "135" Margin="0,78,0,0">Pin 2 Output State(7:0):</
                Label>
92          <CheckBox Height="16"  Width="18" Name="
                Pin2OutputStateBit7" HorizontalAlignment="Left"
                Margin="140,86,0,0" VerticalAlignment="Top" ></
                CheckBox>
93          <CheckBox Height="16"  Width="18" Name="
                Pin2OutputStateBit6" HorizontalAlignment="Left"
                Margin="155,86,0,0" VerticalAlignment="Top"></
                CheckBox>
94          <CheckBox Height="16"  Width="18" Name="
                Pin2OutputStateBit5" HorizontalAlignment="Left"
                Margin="170,86,0,0" VerticalAlignment="Top"></
                CheckBox>
95          <CheckBox Height="16"  Width="18" Name="
                Pin2OutputStateBit4" HorizontalAlignment="Left"
                Margin="185,86,0,0" VerticalAlignment="Top"></
                CheckBox>
96          <CheckBox Height="16"  Width="18" Name="
                Pin2OutputStateBit3" HorizontalAlignment="Left"
                Margin="200,86,0,0" VerticalAlignment="Top"></
                CheckBox>
```

```
97                  <CheckBox Height="16" Name="Pin2OutputStateBit2" Margin
                        ="215,86,0,0" VerticalAlignment="Top"></CheckBox>
98                  <CheckBox Height="16"  Width="18" Name="
                        Pin2OutputStateBit1" HorizontalAlignment="left"
                        Margin="230,86,0,0" VerticalAlignment="Top"></
                        CheckBox>
99                  <CheckBox Height="16"  Width="18" Name="
                        Pin2OutputStateBit0" HorizontalAlignment="left"
                        Margin="245,86,0,0" VerticalAlignment="Top"></
                        CheckBox>
100
101                 <Label HorizontalAlignment="Left" Name="
                        labelPin2TimeAState" Width="123" Margin="0,103,0,113
                        ">Pin 2 Time A State:</Label>
102                 <CheckBox Width="18" Name="Pin2TimeAState"
                        HorizontalAlignment="Left" Margin="120,111,0,117"></
                        CheckBox>
103
104                 <Label HorizontalAlignment="Left" Name="labelPin2TimaA"
                         Width="123" Margin="0,0,0,88" Height="28"
                        VerticalAlignment="Bottom">Pin 2 Time A:</Label>
105                 <TextBox CharacterCasing="Lower"  Margin="96,0,170,93"
                        Name="textBoxPin2TimeA" TextAlignment="Right" Height
                        ="23" VerticalAlignment="Bottom" />
106
107                 <Label Height="28" HorizontalAlignment="Left" Name="
                        labelPin2TimeBState" VerticalAlignment="Bottom"
                        Width="123" Margin="0,0,0,63">Pin 2 Time B State:</
                        Label>
108                 <CheckBox Height="16"  Width="18" Name="Pin2TimeBState"
                         HorizontalAlignment="Left" Margin="120,0,0,67"
                        VerticalAlignment="Bottom" ></CheckBox>
109
110                 <Label Height="28" HorizontalAlignment="Left" Name="
                        labelPin2TimaB" VerticalAlignment="Bottom" Width="
                        123" Margin="0,0,0,38">Pin 2 Time B:</Label>
111                 <TextBox CharacterCasing="Lower"  Margin="96,0,170,43"
                        Name="textBoxPin2TimeB" TextAlignment="Right" Height
                        ="23" VerticalAlignment="Bottom" />
112
113
114             </Grid>
115         </TabItem>
116         <TabItem Header="Pin␣3">
117             <Grid Margin="0,0,0,0" Name="gridPin3" HorizontalAlignment=
                    "Left" Width="413" Height="244" VerticalAlignment="Top">
118                 <CheckBox Height="16"  Width="18" Name="Pin3SettingBit7
                        " HorizontalAlignment="Left" Margin="110,35,0,0"
                        VerticalAlignment="Top" ></CheckBox>
119                 <CheckBox Height="16"  Width="18" Name="Pin3SettingBit6
                        " HorizontalAlignment="Left" Margin="125,35,0,0"
                        VerticalAlignment="Top"></CheckBox>
120                 <CheckBox Height="16"  Width="18" Name="Pin3SettingBit5
                        " HorizontalAlignment="Left" Margin="140,35,0,0"
                        VerticalAlignment="Top"></CheckBox>
121                 <CheckBox Height="16"  Width="18" Name="Pin3SettingBit4
                        " HorizontalAlignment="Left" Margin="155,35,0,0"
                        VerticalAlignment="Top"></CheckBox>
```

```
122                 <CheckBox Height="16"  Width="18" Name="Pin3SettingBit3
                        " HorizontalAlignment="Left" Margin="170,35,0,0"
                        VerticalAlignment="Top"></CheckBox>
123                 <CheckBox Height="16"  Width="18" Name="Pin3SettingBit2
                        " HorizontalAlignment="Left" Margin="185,35,0,0"
                        VerticalAlignment="Top"></CheckBox>
124                 <CheckBox Height="16"  Width="18" Name="Pin3SettingBit1
                        " HorizontalAlignment="Left" Margin="200,35,0,0"
                        VerticalAlignment="Top"></CheckBox>
125                 <CheckBox Height="16" Name="Pin3SettingBit0" Margin="
                        215,35,0,0" VerticalAlignment="Top"></CheckBox>
126                 <Label Height="28" HorizontalAlignment="Left" Name="
                        labelPin3Setting" VerticalAlignment="Top" Width="106
                        "    Margin="0,28,0,0">Pin 3 Settings(7:0):</Label>
127                 <Label Height="28" HorizontalAlignment="Left" Name="
                        labelPin3AffectedPins" VerticalAlignment="Top" Width
                        ="135" Margin="0,54,0,0">Pin 3 Affected Pins(7:0):</
                        Label>
128                 <CheckBox Height="16"  Width="18" Name="
                        Pin3AffectedPinsBit7" HorizontalAlignment="Left"
                        Margin="140,61,0,0" VerticalAlignment="Top" ></
                        CheckBox>
129                 <CheckBox Height="16"  Width="18" Name="
                        Pin3AffectedPinsBit6" HorizontalAlignment="Left"
                        Margin="155,61,0,0" VerticalAlignment="Top"></
                        CheckBox>
130                 <CheckBox Height="16"  Width="18" Name="
                        Pin3AffectedPinsBit5" HorizontalAlignment="Left"
                        Margin="170,61,0,0" VerticalAlignment="Top"></
                        CheckBox>
131                 <CheckBox Height="16"  Width="18" Name="
                        Pin3AffectedPinsBit4" HorizontalAlignment="Left"
                        Margin="185,61,0,0" VerticalAlignment="Top"></
                        CheckBox>
132                 <CheckBox Height="16"  Width="18" Name="
                        Pin3AffectedPinsBit3" HorizontalAlignment="Left"
                        Margin="200,61,0,0" VerticalAlignment="Top"></
                        CheckBox>
133                 <CheckBox Height="16" Name="Pin3AffectedPinsBit2"
                        Margin="215,61,0,0" VerticalAlignment="Top"></
                        CheckBox>
134                 <CheckBox Height="16"  Width="18" Name="
                        Pin3AffectedPinsBit1" HorizontalAlignment="Left"
                        Margin="230,61,0,0" VerticalAlignment="Top"></
                        CheckBox>
135                 <CheckBox Height="16"  Width="18" Name="
                        Pin3AffectedPinsBit0" HorizontalAlignment="left"
                        Margin="245,61,0,0" VerticalAlignment="Top"></
                        CheckBox>
136                 <Label Height="28" HorizontalAlignment="Left" Name="
                        labelPin3OutputState" VerticalAlignment="Top" Width=
                        "135" Margin="0,78,0,0">Pin 3 Output State(7:0):</
                        Label>
137                 <CheckBox Height="16"  Width="18" Name="
                        Pin3OutputStateBit7" HorizontalAlignment="Left"
                        Margin="140,86,0,0" VerticalAlignment="Top" ></
                        CheckBox>
138                 <CheckBox Height="16"  Width="18" Name="
                        Pin3OutputStateBit6" HorizontalAlignment="Left"
```

```
                    Margin="155,86,0,0" VerticalAlignment="Top"></
                    CheckBox>
139                 <CheckBox Height="16"  Width="18" Name="
                    Pin3OutputStateBit5" HorizontalAlignment="Left"
                    Margin="170,86,0,0" VerticalAlignment="Top"></
                    CheckBox>
140                 <CheckBox Height="16"  Width="18" Name="
                    Pin3OutputStateBit4" HorizontalAlignment="Left"
                    Margin="185,86,0,0" VerticalAlignment="Top"></
                    CheckBox>
141                 <CheckBox Height="16"  Width="18" Name="
                    Pin3OutputStateBit3" HorizontalAlignment="Left"
                    Margin="200,86,0,0" VerticalAlignment="Top"></
                    CheckBox>
142                 <CheckBox Height="16" Name="Pin3OutputStateBit2" Margin
                    ="215,86,0,0" VerticalAlignment="Top"></CheckBox>
143                 <CheckBox Height="16"  Width="18" Name="
                    Pin3OutputStateBit1" HorizontalAlignment="left"
                    Margin="230,86,0,0" VerticalAlignment="Top"></
                    CheckBox>
144                 <CheckBox Height="16"  Width="18" Name="
                    Pin3OutputStateBit0" HorizontalAlignment="left"
                    Margin="245,86,0,0" VerticalAlignment="Top"></
                    CheckBox>
145
146                 <Label HorizontalAlignment="Left" Name="
                    labelPin3TimeAState" Width="123" Margin="0,103,0,113
                    ">Pin 3 Time A State:</Label>
147                 <CheckBox Width="18" Name="Pin3TimeAState"
                    HorizontalAlignment="Left" Margin="120,111,0,117"></
                    CheckBox>
148
149                 <Label HorizontalAlignment="Left" Name="labelPin3TimaA"
                     Width="123" Margin="0,0,0,88" Height="28"
                    VerticalAlignment="Bottom">Pin 3 Time A:</Label>
150                 <TextBox CharacterCasing="Lower"  Margin="96,0,170,93"
                    Name="textBoxPin3TimeA" TextAlignment="Right" Height
                    ="23" VerticalAlignment="Bottom" />
151
152                 <Label Height="28" HorizontalAlignment="Left" Name="
                    labelPin3TimeBState" VerticalAlignment="Bottom"
                    Width="123" Margin="0,0,0,63">Pin 3 Time B State:</
                    Label>
153                 <CheckBox Height="16"  Width="18" Name="Pin3TimeBState"
                     HorizontalAlignment="Left" Margin="120,0,0,67"
                    VerticalAlignment="Bottom" ></CheckBox>
154
155                 <Label Height="28" HorizontalAlignment="Left" Name="
                    labelPin3TimaB" VerticalAlignment="Bottom" Width="
                    123" Margin="0,0,0,38">Pin 3 Time B:</Label>
156                 <TextBox CharacterCasing="Lower"  Margin="96,0,170,43"
                    Name="textBoxPin3TimeB" TextAlignment="Right" Height
                    ="23" VerticalAlignment="Bottom" />
157
158
159             </Grid>
160         </TabItem>
161         <TabItem Header="Pin␣4">
```

```
162             <Grid Margin="0,0,0,0" Name="gridPin4" HorizontalAlignment=
                    "Left" Width="413" Height="244" VerticalAlignment="Top">
163                 <CheckBox Height="16"  Width="18" Name="Pin4SettingBit7
                        " HorizontalAlignment="Left" Margin="110,35,0,0"
                        VerticalAlignment="Top" ></CheckBox>
164                 <CheckBox Height="16"  Width="18" Name="Pin4SettingBit6
                        " HorizontalAlignment="Left" Margin="125,35,0,0"
                        VerticalAlignment="Top"></CheckBox>
165                 <CheckBox Height="16"  Width="18" Name="Pin4SettingBit5
                        " HorizontalAlignment="Left" Margin="140,35,0,0"
                        VerticalAlignment="Top"></CheckBox>
166                 <CheckBox Height="16"  Width="18" Name="Pin4SettingBit4
                        " HorizontalAlignment="Left" Margin="155,35,0,0"
                        VerticalAlignment="Top"></CheckBox>
167                 <CheckBox Height="16"  Width="18" Name="Pin4SettingBit3
                        " HorizontalAlignment="Left" Margin="170,35,0,0"
                        VerticalAlignment="Top"></CheckBox>
168                 <CheckBox Height="16"  Width="18" Name="Pin4SettingBit2
                        " HorizontalAlignment="Left" Margin="185,35,0,0"
                        VerticalAlignment="Top"></CheckBox>
169                 <CheckBox Height="16"  Width="18" Name="Pin4SettingBit1
                        " HorizontalAlignment="Left" Margin="200,35,0,0"
                        VerticalAlignment="Top"></CheckBox>
170                 <CheckBox Height="16" Name="Pin4SettingBit0" Margin="
                        215,35,0,0" VerticalAlignment="Top"></CheckBox>
171                 <Label Height="28" HorizontalAlignment="Left" Name="
                        labelPin4Setting" VerticalAlignment="Top" Width="106
                        "  Margin="0,28,0,0">Pin 4 Settings(7:0):</Label>
172                 <Label Height="28" HorizontalAlignment="Left" Name="
                        labelPin4AffectedPins" VerticalAlignment="Top" Width
                        ="135" Margin="0,54,0,0">Pin 4 Affected Pins(7:0):</
                        Label>
173                 <CheckBox Height="16"  Width="18" Name="
                        Pin4AffectedPinsBit7" HorizontalAlignment="Left"
                        Margin="140,61,0,0" VerticalAlignment="Top" ></
                        CheckBox>
174                 <CheckBox Height="16"  Width="18" Name="
                        Pin4AffectedPinsBit6" HorizontalAlignment="Left"
                        Margin="155,61,0,0" VerticalAlignment="Top"></
                        CheckBox>
175                 <CheckBox Height="16"  Width="18" Name="
                        Pin4AffectedPinsBit5" HorizontalAlignment="Left"
                        Margin="170,61,0,0" VerticalAlignment="Top"></
                        CheckBox>
176                 <CheckBox Height="16"  Width="18" Name="
                        Pin4AffectedPinsBit4" HorizontalAlignment="Left"
                        Margin="185,61,0,0" VerticalAlignment="Top"></
                        CheckBox>
177                 <CheckBox Height="16"  Width="18" Name="
                        Pin4AffectedPinsBit3" HorizontalAlignment="Left"
                        Margin="200,61,0,0" VerticalAlignment="Top"></
                        CheckBox>
178                 <CheckBox Height="16" Name="Pin4AffectedPinsBit2"
                        Margin="215,61,0,0" VerticalAlignment="Top"></
                        CheckBox>
179                 <CheckBox Height="16"  Width="18" Name="
                        Pin4AffectedPinsBit1" HorizontalAlignment="Left"
                        Margin="230,61,0,0" VerticalAlignment="Top"></
                        CheckBox>
```

```
180                 <CheckBox Height="16"  Width="18" Name="
                        Pin4AffectedPinsBit0" HorizontalAlignment="left"
                        Margin="245,61,0,0" VerticalAlignment="Top"></
                        CheckBox>
181                 <Label Height="28" HorizontalAlignment="Left" Name="
                        labelPin4OutputState" VerticalAlignment="Top" Width=
                        "135" Margin="0,78,0,0">Pin 4 Output State(7:0):</
                        Label>
182                 <CheckBox Height="16"  Width="18" Name="
                        Pin4OutputStateBit7" HorizontalAlignment="Left"
                        Margin="140,86,0,0" VerticalAlignment="Top" ></
                        CheckBox>
183                 <CheckBox Height="16"  Width="18" Name="
                        Pin4OutputStateBit6" HorizontalAlignment="Left"
                        Margin="155,86,0,0" VerticalAlignment="Top"></
                        CheckBox>
184                 <CheckBox Height="16"  Width="18" Name="
                        Pin4OutputStateBit5" HorizontalAlignment="Left"
                        Margin="170,86,0,0" VerticalAlignment="Top"></
                        CheckBox>
185                 <CheckBox Height="16"  Width="18" Name="
                        Pin4OutputStateBit4" HorizontalAlignment="Left"
                        Margin="185,86,0,0" VerticalAlignment="Top"></
                        CheckBox>
186                 <CheckBox Height="16"  Width="18" Name="
                        Pin4OutputStateBit3" HorizontalAlignment="Left"
                        Margin="200,86,0,0" VerticalAlignment="Top"></
                        CheckBox>
187                 <CheckBox Height="16" Name="Pin4OutputStateBit2" Margin
                        ="215,86,0,0" VerticalAlignment="Top"></CheckBox>
188                 <CheckBox Height="16"  Width="18" Name="
                        Pin4OutputStateBit1" HorizontalAlignment="left"
                        Margin="230,86,0,0" VerticalAlignment="Top"></
                        CheckBox>
189                 <CheckBox Height="16"  Width="18" Name="
                        Pin4OutputStateBit0" HorizontalAlignment="left"
                        Margin="245,86,0,0" VerticalAlignment="Top"></
                        CheckBox>
190
191                 <Label HorizontalAlignment="Left" Name="
                        labelPin4TimeAState" Width="123" Margin="0,103,0,113
                        ">Pin 4 Time A State:</Label>
192                 <CheckBox Width="18" Name="Pin4TimeAState"
                        HorizontalAlignment="Left" Margin="120,111,0,117"></
                        CheckBox>
193
194                 <Label HorizontalAlignment="Left" Name="labelPin4TimaA"
                         Width="123" Margin="0,0,0,88" Height="28"
                        VerticalAlignment="Bottom">Pin 4 Time A:</Label>
195                 <TextBox CharacterCasing="Lower"  Margin="96,0,170,93"
                        Name="textBoxPin4TimeA" TextAlignment="Right" Height
                        ="23" VerticalAlignment="Bottom" />
196
197                 <Label Height="28" HorizontalAlignment="Left" Name="
                        labelPin4TimeBState" VerticalAlignment="Bottom"
                        Width="123" Margin="0,0,0,63">Pin 4 Time B State:</
                        Label>
198                 <CheckBox Height="16"  Width="18" Name="Pin4TimeBState"
                         HorizontalAlignment="Left" Margin="120,0,0,67"
```

```
                    VerticalAlignment="Bottom" ></CheckBox>
199
200                 <Label Height="28" HorizontalAlignment="Left" Name="
                        labelPin4TimaB" VerticalAlignment="Bottom" Width="
                        123" Margin="0,0,0,38">Pin 4 Time B:</Label>
201                 <TextBox CharacterCasing="Lower"  Margin="96,0,170,43"
                        Name="textBoxPin4TimeB" TextAlignment="Right" Height
                        ="23" VerticalAlignment="Bottom" />
202
203
204             </Grid>
205         </TabItem>
206         <TabItem Header="Pin 5">
207             <Grid Margin="0,0,0,0" Name="gridPin5" HorizontalAlignment=
                    "Left" Width="413" Height="244" VerticalAlignment="Top">
208                 <CheckBox Height="16"  Width="18" Name="Pin5SettingBit7
                        " HorizontalAlignment="Left" Margin="110,35,0,0"
                        VerticalAlignment="Top" ></CheckBox>
209                 <CheckBox Height="16"  Width="18" Name="Pin5SettingBit6
                        " HorizontalAlignment="Left" Margin="125,35,0,0"
                        VerticalAlignment="Top"></CheckBox>
210                 <CheckBox Height="16"  Width="18" Name="Pin5SettingBit5
                        " HorizontalAlignment="Left" Margin="140,35,0,0"
                        VerticalAlignment="Top"></CheckBox>
211                 <CheckBox Height="16"  Width="18" Name="Pin5SettingBit4
                        " HorizontalAlignment="Left" Margin="155,35,0,0"
                        VerticalAlignment="Top"></CheckBox>
212                 <CheckBox Height="16"  Width="18" Name="Pin5SettingBit3
                        " HorizontalAlignment="Left" Margin="170,35,0,0"
                        VerticalAlignment="Top"></CheckBox>
213                 <CheckBox Height="16"  Width="18" Name="Pin5SettingBit2
                        " HorizontalAlignment="Left" Margin="185,35,0,0"
                        VerticalAlignment="Top"></CheckBox>
214                 <CheckBox Height="16"  Width="18" Name="Pin5SettingBit1
                        " HorizontalAlignment="Left" Margin="200,35,0,0"
                        VerticalAlignment="Top"></CheckBox>
215                 <CheckBox Height="16" Name="Pin5SettingBit0" Margin="
                        215,35,0,0" VerticalAlignment="Top"></CheckBox>
216                 <Label Height="28" HorizontalAlignment="Left" Name="
                        labelPin5Setting" VerticalAlignment="Top" Width="106
                        "   Margin="0,28,0,0">Pin 5 Settings(7:0):</Label>
217                 <Label Height="28" HorizontalAlignment="Left" Name="
                        labelPin5AffectedPins" VerticalAlignment="Top" Width
                        ="135" Margin="0,54,0,0">Pin 5 Affected Pins(7:0):</
                        Label>
218                 <CheckBox Height="16"  Width="18" Name="
                        Pin5AffectedPinsBit7" HorizontalAlignment="Left"
                        Margin="140,61,0,0" VerticalAlignment="Top" ></
                        CheckBox>
219                 <CheckBox Height="16"  Width="18" Name="
                        Pin5AffectedPinsBit6" HorizontalAlignment="Left"
                        Margin="155,61,0,0" VerticalAlignment="Top"></
                        CheckBox>
220                 <CheckBox Height="16"  Width="18" Name="
                        Pin5AffectedPinsBit5" HorizontalAlignment="Left"
                        Margin="170,61,0,0" VerticalAlignment="Top"></
                        CheckBox>
221                 <CheckBox Height="16"  Width="18" Name="
                        Pin5AffectedPinsBit4" HorizontalAlignment="Left"
```

```
                Margin="185,61,0,0" VerticalAlignment="Top"></
                CheckBox>
222             <CheckBox Height="16"  Width="18" Name="
                Pin5AffectedPinsBit3" HorizontalAlignment="Left"
                Margin="200,61,0,0" VerticalAlignment="Top"></
                CheckBox>
223             <CheckBox Height="16" Name="Pin5AffectedPinsBit2"
                Margin="215,61,0,0" VerticalAlignment="Top"></
                CheckBox>
224             <CheckBox Height="16"  Width="18" Name="
                Pin5AffectedPinsBit1" HorizontalAlignment="Left"
                Margin="230,61,0,0" VerticalAlignment="Top"></
                CheckBox>
225             <CheckBox Height="16"  Width="18" Name="
                Pin5AffectedPinsBit0" HorizontalAlignment="left"
                Margin="245,61,0,0" VerticalAlignment="Top"></
                CheckBox>
226             <Label Height="28" HorizontalAlignment="Left" Name="
                labelPin5OutputState" VerticalAlignment="Top" Width=
                "135" Margin="0,78,0,0">Pin 5 Output State(7:0):</
                Label>
227             <CheckBox Height="16"  Width="18" Name="
                Pin5OutputStateBit7" HorizontalAlignment="Left"
                Margin="140,86,0,0" VerticalAlignment="Top" ></
                CheckBox>
228             <CheckBox Height="16"  Width="18" Name="
                Pin5OutputStateBit6" HorizontalAlignment="Left"
                Margin="155,86,0,0" VerticalAlignment="Top"></
                CheckBox>
229             <CheckBox Height="16"  Width="18" Name="
                Pin5OutputStateBit5" HorizontalAlignment="Left"
                Margin="170,86,0,0" VerticalAlignment="Top"></
                CheckBox>
230             <CheckBox Height="16"  Width="18" Name="
                Pin5OutputStateBit4" HorizontalAlignment="Left"
                Margin="185,86,0,0" VerticalAlignment="Top"></
                CheckBox>
231             <CheckBox Height="16"  Width="18" Name="
                Pin5OutputStateBit3" HorizontalAlignment="Left"
                Margin="200,86,0,0" VerticalAlignment="Top"></
                CheckBox>
232             <CheckBox Height="16" Name="Pin5OutputStateBit2" Margin
                ="215,86,0,0" VerticalAlignment="Top"></CheckBox>
233             <CheckBox Height="16"  Width="18" Name="
                Pin5OutputStateBit1" HorizontalAlignment="left"
                Margin="230,86,0,0" VerticalAlignment="Top"></
                CheckBox>
234             <CheckBox Height="16"  Width="18" Name="
                Pin5OutputStateBit0" HorizontalAlignment="left"
                Margin="245,86,0,0" VerticalAlignment="Top"></
                CheckBox>
235
236             <Label HorizontalAlignment="Left" Name="
                labelPin5TimeAState" Width="123" Margin="0,103,0,113
                ">Pin 5 Time A State:</Label>
237             <CheckBox Width="18" Name="Pin5TimeAState"
                HorizontalAlignment="Left" Margin="120,111,0,117"></
                CheckBox>
238
```

```
239                     <Label HorizontalAlignment="Left" Name="labelPin5TimaA"
                            Width="123" Margin="0,0,0,88" Height="28"
                            VerticalAlignment="Bottom">Pin 5 Time A:</Label>
240                     <TextBox CharacterCasing="Lower"  Margin="96,0,170,93"
                            Name="textBoxPin5TimeA" TextAlignment="Right" Height
                            ="23" VerticalAlignment="Bottom" />
241
242                     <Label Height="28" HorizontalAlignment="Left" Name="
                            labelPin5TimeBState" VerticalAlignment="Bottom"
                            Width="123" Margin="0,0,0,63">Pin 5 Time B State:</
                            Label>
243                     <CheckBox Height="16"  Width="18" Name="Pin5TimeBState"
                            HorizontalAlignment="Left" Margin="120,0,0,67"
                            VerticalAlignment="Bottom" ></CheckBox>
244
245                     <Label Height="28" HorizontalAlignment="Left" Name="
                            labelPin5TimaB" VerticalAlignment="Bottom" Width="
                            123" Margin="0,0,0,38">Pin 5 Time B:</Label>
246                     <TextBox CharacterCasing="Lower"  Margin="96,0,170,43"
                            Name="textBoxPin5TimeB" TextAlignment="Right" Height
                            ="23" VerticalAlignment="Bottom" />
247
248
249                 </Grid>
250             </TabItem>
251             <TabItem Header="Pin␣6">
252                 <Grid Margin="0,0,0,0" Name="gridPin6" HorizontalAlignment=
                        "Left" Width="413" Height="244" VerticalAlignment="Top">
253                     <CheckBox Height="16"  Width="18" Name="Pin6SettingBit7
                            " HorizontalAlignment="Left" Margin="110,35,0,0"
                            VerticalAlignment="Top" ></CheckBox>
254                     <CheckBox Height="16"  Width="18" Name="Pin6SettingBit6
                            " HorizontalAlignment="Left" Margin="125,35,0,0"
                            VerticalAlignment="Top"></CheckBox>
255                     <CheckBox Height="16"  Width="18" Name="Pin6SettingBit5
                            " HorizontalAlignment="Left" Margin="140,35,0,0"
                            VerticalAlignment="Top"></CheckBox>
256                     <CheckBox Height="16"  Width="18" Name="Pin6SettingBit4
                            " HorizontalAlignment="Left" Margin="155,35,0,0"
                            VerticalAlignment="Top"></CheckBox>
257                     <CheckBox Height="16"  Width="18" Name="Pin6SettingBit3
                            " HorizontalAlignment="Left" Margin="170,35,0,0"
                            VerticalAlignment="Top"></CheckBox>
258                     <CheckBox Height="16"  Width="18" Name="Pin6SettingBit2
                            " HorizontalAlignment="Left" Margin="185,35,0,0"
                            VerticalAlignment="Top"></CheckBox>
259                     <CheckBox Height="16"  Width="18" Name="Pin6SettingBit1
                            " HorizontalAlignment="Left" Margin="200,35,0,0"
                            VerticalAlignment="Top"></CheckBox>
260                     <CheckBox Height="16" Name="Pin6SettingBit0" Margin="
                            215,35,0,0" VerticalAlignment="Top"></CheckBox>
261                     <Label Height="28" HorizontalAlignment="Left" Name="
                            labelPin6Setting" VerticalAlignment="Top" Width="106
                            "    Margin="0,28,0,0">Pin 6 Settings(7:0):</Label>
262                     <Label Height="28" HorizontalAlignment="Left" Name="
                            labelPin6AffectedPins" VerticalAlignment="Top" Width
                            ="135" Margin="0,54,0,0">Pin 6 Affected Pins(7:0):</
                            Label>
```

```
263                 <CheckBox Height="16"  Width="18" Name="
                        Pin6AffectedPinsBit7" HorizontalAlignment="Left"
                        Margin="140,61,0,0" VerticalAlignment="Top" ></
                        CheckBox>
264                 <CheckBox Height="16"  Width="18" Name="
                        Pin6AffectedPinsBit6" HorizontalAlignment="Left"
                        Margin="155,61,0,0" VerticalAlignment="Top"></
                        CheckBox>
265                 <CheckBox Height="16"  Width="18" Name="
                        Pin6AffectedPinsBit5" HorizontalAlignment="Left"
                        Margin="170,61,0,0" VerticalAlignment="Top"></
                        CheckBox>
266                 <CheckBox Height="16"  Width="18" Name="
                        Pin6AffectedPinsBit4" HorizontalAlignment="Left"
                        Margin="185,61,0,0" VerticalAlignment="Top"></
                        CheckBox>
267                 <CheckBox Height="16"  Width="18" Name="
                        Pin6AffectedPinsBit3" HorizontalAlignment="Left"
                        Margin="200,61,0,0" VerticalAlignment="Top"></
                        CheckBox>
268                 <CheckBox Height="16" Name="Pin6AffectedPinsBit2"
                        Margin="215,61,0,0" VerticalAlignment="Top"></
                        CheckBox>
269                 <CheckBox Height="16"  Width="18" Name="
                        Pin6AffectedPinsBit1" HorizontalAlignment="Left"
                        Margin="230,61,0,0" VerticalAlignment="Top"></
                        CheckBox>
270                 <CheckBox Height="16"  Width="18" Name="
                        Pin6AffectedPinsBit0" HorizontalAlignment="left"
                        Margin="245,61,0,0" VerticalAlignment="Top"></
                        CheckBox>
271                 <Label Height="28" HorizontalAlignment="Left" Name="
                        labelPin6OutputState" VerticalAlignment="Top" Width=
                        "135" Margin="0,78,0,0">Pin 6 Output State(7:0):</
                        Label>
272                 <CheckBox Height="16"  Width="18" Name="
                        Pin6OutputStateBit7" HorizontalAlignment="Left"
                        Margin="140,86,0,0" VerticalAlignment="Top" ></
                        CheckBox>
273                 <CheckBox Height="16"  Width="18" Name="
                        Pin6OutputStateBit6" HorizontalAlignment="Left"
                        Margin="155,86,0,0" VerticalAlignment="Top"></
                        CheckBox>
274                 <CheckBox Height="16"  Width="18" Name="
                        Pin6OutputStateBit5" HorizontalAlignment="Left"
                        Margin="170,86,0,0" VerticalAlignment="Top"></
                        CheckBox>
275                 <CheckBox Height="16"  Width="18" Name="
                        Pin6OutputStateBit4" HorizontalAlignment="Left"
                        Margin="185,86,0,0" VerticalAlignment="Top"></
                        CheckBox>
276                 <CheckBox Height="16"  Width="18" Name="
                        Pin6OutputStateBit3" HorizontalAlignment="Left"
                        Margin="200,86,0,0" VerticalAlignment="Top"></
                        CheckBox>
277                 <CheckBox Height="16" Name="Pin6OutputStateBit2" Margin
                        ="215,86,0,0" VerticalAlignment="Top"></CheckBox>
278                 <CheckBox Height="16"  Width="18" Name="
                        Pin6OutputStateBit1" HorizontalAlignment="left"
```

```
                    Margin="230,86,0,0" VerticalAlignment="Top"></
                    CheckBox>
279                 <CheckBox Height="16"  Width="18" Name="
                    Pin6OutputStateBit0" HorizontalAlignment="left"
                    Margin="245,86,0,0" VerticalAlignment="Top"></
                    CheckBox>
280
281                 <Label HorizontalAlignment="Left" Name="
                    labelPin6TimeAState" Width="123" Margin="0,103,0,113
                    ">Pin 6 Time A State:</Label>
282                 <CheckBox Width="18" Name="Pin6TimeAState"
                    HorizontalAlignment="Left" Margin="120,111,0,117"></
                    CheckBox>
283
284                 <Label HorizontalAlignment="Left" Name="labelPin6TimaA"
                     Width="123" Margin="0,0,0,88" Height="28"
                    VerticalAlignment="Bottom">Pin 6 Time A:</Label>
285                 <TextBox CharacterCasing="Lower"  Margin="96,0,170,93"
                    Name="textBoxPin6TimeA" TextAlignment="Right" Height
                    ="23" VerticalAlignment="Bottom" />
286
287                 <Label Height="28" HorizontalAlignment="Left" Name="
                    labelPin6TimeBState" VerticalAlignment="Bottom"
                    Width="123" Margin="0,0,0,63">Pin 6 Time B State:</
                    Label>
288                 <CheckBox Height="16"  Width="18" Name="Pin6TimeBState"
                     HorizontalAlignment="Left" Margin="120,0,0,67"
                    VerticalAlignment="Bottom" ></CheckBox>
289
290                 <Label Height="28" HorizontalAlignment="Left" Name="
                    labelPin6TimaB" VerticalAlignment="Bottom" Width="
                    123" Margin="0,0,0,38">Pin 6 Time B:</Label>
291                 <TextBox CharacterCasing="Lower"  Margin="96,0,170,43"
                    Name="textBoxPin6TimeB" TextAlignment="Right" Height
                    ="23" VerticalAlignment="Bottom" />
292
293
294             </Grid>
295         </TabItem>
296         <TabItem Header="Pin␣7">
297             <Grid Margin="0,0,0,0" Name="gridPin7" HorizontalAlignment=
                    "Left" Width="413" Height="244" VerticalAlignment="Top">
298                 <CheckBox Height="16"  Width="18" Name="Pin7SettingBit7
                    " HorizontalAlignment="Left" Margin="110,35,0,0"
                    VerticalAlignment="Top" ></CheckBox>
299                 <CheckBox Height="16"  Width="18" Name="Pin7SettingBit6
                    " HorizontalAlignment="Left" Margin="125,35,0,0"
                    VerticalAlignment="Top"></CheckBox>
300                 <CheckBox Height="16"  Width="18" Name="Pin7SettingBit5
                    " HorizontalAlignment="Left" Margin="140,35,0,0"
                    VerticalAlignment="Top"></CheckBox>
301                 <CheckBox Height="16"  Width="18" Name="Pin7SettingBit4
                    " HorizontalAlignment="Left" Margin="155,35,0,0"
                    VerticalAlignment="Top"></CheckBox>
302                 <CheckBox Height="16"  Width="18" Name="Pin7SettingBit3
                    " HorizontalAlignment="Left" Margin="170,35,0,0"
                    VerticalAlignment="Top"></CheckBox>
303                 <CheckBox Height="16"  Width="18" Name="Pin7SettingBit2
                    " HorizontalAlignment="Left" Margin="185,35,0,0"
```

```
                VerticalAlignment="Top"></CheckBox>
304             <CheckBox Height="16"  Width="18" Name="Pin7SettingBit1
                " HorizontalAlignment="Left" Margin="200,35,0,0"
                VerticalAlignment="Top"></CheckBox>
305             <CheckBox Height="16" Name="Pin7SettingBit0" Margin="
                215,35,0,0" VerticalAlignment="Top"></CheckBox>
306             <Label Height="28" HorizontalAlignment="Left" Name="
                labelPin7Setting" VerticalAlignment="Top" Width="106
                "   Margin="0,28,0,0">Pin 7 Settings(7:0):</Label>
307             <Label Height="28" HorizontalAlignment="Left" Name="
                labelPin7AffectedPins" VerticalAlignment="Top" Width
                ="135" Margin="0,54,0,0">Pin 7 Affected Pins(7:0):</
                Label>
308             <CheckBox Height="16"  Width="18" Name="
                Pin7AffectedPinsBit7" HorizontalAlignment="Left"
                Margin="140,61,0,0" VerticalAlignment="Top" ></
                CheckBox>
309             <CheckBox Height="16"  Width="18" Name="
                Pin7AffectedPinsBit6" HorizontalAlignment="Left"
                Margin="155,61,0,0" VerticalAlignment="Top"></
                CheckBox>
310             <CheckBox Height="16"  Width="18" Name="
                Pin7AffectedPinsBit5" HorizontalAlignment="Left"
                Margin="170,61,0,0" VerticalAlignment="Top"></
                CheckBox>
311             <CheckBox Height="16"  Width="18" Name="
                Pin7AffectedPinsBit4" HorizontalAlignment="Left"
                Margin="185,61,0,0" VerticalAlignment="Top"></
                CheckBox>
312             <CheckBox Height="16"  Width="18" Name="
                Pin7AffectedPinsBit3" HorizontalAlignment="Left"
                Margin="200,61,0,0" VerticalAlignment="Top"></
                CheckBox>
313             <CheckBox Height="16" Name="Pin7AffectedPinsBit2"
                Margin="215,61,0,0" VerticalAlignment="Top"></
                CheckBox>
314             <CheckBox Height="16"  Width="18" Name="
                Pin7AffectedPinsBit1" HorizontalAlignment="Left"
                Margin="230,61,0,0" VerticalAlignment="Top"></
                CheckBox>
315             <CheckBox Height="16"  Width="18" Name="
                Pin7AffectedPinsBit0" HorizontalAlignment="left"
                Margin="245,61,0,0" VerticalAlignment="Top"></
                CheckBox>
316             <Label Height="28" HorizontalAlignment="Left" Name="
                labelPin7OutputState" VerticalAlignment="Top" Width=
                "135" Margin="0,78,0,0">Pin 7 Output State(7:0):</
                Label>
317             <CheckBox Height="16"  Width="18" Name="
                Pin7OutputStateBit7" HorizontalAlignment="Left"
                Margin="140,86,0,0" VerticalAlignment="Top" ></
                CheckBox>
318             <CheckBox Height="16"  Width="18" Name="
                Pin7OutputStateBit6" HorizontalAlignment="Left"
                Margin="155,86,0,0" VerticalAlignment="Top"></
                CheckBox>
319             <CheckBox Height="16"  Width="18" Name="
                Pin7OutputStateBit5" HorizontalAlignment="Left"
                Margin="170,86,0,0" VerticalAlignment="Top"></
```

```
                    CheckBox>
320                 <CheckBox Height="16"  Width="18" Name="
                        Pin7OutputStateBit4" HorizontalAlignment="Left"
                        Margin="185,86,0,0" VerticalAlignment="Top"></
                        CheckBox>
321                 <CheckBox Height="16"  Width="18" Name="
                        Pin7OutputStateBit3" HorizontalAlignment="Left"
                        Margin="200,86,0,0" VerticalAlignment="Top"></
                        CheckBox>
322                 <CheckBox Height="16" Name="Pin7OutputStateBit2" Margin
                        ="215,86,0,0" VerticalAlignment="Top"></CheckBox>
323                 <CheckBox Height="16"  Width="18" Name="
                        Pin7OutputStateBit1" HorizontalAlignment="left"
                        Margin="230,86,0,0" VerticalAlignment="Top"></
                        CheckBox>
324                 <CheckBox Height="16"  Width="18" Name="
                        Pin7OutputStateBit0" HorizontalAlignment="left"
                        Margin="245,86,0,0" VerticalAlignment="Top"></
                        CheckBox>
325
326                 <Label HorizontalAlignment="Left" Name="
                        labelPin7TimeAState" Width="123" Margin="0,103,0,113
                        ">Pin 7 Time A State:</Label>
327                 <CheckBox Width="18" Name="Pin7TimeAState"
                        HorizontalAlignment="Left" Margin="120,111,0,117"></
                        CheckBox>
328
329                 <Label HorizontalAlignment="Left" Name="labelPin7TimaA"
                         Width="123" Margin="0,0,0,88" Height="28"
                        VerticalAlignment="Bottom">Pin 7 Time A:</Label>
330                 <TextBox CharacterCasing="Lower"  Margin="96,0,170,93"
                        Name="textBoxPin7TimeA" TextAlignment="Right" Height
                        ="23" VerticalAlignment="Bottom" />
331
332                 <Label Height="28" HorizontalAlignment="Left" Name="
                        labelPin7TimeBState" VerticalAlignment="Bottom"
                        Width="123" Margin="0,0,0,63">Pin 7 Time B State:</
                        Label>
333                 <CheckBox Height="16"  Width="18" Name="Pin7TimeBState"
                         HorizontalAlignment="Left" Margin="120,0,0,67"
                        VerticalAlignment="Bottom" ></CheckBox>
334
335                 <Label Height="28" HorizontalAlignment="Left" Name="
                        labelPin7TimaB" VerticalAlignment="Bottom" Width="
                        123" Margin="0,0,0,38">Pin 7 Time B:</Label>
336                 <TextBox CharacterCasing="Lower"  Margin="96,0,170,43"
                        Name="textBoxPin7TimeB" TextAlignment="Right" Height
                        ="23" VerticalAlignment="Bottom" />
337
338
339             </Grid>
340         </TabItem>
341         <TabItem Header="Pin␣8">
342             <Grid Margin="0,0,0,0" Name="gridPin8" HorizontalAlignment=
                    "Left" Width="413" Height="244" VerticalAlignment="Top">
343                 <CheckBox Height="16"  Width="18" Name="Pin8SettingBit7
                        " HorizontalAlignment="Left" Margin="110,35,0,0"
                        VerticalAlignment="Top" ></CheckBox>
```

```
344                 <CheckBox Height="16"  Width="18" Name="Pin8SettingBit6
                        " HorizontalAlignment="Left" Margin="125,35,0,0"
                        VerticalAlignment="Top"></CheckBox>
345                 <CheckBox Height="16"  Width="18" Name="Pin8SettingBit5
                        " HorizontalAlignment="Left" Margin="140,35,0,0"
                        VerticalAlignment="Top"></CheckBox>
346                 <CheckBox Height="16"  Width="18" Name="Pin8SettingBit4
                        " HorizontalAlignment="Left" Margin="155,35,0,0"
                        VerticalAlignment="Top"></CheckBox>
347                 <CheckBox Height="16"  Width="18" Name="Pin8SettingBit3
                        " HorizontalAlignment="Left" Margin="170,35,0,0"
                        VerticalAlignment="Top"></CheckBox>
348                 <CheckBox Height="16"  Width="18" Name="Pin8SettingBit2
                        " HorizontalAlignment="Left" Margin="185,35,0,0"
                        VerticalAlignment="Top"></CheckBox>
349                 <CheckBox Height="16"  Width="18" Name="Pin8SettingBit1
                        " HorizontalAlignment="Left" Margin="200,35,0,0"
                        VerticalAlignment="Top"></CheckBox>
350                 <CheckBox Height="16" Name="Pin8SettingBit0" Margin="
                        215,35,0,0" VerticalAlignment="Top"></CheckBox>
351                 <Label Height="28" HorizontalAlignment="Left" Name="
                        labelPin8Setting" VerticalAlignment="Top" Width="106
                        "  Margin="0,28,0,0">Pin 8 Settings(7:0):</Label>
352                 <Label Height="28" HorizontalAlignment="Left" Name="
                        labelPin8AffectedPins" VerticalAlignment="Top" Width
                        ="135" Margin="0,54,0,0">Pin 8 Affected Pins(7:0):</
                        Label>
353                 <CheckBox Height="16"  Width="18" Name="
                        Pin8AffectedPinsBit7" HorizontalAlignment="Left"
                        Margin="140,61,0,0" VerticalAlignment="Top" ></
                        CheckBox>
354                 <CheckBox Height="16"  Width="18" Name="
                        Pin8AffectedPinsBit6" HorizontalAlignment="Left"
                        Margin="155,61,0,0" VerticalAlignment="Top"></
                        CheckBox>
355                 <CheckBox Height="16"  Width="18" Name="
                        Pin8AffectedPinsBit5" HorizontalAlignment="Left"
                        Margin="170,61,0,0" VerticalAlignment="Top"></
                        CheckBox>
356                 <CheckBox Height="16"  Width="18" Name="
                        Pin8AffectedPinsBit4" HorizontalAlignment="Left"
                        Margin="185,61,0,0" VerticalAlignment="Top"></
                        CheckBox>
357                 <CheckBox Height="16"  Width="18" Name="
                        Pin8AffectedPinsBit3" HorizontalAlignment="Left"
                        Margin="200,61,0,0" VerticalAlignment="Top"></
                        CheckBox>
358                 <CheckBox Height="16" Name="Pin8AffectedPinsBit2"
                        Margin="215,61,0,0" VerticalAlignment="Top"></
                        CheckBox>
359                 <CheckBox Height="16"  Width="18" Name="
                        Pin8AffectedPinsBit1" HorizontalAlignment="Left"
                        Margin="230,61,0,0" VerticalAlignment="Top"></
                        CheckBox>
360                 <CheckBox Height="16"  Width="18" Name="
                        Pin8AffectedPinsBit0" HorizontalAlignment="left"
                        Margin="245,61,0,0" VerticalAlignment="Top"></
                        CheckBox>
```

```
361             <Label Height="28" HorizontalAlignment="Left" Name="
                    labelPin8OutputState" VerticalAlignment="Top" Width=
                    "135" Margin="0,78,0,0">Pin 8 Output State(7:0):</
                    Label>
362             <CheckBox Height="16"  Width="18" Name="
                    Pin8OutputStateBit7" HorizontalAlignment="Left"
                    Margin="140,86,0,0" VerticalAlignment="Top" ></
                    CheckBox>
363             <CheckBox Height="16"  Width="18" Name="
                    Pin8OutputStateBit6" HorizontalAlignment="Left"
                    Margin="155,86,0,0" VerticalAlignment="Top"></
                    CheckBox>
364             <CheckBox Height="16"  Width="18" Name="
                    Pin8OutputStateBit5" HorizontalAlignment="Left"
                    Margin="170,86,0,0" VerticalAlignment="Top"></
                    CheckBox>
365             <CheckBox Height="16"  Width="18" Name="
                    Pin8OutputStateBit4" HorizontalAlignment="Left"
                    Margin="185,86,0,0" VerticalAlignment="Top"></
                    CheckBox>
366             <CheckBox Height="16"  Width="18" Name="
                    Pin8OutputStateBit3" HorizontalAlignment="Left"
                    Margin="200,86,0,0" VerticalAlignment="Top"></
                    CheckBox>
367             <CheckBox Height="16" Name="Pin8OutputStateBit2" Margin
                    ="215,86,0,0" VerticalAlignment="Top"></CheckBox>
368             <CheckBox Height="16"  Width="18" Name="
                    Pin8OutputStateBit1" HorizontalAlignment="left"
                    Margin="230,86,0,0" VerticalAlignment="Top"></
                    CheckBox>
369             <CheckBox Height="16"  Width="18" Name="
                    Pin8OutputStateBit0" HorizontalAlignment="left"
                    Margin="245,86,0,0" VerticalAlignment="Top"></
                    CheckBox>
370
371             <Label HorizontalAlignment="Left" Name="
                    labelPin8TimeAState" Width="123" Margin="0,103,0,113
                    ">Pin 8 Time A State:</Label>
372             <CheckBox Width="18" Name="Pin8TimeAState"
                    HorizontalAlignment="Left" Margin="120,111,0,117"></
                    CheckBox>
373
374             <Label HorizontalAlignment="Left" Name="labelPin8TimaA"
                     Width="123" Margin="0,0,0,88" Height="28"
                    VerticalAlignment="Bottom">Pin 8 Time A:</Label>
375             <TextBox CharacterCasing="Lower"  Margin="96,0,170,93"
                    Name="textBoxPin8TimeA" TextAlignment="Right" Height
                    ="23" VerticalAlignment="Bottom" />
376
377             <Label Height="28" HorizontalAlignment="Left" Name="
                    labelPin8TimeBState" VerticalAlignment="Bottom"
                    Width="123" Margin="0,0,0,63">Pin 8 Time B State:</
                    Label>
378             <CheckBox Height="16"  Width="18" Name="Pin8TimeBState"
                     HorizontalAlignment="Left" Margin="120,0,0,67"
                    VerticalAlignment="Bottom" ></CheckBox>
379
380             <Label Height="28" HorizontalAlignment="Left" Name="
                    labelPin8TimaB" VerticalAlignment="Bottom" Width="
```

```
                    123" Margin="0,0,0,38">Pin 8 Time B:</Label>
381                 <TextBox CharacterCasing="Lower"  Margin="96,0,170,43"
                        Name="textBoxPin8TimeB" TextAlignment="Right" Height
                        ="23" VerticalAlignment="Bottom" />
382
383
384                 </Grid>
385             </TabItem>
386         </TabControl>
387         <Button Height="29" IsEnabled="False" Margin="244,223,0,0" Name="
                buttonPinSettingsWrite" VerticalAlignment="Top" Click="
                buttonCommandClick" HorizontalAlignment="Left" Width="130">Pin
                Setting ÃŒbertragen</Button>
388         <Button Height="29" IsEnabled="False" Margin="392,177.48,0,0" Name=
                "buttonTemp" VerticalAlignment="Top"  HorizontalAlignment="Left"
                 Width="71" Click="buttonCommandClick">Temp</Button>
389         <TextBox CharacterCasing="Lower" Height="23" IsEnabled="False"
                Margin="392,226.52,0,0" Name="textBoxTemp" TextAlignment="Right"
                 VerticalAlignment="Top" HorizontalAlignment="Left" Width="71" /
                >
390     </Grid>
391 </Window>
```

Window1.xaml.cs

```
1  ï»¿
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Windows;
7  using System.Windows.Controls;
8  using System.Windows.Data;
9  using System.Windows.Documents;
10 using System.Windows.Input;
11 using System.Windows.Media;
12 using System.Windows.Media.Imaging;
13 using System.Windows.Navigation;
14 using System.Windows.Shapes;
15 using System.Globalization;
16 using System.Collections.ObjectModel;
17 using System.Net.Sockets;
18 using System.Net.NetworkInformation;
19 using System.Net;
20 using System.Threading;
21 using System.Windows.Threading;
22
23 namespace WPFNetzwerk
24 {
25
26     /// <summary>
27     /// Interaction logic for Window1.xaml
28     /// </summary>
29     public partial class Window1 : Window
30     {
31         Int32 Port;
32         Int32 TransactionID=1;
33         char DeviceID = (char) 0x33;
34         IPEndPoint localEndPoint;
```

```
35          UdpClient server = null;
36          IPEndPoint rxPoint;
37          IPEndPoint targetEndPoint;
38          Thread serverThread = null;
39
40          //Warten auf Nachrichten vom Modul
41          private int OpenCode = 0;
42          private UInt16 OpenState = 0;
43
44          public struct Pin_A
45          {
46              public byte Settings;
47              public UInt32 Reaction;
48              public UInt32 Time;
49          };
50
51          public struct PinDetails
52          {
53              public CheckBox[] PinSetting;
54              public CheckBox[] PinAffected;
55              public CheckBox[] PinOutput;
56              public CheckBox PinAState;
57              public CheckBox PinBState;
58              public TextBox  textBoxTimeA;
59              public TextBox  textBoxTimeB;
60          };
61
62          public Window1()
63          {
64              InitializeComponent();
65              Port = 11000;
66              textBox_Port.Text = Port.ToString();
67              textBox_IP.Text = "192.168.20.19";
68          }
69
70          private void textBox_Port_TextChanged(object sender,
                TextChangedEventArgs e)
71          {
72
73              char[] array = textBox_Port.Text.ToCharArray();
74
75             //Ueberprueft jedes Zeichen ob es eine Integer ist
76              for (int i = 0; i < array.Length; i++)
77              {
78                  if (!char.IsDigit(array[i]))
79                  {
80                      textBox_Port.Text = Port.ToString();
81                      return;
82                  }
83              }
84
85              Port = Convert.ToInt32(textBox_Port.Text);
86
87          }
88
89          private void buttonStart_Click(object sender, RoutedEventArgs e)
90          {
91
92              try
```

```
93              {
94                  Port = Convert.ToInt32(textBox_Port.Text);
95                  // Create an instance of IPAddress for the specified
                        address string (in
96                  // dotted-quad, or colon-hexadecimal notation).
97                  IPAddress address = IPAddress.Parse(textBox_IP.Text.
                        ToString());
98                  localEndPoint = new IPEndPoint(address, Port);
99                  rxPoint = new IPEndPoint(IPAddress.Any, 0);
100                 targetEndPoint = new IPEndPoint(IPAddress.Broadcast, Port);
101                 server = new UdpClient(localEndPoint);
102                 serverThread = new Thread(new ThreadStart(
                        WaitForDataFromClient));
103                 serverThread.Start();
104
105                 //Controls freischalten
106                 buttonStart.IsEnabled = false;
107                 textBox_IP.IsEnabled = false;
108                 textBox_Port.IsEnabled = false;
109                 labelIP.IsEnabled = false;
110                 labelPort.IsEnabled = false;
111
112                 buttonUDPBroadcast.IsEnabled = true;
113                 listBoxFoundModul.IsEnabled = true;
114                 labelFoundModul.IsEnabled = true;
115                 labelZielIP.IsEnabled = true;
116                 textBox_ZielIP.IsEnabled = true;
117                 labelInformation.IsEnabled = true;
118                 listBoxInformation.IsEnabled = true;
119                 labelGetClass.IsEnabled = true;
120                 buttonGetClass.IsEnabled = true;
121                 buttonPinSettingsRead.IsEnabled = true;
122                 buttonPinSettingsSave.IsEnabled = true;
123                 buttonPinSettingsWrite.IsEnabled = true;
124                 TabPin.IsEnabled = true;
125                 buttonTemp.IsEnabled = true;
126                 textBox_ZielIP.Text = "192.168.20.50";
127             }
128             catch (Exception ex)
129             {
130                 MessageBox.Show("Message:␣" + ex.Message,"Fehler",
                        MessageBoxButton.OK,MessageBoxImage.Error);
131             }
132         }
133
134         //Funktion ueberprueft ob es sich um ein Protokoll UDP Paket
                handelt
135         //Return 0: Nein
136         //Return 1: Ja
137         char CheckProtokoll(byte[] ClientData)
138         {
139
140             int ShouldLength;
141
142             ShouldLength = ClientData[6] + ((int)ClientData[7] >> 8);
143
144             if (ClientData.Count() != ShouldLength + 8)
145             {
146                 return (char) 0;
```

```
147             }
148             else
149             {
150                 return (char)1;
151             }
152 
153         }
154 
155         private void WaitForDataFromClient()
156         {
157             while (true)
158             {
159                 byte[] ClientData = server.Receive(ref rxPoint);
160 
161                // string ReturnValue = Encoding.ASCII.GetString(ClientData)
                       ;
162 
163                 if (ClientData.Count() < 8)
164                 {
165                     try
166                     {
167                         listBoxFoundModul.Dispatcher.BeginInvoke(
168                         DispatcherPriority.Normal,
169                         new UDPReceive(ShowMessageListInformation), "
                               Received␣Small␣UDP!");
170                     }
171                     catch (Exception ex)
172                     {
173                         MessageBox.Show(ex.ToString());
174                     }
175                 }
176 
177                 if (ClientData.Count() == 22 && CheckProtokoll(ClientData)
                       == 0)
178                 {
179                     string Ausgabe = "IP␣Modul:␣" + ClientData[0].ToString
                           () + "." + ClientData[1].ToString() + "." +
                           ClientData[2].ToString() + "." + ClientData[3].
                           ToString()+ "␣";
180                     Ausgabe = Ausgabe + "MAC␣Modul:␣" + Convert.ToString(
                           ClientData[4], 16) + "-" + Convert.ToString(
                           ClientData[5], 16) + "-" + Convert.ToString(
                           ClientData[6], 16) + "-" + Convert.ToString(
                           ClientData[7], 16) + "-" + Convert.ToString(
                           ClientData[8], 16) + "-" + Convert.ToString(
                           ClientData[9], 16) + "␣";
181                     Ausgabe = Ausgabe +  "IP␣Sender␣letzen␣Befehles:␣" +
                           ClientData[10].ToString() + "." + ClientData[11].
                           ToString() + "." + ClientData[12].ToString() + "." +
                            ClientData[13].ToString() + "␣";
182                     Ausgabe = Ausgabe + "MAC␣Sender␣letzen␣Befehles:␣" +
                           Convert.ToString(ClientData[14], 16) + "-" + Convert
                           .ToString(ClientData[15], 16) + "-" + Convert.
                           ToString(ClientData[16], 16) + "-" + Convert.
                           ToString(ClientData[17], 16) + "-" + Convert.
                           ToString(ClientData[18], 16) + "-" + Convert.
                           ToString(ClientData[19], 16) + "␣";
183                     Ausgabe = Ausgabe + "Ziel␣Port␣letzen␣Befehles:␣" +
                           Convert.ToString((ClientData[20] << 8) + ClientData
```

```
                    [21], 10);
184
185
186                 //listBoxFoundModul.Dispatcher.BeginInvoke(new
                        ShowMessageList(string i)
187                 //, DispatcherPriority.Normal, Ausgabe);
188
189                 try
190                 {
191                     listBoxFoundModul.Dispatcher.BeginInvoke(
192                     DispatcherPriority.Normal,
193                     new BroadcastReceive(ShowMessageList),Ausgabe);
194                 }
195                 catch (Exception ex)
196                 {
197                     MessageBox.Show(ex.ToString());
198                 }
199             }
200
201             if (CheckProtokoll(ClientData) == 1)
202             {
203                 //fuer uns
204                 try
205                 {
206                     listBoxFoundModul.Dispatcher.BeginInvoke(
207                     DispatcherPriority.Normal,
208                     new UDPReceive(ShowMessageListInformation), "
                            Antwort empfangen!");
209                 }
210                 catch (Exception ex)
211                 {
212                     MessageBox.Show(ex.ToString());
213                 }
214
215                 //Paket zum verarbeiten weitergeben
216
217                 try
218                 {
219                     listBoxFoundModul.Dispatcher.BeginInvoke(
220                     DispatcherPriority.Normal,
221                     new UDPReceiveStateMachine(StateMachine),
                            ClientData);
222                 }
223                 catch (Exception ex)
224                 {
225                     MessageBox.Show(ex.ToString());
226                 }
227             }
228             else
229             {
230                 //Was anderes
231                 try
232                 {
233                     listBoxFoundModul.Dispatcher.BeginInvoke(
234                     DispatcherPriority.Normal,
235                     new UDPReceive(ShowMessageListInformation), "
                            Komisches UDP Paket!");
236                 }
237                 catch (Exception ex)
```

```
238                     {
239                         MessageBox.Show(ex.ToString());
240                     }
241                 }
242 
243             }
244         }
245 
246         public delegate void BroadcastReceive(string message);
247 
248         void ShowMessageList(string message)
249         {
250             listBoxFoundModul.Items.Add(message);
251         }
252 
253         public delegate void UDPReceive(string message);
254 
255         void ShowMessageListInformation(string message)
256         {
257             listBoxInformation.Items.Add(message);
258         }
259 
260 
261         private void buttonUDPBroadcast_Click(object sender,
                RoutedEventArgs e)
262         {
263             DateTime time = DateTime.Now;
264 
265             try
266             {
267                 Byte[] sendBytes = Encoding.ASCII.GetBytes("Tell␣me!");
268                 int numBytesSent = server.Send(sendBytes, sendBytes.Length,
                        targetEndPoint);
269 
270                 listBoxInformation.Items.Add(time.ToString() + "␣" + "UDP␣
                    Broadcast");
271 
272 
273                 if (listBoxFoundModul.Items.Count > 0)
274                 {
275                     //Leeren
276                     listBoxFoundModul.Items.Clear();
277                 }
278 
279             }
280             catch (Exception ex)
281             {
282                 MessageBox.Show(ex.ToString());
283             }
284         }
285 
286         //Funktion versendet ein UDP Paket
287         private void SendUDP(int code, int parametercount, byte []
                Parameter)
288         {
289             IPAddress address = IPAddress.Parse(textBox_ZielIP.Text.
                ToString());
290 
```

```
291             SendContainer(code, (char) parametercount, Parameter, (char)
                    TransactionID);
292
293             //TransactionID erhoehen
294             TransactionID++;
295         }
296
297         private void buttonCommandClick(object sender, RoutedEventArgs e)
298         {
299
300             //Code der uebertragen werden solll
301             int code;
302
303             //Anzahl der Parameter
304             int parametercount;
305
306             //Array mit den Parametern
307             byte[] parameter = new byte[20];
308
309             Button Sender = (Button)sender;
310             MessageBoxResult returnValue = MessageBoxResult.None;
311
312             DateTime time = DateTime.Now;
313
314             try
315             {
316                 // Create an instance of IPAddress for the specified
                        address string (in
317                 // dotted-quad, or colon-hexadecimal notation).
318                 IPAddress address = IPAddress.Parse(textBox_ZielIP.Text.
                        ToString());
319             }
320             catch (Exception ex)
321             {
322                 MessageBox.Show("Ziel IP ungÃŒltig! " + ex.Message, "Fehler
                        ", MessageBoxButton.OK, MessageBoxImage.Error);
323                 return;
324             }
325
326             if (OpenCode != 0)
327             {
328                 returnValue = MessageBox.Show("Es ist noch eine Antwort vom
                        Modul ausstÃ€ndig, wollen Sie trotzdem den Befehl
                        senden?", "Antwort ausstÃ€ndig", MessageBoxButton.YesNo,
                         MessageBoxImage.Question);
329             }
330
331             //Feststellen welcher Knopf ausgeloest wurde
332             switch (Sender.Name.ToString())
333             {
334                 case "buttonTemp":
335                     if (returnValue == MessageBoxResult.None || returnValue
                            == MessageBoxResult.Yes)
336                     {
337                         OpenCode = Commands.COMMAND_READ_TEMP;
338                         OpenState = 1;
339
340                         code = ((int)DeviceID << 24) | 0x00020000 |
                                Commands.COMMAND_READ_TEMP; // DeviceID Klasse
```

```
                    AD Klasse Code
341                 parametercount = 0;
342                 SendUDP(code, parametercount, parameter);
343
344                 listBoxInformation.Items.Add(time.ToString() + "␣"
                        + "COMMAND_READ_TEMP");
345
346                 //XX RÃŒckgabe chechek
347             }
348             break;
349
350         case "buttonGetClass":
351             if (returnValue == MessageBoxResult.None || returnValue
                    == MessageBoxResult.Yes)
352             {
353                 OpenCode = Commands.COMMAND_GETCLASS;
354                 OpenState = 1;
355
356                 code = ((int) DeviceID<<24) | 0x00010000 | Commands
                        .COMMAND_GETCLASS; // DeviceID Klasse AD Klasse
                        Code
357                 parametercount = 0;
358                 SendUDP(code, parametercount, parameter);
359
360                 listBoxInformation.Items.Add(time.ToString() + "␣"
                        + "COMMAND_GETCLASS");
361             }
362             break;
363         case "buttonPinSettingsSave":
364             if (returnValue == MessageBoxResult.None || returnValue
                    == MessageBoxResult.Yes)
365             {
366
367                 OpenCode = Commands.COMMAND_PIN_STORE;
368                 OpenState = 1;
369
370                 code = ((int)DeviceID << 24) | 0x00010000 |
                        Commands.COMMAND_PIN_STORE; // DeviceID Klasse
                        AD Klasse Code
371                 parametercount = 0;
372
373                 SendUDP(code, parametercount, parameter);
374
375                 listBoxInformation.Items.Add(time.ToString() + "␣"
                        + "COMMAND_PIN_STORE");
376
377             }
378             break;
379         case "buttonPinSettingsRead":
380             if (returnValue == MessageBoxResult.None || returnValue
                    == MessageBoxResult.Yes)
381             {
382                 TabItem CurrentItem = (TabItem)TabPin.SelectedItem;
383
384                 OpenCode = Commands.COMMAND_PIN_READ;
385                 OpenState = 1;
386
387                 code = ((int)DeviceID << 24) | 0x00010000 |
                        Commands.COMMAND_PIN_READ; // DeviceID Klasse AD
```

```
                        Klasse Code
388                     parametercount = 1;
389
390                     //Welcher PinXX
391                     parameter[0] =  (byte)(Convert.ToByte(CurrentItem.
                            Header.ToString().Replace("Pin ","")) -1);
392
393                     SendUDP(code, parametercount, parameter);
394
395                     listBoxInformation.Items.Add(time.ToString() + " "
                            + "COMMAND_PIN_READ");
396
397
398                 }
399                 break;
400             case "buttonPinSettingsWrite":
401                 if (returnValue == MessageBoxResult.None || returnValue
                        == MessageBoxResult.Yes)
402                 {
403                     TabItem CurrentItem = (TabItem)TabPin.SelectedItem;
404                     Pin_A PinA;
405
406
407
408                     code = ((int)DeviceID << 24) | 0x00010000 |
                            Commands.COMMAND_PIN_SET; // DeviceID Klasse AD
                            Klasse Code
409                     parametercount = 10;
410
411                     //Welcher PinXX
412                     parameter[0] = (byte) (Convert.ToByte(CurrentItem.
                            Header.ToString().Replace("Pin ", "")) - 1);
413                     try
414                     {
415                         PinA = GetPinDetails(Convert.ToByte(CurrentItem
                                .Header.ToString().Replace("Pin ", "")));
416                     }
417                     catch (Exception ex)
418                     {
419                         MessageBox.Show(ex.Message, "Fehler",
                                MessageBoxButton.OK, MessageBoxImage.Error);
420                         return;
421                     }
422
423                     OpenCode = Commands.COMMAND_PIN_SET;
424                     OpenState = 1;
425
426                     parameter[1] = PinA.Settings;
427                     parameter[2] = (byte)(PinA.Reaction & 0xFF);
428                     parameter[3] = (byte)((PinA.Reaction>>8) & 0xFF);
429                     parameter[4] = (byte)((PinA.Reaction >> 16) & 0xFF)
                            ;
430                     parameter[5] = (byte)((PinA.Reaction >> 24) & 0xFF)
                            ;
431                     parameter[6] = (byte)(PinA.Time & 0xFF);
432                     parameter[7] = (byte)((PinA.Time >> 8) & 0xFF);
433                     parameter[8] = (byte)((PinA.Time >> 16) & 0xFF);
434                     parameter[9] = (byte)((PinA.Time >> 24) & 0xFF);
435
```

```
436                         SendUDP(code, parametercount, parameter);
437
438                         listBoxInformation.Items.Add(time.ToString() + "␣"
                                + "COMMAND_PIN_SET");
439
440                     }
441                     break;
442                 default:
443                     MessageBox.Show("Unbekannter␣Befehl!", "unbekannt",
                            MessageBoxButton.OK, MessageBoxImage.Information);
444                     break;
445             }
446
447         }
448
449         //Prozedur nimmt die richtigen Pin Felder
450         private void FillPinFelder(ref PinDetails PinFelder, int Number)
451         {
452             switch (Number)
453             {
454                 case 1:
455                     //Settings
456                     PinFelder.PinSetting[0] = Pin1SettingBit0;
457                     PinFelder.PinSetting[1] = Pin1SettingBit1;
458                     PinFelder.PinSetting[2] = Pin1SettingBit2;
459                     PinFelder.PinSetting[3] = Pin1SettingBit3;
460                     PinFelder.PinSetting[4] = Pin1SettingBit4;
461                     PinFelder.PinSetting[5] = Pin1SettingBit5;
462                     PinFelder.PinSetting[6] = Pin1SettingBit6;
463                     PinFelder.PinSetting[7] = Pin1SettingBit7;
464
465                     //affected
466                     PinFelder.PinAffected[0] = Pin1AffectedPinsBit0;
467                     PinFelder.PinAffected[1] = Pin1AffectedPinsBit1;
468                     PinFelder.PinAffected[2] = Pin1AffectedPinsBit2;
469                     PinFelder.PinAffected[3] = Pin1AffectedPinsBit3;
470                     PinFelder.PinAffected[4] = Pin1AffectedPinsBit4;
471                     PinFelder.PinAffected[5] = Pin1AffectedPinsBit5;
472                     PinFelder.PinAffected[6] = Pin1AffectedPinsBit6;
473                     PinFelder.PinAffected[7] = Pin1AffectedPinsBit7;
474
475                     //Output
476                     PinFelder.PinOutput[0] = Pin1OutputStateBit0;
477                     PinFelder.PinOutput[1] = Pin1OutputStateBit1;
478                     PinFelder.PinOutput[2] = Pin1OutputStateBit2;
479                     PinFelder.PinOutput[3] = Pin1OutputStateBit3;
480                     PinFelder.PinOutput[4] = Pin1OutputStateBit4;
481                     PinFelder.PinOutput[5] = Pin1OutputStateBit5;
482                     PinFelder.PinOutput[6] = Pin1OutputStateBit6;
483                     PinFelder.PinOutput[7] = Pin1OutputStateBit7;
484
485                     //Time A
486                     PinFelder.PinAState = Pin1TimeAState;
487                     PinFelder.textBoxTimeA = textBoxPin1TimeA;
488
489                     //Time B
490                     PinFelder.PinBState = Pin1TimeBState;
491                     PinFelder.textBoxTimeB = textBoxPin1TimeB;
492                     break;
```

```
493                 case 2:
494                     //Settings
495                     PinFelder.PinSetting[0] = Pin2SettingBit0;
496                     PinFelder.PinSetting[1] = Pin2SettingBit1;
497                     PinFelder.PinSetting[2] = Pin2SettingBit2;
498                     PinFelder.PinSetting[3] = Pin2SettingBit3;
499                     PinFelder.PinSetting[4] = Pin2SettingBit4;
500                     PinFelder.PinSetting[5] = Pin2SettingBit5;
501                     PinFelder.PinSetting[6] = Pin2SettingBit6;
502                     PinFelder.PinSetting[7] = Pin2SettingBit7;
503
504                     //affected
505                     PinFelder.PinAffected[0] = Pin2AffectedPinsBit0;
506                     PinFelder.PinAffected[1] = Pin2AffectedPinsBit1;
507                     PinFelder.PinAffected[2] = Pin2AffectedPinsBit2;
508                     PinFelder.PinAffected[3] = Pin2AffectedPinsBit3;
509                     PinFelder.PinAffected[4] = Pin2AffectedPinsBit4;
510                     PinFelder.PinAffected[5] = Pin2AffectedPinsBit5;
511                     PinFelder.PinAffected[6] = Pin2AffectedPinsBit6;
512                     PinFelder.PinAffected[7] = Pin2AffectedPinsBit7;
513
514                     //Output
515                     PinFelder.PinOutput[0] = Pin2OutputStateBit0;
516                     PinFelder.PinOutput[1] = Pin2OutputStateBit1;
517                     PinFelder.PinOutput[2] = Pin2OutputStateBit2;
518                     PinFelder.PinOutput[3] = Pin2OutputStateBit3;
519                     PinFelder.PinOutput[4] = Pin2OutputStateBit4;
520                     PinFelder.PinOutput[5] = Pin2OutputStateBit5;
521                     PinFelder.PinOutput[6] = Pin2OutputStateBit6;
522                     PinFelder.PinOutput[7] = Pin2OutputStateBit7;
523
524                     //Time A
525                     PinFelder.PinAState = Pin2TimeAState;
526                     PinFelder.textBoxTimeA = textBoxPin2TimeA;
527
528                     //Time B
529                     PinFelder.PinBState = Pin2TimeBState;
530                     PinFelder.textBoxTimeB = textBoxPin2TimeB;
531                     break;
532                 case 3:
533                     //Settings
534                     PinFelder.PinSetting[0] = Pin3SettingBit0;
535                     PinFelder.PinSetting[1] = Pin3SettingBit1;
536                     PinFelder.PinSetting[2] = Pin3SettingBit2;
537                     PinFelder.PinSetting[3] = Pin3SettingBit3;
538                     PinFelder.PinSetting[4] = Pin3SettingBit4;
539                     PinFelder.PinSetting[5] = Pin3SettingBit5;
540                     PinFelder.PinSetting[6] = Pin3SettingBit6;
541                     PinFelder.PinSetting[7] = Pin3SettingBit7;
542
543                     //affected
544                     PinFelder.PinAffected[0] = Pin3AffectedPinsBit0;
545                     PinFelder.PinAffected[1] = Pin3AffectedPinsBit1;
546                     PinFelder.PinAffected[2] = Pin3AffectedPinsBit2;
547                     PinFelder.PinAffected[3] = Pin3AffectedPinsBit3;
548                     PinFelder.PinAffected[4] = Pin3AffectedPinsBit4;
549                     PinFelder.PinAffected[5] = Pin3AffectedPinsBit5;
550                     PinFelder.PinAffected[6] = Pin3AffectedPinsBit6;
551                     PinFelder.PinAffected[7] = Pin3AffectedPinsBit7;
```

```
552
553                     //Output
554                     PinFelder.PinOutput[0] = Pin3OutputStateBit0;
555                     PinFelder.PinOutput[1] = Pin3OutputStateBit1;
556                     PinFelder.PinOutput[2] = Pin3OutputStateBit2;
557                     PinFelder.PinOutput[3] = Pin3OutputStateBit3;
558                     PinFelder.PinOutput[4] = Pin3OutputStateBit4;
559                     PinFelder.PinOutput[5] = Pin3OutputStateBit5;
560                     PinFelder.PinOutput[6] = Pin3OutputStateBit6;
561                     PinFelder.PinOutput[7] = Pin3OutputStateBit7;
562
563                     //Time A
564                     PinFelder.PinAState = Pin3TimeAState;
565                     PinFelder.textBoxTimeA = textBoxPin3TimeA;
566
567                     //Time B
568                     PinFelder.PinBState = Pin3TimeBState;
569                     PinFelder.textBoxTimeB = textBoxPin3TimeB;
570                     break;
571                 case 4:
572                     //Settings
573                     PinFelder.PinSetting[0] = Pin4SettingBit0;
574                     PinFelder.PinSetting[1] = Pin4SettingBit1;
575                     PinFelder.PinSetting[2] = Pin4SettingBit2;
576                     PinFelder.PinSetting[3] = Pin4SettingBit3;
577                     PinFelder.PinSetting[4] = Pin4SettingBit4;
578                     PinFelder.PinSetting[5] = Pin4SettingBit5;
579                     PinFelder.PinSetting[6] = Pin4SettingBit6;
580                     PinFelder.PinSetting[7] = Pin4SettingBit7;
581
582                     //affected
583                     PinFelder.PinAffected[0] = Pin4AffectedPinsBit0;
584                     PinFelder.PinAffected[1] = Pin4AffectedPinsBit1;
585                     PinFelder.PinAffected[2] = Pin4AffectedPinsBit2;
586                     PinFelder.PinAffected[3] = Pin4AffectedPinsBit3;
587                     PinFelder.PinAffected[4] = Pin4AffectedPinsBit4;
588                     PinFelder.PinAffected[5] = Pin4AffectedPinsBit5;
589                     PinFelder.PinAffected[6] = Pin4AffectedPinsBit6;
590                     PinFelder.PinAffected[7] = Pin4AffectedPinsBit7;
591
592                     //Output
593                     PinFelder.PinOutput[0] = Pin4OutputStateBit0;
594                     PinFelder.PinOutput[1] = Pin4OutputStateBit1;
595                     PinFelder.PinOutput[2] = Pin4OutputStateBit2;
596                     PinFelder.PinOutput[3] = Pin4OutputStateBit3;
597                     PinFelder.PinOutput[4] = Pin4OutputStateBit4;
598                     PinFelder.PinOutput[5] = Pin4OutputStateBit5;
599                     PinFelder.PinOutput[6] = Pin4OutputStateBit6;
600                     PinFelder.PinOutput[7] = Pin4OutputStateBit7;
601
602                     //Time A
603                     PinFelder.PinAState = Pin4TimeAState;
604                     PinFelder.textBoxTimeA = textBoxPin4TimeA;
605
606                     //Time B
607                     PinFelder.PinBState = Pin4TimeBState;
608                     PinFelder.textBoxTimeB = textBoxPin4TimeB;
609                     break;
610
```

```
611                 case 5:
612                     //Settings
613                     PinFelder.PinSetting[0] = Pin5SettingBit0;
614                     PinFelder.PinSetting[1] = Pin5SettingBit1;
615                     PinFelder.PinSetting[2] = Pin5SettingBit2;
616                     PinFelder.PinSetting[3] = Pin5SettingBit3;
617                     PinFelder.PinSetting[4] = Pin5SettingBit4;
618                     PinFelder.PinSetting[5] = Pin5SettingBit5;
619                     PinFelder.PinSetting[6] = Pin5SettingBit6;
620                     PinFelder.PinSetting[7] = Pin5SettingBit7;
621
622                     //affected
623                     PinFelder.PinAffected[0] = Pin5AffectedPinsBit0;
624                     PinFelder.PinAffected[1] = Pin5AffectedPinsBit1;
625                     PinFelder.PinAffected[2] = Pin5AffectedPinsBit2;
626                     PinFelder.PinAffected[3] = Pin5AffectedPinsBit3;
627                     PinFelder.PinAffected[4] = Pin5AffectedPinsBit4;
628                     PinFelder.PinAffected[5] = Pin5AffectedPinsBit5;
629                     PinFelder.PinAffected[6] = Pin5AffectedPinsBit6;
630                     PinFelder.PinAffected[7] = Pin5AffectedPinsBit7;
631
632                     //Output
633                     PinFelder.PinOutput[0] = Pin5OutputStateBit0;
634                     PinFelder.PinOutput[1] = Pin5OutputStateBit1;
635                     PinFelder.PinOutput[2] = Pin5OutputStateBit2;
636                     PinFelder.PinOutput[3] = Pin5OutputStateBit3;
637                     PinFelder.PinOutput[4] = Pin5OutputStateBit4;
638                     PinFelder.PinOutput[5] = Pin5OutputStateBit5;
639                     PinFelder.PinOutput[6] = Pin5OutputStateBit6;
640                     PinFelder.PinOutput[7] = Pin5OutputStateBit7;
641
642                     //Time A
643                     PinFelder.PinAState = Pin5TimeAState;
644                     PinFelder.textBoxTimeA = textBoxPin5TimeA;
645
646                     //Time B
647                     PinFelder.PinBState = Pin5TimeBState;
648                     PinFelder.textBoxTimeB = textBoxPin5TimeB;
649                     break;
650
651                 case 6:
652                     //Settings
653                     PinFelder.PinSetting[0] = Pin6SettingBit0;
654                     PinFelder.PinSetting[1] = Pin6SettingBit1;
655                     PinFelder.PinSetting[2] = Pin6SettingBit2;
656                     PinFelder.PinSetting[3] = Pin6SettingBit3;
657                     PinFelder.PinSetting[4] = Pin6SettingBit4;
658                     PinFelder.PinSetting[5] = Pin6SettingBit5;
659                     PinFelder.PinSetting[6] = Pin6SettingBit6;
660                     PinFelder.PinSetting[7] = Pin6SettingBit7;
661
662                     //affected
663                     PinFelder.PinAffected[0] = Pin6AffectedPinsBit0;
664                     PinFelder.PinAffected[1] = Pin6AffectedPinsBit1;
665                     PinFelder.PinAffected[2] = Pin6AffectedPinsBit2;
666                     PinFelder.PinAffected[3] = Pin6AffectedPinsBit3;
667                     PinFelder.PinAffected[4] = Pin6AffectedPinsBit4;
668                     PinFelder.PinAffected[5] = Pin6AffectedPinsBit5;
669                     PinFelder.PinAffected[6] = Pin6AffectedPinsBit6;
```

```
670                     PinFelder.PinAffected[7] = Pin6AffectedPinsBit7;
671
672                     //Output
673                     PinFelder.PinOutput[0] = Pin6OutputStateBit0;
674                     PinFelder.PinOutput[1] = Pin6OutputStateBit1;
675                     PinFelder.PinOutput[2] = Pin6OutputStateBit2;
676                     PinFelder.PinOutput[3] = Pin6OutputStateBit3;
677                     PinFelder.PinOutput[4] = Pin6OutputStateBit4;
678                     PinFelder.PinOutput[5] = Pin6OutputStateBit5;
679                     PinFelder.PinOutput[6] = Pin6OutputStateBit6;
680                     PinFelder.PinOutput[7] = Pin6OutputStateBit7;
681
682                     //Time A
683                     PinFelder.PinAState = Pin6TimeAState;
684                     PinFelder.textBoxTimeA = textBoxPin6TimeA;
685
686                     //Time B
687                     PinFelder.PinBState = Pin6TimeBState;
688                     PinFelder.textBoxTimeB = textBoxPin6TimeB;
689                     break;
690
691
692                 case 7:
693                     //Settings
694                     PinFelder.PinSetting[0] = Pin7SettingBit0;
695                     PinFelder.PinSetting[1] = Pin7SettingBit1;
696                     PinFelder.PinSetting[2] = Pin7SettingBit2;
697                     PinFelder.PinSetting[3] = Pin7SettingBit3;
698                     PinFelder.PinSetting[4] = Pin7SettingBit4;
699                     PinFelder.PinSetting[5] = Pin7SettingBit5;
700                     PinFelder.PinSetting[6] = Pin7SettingBit6;
701                     PinFelder.PinSetting[7] = Pin7SettingBit7;
702
703                     //affected
704                     PinFelder.PinAffected[0] = Pin7AffectedPinsBit0;
705                     PinFelder.PinAffected[1] = Pin7AffectedPinsBit1;
706                     PinFelder.PinAffected[2] = Pin7AffectedPinsBit2;
707                     PinFelder.PinAffected[3] = Pin7AffectedPinsBit3;
708                     PinFelder.PinAffected[4] = Pin7AffectedPinsBit4;
709                     PinFelder.PinAffected[5] = Pin7AffectedPinsBit5;
710                     PinFelder.PinAffected[6] = Pin7AffectedPinsBit6;
711                     PinFelder.PinAffected[7] = Pin7AffectedPinsBit7;
712
713                     //Output
714                     PinFelder.PinOutput[0] = Pin7OutputStateBit0;
715                     PinFelder.PinOutput[1] = Pin7OutputStateBit1;
716                     PinFelder.PinOutput[2] = Pin7OutputStateBit2;
717                     PinFelder.PinOutput[3] = Pin7OutputStateBit3;
718                     PinFelder.PinOutput[4] = Pin7OutputStateBit4;
719                     PinFelder.PinOutput[5] = Pin7OutputStateBit5;
720                     PinFelder.PinOutput[6] = Pin7OutputStateBit6;
721                     PinFelder.PinOutput[7] = Pin7OutputStateBit7;
722
723                     //Time A
724                     PinFelder.PinAState = Pin7TimeAState;
725                     PinFelder.textBoxTimeA = textBoxPin7TimeA;
726
727                     //Time B
728                     PinFelder.PinBState = Pin7TimeBState;
```

```
729                     PinFelder.textBoxTimeB = textBoxPin7TimeB;
730                     break;
731
732                 case 8:
733                     //Settings
734                     PinFelder.PinSetting[0] = Pin8SettingBit0;
735                     PinFelder.PinSetting[1] = Pin8SettingBit1;
736                     PinFelder.PinSetting[2] = Pin8SettingBit2;
737                     PinFelder.PinSetting[3] = Pin8SettingBit3;
738                     PinFelder.PinSetting[4] = Pin8SettingBit4;
739                     PinFelder.PinSetting[5] = Pin8SettingBit5;
740                     PinFelder.PinSetting[6] = Pin8SettingBit6;
741                     PinFelder.PinSetting[7] = Pin8SettingBit7;
742
743                     //affected
744                     PinFelder.PinAffected[0] = Pin8AffectedPinsBit0;
745                     PinFelder.PinAffected[1] = Pin8AffectedPinsBit1;
746                     PinFelder.PinAffected[2] = Pin8AffectedPinsBit2;
747                     PinFelder.PinAffected[3] = Pin8AffectedPinsBit3;
748                     PinFelder.PinAffected[4] = Pin8AffectedPinsBit4;
749                     PinFelder.PinAffected[5] = Pin8AffectedPinsBit5;
750                     PinFelder.PinAffected[6] = Pin8AffectedPinsBit6;
751                     PinFelder.PinAffected[7] = Pin8AffectedPinsBit7;
752
753                     //Output
754                     PinFelder.PinOutput[0] = Pin8OutputStateBit0;
755                     PinFelder.PinOutput[1] = Pin8OutputStateBit1;
756                     PinFelder.PinOutput[2] = Pin8OutputStateBit2;
757                     PinFelder.PinOutput[3] = Pin8OutputStateBit3;
758                     PinFelder.PinOutput[4] = Pin8OutputStateBit4;
759                     PinFelder.PinOutput[5] = Pin8OutputStateBit5;
760                     PinFelder.PinOutput[6] = Pin8OutputStateBit6;
761                     PinFelder.PinOutput[7] = Pin8OutputStateBit7;
762
763                     //Time A
764                     PinFelder.PinAState = Pin8TimeAState;
765                     PinFelder.textBoxTimeA = textBoxPin8TimeA;
766
767                     //Time B
768                     PinFelder.PinBState = Pin8TimeBState;
769                     PinFelder.textBoxTimeB = textBoxPin8TimeB;
770                     break;
771
772             }
773         }
774
775         //Prozedure befÃŒllt die ÃŒbergebenen Pin check boxen anhand der
                Struktur von den Pins
776         private void FillsPinA(Pin_A PinA,PinDetails PinFelder )
777         {
778             //Time A fÃŒllen
779             if ((PinA.Time & 0x80000000) == 0x80000000)
780             {
781                 PinFelder.PinAState.IsChecked = true;
782             }
783             else
784             {
785                 PinFelder.PinAState.IsChecked = false;
786             }
```

```
787
788             PinFelder.textBoxTimeA.Text = Convert.ToInt32(((PinA.Time >>
                    16) & 0x7FFF)).ToString();
789
790             //Time B fÃŒllen
791             if ((PinA.Time & 0x00008000) == 0x00008000)
792             {
793                 PinFelder.PinBState.IsChecked = true;
794             }
795             else
796             {
797                 PinFelder.PinBState.IsChecked = false;
798             }
799
800             PinFelder.textBoxTimeB.Text = Convert.ToInt32((PinA.Time & 0
                    x7FFF)).ToString();
801
802             //Setting
803             if ((PinA.Settings & 0x01) == 0x01)
804             {
805                 PinFelder.PinSetting[0].IsChecked = true;
806             }
807             else
808             {
809                 PinFelder.PinSetting[0].IsChecked = false;
810             }
811
812             if ((PinA.Settings & 0x02) == 0x02)
813             {
814                 PinFelder.PinSetting[1].IsChecked = true;
815             }
816             else
817             {
818                 PinFelder.PinSetting[1].IsChecked = false;
819             }
820
821             if ((PinA.Settings & 0x04) == 0x04)
822             {
823                 PinFelder.PinSetting[2].IsChecked = true;
824             }
825             else
826             {
827                 PinFelder.PinSetting[2].IsChecked = false;
828             }
829
830             if ((PinA.Settings & 0x08) == 0x08)
831             {
832                 PinFelder.PinSetting[3].IsChecked = true;
833             }
834             else
835             {
836                 PinFelder.PinSetting[3].IsChecked = false;
837             }
838
839             if ((PinA.Settings & 0x10) == 0x10)
840             {
841                 PinFelder.PinSetting[4].IsChecked = true;
842             }
843             else
```

```
844             {
845                 PinFelder.PinSetting[4].IsChecked = false;
846             }
847 
848             if ((PinA.Settings & 0x20) == 0x20)
849             {
850                 PinFelder.PinSetting[5].IsChecked = true;
851             }
852             else
853             {
854                 PinFelder.PinSetting[5].IsChecked = false;
855             }
856 
857             if ((PinA.Settings & 0x40) == 0x40)
858             {
859                 PinFelder.PinSetting[6].IsChecked = true;
860             }
861             else
862             {
863                 PinFelder.PinSetting[6].IsChecked = false;
864             }
865 
866             if ((PinA.Settings & 0x80) == 0x80)
867             {
868                 PinFelder.PinSetting[7].IsChecked = true;
869             }
870             else
871             {
872                 PinFelder.PinSetting[7].IsChecked = false;
873             }
874 
875             //Output
876             byte Output = (byte)(PinA.Reaction & 0xFF);
877 
878             if ((Output & 0x01) == 0x01)
879             {
880                 PinFelder.PinOutput[0].IsChecked = true;
881             }
882             else
883             {
884                 PinFelder.PinOutput[0].IsChecked = false;
885             }
886 
887             if ((Output & 0x02) == 0x02)
888             {
889                 PinFelder.PinOutput[1].IsChecked = true;
890             }
891             else
892             {
893                 PinFelder.PinOutput[1].IsChecked = false;
894             }
895 
896             if ((Output & 0x04) == 0x04)
897             {
898                 PinFelder.PinOutput[2].IsChecked = true;
899             }
900             else
901             {
902                 PinFelder.PinOutput[2].IsChecked = false;
```

```
903             }
904
905             if ((Output & 0x08) == 0x08)
906             {
907                 PinFelder.PinOutput[3].IsChecked = true;
908             }
909             else
910             {
911                 PinFelder.PinOutput[3].IsChecked = false;
912             }
913
914             if ((Output & 0x10) == 0x10)
915             {
916                 PinFelder.PinOutput[4].IsChecked = true;
917             }
918             else
919             {
920                 PinFelder.PinOutput[4].IsChecked = false;
921             }
922
923             if ((Output & 0x20) == 0x20)
924             {
925                 PinFelder.PinOutput[5].IsChecked = true;
926             }
927             else
928             {
929                 PinFelder.PinOutput[5].IsChecked = false;
930             }
931
932             if ((Output & 0x40) == 0x40)
933             {
934                 PinFelder.PinOutput[6].IsChecked = true;
935             }
936             else
937             {
938                 PinFelder.PinOutput[6].IsChecked = false;
939             }
940
941             if ((Output & 0x80) == 0x80)
942             {
943                 PinFelder.PinOutput[7].IsChecked = true;
944             }
945             else
946             {
947                 PinFelder.PinOutput[7].IsChecked = false;
948             }
949
950             //Affected
951             Output = (byte)((PinA.Reaction>>16) & 0xFF);
952
953             if ((Output & 0x01) == 0x01)
954             {
955                 PinFelder.PinAffected[0].IsChecked = true;
956             }
957             else
958             {
959                 PinFelder.PinAffected[0].IsChecked = false;
960             }
961
```

```
962             if ((Output & 0x02) == 0x02)
963             {
964                 PinFelder.PinAffected[1].IsChecked = true;
965             }
966             else
967             {
968                 PinFelder.PinAffected[1].IsChecked = false;
969             }
970 
971             if ((Output & 0x04) == 0x04)
972             {
973                 PinFelder.PinAffected[2].IsChecked = true;
974             }
975             else
976             {
977                 PinFelder.PinAffected[2].IsChecked = false;
978             }
979 
980             if ((Output & 0x08) == 0x08)
981             {
982                 PinFelder.PinAffected[3].IsChecked = true;
983             }
984             else
985             {
986                 PinFelder.PinAffected[3].IsChecked = false;
987             }
988 
989             if ((Output & 0x10) == 0x10)
990             {
991                 PinFelder.PinAffected[4].IsChecked = true;
992             }
993             else
994             {
995                 PinFelder.PinAffected[4].IsChecked = false;
996             }
997 
998             if ((Output & 0x20) == 0x20)
999             {
1000                PinFelder.PinAffected[5].IsChecked = true;
1001            }
1002            else
1003            {
1004                PinFelder.PinAffected[5].IsChecked = false;
1005            }
1006 
1007            if ((Output & 0x40) == 0x40)
1008            {
1009                PinFelder.PinAffected[6].IsChecked = true;
1010            }
1011            else
1012            {
1013                PinFelder.PinAffected[6].IsChecked = false;
1014            }
1015 
1016            if ((Output & 0x80) == 0x80)
1017            {
1018                PinFelder.PinAffected[7].IsChecked = true;
1019            }
1020            else
```

```
1021            {
1022                PinFelder.PinAffected[7].IsChecked = false;
1023            }
1024
1025            return;
1026        }
1027
1028        //Fills the Parameter Array
1029        //Count is the amount of parameters in bytes
1030        //MaxCount = 2 (actual
1031        public void FillsParameterList(byte[] array, int count, char
               Parameter1)
1032        {
1033            switch (count)
1034            {
1035
1036                case 2:
1037                    array[1] = (byte)(Parameter1 >> 8);
1038                    array[0] = (byte)Parameter1;
1039                    break;
1040                case 1:
1041                    array[0] = (byte)Parameter1;
1042                    break;
1043                default:
1044                    return;
1045            }
1046        }
1047
1048        //Funktion ÃŒberprÃŒft das Codewort in der Response Phase
1049        //0 alles passt
1050        //1 Fehler
1051        private int CheckAnswer(UInt32 code)
1052        {
1053            String ValueReturn;
1054            DateTime time = DateTime.Now;
1055
1056            ValueReturn = "";
1057
1058            switch (code & 0xFFFFFF)
1059            {
1060                case Commands.RESPONSE_OK:
1061                    listBoxInformation.Items.Add(time.ToString() + "_" + "
                           RESPONSE_OK");
1062                    return 0;
1063                case Commands.RESPONSE_WRONG_PARAMETER:
1064                    ValueReturn = "RESPONSE_WRONG_PARAMETER";
1065                    break;
1066                case Commands.RESPONSE_NOT_SUPPORTED_CLASS:
1067                    ValueReturn = "RESPONSE_NOT_SUPPORTED_CLASS";
1068                    break;
1069                case Commands.RESPONSE_NOT_SUPPORTED_ADCLASS:
1070                    ValueReturn = "RESPONSE_NOT_SUPPORTED_ADCLASS";
1071                    break;
1072                case Commands.RESPONSE_NOT_SUPPORTED_PARAMETERCOUNT:
1073                    ValueReturn = "RESPONSE_NOT_SUPPORTED_PARAMETERCOUNT";
1074                    break;
1075                case Commands.RESPONSE_NOT_SUPPORTED:
1076                    ValueReturn = "RESPONSE_NOT_SUPPORTED";
1077                    break;
```

```
1078                case Commands.RESPONSE_NOT_SUPPORTED_PHASE:
1079                    ValueReturn = "RESPONSE_NOT_SUPPORTED_PHASE";
1080                    break;
1081                case Commands.RESPONSE_ERROR:
1082                    ValueReturn = "RESPONSE_ERROR";
1083                    break;
1084                case Commands.RESPONSE_PTP_ERROR:
1085                    ValueReturn = "RESPONSE_PTP_ERROR";
1086                    break;
1087                case Commands.RESPONSE_DATA:
1088                    ValueReturn = "RESPONSE_PTP_ERROR";
1089                    break;
1090            }
1091            listBoxInformation.Items.Add(time.ToString() + "␣" +
                    ValueReturn);
1092            return 1;
1093        }
1094
1095        //CreateContainer
1096        //Funktion die den zu uebertragenden Container generiert
1097        //Es wird keine dynamische Speicherverwaltung verwendet,
1098        //damit auch groessere Daten transferiert werden koennen
1099        // @Param1: Zu uebertragender Code
1100        // @Param2: Anzahl der Parameter
1101        // @Param3: Parameter Array
1102        // @Param4: Ab wann Parameter uebertragen werden (inklusive) (0
                wenn alles in einem Container uebertragen werden kann)
1103        // @Param5: Bis wohin Parameter uebertragen werden (inklusive)
1104        // @Param6: Container mit maximaler Groesse
1105        // @Param7: TransactionID die zu uebertragen ist
1106        private int CreateContainer(int  CodeWort, char countParam, byte []
                arrayParam, char start, char stop, byte [] Container, int
                TransID)
1107        {
1108                int      i;
1109                int      lengthContainer; //LÃ€nge des Containers in Byte
1110
1111                if(start == 0) //Keine Teilung des Containers notwendig
1112                {
1113                        //LÃ€nge des Containers bestimmen
1114                        lengthContainer = 2 /*TransactionID*/ + 4 /*Code*/
                                + 2 /*AnzahlParameter*/ + countParam;
1115
1116                        for (i=0;i<lengthContainer;i++)
1117                        {
1118                                switch(i)
1119                                {
1120                                        case 0:
1121                                        case 1:
1122                                                //TransaktionID
1123                                                Container[i] = (byte) (
                                                        TransID >> i*8) ;
1124                                                break;
1125                                        case 2:
1126                                        case 3:
1127                                        case 4:
1128                                        case 5:
1129                                                //Code
```

```
1130                                                    Container[i] = (byte) (
                                                            CodeWort>> (i-2)*8);
1131                                                    break;
1132                                            case 6:
1133                                            case 7:
1134                                                    //Anzahl Parameter
1135                                                    Container[i] = (byte) (
                                                            countParam>> (i-6)*8);
1136                                                    break;
1137                                            default:
1138                                                    //Parameter
1139                                                    Container[i] = arrayParam[i
                                                            -8];
1140                                                    break;
1141                                    }
1142                            }
1143                    }
1144                    else //Nur die Parameter uebertragen
1145                    {
1146                            //LÃ€nge des Containers bestimmen
1147                            lengthContainer = stop - start + 1;
1148                            for (i=0;i<stop-start +1 ;i++)
1149                            {
1150                                    switch(i)
1151                                    {
1152                                            default:
1153                                                    //Parameter
1154                                                    Container[i] = arrayParam[
                                                            start+i];
1155                                                    break;
1156                                    }
1157                            }
1158                    }
1159
1160                    return  lengthContainer;
1161            }
1162
1163            //SendContainer
1164            //Funktion sendet den Container
1165            //AbhÃ€ngig von der Transportebene
1166            //Container wird hier definiert
1167            // @Param1: Zu uebertragender Code
1168            // @Param2: Anzahl der Parameter
1169            // @Param3: Parameter Array
1170            // @Param4: zusendente TransaktionsID
1171            // @Return Value: 0 success, sonst <>  0
1172            public char SendContainer(int  CodeWort, char countParam, byte []
                    arrayParam, int TransID)
1173            {
1174                    char ParameterLeft;
1175                    int i;
1176                    char countTransmitParameter;
1177
1178                    byte [] buffer = new byte[128];
1179                    char start;
1180                    char stop;
1181                    int lengthContainer;
1182
1183                    start = (char) 0;
```

```
1184            stop = (char)0;
1185            i = (char)0;
1186
1187                for(i=0;i<128;i++)
1188                {
1189                        buffer[i] = 0;
1190                }
1191
1192                //maximale Anzahl der Parameter die uebertragen sind
1193                //64-8 = 56
1194                //Wenn mehr sind, mÃŒssen mehrere Container gesendet werden
1195
1196                ParameterLeft = countParam;
1197
1198                if(ParameterLeft>56)
1199                {
1200                        countTransmitParameter = (char) 56;
1201                }
1202                else
1203                {
1204                        countTransmitParameter = ParameterLeft;
1205                }
1206
1207                do
1208                {
1209                        //Container generieren
1210                        lengthContainer = CreateContainer( CodeWort,
                                countTransmitParameter,arrayParam,start,stop,
                                buffer,TransID);
1211
1212                        //CreateContainer
1213                        //CreateContainer(CodeWort,countTransmitParameter,
                                arrayParam,start,stop,buffer,TransID);
1214
1215                        //AbhÃ€ngig von der Transportebene Start
1216                        //-------------------------------------------------------------------------

1217
1218                        //UDP
1219                try
1220                {
1221                    server.Send(buffer, lengthContainer, textBox_ZielIP.
                            Text.ToString(), Port);
1222
1223                }
1224                catch (Exception ex)
1225                {
1226                    MessageBox.Show(ex.ToString());
1227                }
1228
1229                        //-------------------------------------------------------------------------

1230                        //AbhÃ€ngig von der Transportebene Ende
1231
1232                        if(start == 0)
1233                        {
1234                                //Erster Durchlauf
1235                                if(ParameterLeft>56)
1236                                {
```

```
1237                                        start = (char)56;
1238                                        ParameterLeft-= (char) 56;
1239                                }
1240                                else
1241                                {
1242                            ParameterLeft = (char)0;
1243                                }
1244                            }
1245                            else
1246                            {
1247                                //Xter Durchlauf
1248                        if (ParameterLeft > (char)64)
1249                                {
1250                            start = (char)(start + (char)64);
1251                            ParameterLeft -= (char)64;
1252                                }
1253                                else
1254                                {
1255                            start = (char)(start + (char)64);
1256                            ParameterLeft = (char)0;
1257                                }
1258                            }
1259
1260                            //Stop setzen
1261                            if(ParameterLeft>64)
1262                            {
1263                                stop = (char) (start+64);
1264                        countTransmitParameter = (char)64;
1265                            }
1266                            else
1267                            {
1268                                stop = (char) (start + ParameterLeft);
1269                                countTransmitParameter = ParameterLeft;
1270                            }
1271                    }while(ParameterLeft>0);
1272
1273
1274            return (char) 0;
1275        }
1276
1277        private void Window_Closing(object sender, System.ComponentModel.
                CancelEventArgs e)
1278        {
1279            if (serverThread != null)
1280            {
1281                serverThread.Abort();
1282            }
1283
1284            if(server!=null)
1285            {
1286                server.Close();
1287            }
1288        }
1289
1290        public delegate void UDPReceiveStateMachine(byte[] message);
1291
1292        //Funktion kontrolliert die empfangen UDP Pakete und vergleicht das
                Ergebnis mit dem versendeten
1293        void StateMachine(byte[] data)
```

```
1294        {
1295            UInt32 code = (UInt32)data[2] + ((UInt32)data[3] << 8) + ((
                    UInt32)data[4] << 16) + ((UInt32)data[5] << 24);
1296            UInt32 parametercount = (UInt32)data.Length - 8;
1297            DateTime time = DateTime.Now;
1298
1299            switch (OpenCode)
1300            {
1301                case Commands.COMMAND_READ_TEMP:
1302                    switch (OpenState)
1303                    {
1304                        case 1:
1305                            //Wenn keine Datenphase zurÃŒckkommt, Fehler
                                ausgeben
1306                            if ((code & 0xFF000000) != 0x00000000)
1307                            {
1308                                listBoxInformation.Items.Add(time.ToString
                                    () + "␣" + "Error:␣Aspected␣Dataphase");
1309                                OpenState = 0;
1310                                OpenCode = 0;
1311                                break;
1312                            }
1313
1314                            //Erwartung: Datenphase mit 1 Bytes
1315                            if (parametercount != 2)
1316                            {
1317                                listBoxInformation.Items.Add(time.ToString
                                    () + "␣" + "Error:␣Wrong␣Parameter␣Count
                                    ␣returned!");
1318                                OpenState = 0;
1319                                OpenCode = 0;
1320                                break;
1321                            }
1322                            else
1323                            {
1324                                //Daten auslesen
1325                                if (data[8] == 0x80)
1326                                    textBoxTemp.Text = Convert.ToString(
                                        data[9]) + ",5␣Â°C";
1327                                else
1328                                    textBoxTemp.Text = Convert.ToString(
                                        data[9]) + ",0␣Â°C";
1329
1330                             }
1331
1332                            //Daten sind ok
1333                            OpenState = 2;
1334                            break;
1335                        case 2:
1336                            //Response Phase
1337                            //Wenn keine Response zurÃŒckkommt, Fehler
                                ausgeben
1338                            if ((code & 0xFF000000) != 0xFF000000)
1339                            {
1340                                listBoxInformation.Items.Add(time.ToString
                                    () + "␣" + "Error:␣Aspected␣
                                    Responsephase");
1341                                OpenState = 0;
1342                                OpenCode = 0;
```

```
1343                                    break;
1344                                }
1345
1346                                //Ueberpruefen der Antwort
1347                                if (CheckAnswer(code) == 0)
1348                                {
1349                                    listBoxInformation.Items.Add(time.ToString
                                        () + "␣" + "COMMAND_READ_TEMP␣finish!");
1350                                }
1351
1352                                OpenState = 0;
1353                                OpenCode = 0;
1354                                break;
1355                            }
1356                            break;
1357
1358                        case Commands.COMMAND_PIN_STORE:
1359                            switch (OpenState)
1360                            {
1361                                case 1:
1362                                //Response Phase
1363                                //Wenn keine Response zurÃŒckkommt, Fehler
                                    ausgeben
1364                                if ((code & 0xFF000000) != 0xFF000000)
1365                                {
1366                                    listBoxInformation.Items.Add(time.ToString
                                        () + "␣" + "Error:␣Aspected␣
                                        Responsephase");
1367                                    OpenState = 0;
1368                                    OpenCode = 0;
1369                                    break;
1370                                }
1371
1372                                //Ueberpruefen der Antwort
1373                                if (CheckAnswer(code) == 0)
1374                                {
1375                                    listBoxInformation.Items.Add(time.ToString
                                        () + "␣" + "COMMAND_PIN_STORE␣finish!");
1376                                    //Daten setzen
1377                                }
1378
1379                                OpenState = 0;
1380                                OpenCode = 0;
1381                                break;
1382                            }
1383                            break;
1384                        case Commands.COMMAND_PIN_SET:
1385                            switch (OpenState)
1386                            {
1387                                case 1:
1388                                //Response Phase
1389                                //Wenn keine Response zurÃŒckkommt, Fehler ausgeben
1390                                if ((code & 0xFF000000) != 0xFF000000)
1391                                {
1392                                    listBoxInformation.Items.Add(time.ToString() +
                                        "␣" + "Error:␣Aspected␣Responsephase");
1393                                    OpenState = 0;
1394                                    OpenCode = 0;
1395                                    break;
```

```
1396                        }
1397
1398                        //Ueberpruefen der Antwort
1399                        if (CheckAnswer(code) == 0)
1400                        {
1401                            listBoxInformation.Items.Add(time.ToString() +
                                    " " + "COMMAND_PIN_SET_finish!");
1402                            //Daten setzen
1403                        }
1404
1405                        OpenState = 0;
1406                        OpenCode = 0;
1407                        break;
1408                    }
1409                    break;
1410
1411                case Commands.COMMAND_PIN_READ:
1412                    switch (OpenState)
1413                    {
1414                        case 1:
1415                            //Wenn keine Datenphase zurÃŒckkommt, Fehler
                                ausgeben
1416                            if ((code & 0xFF000000) != 0x00000000)
1417                            {
1418                                listBoxInformation.Items.Add(time.ToString
                                        () + " " + "Error: Aspected Dataphase");
1419                                OpenState = 0;
1420                                OpenCode = 0;
1421                                break;
1422                            }
1423
1424                            //Erwartung: Datenphase mit 1 Bytes
1425                            if (parametercount != 9)
1426                            {
1427                                listBoxInformation.Items.Add(time.ToString
                                        () + " " + "Error: Wrong Parameter Count
                                        returned!");
1428                                OpenState = 0;
1429                                OpenCode = 0;
1430                                break;
1431                            }
1432                            else
1433                            {
1434                                //Daten auslesen
1435                                Pin_A PinA;
1436                                PinDetails PinFelder;
1437                                TabItem CurrentItem = (TabItem)TabPin.
                                        SelectedItem;
1438
1439                                PinFelder.PinOutput = new CheckBox[8];
1440                                PinFelder.PinAffected = new CheckBox[8];
1441                                PinFelder.PinSetting = new CheckBox[8];
1442                                PinFelder.textBoxTimeA = null;
1443                                PinFelder.textBoxTimeB = null;
1444                                PinFelder.PinBState = null;
1445                                PinFelder.PinAState = null;
1446
1447                                PinA.Settings = data[8];
```

```
1448                            PinA.Reaction = (UInt32)data[9] + ((UInt32)
                                    data[10] << 8) + ((UInt32)data[11] <<
                                    16) + ((UInt32)data[12] << 24);
1449                            PinA.Time = (UInt32)data[13] + ((UInt32)
                                    data[14] << 8) + ((UInt32)data[15] <<
                                    16) + ((UInt32)data[16] << 24);
1450
1451                            FillPinFelder(ref PinFelder, Convert.
                                    ToInt32(CurrentItem.Header.ToString().
                                    Replace("Pin␣","")));
1452                            FillsPinA(PinA, PinFelder);
1453
1454                        }
1455
1456                        //Daten sind ok
1457                        OpenState = 2;
1458                        break;
1459                    case 2:
1460                        //Response Phase
1461                        //Wenn keine Response zurÃŒckkommt, Fehler
                                ausgeben
1462                        if ((code & 0xFF000000) != 0xFF000000)
1463                        {
1464                            listBoxInformation.Items.Add(time.ToString
                                    () + "␣" + "Error:␣Aspected␣
                                    Responsephase");
1465                            OpenState = 0;
1466                            OpenCode = 0;
1467                            break;
1468                        }
1469
1470                        //Ueberpruefen der Antwort
1471                        if (CheckAnswer(code) == 0)
1472                        {
1473                            listBoxInformation.Items.Add(time.ToString
                                    () + "␣" + "COMMAND_PIN_READ␣finish!");
1474                            //Daten setzen
1475                        }
1476
1477                        OpenState = 0;
1478                        OpenCode = 0;
1479                        break;
1480                }
1481                break;
1482
1483            case Commands.COMMAND_GETCLASS:
1484
1485                //Status
1486                switch (OpenState)
1487                {
1488                    case 1:
1489                        //Wenn keine Datenphase zurÃŒckkommt, Fehler
                                ausgeben
1490                        if ((code & 0xFF000000) != 0x00000000)
1491                        {
1492                            listBoxInformation.Items.Add(time.ToString
                                    () + "␣" + "Error:␣Aspected␣Dataphase");
1493                            OpenState = 0;
1494                            OpenCode = 0;
```

```
1495                            break;
1496                        }
1497
1498                        //Erwartung: Datenphase mit 1 Bytes
1499                        if (parametercount != 1)
1500                        {
1501                            listBoxInformation.Items.Add(time.ToString
                                    () + " " + "Error: Wrong Parameter Count
                                    returned!");
1502                            OpenState = 0;
1503                            OpenCode = 0;
1504                            break;
1505                        }
1506                        else
1507                        {
1508                            String StringConnection;
1509                            StringConnection = "";
1510
1511                            switch (data[8])
1512                            {
1513                                case (byte)0:
1514                                    StringConnection = "4 E";
1515                                    break;
1516                                case (byte)1:
1517                                    StringConnection = "4 B";
1518                                    break;
1519                                case (byte)2:
1520                                    StringConnection = "5 E";
1521                                    break;
1522                                case (byte)3:
1523                                    StringConnection = "5 B";
1524                                    break;
1525                                default:
1526                                    StringConnection = "unkown";
1527                                    break;
1528                            }
1529
1530                            textBoxGetClass.Text = StringConnection;
1531                        }
1532
1533                        //Daten sind ok
1534                        OpenState = 2;
1535                        break;
1536                    case 2:
1537                        //Response Phase
1538                        //Wenn keine Response zurÃŒckkommt, Fehler
                            ausgeben
1539                        if ((code & 0xFF000000) != 0xFF000000)
1540                        {
1541                            listBoxInformation.Items.Add(time.ToString
                                    () + " " + "Error: Aspected
                                    Responsephase");
1542                            OpenState = 0;
1543                            OpenCode = 0;
1544                            break;
1545                        }
1546
1547                        //Ueberpruefen der Antwort
1548                        if (CheckAnswer(code) == 0)
```

```
1549                                {
1550                                    listBoxInformation.Items.Add(time.ToString
                                            () + "_" + "COMMAND_GETCLASS_finish!");
1551                                    //Daten setzen
1552                                }
1553
1554                                OpenState = 0;
1555                                OpenCode = 0;
1556                                break;
1557                            }
1558                            break;
1559                        }
1560                    }
1561
1562
1563
1564                    //Funktion liefert eine PinA Struktur zurÃŒck
1565                    private Pin_A GetPinDetails(byte number)
1566                    {
1567                        Pin_A PinA;
1568                        PinDetails PinFelder;
1569                        PinFelder.PinOutput = new CheckBox[8];
1570                        PinFelder.PinAffected = new CheckBox[8];
1571                        PinFelder.PinSetting = new CheckBox[8];
1572                        PinFelder.textBoxTimeA = null;
1573                        PinFelder.textBoxTimeB = null;
1574                        PinFelder.PinBState = null;
1575                        PinFelder.PinAState = null;
1576
1577                        FillPinFelder(ref PinFelder, number);
1578
1579                        //Settings fÃŒllen
1580                        PinA.Settings=0;
1581
1582                        if(PinFelder.PinSetting[0].IsChecked == true)
1583                        {
1584                            PinA.Settings |= 0x01;
1585                        }
1586
1587                        if (PinFelder.PinSetting[1].IsChecked == true)
1588                        {
1589                            PinA.Settings |= 0x02;
1590                        }
1591
1592                        if (PinFelder.PinSetting[2].IsChecked == true)
1593                        {
1594                            PinA.Settings |= 0x04;
1595                        }
1596
1597                        if (PinFelder.PinSetting[3].IsChecked == true)
1598                        {
1599                            PinA.Settings |= 0x08;
1600                        }
1601
1602                        if (PinFelder.PinSetting[4].IsChecked == true)
1603                        {
1604                            PinA.Settings |= 0x10;
1605                        }
1606
```

```
1607            if (PinFelder.PinSetting[5].IsChecked == true)
1608            {
1609                PinA.Settings |= 0x20;
1610            }
1611
1612            if (PinFelder.PinSetting[6].IsChecked == true)
1613            {
1614                PinA.Settings |= 0x40;
1615            }
1616
1617            if (PinFelder.PinSetting[7].IsChecked == true)
1618            {
1619                PinA.Settings |= 0x80;
1620            }
1621
1622            //Affected fÃŒllen
1623            UInt32 output = 0;
1624
1625            if (PinFelder.PinAffected[0].IsChecked == true)
1626            {
1627                output |= 0x01;
1628            }
1629
1630            if (PinFelder.PinAffected[1].IsChecked == true)
1631            {
1632                output |= 0x02;
1633            }
1634
1635            if (PinFelder.PinAffected[2].IsChecked == true)
1636            {
1637                output |= 0x04;
1638            }
1639
1640            if (PinFelder.PinAffected[3].IsChecked == true)
1641            {
1642                output |= 0x08;
1643            }
1644
1645            if (PinFelder.PinAffected[4].IsChecked == true)
1646            {
1647                output |= 0x10;
1648            }
1649
1650            if (PinFelder.PinAffected[5].IsChecked == true)
1651            {
1652                output |= 0x20;
1653            }
1654
1655            if (PinFelder.PinAffected[6].IsChecked == true)
1656            {
1657                output |= 0x40;
1658            }
1659
1660            if (PinFelder.PinAffected[7].IsChecked == true)
1661            {
1662                output |= 0x80;
1663            }
1664
1665            output = output << 16;
```

```
1666
1667            //Output setzen
1668            if (PinFelder.PinOutput[0].IsChecked == true)
1669            {
1670                output |= 0x01;
1671            }
1672
1673            if (PinFelder.PinOutput[1].IsChecked == true)
1674            {
1675                output |= 0x02;
1676            }
1677
1678            if (PinFelder.PinOutput[2].IsChecked == true)
1679            {
1680                output |= 0x04;
1681            }
1682
1683            if (PinFelder.PinOutput[3].IsChecked == true)
1684            {
1685                output |= 0x08;
1686            }
1687
1688            if (PinFelder.PinOutput[4].IsChecked == true)
1689            {
1690                output |= 0x10;
1691            }
1692
1693            if (PinFelder.PinOutput[5].IsChecked == true)
1694            {
1695                output |= 0x20;
1696            }
1697
1698            if (PinFelder.PinOutput[6].IsChecked == true)
1699            {
1700                output |= 0x40;
1701            }
1702
1703            if (PinFelder.PinOutput[7].IsChecked == true)
1704            {
1705                output |= 0x80;
1706            }
1707
1708            PinA.Reaction = output;
1709
1710            //Time fuellen
1711
1712            PinA.Time = 0;
1713
1714            //Time A fuellen
1715            if (PinFelder.PinAState.IsChecked == true)
1716            {
1717                PinA.Time |= 0x80000000;
1718            }
1719
1720            PinA.Time |= (Convert.ToUInt32(PinFelder.textBoxTimeA.Text)
                <<16);
1721
1722            //Time B fuellen
1723            if (PinFelder.PinBState.IsChecked == true)
```

```
1724                {
1725                    PinA.Time |= 0x00008000;
1726                }
1727
1728                PinA.Time |= Convert.ToUInt32(PinFelder.textBoxTimeB.Text);
1729
1730
1731                return PinA;
1732            }
1733
1734        }
1735
1736
1737    }
```

Commands.cs

```
1   ï»¿using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5
6   namespace WPFNetzwerk
7   {
8       class Commands {
9
10              //Codes fÃŒr das Protokoll
11              //RÃŒckgabe der Funktion Check Command
12              //Klasse V
13          public const UInt16 COMMAND_PTP_SESSION_OPEN = 0x0001;
14          public const UInt16 COMMAND_PTP_SESSION_CLOSE = 0x0002;
15          public const UInt16 COMMAND_PTP_SHOOT = 0x0003;
16          public const UInt16 COMMAND_PTP_CHANGE_TV = 0x0004;
17          public const UInt16 COMMAND_PTP_CHANGE_AV = 0x0005;
18          public const UInt16 COMMAND_PTP_CHANGE_BELICHTUNG = 0x0006;
19          public const UInt16 COMMAND_PTP_LIVE_VIEW_ON = 0x0007;
20          public const UInt16 COMMAND_PTP_FOKUS = 0x0008;
21          public const UInt16 COMMAND_PTP_GETCONNECTION = 0x0009;
22          public const UInt16 COMMAND_PTP_LIVE_VIEW_OFF = 0x000A;
23
24          //Klasse I
25          public const UInt16 COMMAND_PIN_READ = 0x0001;
26          public const UInt16 COMMAND_PIN_SET = 0x0002;
27          public const UInt16 COMMAND_PIN_STORE = 0x0003;
28          public const UInt16 COMMAND_PIN_STATE = 0x0004;
29          public const UInt16 COMMAND_GETCLASS = 0x0005;
30          public const UInt16 COMMAND_SETOUTPUT = 0x0006;
31
32          //Klasse II
33          public const UInt16 COMMAND_READ_TEMP = 0x0007;
34
35          //RÃŒckgabe der Funktion Check Command
36          public const UInt16 RESPONSE_OK = 0x0000;
37          public const UInt16 RESPONSE_WRONG_PARAMETER = 0x0001;
38          public const UInt16 RESPONSE_NOT_SUPPORTED_CLASS = 0x0002;
39          public const UInt16 RESPONSE_NOT_SUPPORTED_ADCLASS = 0x0003;
40          public const UInt16 RESPONSE_NOT_SUPPORTED_PARAMETERCOUNT = 0x0004
                ;
41          public const UInt16 RESPONSE_NOT_SUPPORTED = 0x0005;
```

```
42          public const UInt16 RESPONSE_NOT_SUPPORTED_PHASE = 0x0006;
43          public const UInt16 RESPONSE_ERROR = 0x0007;
44          public const UInt16 RESPONSE_PTP_ERROR = 0x0008;
45          public const UInt16 RESPONSE_DATA = 0x0009;
46
47          //Klassen interne Definitionen
48          public const UInt16 INTERNAL_COMMAND_TIMELAPS = 0x1000;
49          public const UInt16 INTERNAL_COMMAND_3D = 0x1001;
50          public const UInt16 INTERNAL_COMMAND_3DHDR = 0x1002;
51
52      }
53
54  }
```

L Bilder3D.exe

Window1.xaml

```
1  ï»¿<Window x:Class="Wpfnochmal3D.Window1"
2      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4      Title="3D Modell Aufnahme" Height="504" Width="798">
5      <Grid>
6          <DockPanel
7    Width="Auto"
8    VerticalAlignment="Stretch"
9    Height="Auto"
10   HorizontalAlignment="Stretch"
11   Grid.ColumnSpan="1"
12   Grid.Column="0"
13   Grid.Row="0"
14   Margin="0,0,0,0"
15   Grid.RowSpan="1">
16               <StackPanel>
17                   <StackPanel.Background>
18                       <LinearGradientBrush>
19                           <GradientStop Color="White" Offset="0"/>
20                           <GradientStop Color="DarkKhaki" Offset=".3"/>
21                           <GradientStop Color="DarkKhaki" Offset=".7"/>
22                           <GradientStop Color="White" Offset="1"/>
23                       </LinearGradientBrush>
24                   </StackPanel.Background>
25                   <StackPanel Margin="10">
26                       <TextBlock Text="Camera X Position:"/>
27 <TextBox Name="cameraPositionXTextBox" MaxLength="5"
28     HorizontalAlignment="Left" Text="9"/><Grid Height="25" Name="grid1"
           Width="104"><Button Name="LX_Modell" Click="LX_ModellButtonClick"
           Width="32" Height="21" HorizontalAlignment="Left" VerticalAlignment="
           Top">-</Button><Button Name="RX_Modell" Click="RX_ModellButtonClick"
           Margin="33,0,0,0" Height="21" VerticalAlignment="Top"
           HorizontalAlignment="Left" Width="40">+</Button></Grid>
29 <TextBlock Text="Camera Y Position:"/>
30 <TextBox Name="cameraPositionYTextBox" MaxLength="5"
31     HorizontalAlignment="Left" Text="8"/><Grid Height="25" Name="grid2"
           Width="104"><Button Name="LY_Modell" Click="LY_ModellButtonClick"
           Width="32" Height="21" HorizontalAlignment="Left" VerticalAlignment="
           Top">-</Button><Button Name="RY_Modell" Click="RY_ModellButtonClick"
           Margin="33,0,0,0" Height="21" VerticalAlignment="Top"
           HorizontalAlignment="Left" Width="40">+</Button></Grid>
32 <TextBlock Text="Camera Z Position:"/>
33 <TextBox Name="cameraPositionZTextBox" MaxLength="5"
34     HorizontalAlignment="Left" Text="10"/><Grid Height="25" Name="grid3"
           Width="104"><Button Name="LZ_Modell" Click="LZ_ModellButtonClick"
           Width="32" Height="21" HorizontalAlignment="Left" VerticalAlignment="
           Top">-</Button><Button Name="RZ_Modell" Click="RZ_ModellButtonClick"
           Margin="33,0,0,0" Height="21" VerticalAlignment="Top"
           HorizontalAlignment="Left" Width="40">+</Button></Grid>
35 <Separator/>
36 <TextBlock Text="Look Direction X:"/>
37 <TextBox Name="lookAtXTextBox" MaxLength="5"
38     HorizontalAlignment="Left" Text="-9"/><Grid Height="25" Name="grid4"
           Width="104"><Button Name="LX_Look" Click="Lx_LookButtonClick" Width="
           32" Height="21" HorizontalAlignment="Left" VerticalAlignment="Top">-<
```

```
        /Button><Button Name="RX_Look" Click="RX_LookButtonClick" Margin="
        33,0,0,0" Height="21" VerticalAlignment="Top" HorizontalAlignment="
        Left" Width="40">+</Button></Grid>
39 <TextBlock Text="Look␣Direction␣Y:"/>
40 <TextBox Name="lookAtYTextBox" MaxLength="5"
41     HorizontalAlignment="Left" Text="-8"/><Grid Height="25" Name="grid5"
        Width="104"><Button Name="LY_Look" Click="LY_LookButtonClick" Width="
        32" Height="21" HorizontalAlignment="Left" VerticalAlignment="Top">-<
        /Button><Button Name="RY_Look" Click="RY_LookButtonClick" Margin="
        33,0,0,0" Height="21" VerticalAlignment="Top" HorizontalAlignment="
        Left" Width="40">+</Button></Grid>
42 <TextBlock Text="Look␣Direction␣Z:"/>
43 <TextBox Name="lookAtZTextBox" MaxLength="5"
44     HorizontalAlignment="Left" Text="-10"/><Grid Height="25" Name="grid6"
        Width="104"><Button Name="LZ_Look" Click="LZ_LookButtonClick" Width="
        32" Height="21" HorizontalAlignment="Left" VerticalAlignment="Top">-<
        /Button><Button Name="RZ_Look" Click="RZ_LookButtonClick" Margin="
        33,0,0,0" Height="21" VerticalAlignment="Top" HorizontalAlignment="
        Left" Width="40">+</Button></Grid>
45 <Separator/>
46                     <Button
47       Name="simpleButton"
48       Click="simpleButtonClick">Kamera wechsel</Button>
49                     <Button Name="cubeButton" Click="cubeButtonClick">
                           Modell Ã¶ffnen</Button>
50 <Separator/>
51                 </StackPanel>
52             </StackPanel>
53             <Viewport3D Name="mainViewport" ClipToBounds="True">
54                 <Viewport3D.Camera>
55                     <PerspectiveCamera
56             LookDirection="0,0,-1"
57             UpDirection="0,1,0"
58             NearPlaneDistance="1"
59             Position="90,150,80"
60             FieldOfView="45" />
61                 </Viewport3D.Camera>
62                 <ModelVisual3D>
63                     <ModelVisual3D.Content>
64                         <AmbientLight Color="#FFFFFF" />
65                     </ModelVisual3D.Content>
66                 </ModelVisual3D>
67             </Viewport3D>
68         </DockPanel>
69     </Grid>
70 </Window>
```

Window1.xaml.cs

```
1  ï»¿using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Windows;
6  using System.Windows.Controls;
7  using System.Windows.Data;
8  using System.Windows.Documents;
9  using System.Windows.Input;
10 using System.Windows.Media;
```

```
11 using System.Windows.Media.Imaging;
12 using System.Windows.Navigation;
13 using System.Windows.Shapes;
14 using System.Windows.Media.Media3D;
15 using System.IO;
16 using System.Collections;
17
18
19
20 namespace Wpfnochmal3D
21 {
22     /// <summary>
23     /// Interaction logic for Window1.xaml
24     /// </summary>
25     public partial class Window1 : Window
26     {
27         public Window1()
28         {
29             InitializeComponent();
30         }
31
32         private void simpleButtonClick(object sender, RoutedEventArgs e)
33         {
34             SetCamera();
35         }
36
37         private void cubeButtonClick(object sender, RoutedEventArgs e)
38         {
39             string filename;
40             // Configure open file dialog box
41             Microsoft.Win32.OpenFileDialog dlg = new Microsoft.Win32.
                   OpenFileDialog();
42             dlg.FileName = "Document"; // Default file name
43             dlg.DefaultExt = ".txt"; // Default file extension
44             dlg.Filter = "Text documents (.txt)|*.txt"; // Filter files by
                   extension
45
46             // Show open file dialog box
47             Nullable<bool> result = dlg.ShowDialog();
48
49             // Process open file dialog box results
50             if (result == true)
51             {
52                 // Open document
53                 filename = dlg.FileName;
54             }
55             else
56             {
57                 return;
58             }
59
60             ClearViewport();
61
62             //Datei einlesen
63             StreamReader objReader = new StreamReader(filename);
64
65             string sLine = "";
66             ArrayList arrText = new ArrayList();
67
```

```
68              while (sLine != null)
69              {
70                  sLine = objReader.ReadLine();
71                  if (sLine != null)
72                      arrText.Add(sLine);
73              }
74              objReader.Close();
75
76              int row = Convert.ToInt32(arrText[1]);
77              int column = Convert.ToInt32(arrText[0]);
78
79
80              Model3DGroup cube = new Model3DGroup();
81              Point3D p0;
82              Point3D p1;
83              Point3D p2;
84
85              //Eine Art von Dreiecken erzeugen
86              for (int x = 0; x < row - 1; x++)
87              {
88                  for (int y = 0; y < column - 1; y++)
89                  {
90                      string sp0 = (string)arrText[2 + x * column + y];
91                      string xyz = sp0.Substring(0, sp0.IndexOf(';'));
92                      string rgb = sp0.Substring(sp0.IndexOf(';') + 1, sp0.
                            Length - (sp0.IndexOf(';')) - 1);
93
94                      int x_pos0 = Convert.ToInt32(xyz.Substring(0, xyz.
                            IndexOf(',')));
95                      int y_pos0 = Convert.ToInt32(xyz.Substring(xyz.IndexOf
                            (',') + 1, xyz.LastIndexOf(',') - xyz.IndexOf(',') -
                             1));
96                      int z_pos0 = Convert.ToInt32(xyz.Substring(xyz.
                            LastIndexOf(',') + 1, xyz.Length - xyz.LastIndexOf
                            (',') - 1));
97
98                      int r0 = Convert.ToInt32(rgb.Substring(0, rgb.IndexOf
                            (',')));
99                      int g0 = Convert.ToInt32(rgb.Substring(rgb.IndexOf(',')
                             + 1, rgb.LastIndexOf(',') - rgb.IndexOf(',') - 1));
100                     int b0 = Convert.ToInt32(rgb.Substring(rgb.LastIndexOf
                            (',') + 1, rgb.Length - rgb.LastIndexOf(',') - 1));
101
102                     p0 = new Point3D(x_pos0, y_pos0, z_pos0);
103
104                     string sp1 = (string)arrText[2 + x * column + y + 1];
105                     xyz = sp1.Substring(0, sp1.IndexOf(';'));
106                     rgb = sp1.Substring(sp1.IndexOf(';') + 1, sp1.Length -
                            (sp1.IndexOf(';')) - 1);
107
108                     int x_pos1 = Convert.ToInt32(xyz.Substring(0, xyz.
                            IndexOf(',')));
109                     int y_pos1 = Convert.ToInt32(xyz.Substring(xyz.IndexOf
                            (',') + 1, xyz.LastIndexOf(',') - xyz.IndexOf(',') -
                             1));
110                     int z_pos1 = Convert.ToInt32(xyz.Substring(xyz.
                            LastIndexOf(',') + 1, xyz.Length - xyz.LastIndexOf
                            (',') - 1));
111
```

```
112                     int r1 = Convert.ToInt32(rgb.Substring(0, rgb.IndexOf
                            (',')));
113                     int g1 = Convert.ToInt32(rgb.Substring(rgb.IndexOf(',')
                             + 1, rgb.LastIndexOf(',') - rgb.IndexOf(',') - 1));
114                     int b1 = Convert.ToInt32(rgb.Substring(rgb.LastIndexOf
                            (',') + 1, rgb.Length - rgb.LastIndexOf(',') - 1));
115
116                     p1 = new Point3D(x_pos1, y_pos1, z_pos1);
117
118                     string sp2 = (string)arrText[2 + x * column + y + 1 +
                            column];
119                     xyz = sp2.Substring(0, sp2.IndexOf(';'));
120                     rgb = sp2.Substring(sp2.IndexOf(';') + 1, sp2.Length -
                            (sp2.IndexOf(';')) - 1);
121
122                     int x_pos2 = Convert.ToInt32(xyz.Substring(0, xyz.
                            IndexOf(',')));
123                     int y_pos2 = Convert.ToInt32(xyz.Substring(xyz.IndexOf
                            (',') + 1, xyz.LastIndexOf(',') - xyz.IndexOf(',') -
                             1));
124                     int z_pos2 = Convert.ToInt32(xyz.Substring(xyz.
                            LastIndexOf(',') + 1, xyz.Length - xyz.LastIndexOf
                            (',') - 1));
125
126                     int r2 = Convert.ToInt32(rgb.Substring(0, rgb.IndexOf
                            (',')));
127                     int g2 = Convert.ToInt32(rgb.Substring(rgb.IndexOf(',')
                             + 1, rgb.LastIndexOf(',') - rgb.IndexOf(',') - 1));
128                     int b2 = Convert.ToInt32(rgb.Substring(rgb.LastIndexOf
                            (',') + 1, rgb.Length - rgb.LastIndexOf(',') - 1));
129
130                     p2 = new Point3D(x_pos2, y_pos2, z_pos2);
131
132                     cube.Children.Add(CreateTriangleModel(p0, p1, p2, Color
                            .FromRgb((byte)((r0 + r1 + r2) / 3), (byte)((g0 + g1
                             + g2) / 3), (byte)((b0 + b1 + b2) / 3))));
133                 }
134             }
135
136             //Die andere Art von Dreiecken erzeugen
137             for (int x = 1; x < row; x++)
138             {
139                 for (int y = 0; y < column - 1; y++)
140                 {
141                     string sp0 = (string)arrText[2 + x * column + y];
142                     string xyz = sp0.Substring(0, sp0.IndexOf(';'));
143                     string rgb = sp0.Substring(sp0.IndexOf(';') + 1, sp0.
                            Length - (sp0.IndexOf(';')) - 1);
144
145                     int x_pos0 = Convert.ToInt32(xyz.Substring(0, xyz.
                            IndexOf(',')));
146                     int y_pos0 = Convert.ToInt32(xyz.Substring(xyz.IndexOf
                            (',') + 1, xyz.LastIndexOf(',') - xyz.IndexOf(',') -
                             1));
147                     int z_pos0 = Convert.ToInt32(xyz.Substring(xyz.
                            LastIndexOf(',') + 1, xyz.Length - xyz.LastIndexOf
                            (',') - 1));
148
```

```
149                     int r0 = Convert.ToInt32(rgb.Substring(0, rgb.IndexOf
                            (',')));
150                     int g0 = Convert.ToInt32(rgb.Substring(rgb.IndexOf(',')
                             + 1, rgb.LastIndexOf(',') - rgb.IndexOf(',') - 1));
151                     int b0 = Convert.ToInt32(rgb.Substring(rgb.LastIndexOf
                            (',') + 1, rgb.Length - rgb.LastIndexOf(',') - 1));
152
153                     p0 = new Point3D(x_pos0, y_pos0, z_pos0);
154
155                     string sp1 = (string)arrText[2 + x * column + y + 1];
156                     xyz = sp1.Substring(0, sp1.IndexOf(';'));
157                     rgb = sp1.Substring(sp1.IndexOf(';') + 1, sp1.Length -
                            (sp1.IndexOf(';')) - 1);
158
159                     int x_pos1 = Convert.ToInt32(xyz.Substring(0, xyz.
                            IndexOf(',')));
160                     int y_pos1 = Convert.ToInt32(xyz.Substring(xyz.IndexOf
                            (',') + 1, xyz.LastIndexOf(',') - xyz.IndexOf(',') -
                             1));
161                     int z_pos1 = Convert.ToInt32(xyz.Substring(xyz.
                            LastIndexOf(',') + 1, xyz.Length - xyz.LastIndexOf
                            (',') - 1));
162
163                     int r1 = Convert.ToInt32(rgb.Substring(0, rgb.IndexOf
                            (',')));
164                     int g1 = Convert.ToInt32(rgb.Substring(rgb.IndexOf(',')
                             + 1, rgb.LastIndexOf(',') - rgb.IndexOf(',') - 1));
165                     int b1 = Convert.ToInt32(rgb.Substring(rgb.LastIndexOf
                            (',') + 1, rgb.Length - rgb.LastIndexOf(',') - 1));
166
167                     p1 = new Point3D(x_pos1, y_pos1, z_pos1);
168
169                     string sp2 = (string)arrText[2 + (x - 1) * column + y];
170                     xyz = sp2.Substring(0, sp2.IndexOf(';'));
171                     rgb = sp2.Substring(sp2.IndexOf(';') + 1, sp2.Length -
                            (sp2.IndexOf(';')) - 1);
172
173                     int x_pos2 = Convert.ToInt32(xyz.Substring(0, xyz.
                            IndexOf(',')));
174                     int y_pos2 = Convert.ToInt32(xyz.Substring(xyz.IndexOf
                            (',') + 1, xyz.LastIndexOf(',') - xyz.IndexOf(',') -
                             1));
175                     int z_pos2 = Convert.ToInt32(xyz.Substring(xyz.
                            LastIndexOf(',') + 1, xyz.Length - xyz.LastIndexOf
                            (',') - 1));
176
177                     int r2 = Convert.ToInt32(rgb.Substring(0, rgb.IndexOf
                            (',')));
178                     int g2 = Convert.ToInt32(rgb.Substring(rgb.IndexOf(',')
                             + 1, rgb.LastIndexOf(',') - rgb.IndexOf(',') - 1));
179                     int b2 = Convert.ToInt32(rgb.Substring(rgb.LastIndexOf
                            (',') + 1, rgb.Length - rgb.LastIndexOf(',') - 1));
180
181                     p2 = new Point3D(x_pos2, y_pos2, z_pos2);
182
183                     cube.Children.Add(CreateTriangleModel(p0, p2, p1, Color
                            .FromRgb((byte)((r0 + r1 + r2) / 3), (byte)((g0 + g1
                             + g2) / 3), (byte)((b0 + b1 + b2) / 3))));
184                 }
```

```
185             }
186
187             ModelVisual3D model = new ModelVisual3D();
188             model.Content = cube;
189             this.mainViewport.Children.Add(model);
190         }
191
192         private Model3DGroup CreateTriangleModel(Point3D p0, Point3D p1,
                Point3D p2, Color Farbe)
193         {
194             MeshGeometry3D mesh = new MeshGeometry3D();
195             mesh.Positions.Add(p0);
196             mesh.Positions.Add(p1);
197             mesh.Positions.Add(p2);
198             mesh.TriangleIndices.Add(0);
199             mesh.TriangleIndices.Add(1);
200             mesh.TriangleIndices.Add(2);
201             Vector3D normal = CalculateNormal(p0, p1, p2);
202             mesh.Normals.Add(normal);
203             mesh.Normals.Add(normal);
204             mesh.Normals.Add(normal);
205             Material material = new DiffuseMaterial(
206                 new SolidColorBrush(Farbe));
207             GeometryModel3D model = new GeometryModel3D(
208                 mesh, material);
209             Model3DGroup group = new Model3DGroup();
210             group.Children.Add(model);
211
212             return group;
213         }
214         private Vector3D CalculateNormal(Point3D p0, Point3D p1, Point3D p2
                )
215         {
216             Vector3D v0 = new Vector3D(
217                 p1.X - p0.X, p1.Y - p0.Y, p1.Z - p0.Z);
218             Vector3D v1 = new Vector3D(
219                 p2.X - p1.X, p2.Y - p1.Y, p2.Z - p1.Z);
220             return Vector3D.CrossProduct(v0, v1);
221         }
222
223
224         private void ClearViewport()
225         {
226             ModelVisual3D m;
227             for (int i = mainViewport.Children.Count - 1; i >= 0; i--)
228             {
229                 m = (ModelVisual3D)mainViewport.Children[i];
230                 if (m.Content is AmbientLight == false)
231                     mainViewport.Children.Remove(m);
232             }
233         }
234
235         private void SetCamera()
236         {
237             PerspectiveCamera camera = (PerspectiveCamera)mainViewport.
                    Camera;
238             Point3D position = new Point3D(
239                 Convert.ToDouble(cameraPositionXTextBox.Text),
240                 Convert.ToDouble(cameraPositionYTextBox.Text),
```

```
241                 Convert.ToDouble(cameraPositionZTextBox.Text)
242             );
243             Vector3D lookDirection = new Vector3D(
244                 Convert.ToDouble(lookAtXTextBox.Text),
245                 Convert.ToDouble(lookAtYTextBox.Text),
246                 Convert.ToDouble(lookAtZTextBox.Text)
247             );
248             camera.Position = position;
249             camera.LookDirection = lookDirection;
250 
251             //DirectionalLight myDirLight = new DirectionalLight();
252             //myDirLight.Color = Colors.White;
253             //myDirLight.Direction = lookDirection;
254 
255             //mainViewport.Children.Add(myDirLight);
256         }
257 
258         private void LX_ModellButtonClick(object sender, RoutedEventArgs e)
259         {
260             PerspectiveCamera camera = (PerspectiveCamera)mainViewport.
                    Camera;
261             Point3D position = new Point3D(
262                 Convert.ToDouble(cameraPositionXTextBox.Text)-1,
263                 Convert.ToDouble(cameraPositionYTextBox.Text),
264                 Convert.ToDouble(cameraPositionZTextBox.Text)
265             );
266             Vector3D lookDirection = new Vector3D(
267                 Convert.ToDouble(lookAtXTextBox.Text),
268                 Convert.ToDouble(lookAtYTextBox.Text),
269                 Convert.ToDouble(lookAtZTextBox.Text)
270             );
271             camera.Position = position;
272             camera.LookDirection = lookDirection;
273             cameraPositionXTextBox.Text = (Convert.ToDouble(
                    cameraPositionXTextBox.Text) - 1).ToString();
274         }
275 
276         private void RX_ModellButtonClick(object sender, RoutedEventArgs e)
277         {
278             PerspectiveCamera camera = (PerspectiveCamera)mainViewport.
                    Camera;
279             Point3D position = new Point3D(
280                 Convert.ToDouble(cameraPositionXTextBox.Text) + 1,
281                 Convert.ToDouble(cameraPositionYTextBox.Text),
282                 Convert.ToDouble(cameraPositionZTextBox.Text)
283             );
284             Vector3D lookDirection = new Vector3D(
285                 Convert.ToDouble(lookAtXTextBox.Text),
286                 Convert.ToDouble(lookAtYTextBox.Text),
287                 Convert.ToDouble(lookAtZTextBox.Text)
288             );
289             camera.Position = position;
290             camera.LookDirection = lookDirection;
291             cameraPositionXTextBox.Text = (Convert.ToDouble(
                    cameraPositionXTextBox.Text) + 1).ToString();
292 
293         }
294 
295         private void RY_ModellButtonClick(object sender, RoutedEventArgs e)
```

```
296         {
297             PerspectiveCamera camera = (PerspectiveCamera)mainViewport.
                Camera;
298             Point3D position = new Point3D(
299                 Convert.ToDouble(cameraPositionXTextBox.Text) ,
300                 Convert.ToDouble(cameraPositionYTextBox.Text)+1,
301                 Convert.ToDouble(cameraPositionZTextBox.Text)
302             );
303             Vector3D lookDirection = new Vector3D(
304                 Convert.ToDouble(lookAtXTextBox.Text),
305                 Convert.ToDouble(lookAtYTextBox.Text),
306                 Convert.ToDouble(lookAtZTextBox.Text)
307             );
308             camera.Position = position;
309             camera.LookDirection = lookDirection;
310             cameraPositionYTextBox.Text = (Convert.ToDouble(
                    cameraPositionYTextBox.Text) + 1).ToString();
311         }
312
313         private void LY_ModellButtonClick(object sender, RoutedEventArgs e)
314         {
315             PerspectiveCamera camera = (PerspectiveCamera)mainViewport.
                Camera;
316             Point3D position = new Point3D(
317                 Convert.ToDouble(cameraPositionXTextBox.Text),
318                 Convert.ToDouble(cameraPositionYTextBox.Text) - 1,
319                 Convert.ToDouble(cameraPositionZTextBox.Text)
320             );
321             Vector3D lookDirection = new Vector3D(
322                 Convert.ToDouble(lookAtXTextBox.Text),
323                 Convert.ToDouble(lookAtYTextBox.Text),
324                 Convert.ToDouble(lookAtZTextBox.Text)
325             );
326             camera.Position = position;
327             camera.LookDirection = lookDirection;
328             cameraPositionYTextBox.Text = (Convert.ToDouble(
                    cameraPositionYTextBox.Text) - 1).ToString();
329         }
330
331         private void RZ_ModellButtonClick(object sender, RoutedEventArgs e)
332         {
333             PerspectiveCamera camera = (PerspectiveCamera)mainViewport.
                Camera;
334             Point3D position = new Point3D(
335                 Convert.ToDouble(cameraPositionXTextBox.Text),
336                 Convert.ToDouble(cameraPositionYTextBox.Text),
337                 Convert.ToDouble(cameraPositionZTextBox.Text)+1
338             );
339             Vector3D lookDirection = new Vector3D(
340                 Convert.ToDouble(lookAtXTextBox.Text),
341                 Convert.ToDouble(lookAtYTextBox.Text),
342                 Convert.ToDouble(lookAtZTextBox.Text)
343             );
344             camera.Position = position;
345             camera.LookDirection = lookDirection;
346             cameraPositionZTextBox.Text = (Convert.ToDouble(
                    cameraPositionZTextBox.Text) + 1).ToString();
347         }
348
```

```
349         private void LZ_ModellButtonClick(object sender, RoutedEventArgs e)
350         {
351             PerspectiveCamera camera = (PerspectiveCamera)mainViewport.
                Camera;
352             Point3D position = new Point3D(
353                 Convert.ToDouble(cameraPositionXTextBox.Text),
354                 Convert.ToDouble(cameraPositionYTextBox.Text),
355                 Convert.ToDouble(cameraPositionZTextBox.Text) - 1
356             );
357             Vector3D lookDirection = new Vector3D(
358                 Convert.ToDouble(lookAtXTextBox.Text),
359                 Convert.ToDouble(lookAtYTextBox.Text),
360                 Convert.ToDouble(lookAtZTextBox.Text)
361             );
362             camera.Position = position;
363             camera.LookDirection = lookDirection;
364             cameraPositionZTextBox.Text = (Convert.ToDouble(
                    cameraPositionZTextBox.Text) - 1).ToString();
365         }
366
367         private void Lx_LookButtonClick(object sender, RoutedEventArgs e)
368         {
369             PerspectiveCamera camera = (PerspectiveCamera)mainViewport.
                Camera;
370             Point3D position = new Point3D(
371                 Convert.ToDouble(cameraPositionXTextBox.Text),
372                 Convert.ToDouble(cameraPositionYTextBox.Text),
373                 Convert.ToDouble(cameraPositionZTextBox.Text)
374             );
375             Vector3D lookDirection = new Vector3D(
376                 Convert.ToDouble(lookAtXTextBox.Text)-1,
377                 Convert.ToDouble(lookAtYTextBox.Text),
378                 Convert.ToDouble(lookAtZTextBox.Text)
379             );
380             camera.Position = position;
381             camera.LookDirection = lookDirection;
382             lookAtXTextBox.Text = (Convert.ToDouble(lookAtXTextBox.Text) -
                    1).ToString();
383         }
384
385         private void RX_LookButtonClick(object sender, RoutedEventArgs e)
386         {
387             PerspectiveCamera camera = (PerspectiveCamera)mainViewport.
                Camera;
388             Point3D position = new Point3D(
389                 Convert.ToDouble(cameraPositionXTextBox.Text),
390                 Convert.ToDouble(cameraPositionYTextBox.Text),
391                 Convert.ToDouble(cameraPositionZTextBox.Text)
392             );
393             Vector3D lookDirection = new Vector3D(
394                 Convert.ToDouble(lookAtXTextBox.Text) + 1,
395                 Convert.ToDouble(lookAtYTextBox.Text),
396                 Convert.ToDouble(lookAtZTextBox.Text)
397             );
398             camera.Position = position;
399             camera.LookDirection = lookDirection;
400             lookAtXTextBox.Text = (Convert.ToDouble(lookAtXTextBox.Text) +
                    1).ToString();
401         }
```

```
402
403         private void LY_LookButtonClick(object sender, RoutedEventArgs e)
404         {
405             PerspectiveCamera camera = (PerspectiveCamera)mainViewport.
                Camera;
406             Point3D position = new Point3D(
407                 Convert.ToDouble(cameraPositionXTextBox.Text),
408                 Convert.ToDouble(cameraPositionYTextBox.Text),
409                 Convert.ToDouble(cameraPositionZTextBox.Text)
410             );
411             Vector3D lookDirection = new Vector3D(
412                 Convert.ToDouble(lookAtXTextBox.Text),
413                 Convert.ToDouble(lookAtYTextBox.Text)-1,
414                 Convert.ToDouble(lookAtZTextBox.Text)
415             );
416             camera.Position = position;
417             camera.LookDirection = lookDirection;
418             lookAtYTextBox.Text = (Convert.ToDouble(lookAtYTextBox.Text) -
                1).ToString();
419         }
420
421         private void RY_LookButtonClick(object sender, RoutedEventArgs e)
422         {
423             PerspectiveCamera camera = (PerspectiveCamera)mainViewport.
                Camera;
424             Point3D position = new Point3D(
425                 Convert.ToDouble(cameraPositionXTextBox.Text),
426                 Convert.ToDouble(cameraPositionYTextBox.Text),
427                 Convert.ToDouble(cameraPositionZTextBox.Text)
428             );
429             Vector3D lookDirection = new Vector3D(
430                 Convert.ToDouble(lookAtXTextBox.Text),
431                 Convert.ToDouble(lookAtYTextBox.Text) + 1,
432                 Convert.ToDouble(lookAtZTextBox.Text)
433             );
434             camera.Position = position;
435             camera.LookDirection = lookDirection;
436             lookAtYTextBox.Text = (Convert.ToDouble(lookAtYTextBox.Text) +
                1).ToString();
437         }
438
439         private void LZ_LookButtonClick(object sender, RoutedEventArgs e)
440         {
441             PerspectiveCamera camera = (PerspectiveCamera)mainViewport.
                Camera;
442             Point3D position = new Point3D(
443                 Convert.ToDouble(cameraPositionXTextBox.Text),
444                 Convert.ToDouble(cameraPositionYTextBox.Text),
445                 Convert.ToDouble(cameraPositionZTextBox.Text)
446             );
447             Vector3D lookDirection = new Vector3D(
448                 Convert.ToDouble(lookAtXTextBox.Text),
449                 Convert.ToDouble(lookAtYTextBox.Text) ,
450                 Convert.ToDouble(lookAtZTextBox.Text)-1
451             );
452             camera.Position = position;
453             camera.LookDirection = lookDirection;
454             lookAtZTextBox.Text = (Convert.ToDouble(lookAtZTextBox.Text) -
                1).ToString();
```

```
455         }
456
457         private void RZ_LookButtonClick(object sender, RoutedEventArgs e)
458         {
459             PerspectiveCamera camera = (PerspectiveCamera)mainViewport.
                Camera;
460             Point3D position = new Point3D(
461                 Convert.ToDouble(cameraPositionXTextBox.Text),
462                 Convert.ToDouble(cameraPositionYTextBox.Text),
463                 Convert.ToDouble(cameraPositionZTextBox.Text)
464             );
465             Vector3D lookDirection = new Vector3D(
466                 Convert.ToDouble(lookAtXTextBox.Text),
467                 Convert.ToDouble(lookAtYTextBox.Text),
468                 Convert.ToDouble(lookAtZTextBox.Text) + 1
469             );
470             camera.Position = position;
471             camera.LookDirection = lookDirection;
472             lookAtZTextBox.Text = (Convert.ToDouble(lookAtZTextBox.Text) +
                1).ToString();
473         }
474     }
475 }
```

M Firmware Controller

uart.h

```
1
2
3  #ifndef UART_H_
4  #define UART_H_
5
6
7  unsigned char ser_getc_1 (void);
8  void uart_putc_1(unsigned char);
9  void uart_puts_1 (char *);
10 void uart_ini_1 (void);
11
12 unsigned char ser_getc_0 (void);
13 void uart_putc_0(unsigned char);
14 void uart_puts_0 (char *);
15 void uart_ini_0 (void);
16
17 unsigned char Getrbuffcnt_0(void);
18
19 #endif
```

uart.c

```
1  /* Eingaben über ein Hyperterminal werden als ECHO zurückgesendet oder auf
       dem LCD ausgegeben
2   *
3   */
4
5  #ifndef F_CPU
6  #define F_CPU 12000000UL
7  #endif
8
9  #include <avr/signal.h>
10 #include <avr/interrupt.h>
11 #include "uart.h"
12
13 #define UART_BAUD_RATE 19200L
14 #define UART_BAUD_CALC(UART_BAUD_RATE,F_CPU)  ((F_CPU)/((UART_BAUD_RATE)*16L
       )-1)
15
16 #define RBUFFLEN 80 //Pufferlänge für seriellen Empfang
17
18 volatile unsigned char rbuff_1[RBUFFLEN];        // Ringpuffer
19 volatile uint8_t        rbuffpos_1,      // Position, die als nächstes
       gelesen werden muß im Ringpuffer
20                                          rbuffcnt_1,      // Anzahl zu
                                                lesender Zeichen im Puffer
21                                          udr_data_1;              // Daten
                                                aus dem UART (volatile, damit
                                                nicht wegoptimiert wird vom
                                                Präprozessor)
22
23 volatile unsigned char rbuff_0[RBUFFLEN];        // Ringpuffer
24 volatile uint8_t        rbuffpos_0,      // Position, die als nächstes
       gelesen werden muß im Ringpuffer
```

```
25                                              rbuffcnt_0,     // Anzahl zu
                                                    lesender Zeichen im Puffer
26                                              udr_data_0;             // Daten
                                                    aus dem UART (volatile, damit
                                                    nicht wegoptimiert wird vom
                                                    Präprozessor)
27
28 // Interruptroutine, die Zeichen aus dem UART sofort ausliest, wenn
       empfangen
29 ISR (USART1_RX_vect)
30 {
31         //Byte auf jeden Fall abholen, sonst Endlosinterrupt
32         udr_data_1= UDR1;
33
34         if(rbuffcnt_1 < RBUFFLEN)       // kein Zeichen in einem vollen
               Ringpuffer überschreiben
35                 rbuff_1[(rbuffpos_1+rbuffcnt_1++) % RBUFFLEN] = udr_data_1;
36         // welche Position? Gelesene Zeichenpos + Anzahl Zeichen MODULO
               Pufferlänge
37         // (von 0 wieder anfangen, wenn Ende erreicht)
38 }
39
40 // Nächstes zu lesendes Zeichen aus Ringpuffer zurückgeben
41 unsigned char ser_getc_1 (void)
42 {
43         unsigned char c;
44         // Warte bis ein Zeichen vorhanden ist
45         while(!rbuffcnt_1);
46
47         cli();
48         // Interruptbehandlung kurz aussetzen. Ab jetzt muß es schnell
               gehen (wenig Befehle), damit Zeichen, die
49         // ab jetzt eintreffen nicht verloren gehen.
50         rbuffcnt_1--;   // anschl. ein Zeichen weniger zum ausgeben
51         c = rbuff_1 [rbuffpos_1++];     // Zeichen holen, was nach dem
               bereits gelesenen liegt
52         if (rbuffpos_1 >= RBUFFLEN)  rbuffpos_1 = 0;
53         // wenn hinterstes Zeichen (rechts im Puffer) gelesen wurde, dann
               wieder vorne anfangen
54
55         sei();  // Interruptbehandlung wieder aktivieren
56
57         return (c);     // Zeichen zurückgeben
58 }
59
60 // Ein Zeichen senden
61 void uart_putc_1(unsigned char c)
62 {
63     while(!(UCSR1A & (1 << UDRE1)));     // warte, bis UDR bereit
64     UDR1 = c;    // sende Zeichen
65 }
66
67 // Einen String senden
68 void uart_puts_1(char *s)
69 {
70           while (*s != '\0') { // send all chars except \0 (end of string)
71       uart_putc_1(*s);
72       s++; // increment pointer
73           }
```

```
74 }
75
76 // USART initialisieren
77 void uart_ini_1 ()
78 {
79         sei(); // Interruptbehandlung aktivieren
80
81         // Baudrate wählen
82         UBRR1H=(uint8_t)(UART_BAUD_CALC(UART_BAUD_RATE,F_CPU)>>8);
83         UBRR1L=(uint8_t)UART_BAUD_CALC(UART_BAUD_RATE,F_CPU);
84
85         UCSR1B |= (1 << TXEN1); // UART TX (senden) einschalten
86         UCSR1B |= (1 << RXEN1 ); // UART RX (empfangen) einschalten
87         UCSR1B |= (1 << RXCIE1 );
88         UCSR1C  |= (1<<UCSZ10)|(1<<UCSZ10); // Asynchron, 8N1
89 }
90
91
92 // Interruptroutine, die Zeichen aus dem UART sofort ausliest, wenn
       empfangen
93 ISR (USART0_RX_vect)
94 {
95         //Byte auf jeden Fall abholen, sonst Endlosinterrupt
96         udr_data_0= UDR0;
97         // kein Zeichen in einem vollen Ringpuffer überschreiben
98         if(rbuffcnt_0 < RBUFFLEN)
99                 rbuff_0[(rbuffpos_0+rbuffcnt_0++) % RBUFFLEN] = udr_data_0;
100        // welche Position? Gelesene Zeichenpos + Anzahl Zeichen MODULO
           Pufferlänge
101        // (von 0 wieder anfangen, wenn Ende erreicht)
102
103        CheckConnection(udr_data_0);
104 }
105
106 // Nächstes zu lesendes Zeichen aus Ringpuffer zurückgeben
107 unsigned char ser_getc_0 (void)
108 {
109        unsigned char c;
110        // Warte bis ein Zeichen vorhanden ist
111        while(!rbuffcnt_0);
112
113        cli();
114        // Interruptbehandlung kurz aussetzen. Ab jetzt muß es schnell
           gehen (wenig Befehle)
115        //, damit Zeichen, die ab jetzt eintreffen nicht verloren gehen.
116        rbuffcnt_0--;  // anschl. ein Zeichen weniger zum ausgeben
117        c = rbuff_0 [rbuffpos_0++];      // Zeichen holen, was nach dem
           bereits gelesenen liegt
118        if (rbuffpos_0 >= RBUFFLEN)  rbuffpos_0 = 0;
119        // wenn hinterstes Zeichen (rechts im Puffer) gelesen wurde, dann
           wieder vorne anfangen
120
121        sei(); // Interruptbehandlung wieder aktivieren
122
123        return (c);      // Zeichen zurückgeben
124 }
125
126 // Ein Zeichen senden
127 void uart_putc_0(unsigned char c)
```

```
128 {
129     // warte, bis UDR bereit
130         while(!(UCSR0A & (1 << UDRE0)));
131
132     UDR0= c;      // sende Zeichen
133 }
134
135 // Einen String senden
136 void uart_puts_0(char *s)
137 {
138           // send all chars except |0 (end of string)
139           while (*s != '\0') {
140       uart_putc_0(*s);
141       s++; // increment pointer
142           }
143 }
144
145 // USART initialisieren
146 void uart_ini_0 ()
147 {
148         sei();  // Interruptbehandlung aktivieren
149
150         // Baudrate wählen
151         UBRR0H=(uint8_t)(UART_BAUD_CALC(UART_BAUD_RATE,F_CPU)>>8);
152         UBRR0L=(uint8_t)UART_BAUD_CALC(UART_BAUD_RATE,F_CPU);
153
154         UCSR0B |= (1 << TXEN0);            // UART TX (senden) einschalten
155         UCSR0B |= (1 << RXEN0 );           // UART RX (empfangen) einschalten
156         UCSR0B |= (1 << RXCIE0 );
157         UCSR0C  |= (1<<UCSZ00)|(1<<UCSZ00);       // Asynchron, 8N1
158 }
159
160 //Returns the Buffer Count from USART 0
161 unsigned char Getrbuffcnt_0(void)
162 {
163         return rbuffcnt_0;
164 }
```

Commands.h

```
1  /*
2   * Commands.h
3   *
4   * Created: 07.10.2011 16:39:51
5   *  Author: Niki
6   */
7
8
9  #ifndef COMMANDS_H_
10 #define COMMANDS_H_
11
12 //Rückgabe der Funktion Check Command
13 //Klasse V
14 #define COMMAND_PTP_SESSION_OPEN                0x0001
15 #define COMMAND_PTP_SESSION_CLOSE               0x0002
16 #define COMMAND_PTP_SHOOT                       0x0003
17 #define COMMAND_PTP_CHANGE_TV                   0x0004
18 #define COMMAND_PTP_CHANGE_AV                   0x0005
19 #define COMMAND_PTP_CHANGE_BELICHTUNG           0x0006
```

```
20 #define COMMAND_PTP_LIVE_VIEW_ON                    0x0007
21 #define COMMAND_PTP_FOKUS                           0x0008
22 #define COMMAND_PTP_GETCONNECTION                   0x0009
23 #define COMMAND_PTP_LIVE_VIEW_OFF                   0x000A
24
25 //Klasse I
26 #define COMMAND_PIN_READ                            0x0001
27 #define COMMAND_PIN_SET                             0x0002
28 #define COMMAND_PIN_STORE                           0x0003
29 #define COMMAND_PIN_STATE                           0x0004
30 #define COMMAND_GETCLASS                            0x0005
31 #define COMMAND_SET_OUTPUT                          0x0006
32
33 //Klasse II
34 #define COMMAND_READ_TEMP                           0x0007
35
36 //Rückgabe der Funktion Check Command
37 #define RESPONSE_OK                                 0x0000
38 #define RESPONSE_WRONG_PARAMETER                    0x0001
39 #define RESPONSE_NOT_SUPPORTED_CLASS                0x0002
40 #define RESPONSE_NOT_SUPPORTED_ADCLASS              0x0003
41 #define RESPONSE_NOT_SUPPORTED_PARAMETERCOUNT       0x0004
42 #define RESPONSE_NOT_SUPPORTED                      0x0005
43 #define RESPONSE_NOT_SUPPORTED_PHASE                0x0006
44 #define RESPONSE_ERROR                              0x0007
45 #define RESPONSE_PTP_ERROR                          0x0008
46 #define RESPONSE_DATA                               0x0009
47
48 #endif /* COMMANDS_H_ */
```

DS1621.h

```
1  /*
2   * DS1621.h
3   *
4   * Created: 09.03.2012 08:18:59
5   *  Author: Niki
6   */
7
8
9  #ifndef DS1621_H_
10 #define DS1621_H_
11
12 #include <avr/io.h>
13 #include "TWI.h"
14
15 #define TW_READ          1                  // LSB bei Busadresse zur
       Kennzeichnung eines Lesezugriffes
16 #define TW_WRITE         0                  // LSB bei Busadresse zur
       Kennzeichnung eines Schreibzugriffes
17
18 //Funktion liest die Temperatur aus
19 // Param: Adresse des DS1621 A2:0
20 // Return: 2 Byte Temperatur
21 uint16_t ReadTemp(char Address);
22
23 //Funktion initialisiert den DS1621
24 // Param: Adresse des DS1621 A2:0
25 void InitDS1621(char Address);
```

```
26
27 //Funktion startet eine Messung
28 // Param: Adresse des DS1621 A2:0
29 void StartConvert(char Address);
30
31 //Funktion liest nur ein Byte aus
32 uint8_t ReadByte(char Address);
33
34 #endif /* DS1621_H_ */
```

DS1621.c

```
1  /*
2   * DS1621.c
3   *
4   * Created: 09.03.2012 08:19:39
5   *  Author: Niki
6   */
7
8  #include "DS1621.h"
9
10 //Funktion initialisiert den DS1621
11 // Param: Adresse des DS1621 A2:0
12 void InitDS1621(char Address)
13 {
14         uint8_t command;
15         uint8_t add;
16
17         command = 0x01;//1Shot = 1
18
19         TWI_start();
20
21         add = (uint8_t) 0x90 + ((uint8_t)Address<<1) + (uint8_t) 0;//1001
               control bits + adresse + write
22
23         //Address Byte
24         TWI_send(add);
25
26         //Command Byte
27         TWI_send(0xAC);
28
29         //Daten
30         TWI_send(command);
31
32         //Stop
33         TWI_stop();
34 }
35
36 //Funktion liest nur ein Byte aus
37 uint8_t ReadByte(char Address)
38 {
39         uint8_t add;
40         uint8_t Temp;
41
42         Temp = 0;
43
44         TWI_start();
45
46
```

```
47          add = (uint8_t) 0x90 + ((uint8_t)Address<<1) + (uint8_t) 0;//1001
                control bits + adresse + write
48
49          //Address Byte
50          TWI_send(add);
51
52          //Command Byte
53          TWI_send(0xAC);
54
55          //Repeated Start
56          TWI_start();
57
58          add = (uint8_t) 0x90 + ((uint8_t)Address<<1) + (uint8_t) 1;//1001
                control bits + adresse + Read
59
60          //Address Byte
61          TWI_send(add);
62
63
64          //LS Byte
65          Temp |= TWI_receive(0);
66
67          //Stop
68          TWI_stop();
69
70          return Temp;
71  }
72
73
74  //Funktion liest die Temperatur aus
75  // Param: Adresse des DS1621 A2:0
76  // Return: 2 Byte Temperatur
77  uint16_t ReadTemp(char Address)
78  {
79          uint8_t add;
80          char HByte;
81          char LByte;
82          uint16_t Temp;
83
84          Temp = 0;
85
86          TWI_start();
87
88
89          add = (uint8_t) 0x90 + ((uint8_t)Address<<1) + (uint8_t) 0;//1001
                control bits + adresse + write
90
91          //Address Byte
92          TWI_send(add);
93
94          //Command Byte
95          TWI_send(0xAA);
96
97          //Repeated Start
98          TWI_start();
99
100         add = (uint8_t) 0x90 + ((uint8_t)Address<<1) + (uint8_t) 1;//1001
                control bits + adresse + Read
101
```

```
102         //Address Byte
103         TWI_send(add);
104
105         //MS Byte
106         HByte = TWI_receive(1);
107
108         //LS Byte
109         LByte |= TWI_receive(0);
110
111         //Stop
112         TWI_stop();
113
114         Temp = ((uint16_t)HByte<<8) + LByte;
115
116         return Temp;
117 }
118
119 //Funktion startet eine Messung
120 // Param: Adresse des DS1621 A2:0
121 void StartConvert(char Address)
122 {
123         uint8_t add;
124
125         TWI_start();
126
127         add = (uint8_t) 0x90 + ((uint8_t)Address<<1) + (uint8_t) 0;//1001
              control bits + adresse + write
128
129         //Address Byte
130         TWI_send(add);
131
132         //Command Byte
133         TWI_send(0xEE);
134
135         //Stop
136         TWI_stop();
137
138 }
```

EEPROM_PIN.h

```
1  /*
2   * EEPROM_PIN.h
3   *
4   * Created: 23.01.2012 19:46:59
5   *  Author: Niki
6   */
7
8
9  #ifndef EEPROM_PIN_H_
10 #define EEPROM_PIN_H_
11
12 #include <avr/eeprom.h>
13
14 struct Pin_A {
15         char Settings;
16         uint32_t Reaction;
17         uint32_t Time;
18 };
```

```
19
20 //Read EEPROM Value
21 //Function Read the EEPROM Value of the Pin
22 //@ Param Pin_A that should be read
23 void ReadEEPROMPin(struct Pin_A *Pin, char Number);
24
25
26
27 #endif /* EEPROM_PIN_H_ */
```

EEPROM_PIN.c

```
1  /*
2   * EEPROM_PIN.c
3   *
4   * Created: 23.01.2012 19:50:07
5   *  Author: Niki
6   */
7
8  #include "EEPROM_PIN.h"
9
10 //Struct im Speicher ablegen
11 EEMEM struct Pin_A  Pin_A_EEMEM[8] =
12         {
13                 { 0,0,0},
14                 { 0,0,0},
15                 { 0,0,0},
16                 { 0,0,0},
17                 { 0,0,0},
18                 { 0,0,0},
19                 { 0,0,0},
20                 { 0,0,0}
21         };
22
23 //Read EEPROM Value
24 //Function Read the EEPROM Value of the Pin
25 //@ Param Pin_A that should be read
26 //@ Param 2 Number of Pin in EEPROM
27 void ReadEEPROMPin(struct Pin_A *Pin, char Number)
28 {
29         eeprom_busy_wait();
30         eeprom_read_block(Pin,&Pin_A_EEMEM[Number], sizeof(struct Pin_A));
31 }
32
33
34 //Write EEPROM Value
35 //Function write the Pin Value to the EEPROM
36 //@ Param 1 Pin_A that should be write
37 //@ Param 2 Number of Pin in EEPROM
38 void WriteEEPROMPin(struct Pin_A *Pin, char Number)
39 {
40         eeprom_busy_wait();
41         eeprom_write_block(Pin,&Pin_A_EEMEM[Number], sizeof(struct Pin_A));
42 }
```

Protokoll.h

```
1  /*
2   * Protokoll.h
```

```
3   *
4   * Created: 01.10.2011 16:45:39
5   *  Author: Niki
6   */
7
8
9  #ifndef PROTOKOLL_H_
10 #define PROTOKOLL_H_
11
12 #include "SPI_Master.h"
13 #include "Commands.h"
14 #include "EEPROM_PIN.h"
15 #include <avr/io.h>
16 #include <util/delay.h>
17 #include <stdio.h>
18 #include <stdlib.h>
19
20
21
22 //CreateContainer
23 //Funktion die den zu übertragenden Container generiert
24 //Es wird keine dynamische Speicherverwaltung verwendet,
25 //damit auch größere Daten transferiert werden können
26 // @ Param1: Zu Übertragender Code
27 // @ Param2: Anzahl der Parameter
28 // @ Param3: Parameter Array
29 // @ Param4: Ab wann Parameter übertragen werden
30 // @ Param5: Bis wohin Parameter übertragen werden
31 // @ Param6: Container mit maximaler Größe
32 // @ Param7: TransactionID die zu übertragen ist
33 void CreateContainer(uint32_t CodeWort, unsigned short countParam, char *
       arrayParam, unsigned short start, unsigned short stop, char * Container,
        unsigned short TransID);
34
35 //SendContainer
36 //Funktion sendet den Container
37 //Abhängig von der Transportebene
38 //Container wird hier definiert
39 // @ Param1: Zu übertragender Code
40 // @ Param2: Anzahl der Parameter
41 // @ Param3: Parameter Array
42 // @ Param4: zusendende TransaktionsID
43 // @ Param5: Übertragungsmethode
44 // @ Return Value: 0 success, sonst <>  0
45 char SendContainer(uint32_t CodeWort, unsigned short countParam, char *
       arrayParam, unsigned short TransID, char Mode);
46
47 //SPI Commandphase and VNC II senden
48 // @ Param1: Unterscheidung ob man lesen oder schreiben will
49 //Modus = 0: Write
50 //Modus <>0: Read
51 void SendSPICommandPhase(char modus);
52
53 //ReadContainerSPI
54 //Funktion liest einen Container ein
55 //@Param1: ausgelesener Code
56 //@Param2: ausgelesene Anzahl der Parameter
57 //@return: Parameterfeld (free nicht vergessen)
```

```
58 char *  ReadContainerSPI(uint32_t *CodeResponseWort, unsigned short *
       countParam);
59
60 //ReadContainerUSART
61 //Funktion liest einen Container ein
62 //@Param1: ausgelesener Code
63 //@Param2: ausgelesene Anzahl der Parameter
64 //@return: Parameterfeld (free nicht vergessen)
65 char* ReadContainerUSART(uint32_t *CodeWort, unsigned short *countParam);
66
67 //LogicUnit
68 //Funktion managed die Kommunikation
69 // @Param1: Comment oder Response Code
70 // @Param2: Anzahl der Parameter
71 // @Param3: Parameter Array
72 void LogicUnit(uint32_t CommandResponseCode, unsigned short countParam,
       char * arrayParam);
73
74 //CheckCommand
75 //Funktion überprüft den Command Code ob er unterstützt wird und ob die
       Parameter stimmen
76 //Muss für jedes Device angepasst werden
77 // @Param1: Command Code
78 // @Param2: Anzahl der Parameter
79 //Return Command.h
80 unsigned short CheckCommand(uint32_t CommandResponseCode, unsigned short
       countParam);
81
82 //GetTransactionID
83 //Funktion liefert die gesendete TransactionID zurück
84 //@Return: TransactionID
85 unsigned short GetTransactionID();
86
87 //SetTransactionID
88 //Funktion setzt die gesendete TransactionID zurück
89 //@Param1: TransactionID
90 void SetTransactionID(unsigned short transID);
91
92 //GetTransactionIDRead
93 //Funktion liefert die gesendete TransactionIDRead zurück
94 //@Return: TransactionID
95 unsigned short GetTransactionIDRead();
96
97 //SetTransactionIDRead
98 //Funktion setzt die gesendete TransactionIDRead zurück
99 //@Param1: TransactionID
100 void SetTransactionIDRead(unsigned short transID);
101
102 //GetKlasse
103 //Funktion liefert die Klasse zurück
104 char GetKlasse(void);
105
106 //Funktion gibt des Modus für die Übertragung zurück
107 //@Return: Modus für die Übertragung
108 char GetModus(void);
109
110 #endif /* PROTOKOLL_H_ */
```

Protokoll.c

```
1  /*
2   * Protokoll.c
3   *
4   * Created: 01.10.2011 16:49:36
5   *  Author: Niki
6   */
7
8  #include "Protokoll.h"
9  #include "uart.h"
10 #include "BTM_222.h"
11 #include <avr/io.h>
12
13 #ifndef DEBUG
14 #define DEBUG 1
15 #endif
16
17 unsigned short TransactionID; //ID für die Transaktion Senden
18 unsigned short TransactionIDRead; //ID für TransaktionID lesen
19 unsigned short TransactionIDReadSPI; //ID für TransaktionID SPI lesen,
       Information wird nicht verwendet
20 char volatile Klasse; //Klasse des Moduls
21
22 //GetKlasse
23 //Funktion liefert die Klasse zurück
24 //@Return: Klasse
25 char GetKlasse(void)
26 {
27         return Klasse;
28 }
29
30 //SetKlasse
31 //Funktion setzt die Klasse
32 //@Param1: Neuer Wert
33 void SetKlasse(char wert)
34 {
35         Klasse = wert;
36 }
37
38 //Funktion gibt des Modus für die Übertragung zurück
39 //@Return: Modus für die Übertragung
40 char GetModus(void)
41 {
42         if(GetKlasse() == 0 || GetKlasse() == 2)
43         {
44                 //Ethernet
45                 return 2;
46         }
47
48         //Bluetooth
49         return 1;
50 }
51
52
53
54  //GetTransactionID
55 //Funktion liefert die gesendete TransactionID zurück
56 //@Return: TransactionID
57 unsigned short GetTransactionID()
58 {
```

```
59          return    TransactionID;
60  }
61
62  //SetTransactionID
63  //Funktion setzt die gesendete TransactionID zurück
64  //@Param1: TransactionID
65  void SetTransactionID(unsigned short transID)
66  {
67          TransactionID = transID;
68  }
69
70  //GetTransactionIDRead
71  //Funktion liefert die gesendete TransactionIDRead zurück
72  //@Return: TransactionID
73  unsigned short GetTransactionIDRead()
74  {
75          return    TransactionIDRead;
76  }
77
78
79  //SetTransactionIDRead
80  //Funktion setzt die gesendete TransactionIDRead zurück
81  //@Param1: TransactionID
82  void SetTransactionIDRead(unsigned short transID)
83  {
84          TransactionIDRead = transID;
85  }
86
87  //CreateContainer
88  //Funktion die den zu übertragenden Container generiert
89  //Es wird keine dynamische Speicherverwaltung verwendet,
90  //damit auch größere Daten transferiert werden können
91  // @ Param1: Zu Übertragender Code
92  // @ Param2: Anzahl der Parameter
93  // @ Param3: Parameter Array
94  // @ Param4: Ab wann Parameter übertragen werden (inklusive) (0 wenn alles
        in einem Container übertragen werden kann)
95  // @ Param5: Bis wohin Parameter übertragen werden (inklusive)
96  // @ Param6: Container mit maximaler Größe
97  // @ Param7: TransactionID die zu übertragen ist
98  void CreateContainer(uint32_t CodeWort, unsigned short countParam, char *
        arrayParam, unsigned short start, unsigned short stop, char * Container,
         unsigned short TransID)
99  {
100         uint32_t          i;
101         uint32_t          lengthContainer; //Länge des Containers in Byte
102
103         if(start == 0) //Keine Teilung des Containers notwendig
104         {
105                 //Länge des Containers bestimmen
106                 lengthContainer = 2 /*TransactionID*/ + 4 /*Code*/ + 2 /*
                        AnzahlParameter*/ + countParam;
107
108                 for (i=0;i<lengthContainer;i++)
109                 {
110                         switch(i)
111                         {
112                                 case 0:
113                                 case 1:
```

```
114                                             //TransaktionID
115                                             Container[i] = (char) (TransID>> i
                                                    *8) & 0xFF; //nachträglich
                                                    geändert, schauen ob es jetzt
                                                    stimmt
116                                             break;
117                                     case 2:
118                                     case 3:
119                                     case 4:
120                                     case 5:
121                                             //Code
122                                             Container[i] = (char) (CodeWort>> (
                                                    i-2)*8) & 0xFF;
123                                             break;
124                                     case 6:
125                                     case 7:
126                                             //Anzahl Parameter
127                                             Container[i] = (char) (countParam>>
                                                     (i-6)*8) & 0xFF;
128                                             break;
129                                     default:
130                                             //Parameter
131                                             Container[i] = *(arrayParam+(i-8));
132                                             break;
133                             }
134                     }
135             }
136             else //Nur die Parameter übertragen
137             {
138                     //Länge des Containers bestimmen
139                     lengthContainer = stop - start + 1;
140                     for (i=0;i<stop-start +1 ;i++)
141                     {
142                             switch(i)
143                             {
144                                     default:
145                                             //Parameter
146                                             Container[i] = *(arrayParam+start+i
                                                    );
147                                             break;
148                             }
149                     }
150             }
151     }
152
153     //SendContainer
154     //Funktion sendet den Container
155     //Abhängig von der Transportebene
156     //Container wird hier definiert
157     // @ Param1: Zu übertragender Code
158     // @ Param2: Anzahl der Parameter
159     // @ Param3: Parameter Array
160     // @ Param4: zusendende TransaktionsID
161     // @ Param5: Übertragungsmethode
162     // @ Return Value: 0 success, sonst <>  0
163     char SendContainer(uint32_t CodeWort, unsigned short countParam, char *
            arrayParam, unsigned short TransID, char Mode)
164     {
165             unsigned short ParameterLeft;
```

```
166         char i;
167         char countTransmitParameter;
168
169         //SPI Buffer
170         //USART Buffer auf Gegenseite, sicherheitshalber auch auf 64 Byte
                beschränken
171         char buffer[64];
172         unsigned short start;
173         unsigned short stop;
174         unsigned short j;
175
176         start = 0;
177         stop = 0;
178         i = 0;
179
180         for(i=0;i<64;i++)
181         {
182                 buffer[i] = 0;
183         }
184
185         //maximale Anzahl der Parameter die übertragen sind
186         //64-8 = 56
187         //Wenn mehr sind, müssen mehrere Container gesendet werden
188
189         ParameterLeft = countParam;
190
191         if(ParameterLeft>56)
192         {
193                 countTransmitParameter = 56;
194         }
195         else
196         {
197                 countTransmitParameter = ParameterLeft;
198         }
199
200         do
201         {
202                 //Container generieren
203                 CreateContainer(CodeWort,countTransmitParameter,arrayParam,
                        start,stop,buffer,TransID);
204
205                 //Abhängig von der Transportebene Start
206                 //
                        ------------------------------------------------------------------------
207
208                 if(Mode==0)
209                 {
210                         //SPI an VNC senden
211                         //Command Byte übertragen
212                         SendSPICommandPhase(0);
213                         for(i=0;i<(8+countTransmitParameter);i++)
214                         {
215                                 SPI_MasterTransmit(buffer[i]);
216                         }
217                 }
218
219
220                 if(Mode == 1)
```

```
221                 {
222                         //Bluetooth
223                         for(i=0;i<(8+countTransmitParameter);i++)
224                         {
225                                 uart_putc_0(buffer[i]);
226 
227                                 //warten bis gesendet Zeichen verarbeitet
                                        wurde
228                                 while(ser_getc_0() != 'o');
229                         }
230                 }
231 
232 
233                 //Ethernet
234                 if(Mode == 2)
235                 {
236                         //Ethernet
237                         for(i=0;i<(8+countTransmitParameter);i++)
238                         {
239                                 #ifdef DEBUG
240                                         uart_putc_1('A'+buffer[i]);
241                                 #endif
242                                 uart_putc_0(buffer[i]);
243                         }
244                 }
245 
246                 //
                        ------------------------------------------------------------------

247                 //Abhängig von der Transportebene Ende
248 
249                 if(start == 0)
250                 {
251                         //Erster Durchlauf
252                         if(ParameterLeft>56)
253                         {
254                                 start = 56;
255                                 ParameterLeft-=56;
256                         }
257                         else
258                         {
259                                 ParameterLeft=0;
260                         }
261                 }
262                 else
263                 {
264                         //Xter Durchlauf
265                         if(ParameterLeft>64)
266                         {
267                                 start = start + 64;
268                                 ParameterLeft-=64;
269                         }
270                         else
271                         {
272                                 start = start + 64;
273                                 ParameterLeft=0;
274                         }
275                 }
276 
```

```
277                     //Stop setzen
278                     if(ParameterLeft>64)
279                     {
280                             stop = start+64;
281                             countTransmitParameter = 64;
282                     }
283                     else
284                     {
285                             stop = start + ParameterLeft;
286                             countTransmitParameter = ParameterLeft;
287                     }
288             }while(ParameterLeft>0);
289
290
291             return 0;
292     }
293
294
295     //SPI Commandphase and VNC II senden
296     //Modus = 0: Write
297     //Modus <>0: Read
298     void SendSPICommandPhase(char modus)
299     {
300             char trans;
301
302             if (modus == 0)
303                     trans = 0x0F;
304             else
305                     trans = 0x1F;
306
307             SPI_MasterTransmit(trans);
308     }
309
310     //ReadContainerUSART
311     //Funktion liest einen Container ein
312     //@Param1: ausgelesener Code
313     //@Param2: ausgelesene Anzahl der Parameter
314     // return : Parameterfeld (free nicht vergessen)
315     char * ReadContainerUSART(uint32_t *CodeResponseWort, unsigned short *
        countParam)
316     {
317             char buffer[8];
318             char * arrayParam;
319             unsigned short i;
320
321             arrayParam =0;
322             //USART Implementation ATmega
323             //Wenn 8 Zeichen im Buffer sind, lese sie aus
324             i = 0;
325
326             #ifdef DEBUG
327                     uart_puts_1("Read␣USART\r\n");
328             #endif
329
330             for(i=0;i<8;i++)
331             {
332                     buffer[i]= ser_getc_0();
333                     uart_putc_1('A'+buffer[i]);
334             }
```

```
335
336         //Buffer auswerten
337         SetTransactionIDRead((uint16_t) (buffer[1]<<8) + (uint16_t)buffer
                [0]);
338
339         *countParam=(uint16_t)(buffer[7]<<8) + (uint16_t) buffer[6];
340
341
342         *CodeResponseWort = ((uint32_t) buffer[5] <<24) +((uint32_t) buffer
                [4] <<16) +((uint32_t)buffer[3] <<8) + (uint32_t) buffer[2];
343
344         //Parameter einlesen
345         if((*countParam)>0)
346         {
347
348                 #ifdef DEBUG
349                         uart_puts_1("Parameter\r\n");
350                 #endif
351
352                 arrayParam = malloc(sizeof(char)*(*countParam));
353
354                 if (arrayParam==0)
355                 {
356                         //Kein PLatz mehr
357                         #ifdef DEBUG
358                                 uart_puts_1("Kein Platz USART\r\n");
359                         #endif
360                         return 0;
361                 }
362
363                 for(i=0;i<(*countParam);i++)
364                 {
365                         arrayParam[i]=ser_getc_0();
366                 }
367                 //Free nicht vergessen
368
369         }
370         return arrayParam;
371 }
372
373 //ReadContainerSPI
374 //Funktion liest einen Container ein
375 //@Param1: ausgelesener Code
376 //@Param2: ausgelesene Anzahl der Parameter
377 //@return: Parameterfeld (free nicht vergessen)
378 char *ReadContainerSPI(uint32_t *CodeResponseWort, unsigned short *
        countParam)
379 {
380         char buffer[8]; //Empfangsbuffer, abhängig von der Transportebene
381         unsigned short i;
382         char check;
383         char * arrayParam;
384         arrayParam =0;
385         PORTB |= (0x08);
386         //SPI Implementation ATmega
387         //Die ersten 8 Zeichen einlesen
388         //SPI an VNC senden
389         //Command Byte übertragen
390         i = 0;
```

```
391         check = 1;
392         do
393         {
394                 //warten bis die Daten bereit sind
395                 PORTB &= ~(0x08);
396                 SendSPICommandPhase(1);
397                 buffer[0] = SPI_MasterReceive();
398                 PORTB |= (0x08);
399                 check = buffer[0]&0x08;
400         } while (check > 0); //TXE Bit Check Status Byte
401
402         PORTB &= ~(0x08);
403         SendSPICommandPhase(1);
404         SPI_MasterReceive();//Status brauchen wir nicht
405
406         for(i=0;i<8;i++)
407         {
408                 SPI_MasterTransmit(0);//Dummy Übertragung
409                 buffer[i]= SPI_MasterReceive();
410
411                 if(i<7)
412                 {
413                         SPI_MasterTransmit(0);//Dummy Übertragung
414                         SPI_MasterReceive();//Status brauchen wir nicht
415                 }
416
417         }
418
419         //Buffer auswerten
420         TransactionIDReadSPI = (uint16_t) (buffer[1]<<8) + (uint16_t)buffer
                [0];
421
422         *countParam=(uint16_t)(buffer[7]<<8) + (uint16_t) buffer[6];
423
424         *CodeResponseWort = ((uint32_t) buffer[5] <<24) +((uint32_t) buffer
                [4] <<16) +((uint32_t)buffer[3] <<8) + (uint32_t) buffer[2];
425
426         //Parameter einlesen
427         if((*countParam)>0)
428         {
429                 arrayParam = malloc(sizeof(char)*(*countParam));
430
431                 if (arrayParam==0)
432                 {
433                         //Kein PLatz mehr
434                         #ifdef DEBUG
435                                 uart_puts_1("Kein Platz SPI\r\n");
436                         #endif
437                         return;
438                 }
439
440                 SPI_MasterTransmit(0);//Dummy Übertragung
441                 SPI_MasterReceive();//Status brauchen wir nicht
442
443                 for(i=0;i<(*countParam);i++)
444                 {
445                         SPI_MasterTransmit(0); //Dummy Übertragung
446                         arrayParam[i]=SPI_MasterReceive();
447
```

```
448                             if(i<((*countParam)-1))
449                             {
450                                     SPI_MasterTransmit(0);//Dummy Übertragung
451                                     SPI_MasterReceive();//Status brauchen wir
                                            nicht
452                             }
453                     }
454                     //Free nicht vergessen
455
456             }
457             PORTB |= (0x08);
458             return arrayParam;
459     }
460
461     //LogicUnit
462     //Funktion managed die Kommunikation
463     // @Param1: Comment oder Response Code
464     // @Param2: Anzahl der Parameter
465     // @Param3: Parameter Array
466     void LogicUnit(uint32_t CommandResponseCode, unsigned short countParam,
            char * arrayParam)
467     {
468             char check;
469             char tmp;
470             char Klasse;
471             char Code;
472             char x;
473             char pin;
474             unsigned short returnValue;
475             char SendParameter[10];
476             char countSendParameter;
477             struct Pin_A SelectedPin;
478             uint32_t CodeReturnSPI;
479             unsigned short countParamSPI;
480             char *arrayParamSPI;
481
482             #ifdef DEBUG
483                     char buff[100];
484             #endif
485
486             arrayParamSPI=0;
487             countParamSPI=0;
488             CodeReturnSPI=0;
489
490             #ifdef DEBUG
491                     uart_puts_1("Logic Unit\n\r");
492             #endif
493
494             //Container auswerten
495             returnValue = CheckCommand(CommandResponseCode,countParam);
496
497             if(returnValue != RESPONSE_OK)
498             {
499                     //Fehler
500                     //ErrorMessage erzeugen
501                     if(arrayParam != 0)
502                     {
503                             free(arrayParam);
504                             arrayParam = 0;
```

```
505                 }
506
507                 #ifdef DEBUG
508                         uart_puts_1("Check Failed!\r\n");
509                 #endif
510
511                 CommandResponseCode = 0xFF000000 | returnValue;
512
513                 //Fehler senden
514                 SendContainer(CommandResponseCode,0,arrayParam,
                        GetTransactionIDRead(),GetModus());
515         }
516         else
517         {
518                 //Klasse herausfinden
519                 Klasse = (char) (CommandResponseCode>>16 & 0xFF);
520                 Code = (char) (CommandResponseCode & 0xFF);
521
522                 //Pin Befehle
523                 //Klasse I
524                 //-----------------------------------------
525                 if(Klasse == 1)
526                 {
527                         //Code überprüfen
528                         switch(Code)
529                                 {
530                                         case COMMAND_PIN_STORE:
531                                                 #ifdef DEBUG
532                                                         uart_puts_1("
                                                            COMMAND_PIN_STORE
                                                            \r\n");
533                                                 #endif
534
535                                                 //Funktion zum speichern
                                                        der aktuellen Pin
                                                        Einstellungen
536                                                 for(x=0;x<8;x++)
537                                                 {
538                                                         WriteEEPROMPin(
                                                            GetPin(x),x);
539                                                 }
540
541                                                 //ok
542                                                 //Response Phase
543                                                 CommandResponseCode = 0
                                                        xFF000000 | RESPONSE_OK;
544                                                 #ifdef DEBUG
545                                                         uart_puts_1("
                                                            Response OK\r\n"
                                                            );
546                                                 #endif
547
548                                                 //Response Phase
549                                                 SendContainer(
                                                        CommandResponseCode,0,
                                                        SendParameter,
                                                        GetTransactionIDRead(),
                                                        GetModus());
550
```

```
551                     break;
552             //      ------------------------------------------------------------

553             case COMMAND_PIN_STATE:
554                     //2 Bytes in der Datenphase zurück
555                     //1. Byte I/O
556                     //2. Byte Zustände
557
558                     #ifdef DEBUG
559                             uart_puts_1("COMMAND_PIN_STATE\r\n");
560                     #endif
561
562                     countSendParameter = 2;
563                     SendParameter[0] = (char)DDRA;
564                     SendParameter[1] = (char)PINA;
565
566                     //Daten Phase
567                     CommandResponseCode = 0x00000000 | RESPONSE_DATA;
568                     if(SendContainer(CommandResponseCode, countSendParameter, SendParameter, GetTransactionIDRead(), GetModus()) != 0)
569                     {
570                             //Fehler
571                             CommandResponseCode = 0xFF000000 | RESPONSE_ERROR;
572                             #ifdef DEBUG
573                                     uart_puts_1("Response_ERROR\r\n");
574                             #endif
575                     }
576                     else
577                     {
578                             //ok
579                             //Response Phase
580                             CommandResponseCode = 0xFF000000 | RESPONSE_OK;
581                             #ifdef DEBUG
582                                     uart_puts_1("Response_OK\r\n");
583                             #endif
```

```
584
585                         }
586
587                         //Response Phase
588                         SendContainer(
                                CommandResponseCode,0,
                                SendParameter,
                                GetTransactionIDRead(),
                                GetModus());
589
590                         break;
591                 //
                    ------------------------------------------------------------

592                 case COMMAND_PIN_READ:
593                         //1 Byte Welcher Port
594
595                         #ifdef DEBUG
596                                 uart_puts_1("
                                    COMMAND_PIN_READ
                                    \r\n");
597                         #endif
598
599                         //Parameter einlesen
600                         pin = arrayParam[0];
601
602                         if(pin>7)
603                         {
604                                 //Pin Nummer stimmt
                                      nicht
605                                 //Response Phase
606                                 CommandResponseCode
                                      = 0xFF000000 |
                                    RESPONSE_WRONG_PARAMETER
                                    ;
607
608                                 #ifdef DEBUG
609                                         uart_puts_1
                                            ("
                                            Response
                                            wrong
                                            Parameter
                                            \r\n");
610                                 #endif
611                         }
612                         else
613                         {
614                                 //gewünschten Pin
                                      auslesen
615                                 SelectedPin.
                                      Settings = ((
                                      struct Pin_A *)
                                      GetPin(pin))->
                                      Settings;
616                                 SelectedPin.
                                      Reaction = ((
                                      struct Pin_A *)
                                      GetPin(pin))->
                                      Reaction;
```

```
617                                     SelectedPin.Time =
                                            ((struct Pin_A
                                            *)GetPin(pin))->
                                            Time;
618
619
620
621                                     //Parameter für das
                                             Senden einlesen
622                                     countSendParameter
                                            = 9;
623                                     SendParameter[0] =
                                            (char)
                                            SelectedPin.
                                            Settings;
624                                     SendParameter[1] =
                                            (char) (
                                            SelectedPin.
                                            Reaction & 0xFF)
                                            ;
625                                     SendParameter[2] =
                                            (char) ((
                                            SelectedPin.
                                            Reaction >> 8)&
                                            0xFF);
626                                     SendParameter[3] =
                                            (char) ((
                                            SelectedPin.
                                            Reaction >> 16)&
                                             0xFF);
627                                     SendParameter[4] =
                                            (char) ((
                                            SelectedPin.
                                            Reaction >> 24)&
                                             0xFF);
628                                     SendParameter[5] =
                                            (char) (
                                            SelectedPin.Time
                                             & 0xFF);
629                                     SendParameter[6] =
                                            (char) ((
                                            SelectedPin.Time
                                             >> 8)& 0xFF);
630                                     SendParameter[7] =
                                            (char) ((
                                            SelectedPin.Time
                                             >> 16)& 0xFF);
631                                     SendParameter[8] =
                                            (char) ((
                                            SelectedPin.Time
                                             >> 24)& 0xFF);
632
633                                     //Daten Phase
634                                     CommandResponseCode
                                             = 0x00000000 |
                                            RESPONSE_DATA;
635                                     if(SendContainer(
                                            CommandResponseCode
                                            ,
```

```
                countSendParameter
                ,SendParameter,
                GetTransactionIDRead
                (),GetModus())
                !=0)
636             {
637                 //Fehler
638                 CommandResponseCode
                        =0
                    xFF000000
                    |
                    RESPONSE_ERROR
                    ;
639                 #ifdef
                    DEBUG
640                     uart_puts_1
                            (
                            "
                            Response
                            _
                            ERROR
                            \
                            r
                            \
                            n
                            "
                            )
                            ;
641                 #endif
642             }
643             else
644             {
645                 //ok
646                 //Response
                        Phase
647                 CommandResponseCode
                        =0
                    xFF000000
                    |
                    RESPONSE_OK
                    ;
648                 #ifdef
                    DEBUG
649                     uart_puts_1
                            (
                            "
                            Response
                            _
                            OK
                            \
                            r
                            \
                            n
                            "
                            )
                            ;
650                 #endif
```

```
651                                     }
652                                 }
653
654                                 //Response Phase
655                                 SendContainer(
                                        CommandResponseCode,0,
                                        SendParameter,
                                        GetTransactionIDRead(),
                                        GetModus());
656
657                                 break;
658                             //
                                ------------------------------------------------------------

659                             case COMMAND_PIN_SET:
660                                 //1. Byte Welcher Port Pin
661                                 //2. Byte SelectedPin->
                                        Settings;
662                                 //3. Byte       (
                                        SelectedPin->Reaction &
                                        0xFF);
663                                 //4. Byte       ((
                                        SelectedPin->Reaction >>
                                         8)& 0xFF);
664                                 //5. Byte       ((
                                        SelectedPin->Reaction >>
                                         16)& 0xFF);
665                                 //6. Byte       ((
                                        SelectedPin->Reaction >>
                                         24)& 0xFF);
666                                 //7. Byte       (
                                        SelectedPin->Time & 0xFF
                                        );
667                                 //8. Byte       ((
                                        SelectedPin->Time >> 8)&
                                         0xFF);
668                                 //9. Byte       ((
                                        SelectedPin->Time >> 16)
                                        & 0xFF);
669                                 //10. Byte      ((
                                        SelectedPin->Time >> 24)
                                        & 0xFF);
670
671                                 #ifdef DEBUG
672                                         uart_puts_1("
                                            COMMAND_PIN_SET\
                                            r\n");
673                                 #endif
674
675                                 //Parameter einlesen
676                                 pin = arrayParam[0];
677
678
679                                 if(pin>7)
680                                 {
681                                         //Pin Nummer stimmt
                                                nicht
682                                         //Response Phase
```

```
683                                 CommandResponseCode
                                        = 0xFF000000 |
                                    RESPONSE_WRONG_PARAMETER
                                    ;
684
685                                 #ifdef DEBUG
686                                         uart_puts_1
                                            ("
                                            Response
                                            wrong
                                            Parameter
                                            \r\n");
687                                 #endif
688                             }
689                             else
690                             {
691                                 //Parameter
                                        einlesen
692                                 SelectedPin.
                                        Settings =
                                        arrayParam[1];
693                                 SelectedPin.
                                        Reaction = (
                                        uint32_t) ((
                                        uint32_t)
                                        arrayParam
                                        [5]<<24)+ (
                                        uint32_t) ((
                                        uint32_t)
                                        arrayParam
                                        [4]<<16)+ (
                                        uint32_t) ((
                                        uint32_t)
                                        arrayParam
                                        [3]<<8)+  (
                                        uint32_t)
                                        arrayParam[2];
694                                 SelectedPin.Time =
                                        ((uint32_t)
                                        arrayParam
                                        [9]<<24)+ ((
                                        uint32_t)
                                        arrayParam
                                        [8]<<16)+ ((
                                        uint32_t)
                                        arrayParam
                                        [7]<<8)+ (
                                        uint32_t)
                                        arrayParam[6];
695
696
697                                 //(uint32_t)  ((
                                        uint32_t)
                                        arrayParam
                                        [9]<<24)+(
                                        uint32_t) ((
                                        uint32_t)
                                        arrayParam
                                        [8]<<16)+(
```

```
                                    uint32_t) ((
                                    uint32_t)
                                    arrayParam
                                    [7]<<8)+ (
                                    uint32_t)
                                    arrayParam[6];
698
699                                 //Pin übernehmen
700                                 SavePin(&
                                        SelectedPin,pin)
                                        ;
701
702                                 //Interrupts
                                        aktivieren
703                                 UpdateInterruptRegister
                                        ();
704
705                                 //Timer updaten
706                                 UpdateTime(((
                                        SelectedPin.Time
                                        >>16)&0x3FFF),
                                        arrayParam[0]);
707
708                                 //Ausgang des Pins
                                        richtig setzen
709                                 SetOutputPin(pin);
710
711                                 #ifdef DEBUG
712                                         sprintf(
                                                buff,"
                                                TimeA:␣%
                                                x␣TimeB:
                                                ␣%x\r\n"
                                                ,((
                                                SelectedPin
                                                .Time
                                                >>16)&0
                                                x3FFF),(
                                                SelectedPin
                                                .Time&0
                                                x3FFF));
713                                             uart_puts_1
                                                (buff);
714                                             sprintf(
                                                buff,"
                                                Time␣
                                                array:␣%
                                                x␣%x␣%x␣
                                                %x\r\n",
                                                arrayParam
                                                [9],
                                                arrayParam
                                                [8],
                                                arrayParam
                                                [7],
                                                arrayParam
                                                [6]);
715                                             uart_puts_1
                                                (buff);
```

```
716                                     #endif
717
718                                     //ok
719                                     //Response Phase
720                                     CommandResponseCode
                                            = 0xFF000000 |
                                        RESPONSE_OK;
721                                     #ifdef DEBUG
722                                             uart_puts_1
                                                ("
                                                Response
                                                _OK\r\n"
                                                );
723                                     #endif
724
725                                 }
726
727                                 //Response Phase
728                                 SendContainer(
                                        CommandResponseCode,0,
                                        SendParameter,
                                        GetTransactionIDRead(),
                                        GetModus());
729
730                                 break;
731                         //      ------------------------------------------------------------------

732                         case COMMAND_GETCLASS:
733                                 //1. Byte retour
734                                 #ifdef DEBUG
735                                         uart_puts_1("
                                            COMMAND_GETCLASS
                                            \r\n");
736                                 #endif
737
738                                 countSendParameter = 1;
739                                 SendParameter[0] =
                                        GetKlasse();
740
741                                 //Daten Phase
742                                 CommandResponseCode = 0
                                        x00000000 |
                                        RESPONSE_DATA;
743                                 if(SendContainer(
                                        CommandResponseCode,
                                        countSendParameter,
                                        SendParameter,
                                        GetTransactionIDRead(),
                                        GetModus()) != 0)
744                                 {
745                                         //Fehler
746                                         CommandResponseCode
                                                = 0xFF000000 |
                                            RESPONSE_ERROR;
747                                         #ifdef DEBUG
748                                                 uart_puts_1
                                                    ("
                                                    Response
```

```
                                                    _ERROR\r
                                                    \n");
749                                         #endif
750                                 }
751                                 else
752                                 {
753                                         //ok
754                                         //Response Phase
755                                         CommandResponseCode
                                                = 0xFF000000 |
                                                RESPONSE_OK;
756                                         #ifdef DEBUG
757                                                 uart_puts_1
                                                    ("
                                                    Response
                                                    _OK\r\n"
                                                    );
758                                         #endif
759
760                                 }
761
762                                 //Response Phase
763                                 SendContainer(
                                        CommandResponseCode,0,
                                        SendParameter,
                                        GetTransactionIDRead(),
                                        GetModus());
764                                 break;
765                         //      ------------------------------------------------------------

766                         case COMMAND_SET_OUTPUT:
767                                 //1. Byte I/O
768                                 //2. Byte Zustände
769                                 #ifdef DEBUG
770                                         uart_puts_1("
                                            COMMAND_SET_OUTPUT
                                            \r\n");
771                                 #endif
772
773                                 //Parameter einlesen
774                                 for (x=0;x<8;x++)
775                                 {
776                                         if((arrayParam[0]>>
                                            x)&0x01 == 0x01)
777                                         {
778                                                 DDRA |=
                                                    (1<<x);
779
780                                                 //Parameter

                                                    einlesen
781                                                 SelectedPin
                                                    .
                                                    Settings
                                                     = 0x80;
782                                                 SelectedPin
                                                    .
                                                    Reaction
```

```
                                                    = 0;
783                                                 SelectedPin
                                                        .Time =
                                                        0;
784
785                                                 //Pin
                                                        übernehmen
786                                                 SavePin(&
                                                        SelectedPin
                                                        ,pin);
787
788                                                 //
                                                        Interrupts
                                                        aktivieren
789                                                 UpdateInterruptRegister
                                                        ();
790
791                                                 //Timer
                                                        updaten
792                                                 UpdateTime
                                                        (((
                                                        SelectedPin
                                                        .Time
                                                        >>16)&0
                                                        x3FFF),
                                                        pin);
793
794                                                 if((
                                                        arrayParam
                                                        [1]>>x)
                                                        &0x01 ==
                                                         0x01)
795                                                 {
796                                                         //
                                                                Output
797                                                         PORTA
                                                                |=
                                                                (1<<
                                                                x
                                                                )
                                                                ;
798                                                 }
799                                                 else
800                                                 {
801                                                         PORTA
                                                                &=
                                                                ~(1<<
                                                                x
                                                                )
                                                                ;
```

```
802                                                             }
803                                                     }
804
805                                             }
806
807                                             //ok
808                                             //Response Phase
809                                             CommandResponseCode = 0
                                                    xFF000000 | RESPONSE_OK;
810                                             #ifdef DEBUG
811                                                     uart_puts_1("
                                                        Response OK\r\n"
                                                        );
812                                             #endif
813
814                                             //Response Phase
815                                             SendContainer(
                                                    CommandResponseCode,0,
                                                    SendParameter,
                                                    GetTransactionIDRead(),
                                                    GetModus());
816                                             break;
817                                     }
818             }
819
820             //TWI Befehle
821             if(Klasse == 2)
822             {
823                     //Code überprüfen
824                     switch(Code)
825                             {
826                                     case COMMAND_READ_TEMP:
827                                             //2 Byte retour
828                                             #ifdef DEBUG
829                                                     uart_puts_1("
                                                        COMMAND_READ_TEMP
                                                        \r\n");
830                                             #endif
831
832                                             countSendParameter = 2;
833                                             SendParameter[0] = (char) (
                                                    ActualTemp() & 0xFF);
834                                             SendParameter[1] = (char)
                                                    ((ActualTemp()>>8) & 0
                                                    xFF);
835
836                                             //Daten Phase
837                                             CommandResponseCode = 0
                                                    x00000000 |
                                                    RESPONSE_DATA;
838                                             if(SendContainer(
                                                    CommandResponseCode,
                                                    countSendParameter,
                                                    SendParameter,
                                                    GetTransactionIDRead(),
                                                    GetModus()) != 0)
839                                             {
840                                                     //Fehler
```

```
841                                                     CommandResponseCode
                                                            = 0xFF000000 |
                                                        RESPONSE_ERROR;
842                                                     #ifdef DEBUG
843                                                             uart_puts_1
                                                                ("
                                                                Response
                                                                _ERROR\r
                                                                \n");
844                                                     #endif
845                                             }
846                                             else
847                                             {
848                                                     //ok
849                                                     //Response Phase
850                                                     CommandResponseCode
                                                            = 0xFF000000 |
                                                        RESPONSE_OK;
851                                                     #ifdef DEBUG
852                                                             uart_puts_1
                                                                ("
                                                                Response
                                                                _OK\r\n"
                                                                );
853                                                     #endif
854
855                                             }
856
857                                             //Response Phase
858                                             SendContainer(
                                                    CommandResponseCode,0,
                                                    SendParameter,
                                                    GetTransactionIDRead(),
                                                    GetModus());
859                                             break;
860                             }
861             }
862
863             //Klasse 5 Befehle
864             //-----------------------------------------------
865             if(Klasse == 5)
866             {
867                     switch(Code)
868                     {
869                             case COMMAND_PTP_SESSION_OPEN:
870
871                                     #ifdef DEBUG
872                                             uart_puts_1("
                                                    COMMAND_PTP_SESSION_OPEN
                                                    \r\n");
873                                     #endif
874
875                                     PORTB &= ~(0x08);
876
877                                     //Befehl über SPI senden
878                                     if(SendContainer(
                                            CommandResponseCode,countParam,
                                            arrayParam,GetTransactionIDRead
                                            (),0) != 0)
```

```
879                                     {
880                                             //Fehler
881                                             CommandResponseCode = 0
                                                    xFF000000 |
                                                    RESPONSE_ERROR;
882                                             #ifdef DEBUG
883                                                     uart_puts_1("ERROR_
                                                            SPI_Transmission
                                                            \r\n");
884                                             #endif
885                                             PORTB |= (0x08);
886                                     }
887                                     else
888                                     {
889                                             PORTB |= (0x08);
890                                             //Antwort lesen
891                                             arrayParamSPI=
                                                    ReadContainerSPI(&
                                                    CodeReturnSPI,&
                                                    countParamSPI);
892
893                                             #ifdef DEBUG
894                                                     uart_puts_1("
                                                            Antwort_SPI_
                                                            gelesen\r\n");
895                                             #endif
896
897                                             //Es sollte kein Parameter
                                                    zurückübergeben werden
898                                             if(arrayParamSPI!=0)
899                                             {
900                                                     free(arrayParamSPI)
                                                            ;
901                                                     arrayParamSPI = 0;
902                                             }
903
904                                             CommandResponseCode =
                                                    CodeReturnSPI;
905                                     }
906
907                                     #ifdef DEBUG
908                                             uart_puts_1("Response\r\n")
                                                    ;
909                                     #endif
910
911                                     //Response Phase
912                                     SendContainer(CommandResponseCode
                                            ,0,SendParameter,
                                            GetTransactionIDRead(),GetModus
                                            ());
913
914                                     break;
915                             case COMMAND_PTP_SESSION_CLOSE:
916                                     #ifdef DEBUG
917                                             uart_puts_1("
                                                    COMMAND_PTP_SESSION_CLOSE
                                                    \r\n");
918                                     #endif
919
```

```
920                     PORTB &= ~(0x08);
921
922                     //Befehl über SPI senden
923                     if(SendContainer(
                            CommandResponseCode,countParam,
                            arrayParam,GetTransactionIDRead
                            (),0) != 0)
924                     {
925                             //Fehler
926                             CommandResponseCode = 0
                                    xFF000000 |
                                    RESPONSE_ERROR;
927                             #ifdef DEBUG
928                                     uart_puts_1("ERROR␣
                                            SPI␣Transmission
                                            \r\n");
929                             #endif
930                             PORTB |= (0x08);
931                     }
932                     else
933                     {
934                             PORTB |= (0x08);
935                             //Antwort lesen
936                             arrayParamSPI=
                                    ReadContainerSPI(&
                                    CodeReturnSPI,&
                                    countParamSPI);
937
938                             #ifdef DEBUG
939                                     uart_puts_1("
                                            Antwort␣SPI␣
                                            gelesen\r\n");
940                             #endif
941
942                             //Es sollte kein Parameter
                                    zurückübergeben werden
943                             if(arrayParamSPI!=0)
944                             {
945                                     free(arrayParamSPI)
                                            ;
946                                     arrayParamSPI = 0;
947                             }
948
949                             CommandResponseCode =
                                    CodeReturnSPI;
950                     }
951
952                     #ifdef DEBUG
953                             uart_puts_1("Response\r\n")
                                    ;
954                     #endif
955
956                     //Response Phase
957                     SendContainer(CommandResponseCode
                            ,0,SendParameter,
                            GetTransactionIDRead(),GetModus
                            ());
958
959                     break;
```

```
960                             case COMMAND_PTP_SHOOT:
961                                     #ifdef DEBUG
962                                             uart_puts_1("
                                                    COMMAND_PTP_SHOOT\r\n");
963                                     #endif
964
965                                     PORTB &= ~(0x08);
966
967                                     //Befehl über SPI senden
968                                     if(SendContainer(
                                            CommandResponseCode,countParam,
                                            arrayParam,GetTransactionIDRead
                                            (),0) != 0)
969                                     {
970                                             //Fehler
971                                             CommandResponseCode = 0
                                                xFF000000 |
                                                RESPONSE_ERROR;
972                                             #ifdef DEBUG
973                                                     uart_puts_1("ERROR␣
                                                        SPI␣Transmission
                                                        \r\n");
974                                             #endif
975                                             PORTB |= (0x08);
976                                     }
977                                     else
978                                     {
979                                             PORTB |= (0x08);
980                                             //Antwort lesen
981                                             arrayParamSPI=
                                                ReadContainerSPI(&
                                                CodeReturnSPI,&
                                                countParamSPI);
982
983                                             #ifdef DEBUG
984                                                     uart_puts_1("
                                                        Antwort␣SPI␣
                                                        gelesen\r\n");
985                                             #endif
986
987                                             //Es sollte kein Parameter
                                                zurückübergeben werden
988                                             if(arrayParamSPI!=0)
989                                             {
990                                                     free(arrayParamSPI)
                                                        ;
991                                                     arrayParamSPI = 0;
992                                             }
993
994                                             CommandResponseCode =
                                                CodeReturnSPI;
995                                     }
996
997                                     #ifdef DEBUG
998                                             uart_puts_1("Response\r\n")
                                                ;
999                                     #endif
1000
1001                                    //Response Phase
```

```
1002                                    SendContainer(CommandResponseCode
                                            ,0,SendParameter,
                                            GetTransactionIDRead(),GetModus
                                            ());
1003
1004                                    break;
1005                                case COMMAND_PTP_CHANGE_TV:
1006                                    //1. Byte ist der neue Wert
1007                                    #ifdef DEBUG
1008                                            uart_puts_1("
                                                COMMAND_PTP_CHANGE_TV\r\
                                                n");
1009                                    #endif
1010
1011                                    PORTB &= ~(0x08);
1012
1013                                    //Befehl über SPI senden
1014                                    if(SendContainer(
                                            CommandResponseCode,countParam,
                                            arrayParam,GetTransactionIDRead
                                            (),0) != 0)
1015                                    {
1016                                            //Fehler
1017                                            CommandResponseCode = 0
                                                xFF000000 |
                                                RESPONSE_ERROR;
1018                                            #ifdef DEBUG
1019                                                    uart_puts_1("ERROR␣
                                                        SPI␣Transmission
                                                        \r\n");
1020                                            #endif
1021                                            PORTB |= (0x08);
1022                                    }
1023                                    else
1024                                    {
1025                                            PORTB |= (0x08);
1026                                            //Antwort lesen
1027                                            arrayParamSPI=
                                                ReadContainerSPI(&
                                                CodeReturnSPI,&
                                                countParamSPI);
1028
1029                                            #ifdef DEBUG
1030                                                    uart_puts_1("
                                                        Antwort␣SPI␣
                                                        gelesen\r\n");
1031                                            #endif
1032
1033                                            //Es sollte kein Parameter
                                                zurückübergeben werden
1034                                            if(arrayParamSPI!=0)
1035                                            {
1036                                                    free(arrayParamSPI)
                                                        ;
1037                                                    arrayParamSPI = 0;
1038                                            }
1039
1040                                            CommandResponseCode =
                                                CodeReturnSPI;
```

```
1041                                    }
1042
1043                                    #ifdef DEBUG
1044                                            uart_puts_1("Response\r\n")
                                                    ;
1045                                    #endif
1046
1047                                    //Response Phase
1048                                    SendContainer(CommandResponseCode
                                            ,0,SendParameter,
                                            GetTransactionIDRead(),GetModus
                                            ());
1049                                    break;
1050                            case COMMAND_PTP_CHANGE_AV:
1051                                    //1. Byte ist der neue Wert
1052                                    #ifdef DEBUG
1053                                            uart_puts_1("
                                                    COMMAND_PTP_CHANGE_AV\r\
                                                    n");
1054                                    #endif
1055
1056                                    PORTB &= ~(0x08);
1057
1058                                    //Befehl über SPI senden
1059                                    if(SendContainer(
                                            CommandResponseCode,countParam,
                                            arrayParam,GetTransactionIDRead
                                            (),0) != 0)
1060                                    {
1061                                            //Fehler
1062                                            CommandResponseCode = 0
                                                    xFF000000 |
                                                    RESPONSE_ERROR;
1063                                            #ifdef DEBUG
1064                                                    uart_puts_1("ERROR␣
                                                            SPI␣Transmission
                                                            \r\n");
1065                                            #endif
1066                                            PORTB |= (0x08);
1067                                    }
1068                                    else
1069                                    {
1070                                            PORTB |= (0x08);
1071                                            //Antwort lesen
1072                                            arrayParamSPI=
                                                    ReadContainerSPI(&
                                                    CodeReturnSPI,&
                                                    countParamSPI);
1073
1074                                            #ifdef DEBUG
1075                                                    uart_puts_1("
                                                            Antwort␣SPI␣
                                                            gelesen\r\n");
1076                                            #endif
1077
1078                                            //Es sollte kein Parameter
                                                    zurückübergeben werden
1079                                            if(arrayParamSPI!=0)
1080                                            {
```

```
1081                                                    free(arrayParamSPI)
                                                            ;
1082                                                    arrayParamSPI = 0;
1083                                            }
1084
1085                                            CommandResponseCode =
                                                    CodeReturnSPI;
1086                                    }
1087
1088                                    #ifdef DEBUG
1089                                            uart_puts_1("Response\r\n")
                                                    ;
1090                                    #endif
1091
1092                                    //Response Phase
1093                                    SendContainer(CommandResponseCode
                                        ,0,SendParameter,
                                        GetTransactionIDRead(),GetModus
                                        ());
1094                                    break;
1095                            case COMMAND_PTP_CHANGE_BELICHTUNG:
1096                                    //1. Byte ist der neue Wert
1097                                    #ifdef DEBUG
1098                                            uart_puts_1("
                                                COMMAND_PTP_CHANGE_BELICHTUNG
                                                \r\n");
1099                                    #endif
1100
1101                                    PORTB &= ~(0x08);
1102
1103                                    //Befehl über SPI senden
1104                                    if(SendContainer(
                                        CommandResponseCode,countParam,
                                        arrayParam,GetTransactionIDRead
                                        (),0) != 0)
1105                                    {
1106                                            //Fehler
1107                                            CommandResponseCode = 0
                                                xFF000000 |
                                                RESPONSE_ERROR;
1108                                            #ifdef DEBUG
1109                                                    uart_puts_1("ERROR␣
                                                        SPI␣Transmission
                                                        \r\n");
1110                                            #endif
1111                                            PORTB |= (0x08);
1112                                    }
1113                                    else
1114                                    {
1115                                            PORTB |= (0x08);
1116                                            //Antwort lesen
1117                                            arrayParamSPI=
                                                ReadContainerSPI(&
                                                CodeReturnSPI,&
                                                countParamSPI);
1118
1119                                            #ifdef DEBUG
1120                                                    uart_puts_1("
                                                        Antwort␣SPI␣
```

```
                                                gelesen\r\n");
1121                                    #endif
1122
1123                                    //Es sollte kein Parameter
                                            zurückübergeben werden
1124                                    if(arrayParamSPI!=0)
1125                                    {
1126                                            free(arrayParamSPI)
                                                ;
1127                                            arrayParamSPI = 0;
1128                                    }
1129
1130                                    CommandResponseCode =
                                            CodeReturnSPI;
1131                            }
1132
1133                            #ifdef DEBUG
1134                                    uart_puts_1("Response\r\n")
                                        ;
1135                            #endif
1136
1137                            //Response Phase
1138                            SendContainer(CommandResponseCode
                                ,0,SendParameter,
                                GetTransactionIDRead(),GetModus
                                ());
1139                            break;
1140                    case COMMAND_PTP_LIVE_VIEW_ON:
1141                            #ifdef DEBUG
1142                                    uart_puts_1("
                                        COMMAND_PTP_LIVE_VIEW_ON
                                        \r\n");
1143                            #endif
1144
1145                            PORTB &= ~(0x08);
1146
1147                            //Befehl über SPI senden
1148                            if(SendContainer(
                                CommandResponseCode,countParam,
                                arrayParam,GetTransactionIDRead
                                (),0) != 0)
1149                            {
1150                                    //Fehler
1151                                    CommandResponseCode = 0
                                        xFF000000 |
                                        RESPONSE_ERROR;
1152                                    #ifdef DEBUG
1153                                            uart_puts_1("ERROR␣
                                                SPI␣Transmission
                                                \r\n");
1154                                    #endif
1155                                    PORTB |= (0x08);
1156                            }
1157                            else
1158                            {
1159                                    PORTB |= (0x08);
1160                                    //Antwort lesen
1161                                    arrayParamSPI=
                                            ReadContainerSPI(&
```

```
                                        CodeReturnSPI,&
                                        countParamSPI);
1162
1163                                    #ifdef DEBUG
1164                                            uart_puts_1("
                                                Antwort SPI
                                                gelesen\r\n");
1165                                    #endif
1166
1167                                    //Es sollte kein Parameter
                                            zurückübergeben werden
1168                                    if(arrayParamSPI!=0)
1169                                    {
1170                                            free(arrayParamSPI)
                                                ;
1171                                            arrayParamSPI = 0;
1172                                    }
1173
1174                                    CommandResponseCode =
                                            CodeReturnSPI;
1175                            }
1176
1177                            #ifdef DEBUG
1178                                    uart_puts_1("Response\r\n")
                                        ;
1179                            #endif
1180
1181                            //Response Phase
1182                            SendContainer(CommandResponseCode
                                ,0,SendParameter,
                                GetTransactionIDRead(),GetModus
                                ());
1183                            break;
1184                    case COMMAND_PTP_LIVE_VIEW_OFF:
1185                            #ifdef DEBUG
1186                                    uart_puts_1("
                                        COMMAND_PTP_LIVE_VIEW_OFF
                                        \r\n");
1187                            #endif
1188
1189                            PORTB &= ~(0x08);
1190
1191                            //Befehl über SPI senden
1192                            if(SendContainer(
                                CommandResponseCode,countParam,
                                arrayParam,GetTransactionIDRead
                                (),0) != 0)
1193                            {
1194                                    //Fehler
1195                                    CommandResponseCode = 0
                                        xFF000000 |
                                        RESPONSE_ERROR;
1196                                    #ifdef DEBUG
1197                                            uart_puts_1("ERROR
                                                SPI Transmission
                                                \r\n");
1198                                    #endif
1199                                    PORTB |= (0x08);
1200                            }
```

```
1201                                    else
1202                                    {
1203                                            PORTB |= (0x08);
1204                                            //Antwort lesen
1205                                            arrayParamSPI=
                                                    ReadContainerSPI(&
                                                    CodeReturnSPI,&
                                                    countParamSPI);
1206
1207                                            #ifdef DEBUG
1208                                                    uart_puts_1("
                                                        Antwort SPI
                                                        gelesen\r\n");
1209                                            #endif
1210
1211                                            //Es sollte kein Parameter
                                                    zurückübergeben werden
1212                                            if(arrayParamSPI!=0)
1213                                            {
1214                                                    free(arrayParamSPI)
                                                        ;
1215                                                    arrayParamSPI = 0;
1216                                            }
1217
1218                                            CommandResponseCode =
                                                    CodeReturnSPI;
1219                                    }
1220
1221                                    #ifdef DEBUG
1222                                            uart_puts_1("Response\r\n")
                                                ;
1223                                    #endif
1224
1225                                    //Response Phase
1226                                    SendContainer(CommandResponseCode
                                        ,0,SendParameter,
                                        GetTransactionIDRead(),GetModus
                                        ());
1227                                    break;
1228                            case COMMAND_PTP_FOKUS:
1229                                    //1. Byte und 2. Byte ist der neue
                                        Wert
1230
1231                                    #ifdef DEBUG
1232                                            uart_puts_1("
                                                COMMAND_PTP_FOKUS\r\n");
1233                                    #endif
1234
1235                                    PORTB &= ~(0x08);
1236
1237                                    //Befehl über SPI senden
1238                                    if(SendContainer(
                                        CommandResponseCode,countParam,
                                        arrayParam,GetTransactionIDRead
                                        (),0) != 0)
1239                                    {
1240                                            //Fehler
1241                                            CommandResponseCode = 0
                                                xFF000000 |
```

```
                                        RESPONSE_ERROR;
1242                                    #ifdef DEBUG
1243                                            uart_puts_1("ERROR SPI Transmission \r\n");
1244                                    #endif
1245                                    PORTB |= (0x08);
1246                            }
1247                            else
1248                            {
1249                                    PORTB |= (0x08);
1250                                    //Antwort lesen
1251                                    arrayParamSPI= ReadContainerSPI(& CodeReturnSPI,& countParamSPI);
1252
1253                                    #ifdef DEBUG
1254                                            uart_puts_1("Antwort SPI gelesen\r\n");
1255                                    #endif
1256
1257                                    //Es sollte kein Parameter zurückübergeben werden
1258                                    if(arrayParamSPI!=0)
1259                                    {
1260                                            free(arrayParamSPI);
1261                                            arrayParamSPI = 0;
1262                                    }
1263
1264                                    CommandResponseCode = CodeReturnSPI;
1265                            }
1266
1267                            #ifdef DEBUG
1268                                    uart_puts_1("Response\r\n");
1269                            #endif
1270
1271                            //Response Phase
1272                            SendContainer(CommandResponseCode,0,SendParameter, GetTransactionIDRead(),GetModus());
1273                            break;
1274                    case COMMAND_PTP_GETCONNECTION:
1275                            //Return Data Phase 1 Byte mit Ergebnis
1276                            //Response ok
1277                            #ifdef DEBUG
1278                                    uart_puts_1("COMMAND_PTP_GETCONNECTION \r\n");
1279                            #endif
1280
1281
1282                            PORTB &= ~(0x08);
```

```
1283
1284                                    //Befehl über SPI senden
1285                                    if(SendContainer(
                                            CommandResponseCode,countParam,
                                            arrayParam,GetTransactionIDRead
                                            (),0) != 0)
1286                                    {
1287                                            //Fehler
1288                                            CommandResponseCode = 0
                                                    xFF000000 |
                                                    RESPONSE_ERROR;
1289                                            #ifdef DEBUG
1290                                                    uart_puts_1("ERROR␣
                                                            SPI␣Transmission
                                                            \r\n");
1291                                            #endif
1292                                            PORTB |= (0x08);
1293                                    }
1294                                    else
1295                                    {
1296                                            PORTB |= (0x08);
1297                                            //Dataphase
1298                                            //Antwort lesen
1299                                            arrayParamSPI =
                                                    ReadContainerSPI(&
                                                    CodeReturnSPI,&
                                                    countParamSPI);
1300
1301                                            #ifdef DEBUG
1302                                                    uart_puts_1("
                                                            Antwort␣SPI␣
                                                            gelesen\r\n");
1303                                            #endif
1304
1305                                            //Es sollte ein Parameter
                                                    zurückübergeben werden
1306                                            if(countParamSPI!=1)
1307                                            {
1308                                                    free(arrayParamSPI)
                                                            ;
1309                                                    arrayParamSPI = 0;
1310                                                    //Fehler
1311                                                    #ifdef DEBUG
1312                                                            uart_puts_1
                                                                    ("Fehler
                                                                    ␣kein␣
                                                                    Parameter
                                                                    \r\n");
1313                                                    #endif
1314
1315                                                    CommandResponseCode
                                                            = CodeReturnSPI
                                                            ;
1316                                            }
1317                                            else
1318                                            {
1319                                                    //Parameter
                                                            zurücksenden
1320                                                    //Daten Phase
```

```
1321                                    CommandResponseCode
                                            = 0x00000000 |
                                        RESPONSE_DATA;
1322                                    if(SendContainer(
                                            CommandResponseCode
                                            ,1,arrayParamSPI
                                            ,
                                            GetTransactionIDRead
                                            (),GetModus())
                                            != 0)
1323                                    {
1324                                        //Fehler
1325                                        CommandResponseCode
                                                = 0
                                            xFF000000
                                            |
                                            RESPONSE_ERROR
                                            ;
1326                                        #ifdef
                                            DEBUG
1327                                            uart_puts_1
                                                (
                                                "
                                                Response
                                                ␣
                                                ERROR
                                                \
                                                r
                                                \
                                                n
                                                "
                                                )
                                                ;

1328                                        #endif
1329                                    }
1330                                    else
1331                                    {
1332                                        //Antwort
                                            lesen
1333                                        arrayParamSPI
                                            =
                                            ReadContainerSPI
                                            (&
                                            CodeReturnSPI
                                            ,&
                                            countParamSPI
                                            );
1334
1335                                        #ifdef
                                            DEBUG
1336                                            uart_puts_1
                                                (
                                                "
                                                Antwort
                                                ␣
                                                SPI
                                                ␣
                                                gelesen
```

```
                                                                                                \
                                                                                                r
                                                                                                \
                                                                                                n
                                                                                                "
                                                                                                )
                                                                                                ;
1337                                                                                    #endif
1338
1339                                                                                    //Es sollte kein Parameter zurückübergeben werden
1340                                                                                    if(arrayParamSPI!=0)
1341                                                                                    {
1342                                                                                            free(arrayParamSPI);
1343                                                                                            arrayParamSPI = 0;
1344                                                                                    }
1345
1346                                                                                    CommandResponseCode = CodeReturnSPI;
1347                                                                                            }
1348                                                                                    }
1349                                                                            }
1350
1351                                                                            #ifdef DEBUG
1352                                                                                    uart_puts_1("Response\r\n");
1353                                                                            #endif
1354
1355                                                                            //Response Phase
1356                                                                            SendContainer(CommandResponseCode,0,SendParameter,GetTransactionIDRead(),GetModus());
1357                                                                            break;
1358                                                                    default:
1359                                                                            return;
1360                                                            }
1361                                                    }
1362                                            }
1363
```

```
1364 }
1365
1366 //CheckCommand
1367 //Funktion überprüft den Command Code ob er unterstützt wird und ob die
        Parameter stimmen
1368 //Muss für jedes Device angepasst werden
1369 // @Param1: Command Code
1370 // @Param2: Anzahl der Parameter
1371 //Return Command.h Definitionen
1372 unsigned short CheckCommand(uint32_t CommandResponseCode, unsigned short
        countParam)
1373 {
1374         char DeviceID;
1375         char Klasse;
1376         char ADKlasse;
1377         char Code;
1378
1379         //DeviceID
1380         DeviceID = (char) (CommandResponseCode>>24 & 0xFF);
1381
1382         if(DeviceID == 0 || DeviceID == 0xFF)
1383         {
1384                 //ResponseCode oder Dataphase wird aktuell nicht
                        unterstützt
1385                 return RESPONSE_NOT_SUPPORTED_PHASE;
1386         }
1387         else
1388         {
1389                 //Klasse herausfinden
1390                 Klasse = (char) (CommandResponseCode>>16 & 0xFF);
1391
1392                 if(Klasse != 1 && Klasse != 2 && Klasse != 5)
1393                 {
1394                         //Andere Klassen werden aktuell nicht unterstützt
1395                         return RESPONSE_NOT_SUPPORTED_CLASS;
1396                 }
1397
1398                 ADKlasse = (char) (CommandResponseCode>>8 & 0xFF);
1399                 Code = (char) (CommandResponseCode & 0xFF);
1400
1401
1402                 //Pin Befehle
1403                 //Klasse I
1404                 //-----------------------------------------------
1405                 if(Klasse == 1)
1406                 {
1407                         //Nur 7 Port Pins
1408                         if(ADKlasse > 7)
1409                         {
1410                                 return RESPONSE_NOT_SUPPORTED_ADCLASS;
1411                         }
1412
1413                         //Code überprüfen
1414                         switch(Code)
1415                                 {
1416                                         case COMMAND_PIN_STORE:
1417                                         case COMMAND_PIN_STATE:
1418                                         case COMMAND_GETCLASS:
1419                                                 if(countParam!=0)
```

```
1420                                                    {
1421                                                            //Anzahl stimmt
                                                                nicht
1422                                                            return
                                                                RESPONSE_NOT_SUPPORTED_PARAME
                                                                ;
1423                                                    }
1424                                                    break;
1425                                            case COMMAND_PIN_READ:
1426                                                    if(countParam!=1)
1427                                                    {
1428                                                            //Anzahl stimmt
                                                                nicht
1429                                                            return
                                                                RESPONSE_NOT_SUPPORTED_PARAME
                                                                ;
1430                                                    }
1431                                                    break;
1432                                            case COMMAND_SET_OUTPUT:
1433                                                    if(countParam!=2)
1434                                                    {
1435                                                            //Anzahl stimmt
                                                                nicht
1436                                                            return
                                                                RESPONSE_NOT_SUPPORTED_PARAME
                                                                ;
1437                                                    }
1438                                                    break;
1439                                            case COMMAND_PIN_SET:
1440                                                    if(countParam!=10)
1441                                                    {
1442                                                            //Anzahl stimmt
                                                                nicht
1443                                                            return
                                                                RESPONSE_NOT_SUPPORTED_PARAME
                                                                ;
1444                                                    }
1445                                                    break;
1446                                            default:
1447                                                    return
                                                        RESPONSE_NOT_SUPPORTED;
1448                                    }
1449
1450                    }
1451                    //-----------------------------------------
1452
1453                    //TWI Befehle
1454                    //-----------------------------------------
1455                    if(Klasse == 2)
1456                    {
1457                            //Nur 120 Adressen möglich
1458                            if(ADKlasse > 120)
1459                            {
1460                                    return RESPONSE_NOT_SUPPORTED_ADCLASS;
1461                            }
1462
1463                            //Code überprüfen
1464                            switch(Code)
1465                                    {
```

```
1466                                    case COMMAND_READ_TEMP:
1467                                            if(countParam!=0)
1468                                            {
1469                                                    //Anzahl stimmt
                                                            nicht
1470                                                    return
                                                            RESPONSE_NOT_SUPPORTED_PARAME
                                                            ;
1471                                            }
1472                                            break;
1473                                    default:
1474                                            return
                                                    RESPONSE_NOT_SUPPORTED;
1475                            }
1476
1477                    }
1478                    //-----------------------------------------------
1479
1480
1481                    //Klasse 5 Befehle
1482                    //-----------------------------------------------
1483                    if(Klasse == 5)
1484                    {
1485                            if(GetKlasse() < 2)
1486                            {
1487                                    //Modul kann die Befehle nicht
1488                                    return RESPONSE_NOT_SUPPORTED_CLASS;
1489                            }
1490
1491                            //Nur USB Host 1 und 0
1492                            if(ADKlasse > 1)
1493                            {
1494                                    return RESPONSE_NOT_SUPPORTED_ADCLASS;
1495                            }
1496
1497                            //Code überprüfen
1498                            switch(Code)
1499                                    {
1500                                            case COMMAND_PTP_SESSION_OPEN:
1501                                            case COMMAND_PTP_SESSION_CLOSE:
1502                                            case COMMAND_PTP_SHOOT:
1503                                            case COMMAND_PTP_GETCONNECTION:
1504                                            case COMMAND_PTP_LIVE_VIEW_ON:
1505                                            case COMMAND_PTP_LIVE_VIEW_OFF:
1506                                                    if(countParam!=0)
1507                                                    {
1508                                                            //Anzahl stimmt
                                                                    nicht
1509                                                            return
                                                                    RESPONSE_NOT_SUPPORTED_PARAME
                                                                    ;
1510                                                    }
1511                                                    break;
1512                                            case COMMAND_PTP_CHANGE_TV:
1513                                            case COMMAND_PTP_CHANGE_AV:
1514                                            case COMMAND_PTP_CHANGE_BELICHTUNG:
1515                                                    if(countParam!=1)
1516                                                    {
```

```
1517                                                    //Anzahl stimmt
                                                            nicht
1518                                                    return
                                                            RESPONSE_NOT_SUPPORTED_PARAME
                                                            ;
1519                                            }
1520                                            break;
1521                                    case COMMAND_PTP_FOKUS:
1522                                            if(countParam!=2)
1523                                            {
1524                                                    //Anzahl stimmt
                                                            nicht
1525                                                    return
                                                            RESPONSE_NOT_SUPPORTED_PARAME
                                                            ;
1526                                            }
1527                                            break;
1528                                    default:
1529                                            return
                                                    RESPONSE_NOT_SUPPORTED;
1530                                    }
1531                            }
1532                    }
1533                    return RESPONSE_OK;
1534    }
```

RS232_ATmega324a.h

```
1  /*
2   * RS232_ATmega324a.h
3   *
4   * Created: 24.01.2012 19:03:29
5   *  Author: Niki
6   */
7
8
9  #ifndef RS232_ATMEGA324A_H_
10 #define RS232_ATMEGA324A_H_
11
12 #include "SPI_Master.h"
13 #include <avr/io.h>
14 #include <util/delay.h>
15 #include <avr/interrupt.h>
16 #include <string.h>
17 #include <stdio.h>
18 #include <stdlib.h>
19 #include "uart.h"
20 #include "BTM_222.h"
21 #include "Protokoll.h"
22 #include "Commands.h"
23 #include "EEPROM_PIN.h"
24 #include "DS1621.h"
25 #include "TWI.h"
26
27 //Get Pin
28 //Function return the Pin from RAM
29 //@Param: Parameter of the Pin
30 struct Pin_A * GetPin(char Number);
31
```

```
32 //SetOutput Pin
33 //Function ändert DDRA entprechend der Einstellung
34 //@Param: Parameter of the Pin
35 void SetOutputPin(char Number);
36
37 //Save Pin
38 //Funktion speichert die den neuen Pin
39 //@Param1: Neue Werte vom Pin
40 //@Param2: Welcher Pin geändert werden soll
41 void SavePin(struct Pin_A * Pin, char Number);
42
43
44 //UpdateInterruptRegister
45 //Funktion setzt PCMSK0 richtig
46 void UpdateInterruptRegister();
47
48 //Funktion updatet die Hilfvariablen für die PWM Implementation pro Pin
49 //@Param1: Neuer TIME A Wert
50 //@Param2: Welchen Pin es betrifft
51 void UpdateTime(uint32_t number,char pin);
52
53 #endif /* RS232_ATMEGA324A_H_ */
```

RS232_ATmega324a.c

```
1  /*
2   * RS232_ATmega324a.c
3   *
4   * Created: 13.11.2011 13:06:23
5   *  Author: Niki
6   */
7
8  #ifndef DEBUG
9  #define DEBUG 1
10 #endif
11
12 #ifndef F_CPU
13 #define F_CPU 12000000UL
14 #endif
15
16 #define UART_BAUD_RATE 19200L
17 #define UART_BAUD_CALC(UART_BAUD_RATE,F_CPU) ((F_CPU)/((UART_BAUD_RATE)*16L
       )-1)
18
19 #ifndef BLUETOOTH_STATE_DEFINITION
20 #define BLUETOOTH_STATE_DEFINITION              1
21
22 #define BLUETOOTH_STATE_ERROR                   0
23 #define BLUETOOTH_STATE_AVAILABLE               1
24 #define BLUETOOTH_STATE_NOT_AVAILABLE   2
25
26 #define BLUETOOTH_CONNECTION_DISCONNECTED       0
27 #define BLUETOOTH_CONNECTION_CONNECTED          1
28
29 #define CONNECTSTRINGLENGTH             9
30 #define DISCONNECTSTRINGLENGTH  12
31 #endif
32
33 #ifndef DS1621
```

```
34 #define DS1621 1
35 #endif
36
37 #include "RS232_ATmega324a.h"
38
39 uint16_t volatile TickTack;
40 #ifdef DEBUG
41 char CheckTickTack =0;
42 char CheckDS1621 =0;
43 #endif
44
45 char volatile PinChangeState;
46 char volatile PinChange;
47 uint16_t volatile CurrentTemp;
48
49 struct Pin_A PinsPortA[8];
50
51 //Hilfvariablen für die PWM Implementation pro Pin
52 uint16_t volatile PinTimerChange[8];
53 char volatile PinTimerState;
54
55 #ifdef DEBUG
56 void Debug_ActualBluetoothConnection()
57 {
58                 char x;
59                 char i;
60                 char result[30];
61                 x = GetBluetoothConnection();
62
63                 switch(x)
64                 {
65                         case BLUETOOTH_CONNECTION_DISCONNECTED:
66                                 uart_puts_1("BT Disconnected!");
67                                 break;
68                         case BLUETOOTH_CONNECTION_CONNECTED:
69                                 uart_puts_1("BT Connected!");
70                                 i = 0;
71                                 //Read out until Buffer is Empty
72                                 while(Getrbuffcnt_0() != 0)
73                                 {
74                                         x = ser_getc_0();
75                                         result[i] = x;
76                                         i++;
77                                 }
78
79                                 break;
80                 }
81
82                 uart_putc_1 (0x0A);
83                 uart_putc_1 (0x0D);
84 }
85
86 void Debug_ActualBluetoothState()
87 {
88                 char x;
89                 x = GetBluetoothState();
90
91                 switch(x)
92                 {
```

```
93                          case BLUETOOTH_STATE_ERROR:
94                                  uart_puts_1("BT Error!");
95                                  break;
96                          case BLUETOOTH_STATE_AVAILABLE:
97                                  uart_puts_1("BT available!");
98                                  break;
99                          case BLUETOOTH_STATE_NOT_AVAILABLE:
100                                 uart_puts_1("BT not available!");
101                                 break;
102                 }
103
104                 uart_putc_1 (0x0A);
105                 uart_putc_1 (0x0D);
106 }
107 #endif
108
109 //Funktion updatet die Hilfvariablen für die PWM Implementation pro Pin
110 //@Param1: Neuer TIME A Wert
111 //@Param2: Welchen Pin es betrifft
112 void UpdateTime(uint32_t number,char pin)
113 {
114
115         #ifdef DEBUG
116                 char buff[100];
117                 sprintf(buff,"Update Timer: %d - %d\n\r",pin,number);
118                 uart_puts_1(buff);
119         #endif
120
121         PinTimerChange[pin]= number;
122
123         //Jetzt beginnt die 1. Hälfte
124         PinTimerState |= (1<<pin);
125 }
126
127 // Interruptroutine, die alle 0,5 Sekunden aufgerufen wird
128 ISR (TIMER1_COMPA_vect)
129 {
130         char x;
131         TickTack++;
132         char buf[100];
133
134
135         //Zähler wieder auf 0 stellen
136         TCNT1L=0;
137         TCNT1H=0;
138
139         //Kontrollieren ob der Pin im PWM Modus ist
140         for(x=0;x<8;x++)
141         {
142                 //Pulsweitenmodulation
143                 if((PinsPortA[x].Settings & 0x3F)==3)
144                 {
145                         //PWM wird am Pin durchgeführt
146                         //Timer verringern
147                         PinTimerChange[x]--;
148
149                         #ifdef DEBUG
150                                 sprintf(buf,"Period --: Pin %d New Value: %d
                                        TimeA: %d TimeB: %d\n\r",x,
```

```
                        PinTimerChange[x],((PinsPortA[x].Time
                        >>16)&0x3FFF),(PinsPortA[x].Time&0x3FFF)
                        );
151                     uart_puts_1(buf);
152             #endif
153
154             if(PinTimerChange[x] == 0 || PinTimerChange[x] ==
                    UINT16_MAX)
155             {
156
157                     #ifdef DEBUG
158                             uart_puts_1("PeriodChange2:␣");
159                             uart_putc_1('A'+x);
160                             uart_puts_1("\n\r");
161                     #endif
162
163                     //Nächste Periode ist dran
164                     if(0x01 & (PinTimerState>>x) == 0x01)
165                     {
166                             //Jetzt beginnt die 2. Hälfte
167                             PinTimerState &= ~(1<<x);
168
169                             //Neue Zeit eintragen
170                             PinTimerChange[x] = PinsPortA[x].
                                    Time & 0x7FFF;
171
172                             //Ausgang setzen
173                             if(((PinsPortA[x].Time >> 15) & 0
                                    x01) == 0x01)
174                             {
175                                     PORTA |= (1<<x);
176                             }
177                             else
178                             {
179                                     PORTA &= ~(1<<x);
180                             }
181
182                     }
183                     else
184                     {
185
186                             #ifdef DEBUG
187                                     uart_puts_1("PeriodChange1:
                                            ␣");
188                                     uart_putc_1('A'+x);
189                                     uart_puts_1("\n\r");
190                             #endif
191
192                             //Jetzt beginnt die 1. Hälfte
193                             PinTimerState |= (1<<x);
194                             //Neue Zeit eintragen
195                             PinTimerChange[x] = (PinsPortA[x].
                                    Time >> 16) & 0x7FFF;
196
197                             //Ausgang setzen
198                             if(((PinsPortA[x].Time >> 31) & 0
                                    x01) == 0x01)
199                             {
200                                     PORTA |= (1<<x);
```

```
201                                             }
202                                             else
203                                             {
204                                                     PORTA &= ~(1<<x);
205                                             }
206                                     }
207                             }
208                     }
209                     else if(!(PinsPortA[x].Settings & 0x80) && (((PinsPortA[x].
                            Settings & 0x3F)==2)|| ((PinsPortA[x].Settings & 0x3F)
                            ==1))) //Pin = Eingang und Modus = Interrupt oder Aktiv
210                     {
211                             //Timer wenn kleiner als TimeA-1 und > 0,
                                    verringern
212                             if(((PinsPortA[x].Time & 0x7FFF) != ((PinsPortA[x].
                                Time >> 16) & 0x7FFF)) && ((PinsPortA[x].Time &
                                0x7FFF) > 0))
213                             {
214                                     PinsPortA[x].Time = (PinsPortA[x].Time & 0
                                        xFFFF8000) | ((PinsPortA[x].Time & 0
                                        x7FFF)-1);
215
216                                     #ifdef DEBUG
217                                             uart_puts_1("Zeit--:␣");
218                                             uart_putc_1('A'+x);
219                                             uart_puts_1("\n\r");
220                                     #endif
221                             }
222                     }
223             }
224 }
225
226 //Interruptroutine für Eingänge
227 ISR(PCINT0_vect)
228 {
229         char x,i;
230         char PinState;
231         char PinAffected;
232
233         #ifdef DEBUG
234                 char buf[100];
235                 uart_puts_1("ISR\r\n");
236         #endif
237
238         //Herausfinden welche Pin sich geändert hat
239         PinChange = PCMSK0 & (PinChangeState ^ (PINA & PCMSK0));
240         //Speichern der aktuellen Pin Zustände
241         PinChangeState = (PINA & PCMSK0);
242
243         //Kontrollieren ob der Pin ausgeführt werden soll
244         for(x=0;x<8;x++)
245         {
246                 //Input & Modus = Interrupt & Signalflanke richtig &
                        betroffener Pin
247                 if(!(PinsPortA[x].Settings & 0x80) && ((PinsPortA[x].
                        Settings & 0x3F)==2) && ( (0x01 & (PinChangeState >> x))
                         == (0x01 & PinsPortA[x].Settings>>6)) && (0x01 & (
                        PinChange>>x)))
248                 {
```

```
249                             //Aktiv Pin Ändern
250                             PinAffected = (PinsPortA[x].Reaction>>16) & 0xFF;
251                             PinState = PinsPortA[x].Reaction & 0xFF;
252
253                             #ifdef DEBUG
254                                     sprintf(buf,"Pin: %d PinAff: %x PinS: %x \n
                                            \r",x,PinAffected,PinState);
255                                     uart_puts_1(buf);
256                             #endif
257
258                             for(i=0;i<8;i++)
259                             {
260                                     if(((PinAffected>>i)&0x01) == 0x01)
261                                     {
262                                             //Pin setzen
263                                             if( ((PinState >> i) & 0x01 )== 0
                                                    x01)
264                                             {
265                                                     PORTA |= (1<<i);
266                                             }
267                                             else
268                                             {
269                                                     PORTA &= ~(1<<i);
270                                             }
271
272                                     }
273                             }
274
275
276                             //Ist eine Zeit eingetragen?
277                             //Wenn nicht, Zustand nur einmal ausgelöst
278                             if(((PinsPortA[x].Time>>16) & 0x7FFF) != 0)
279                             {
280                                     //in Time B Zeit speichern und herunter
                                            zählen im Ticker
281                                     PinsPortA[x].Time = (PinsPortA[x].Time & 0
                                            xFFFF8000) | (((PinsPortA[x].Time>>16) &
                                             0x7FFF)-1);
282                             }
283                     }
284             }
285     }
286
287     //void InitPin_A(struct Pin_A *PinA)
288     //{
289             //PinA->Reaction=0;
290             //PinA->Settings=0;
291             //PinA->Time = 0;
292     //}
293
294     //Funktion gibt die aktuelle Temepartur zurück
295     uint16_t ActualTemp(void)
296     {
297             return CurrentTemp;
298     }
299
300     //Funktion setzt die aktuelle Temperatur
301     void SetTemp(uint16_t Temp)
302     {
```

```
303         CurrentTemp = Temp;
304 }
305
306 //Funktion initialisiert die Pins und den Timer 1
307 void Init(void)
308 {
309         char x;
310
311         //Temperatur setzen
312         SetTemp(0);
313
314         //Pinstruktur + Hilfvariable initialisieren
315         PinTimerState = 0;
316         for(x=0;x<8;x++)
317         {
318                 //InitPin_A(&PinsPortA[x]);
319                 ReadEEPROMPin(&PinsPortA[x],x);
320
321                 //Wenn die Settings 0 sind, setzen wir es lieber auf Output
322                 if(PinsPortA[x].Settings==0)
323                 {
324                         PinsPortA[x].Settings= 0x80;
325                 }
326
327                 //Pulsweitenmodulation
328                 if((PinsPortA[x].Settings & 0x3F)==3)
329                 {
330                         //Mit der zweiten Hälfte beginnen
331                         PinTimerChange[x] = PinsPortA[x].Time & 0x7FFF;
332                 }
333                 else
334                 {
335                         PinTimerChange[x] = 0;
336                 }
337                 SetOutputPin(x);
338         }
339
340
341         //Klasse III FixEinstellungen hier vornehmen
342         //Das Einlesen über die serielle Schnittstelle
343         //ist noch nicht entwickelt
344         //-------------------------------------------------------
345
346         //-------------------------------------------------------
347
348         //Eingänge richtig setzen
349         //PORTD Ausgänge, werden automatisch vom USART gesetzt
350         DDRD = 0xFF;
351
352         //PORTC Eingänge für Modus sonst Ausgänge -> XXX TWI kontrollieren
353         DDRC = 0xE7;
354         PORTC = 0xFF;
355
356         //PORTB alles Ausgänge SPI_Master_Init setzt es richtig
357         DDRB = 0xFF;
358
359
360         //Timer einstellen
361         //Compare einstellen
```

```
362         TickTack = 0;
363         OCR1AH = 0x16;
364         OCR1AL = 0xE3;
365         TCCR1B = 0x05;
366         TIMSK1 = 0x02;
367
368         //Externe Interrupts PCINT7:0 aktivieren
369         PCICR = 0x01;
370
371         //Interrupts aktivieren
372         UpdateInterruptRegister();
373
374         //aktuellen Pegel an den Interrupts einlesen
375         PinChange = 0;
376         PinChangeState = (PINA & PCMSK0);
377
378         //JTAG deaktivieren
379         MCUCR |= (1<<JTD);
380         MCUCR |= (1<<JTD);
381
382         //TWI aktivieren
383         Init_TWI();
384
385         sei();  // Interruptbehandlung aktivieren
386 }
387
388 //Funktion überprüft als welche Klasse das Gerät verwendet wird
389 void CheckKlasse(void)
390 {
391         char Klasse;
392
393         Klasse=0;
394
395         //Eingangsstatus einlesen
396         if(PINC & (1<<PINC3))
397         {
398                 Klasse |= 1;
399         }
400
401         //Oder verknüpfen
402         if (PINC & (1<<PINC4))
403         {
404                 Klasse |= 2;
405         }
406
407         SetKlasse(Klasse);
408 }
409
410 //Block: TWI Status abfragen
411 void TWIAbfragen(void)
412 {
413         //#ifdef DEBUG
414                 //uart_puts_1("TWI abfragen\n\r");
415                 //ser_getc_1();
416         //#endif
417
418         uint16_t Temp;
419         #ifdef DEBUG
420         char Buff[100];
```

```
421         #endif
422
423         #ifdef DS1621
424                 if(TickTack%20 == CheckDS1621)
425                 {
426
427                         if(CheckDS1621==10)
428                         {
429                                 CheckDS1621=1;
430                         }
431
432                         //Start der Temperatur Umwandlung
433                         if(CheckDS1621==0)
434                         {
435                                 #ifdef DEBUG
436                                         uart_puts_1("Start Umwandlung\n\r")
                                            ;
437                                 #endif
438                                 StartConvert(5);
439
440                                 CheckDS1621=10;
441                         }
442
443                         //Temperatur auslesen
444                         if(CheckDS1621==1)
445                         {
446
447                                 Temp = ReadTemp(5);
448                                 SetTemp(Temp);
449
450                                 #ifdef DEBUG
451                                         sprintf(Buff,"Temperatur: %d,%d C\r
                                            \n",(ActualTemp()>>8),((
                                            ActualTemp() & 0xFF) == 0x80 ?
                                            5:0));
452                                         uart_puts_1(Buff);
453                                 #endif
454
455                                 CheckDS1621=0;
456                         }
457                 }
458         #endif
459 }
460
461 //Block: Pins im Modus Aktiv abfragen
462 void PinsAbfragen(void)
463 {
464         char x, i;
465         char PinState;
466         char PinAffected;
467
468
469         for(x=0;x<8;x++)
470         {
471                 //Input & Modus = Aktiv   & Signalflanke richtig & noch
                        nicht aktiviert
472                 if(     !(PinsPortA[x].Settings & 0x80)
473                         && ((PinsPortA[x].Settings & 0x3F)==1)
```

```
474                     && ( (0x01 & (PINA >> x)) == (0x01 & (PinsPortA[x].
                            Settings>>6)))
475                     && ((PinsPortA[x].Time & 0x7FFF) == ((PinsPortA[x].
                            Time >> 16) & 0x7FFF) )
476             )
477             {
478                     //Aktiv Pin Ändern
479                     PinAffected = (PinsPortA[x].Reaction>>16) & 0xFF;
480                     PinState = PinsPortA[x].Reaction & 0xFF;
481
482                     for(i=0;i<8;i++)
483                     {
484                             if(((PinAffected>>i)&0x01) == 0x01)
485                             {
486                                     //Pin setzen
487                                     if( ((PinState >> i) & 0x01 )== 0
                                            x01)
488                                     {
489                                             PORTA |= (1<<i);
490                                     }
491                                     else
492                                     {
493                                             PORTA &= ~(1<<i);
494                                     }
495
496                             }
497                     }
498
499                     #ifdef DEBUG
500                             uart_puts_1("Pin Aktiv ausgelöst: ");
501                             uart_putc_1('A'+x);
502                             uart_puts_1("\n\r");
503                     #endif
504
505                     //Ist eine Zeit eingetragen?
506                     //Wenn nicht, Zustand nur einmal ausgelöst
507                     if(((PinsPortA[x].Time>>16) & 0x7FFF) != 0)
508                     {
509                             //in Time B Zeit speichern und herunter
                                    zählen im Ticker
510                             PinsPortA[x].Time = (PinsPortA[x].Time & 0
                                    xFFFF8000) | (((PinsPortA[x].Time>>16) &
                                     0x7FFF)-1);
511                     }
512
513             }
514             else if(!(PinsPortA[x].Settings & 0x80) && (((PinsPortA[x].
                    Settings & 0x3F)==2) || ((PinsPortA[x].Settings & 0x3F)
                    ==1)))//Input & Modus = Interrupt oder Aktiv - Bedingung
                     nicht mehr erfüllt
515             {
516                     //Wenn sie gleich sind, ist die Bedingung nicht
                            aktiviert worden
517                     //Wenn sie 0 ist, wurde sie schon aktiviert und ist
                             jetzt zu ende
518                     //Aufpassen in der ISR, bei 0 nicht noch einmal
                            verringern
519                     if ( ((PinsPortA[x].Time & 0x7FFF) != ((PinsPortA[x
                            ].Time >> 16) & 0x7FFF) )&&((PinsPortA[x].Time &
```

```
                                0x7FFF) == 0))
520                         {
521                                 //Pins wieder umkehren
522                                 //Aktiv Pin Ändern
523                                 PinAffected = (PinsPortA[x].Reaction>>16) &
                                        0xFF;
524                                 //Genau umgekehrt ändern
525                                 PinState = ~(PinsPortA[x].Reaction & 0xFF);
526
527                                 for(i=0;i<8;i++)
528                                 {
529                                         if(((PinAffected>>i)&0x01) == 0x01)
530                                         {
531                                                 //Pin setzen
532                                                 if( ((PinState >> i) & 0x01
                                                        )== 0x01)
533                                                 {
534                                                         PORTA |= (1<<i);
535                                                 }
536                                                 else
537                                                 {
538                                                         PORTA &= ~(1<<i);
539                                                 }
540
541                                         }
542                                 }
543
544                                 //Zeit wieder setzen
545                                 PinsPortA[x].Time = (PinsPortA[x].Time & 0
                                        xFFFF8000) | (((PinsPortA[x].Time>>16) &
                                         0x7FFF));
546
547                                 #ifdef DEBUG
548                                         uart_puts_1("Pin Aktiv/Interrupt
                                                Zeit abgelaufen: ");
549                                         uart_putc_1('A'+x);
550                                         uart_puts_1("\n\r");
551                                 #endif
552                         }
553                 }
554
555         }
556
557 }
558
559
560 int main(void)
561 {
562         char i,x,j;
563         uint16_t delayChange; //Variable, damit genug Zeit vergeht, bist
                die BT Adresse übertragen wurde
564         char *param;
565         uint32_t CodeWort;
566         unsigned short countParam;
567         char *arrayParam;
568
569         #ifdef DEBUG
570                 char buff[100];
571         #endif
```

```
572
573         Init();
574
575         i=0;
576         j = 0;
577         x = 0;
578         delayChange=0;
579         uart_ini_1();
580
581         //Klasse kontrollieren
582         CheckKlasse();
583
584         #ifdef DS1621
585                 InitDS1621(5);
586         #endif
587
588         //Nur wenn Klasse V
589         //SPI Master Init
590         if(GetKlasse() > 1)
591         {
592                 SPI_MasterInit();
593                 PORTB |= (0x08);
594         }
595
596         ///Wenn Bluetooth - Rest und Pin Struktur richtig stellen
597         if (GetKlasse() == 1 || GetKlasse() == 3 )
598         {
599                 PinsPortA[7].Settings = 0x80;//1000 0000
600                 PinsPortA[7].Reaction = 0x00;
601                 PinsPortA[7].Time = 0x00;
602                 SetOutputPin(7);
603
604                 Reset_BTM();
605         }
606
607         uart_ini_0();
608
609         #ifdef DEBUG
610     uart_puts_1("Ready V13\n\r");
611         //ser_getc_1();
612         uart_puts_1("Run V13\n\r");
613
614         switch(GetKlasse())
615         {
616                 case 0:
617                         uart_puts_1("Klasse 4 E\n\r");
618                         break;
619                 case 1:
620                         uart_puts_1("Klasse 4 B\n\r");
621                         break;
622                 case 2:
623                         uart_puts_1("Klasse 5 E\n\r");
624                         break;
625                 case 3:
626                         uart_puts_1("Klasse 5 B\n\r");
627                         break;
628         }
629         #endif
630
```

```
631         while(1)
632     {
633                 if(TickTack%20 == CheckTickTack)
634                 {
635                         //Damit erkennbar ist, ob das System läuft
636                         #ifdef DEBUG
637                         if(CheckTickTack == 0)
638                         {
639                                 uart_puts_1("Beep\n\r");
640                                 CheckTickTack = 1;
641                         }
642                         else
643                         {
644                                 CheckTickTack = 0;
645                         }
646                         #endif
647                 }
648
649                 //Kontrollieren ob sich die Verbindung beim Bluetooth
                        verändert hat
650                 if(GetConnectionChange() == 1)
651                 {
652                         //Die Übertragung der BT Adresse dauert manchmal
                            länger
653                         if(delayChange<2000)
654                         {
655                                 delayChange++;
656                         }
657                         else
658                         {
659                                 delayChange = 0;
660
661                                 //Solange auslesen bis der Buffer leer ist
662                                 while(Getrbuffcnt_0() != 0)
663                                 {
664                                         x = ser_getc_0();
665                                 }
666
667                                 SetConnectionChange(0);
668
669                                 #ifdef DEBUG
670                                         uart_puts_1("Änderung\n\r");
671                                 #endif
672                         }
673                 }
674
675
676                 //Mehr als 7 Byte im Empfangsbuffer?
677                 //Bluetooth: Bluetooth Status muss aktiviert sein, und die
                        Verbindung darf sich nicht geändert haben
678                 //Ethernet: Klasse muss stimmen
679                 if((Getrbuffcnt_0() > 7 && (GetKlasse() == 0 || GetKlasse()
                        == 2)) || (Getrbuffcnt_0()>7 && GetBluetoothConnection
                        () == BLUETOOTH_CONNECTION_CONNECTED &&
                        GetConnectionChange() == 0 ))
680                 {
681                         //Datenpaket vorhanden
682
683                         #ifdef DEBUG
```

```
684                     uart_puts_1("Datenpaket vorhanden\n\r");
685             #endif
686             //Die Änderung kann zwischen der Ausgabe auftreten,
                    deswegen wird es noch einmal abgefragt
687             if((Getrbuffcnt_0() > 7 && (GetKlasse() == 0 ||
                GetKlasse() == 2)) || (Getrbuffcnt_0()>7 &&
                GetBluetoothConnection() ==
                BLUETOOTH_CONNECTION_CONNECTED &&
                GetConnectionChange() == 0 ))
688             {
689                     //ReadContainer
690                     CodeWort= 0;
691                     arrayParam = 0;
692                     countParam = 0;
693
694                     arrayParam = ReadContainerUSART(&CodeWort,&
                            countParam);
695
696                     #ifdef DEBUG
697                             if(countParam>0)
698                                     sprintf(buff , "%i %i %i %i
                                        Par: %i-%i", (char) (
                                        CodeWort>>24 & 0xFF), (
                                        char) (CodeWort>>16 & 0
                                        xFF), (char) (CodeWort
                                        >>8 & 0xFF), (char) (
                                        CodeWort & 0xFF),
                                        countParam,arrayParam
                                        [0]);
699                             else
700                                     sprintf(buff , "%i %i %i %i
                                        Par: %i", (char) (
                                        CodeWort>>24 & 0xFF), (
                                        char) (CodeWort>>16 & 0
                                        xFF), (char) (CodeWort
                                        >>8 & 0xFF), (char) (
                                        CodeWort & 0xFF),
                                        countParam);
701
702                             uart_puts_1(buff);
703                     #endif
704
705                     //Kontrollieren ob ein Fehler aufgetreten
                            ist
706                     if(countParam > 0 && arrayParam == 0)
707                     {
708                             //Fehler zurücksenden
709                             #ifdef DEBUG
710                                     uart_puts_1("1: Fehler
                                        Parameter\n\r");
711                             #endif
712                     }
713                     else
714                     {
715                             //Logik Einheit aufrufen
716                             LogicUnit(CodeWort,      countParam,
                                    arrayParam);
717                     }
718
```

```
719                                     //Parameter wieder freigeben
720                                     if(arrayParam!=0)
721                                     {
722                                             free(arrayParam);
723                                             arrayParam = 0;
724                                     }
725
726                             }
727
728                     }
729                     else
730                     {
731                             //Nein
732                             //Einzelne Pins Abfragen
733                             //Kontrollieren ob der Pin ausgeführt werden soll
734                             //-----------------------------------------
735                             PinsAbfragen();
736                             //-----------------------------------------
737
738                             //TWI Implementation
739                             //-----------------------------------------
740                             TWIAbfragen();
741                             //-----------------------------------------
742
743
744                     }
745     }
746 }
747
748
749
750 //Get Pin
751 //Function return the Pin from RAM
752 //@Param: Parameter of the Pin
753 struct Pin_A * GetPin(char Number)
754 {
755         return &PinsPortA[Number];
756 }
757
758
759 //SetOutput Pin
760 //Function ändert DDRA entprechend der Einstellung
761 //@Param: Parameter of the Pin
762 void SetOutputPin(char Number)
763 {
764         if((PinsPortA[Number].Settings & 0x80) == 0x80)
765         {
766                 //Output
767                 DDRA |= (1<<Number);
768         }
769         else
770         {
771                 DDRA &= ~(1<<Number);
772         }
773 }
774
775
776 //Save Pin
777 //Funktion speichert die den neuen Pin
```

```
778 //@Param1: Neue Werte vom Pin
779 //@Param2: Welcher Pin geändert werden soll
780 void SavePin(struct Pin_A * Pin, char Number)
781 {
782         PinsPortA[Number].Settings = Pin->Settings;
783         PinsPortA[Number].Reaction = Pin->Reaction;
784         PinsPortA[Number].Time = Pin->Time;
785 }
786
787
788 //UpdateInterruptRegister
789 //Funktion setzt PCMSK0 richtig
790 void UpdateInterruptRegister()
791 {
792         char x;
793
794         for(x=0;x<8;x++)
795         {
796                 if((PinsPortA[x].Settings & 0x3f)==2)
797                 {
798                         PCMSK0 |= (1<<x);
799
800                         #ifdef DEBUG
801                                 uart_puts_1("Pin is Int: ");
802                                 uart_putc_1('A'+x);
803                                 uart_puts_1("\n\r");
804                         #endif
805                 }
806                 else
807                 {
808                         PCMSK0 &= ~(1<<x);
809                 }
810         }
811 }
```

SPI_Master.h

```
1  /*
2   * SPI_Master.h
3   *
4   * Created: 28.09.2011 12:59:39
5   *  Author: Niki
6   */
7
8
9  #ifndef SPI_MASTER_H_
10 #define SPI_MASTER_H_
11
12
13 #include <avr/io.h>
14
15
16 #define DDR_SPI DDRB
17 #define DD_MOSI DDB5
18 #define DD_MISO DDB6
19 #define DD_SCK  DDB7
20 #define DD_SPISS DDB4
21
22 //Chip Select für den Slave (Active Low)
```

```
23 #define DDR_SS DDRB
24 #define DD_SS  DDB3
25
26 //Initialisierung für den Master
27 void SPI_MasterInit(void);
28
29 //Übertragen Daten
30 void SPI_MasterTransmit(char data);
31
32 //Empfange Daten
33 char SPI_MasterReceive(void);
34
35
36 #endif /* SPI_MASTER_H_ */
```

SPI_Master.c

```
1  /*
2   * SPI_Master.c
3   *
4   * Created: 28.09.2011 12:42:14
5   *  Author: Niki
6   */
7
8  #include "SPI_Master.h"
9
10 #define SPCR    _SFR_IO8(0x2C)
11 #define SPIE    7
12 #define SPE             6
13 #define DORD    5
14 #define MSTR    4
15 #define CPOL    3
16 #define CPHA    2
17 #define SPR1    1
18 #define SPR0    0
19
20 #define SPDR    _SFR_IO8(0x2E)
21
22 #define SPSR    _SFR_IO8(0x2D)
23 #define SPIF    7
24 #define WCOL    6
25 #define SPI2X   0
26
27
28 //Initialisiert den Master bei einer SPI Verbindung
29 //Setzt die Portrichtigungen richtig
30 //Tragt die richtigen Werte in das SPI Control Register ein
31 void SPI_MasterInit(void)
32 {
33         DDR_SPI |= (1<<DD_MOSI) | (1<<DD_SCK) | (1<<DD_SPISS);  //Mosi, SCK
               und SS vom SPI als Ausgang schalten, damit es funktioniert
34         DDR_SPI &= ~(1<<DD_MISO); //Miso als Eingang
35
36         DDR_SS |= (1<<DD_SS); //SS für den Slave als Ausgang schalten
37
38         SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR1); //SPI Enablen, Master, 1/64
               Clk
39 }
40
```

```
41 //Übertragen Daten
42 void SPI_MasterTransmit(char data)
43 {
44         SPDR = data;                                    // Übertragung
               starten
45         while(!(SPSR & (1<<SPIF)));                     // Warten bis fertig
46 }
47
48 //Empfange Daten
49 char SPI_MasterReceive(void)
50 {
51         while(!(SPSR & (1<<SPIF)));                     // warte bis Empfang
               komplett
52         return SPDR;                                    // empfangenes Byte
               zurueckgeben
53 }
```

TWI.h

```
1  /*
2   * TWI.h
3   *
4   * Created: 09.03.2012 08:37:01
5   *  Author: Niki
6   */
7
8
9  #ifndef TWI_H_
10 #define TWI_H_
11
12 #include <avr/io.h>
13
14 //Funktion startet eine TWI Übertragung
15 void TWI_start(void);
16
17 //Funktion stopped eine TWI Übertragung
18 void TWI_stop(void);
19
20 //Funktion sendet ein Byte über TWI
21 //Param 1 Byte das zu senden ist
22 void TWI_send(uint8_t DataByte);
23
24 //Funktion empfängt ein Byte über TWI
25 //Param gibt an ob ein ACK erforderlich ist
26 uint8_t TWI_receive(uint8_t ack);
27
28 //Funktion initialisiert die TWI Schnittstelle
29 void Init_TWI(void);
30
31 #endif /* TWI_H_ */
```

TWI.c

```
1  /*
2   * TWI.c
3   *
4   * Created: 09.03.2012 08:38:15
5   *  Author: Niki
6   */
```

```
 7
 8 #include "TWI.h"
 9
10 #ifndef F_CPU
11 #define F_CPU 12000000UL
12 #endif
13
14 //Funktion initialisiert die TWI Schnittstelle
15 void Init_TWI(void)
16 {
17         TWBR = (F_CPU / 100000UL - 16) / 2;
18 }
19
20 //Funktion startet eine TWI Übertragung
21 void TWI_start(void)
22 {
23         TWCR = _BV(TWINT) | _BV(TWSTA) | _BV(TWEN);      // start
24         while ((TWCR & _BV(TWINT)) == 0) ;                                  //
               Warten bis Ende
25 }
26
27 //Funktion stopped eine TWI Übertragung
28 void TWI_stop(void)
29 {
30         TWCR = _BV(TWINT) | _BV(TWSTO) | _BV(TWEN);               // Stop
               senden
31 }
32
33 //Funktion sendet ein Byte über TWI
34 //Param 1 Byte das zu senden ist
35 void TWI_send(uint8_t DataByte)
36 {
37         TWDR = DataByte;
38         TWCR = _BV(TWINT) | _BV(TWEN);                    // clear interrupt
               to start transmission
39         while ((TWCR & _BV(TWINT)) == 0) ;                // wait for
               transmission
40 }
41
42 //Funktion empfängt ein Byte über TWI
43 //Param gibt an ob ein ACK erforderlich ist
44 uint8_t TWI_receive(uint8_t ack)
45 {
46         if (ack == 1)
47                 TWCR = _BV(TWINT) | _BV(TWEA) | _BV(TWEN);// Empfänger
                       starten, ACK notwendig
48         else
49                 TWCR = _BV(TWINT) | _BV(TWEN);                    //
                       Empfänger starten
50         while ((TWCR & _BV(TWINT)) == 0) ;                // Warten
51         return TWDR;
52 }
```