Ostfalia
Hochschule für angewandte Wissenschaften

Studienarbeit

Entwicklung des CITRIC Stroboskop-Kamerasystems für den Rotorversuchstand des DLR

Abgabetermin: 21.09.2011

Erstellt von: Tim Schinke Matrikelnr.: 20864103

Betreuender Professor: Prof. Dr. rer. nat. C.- W. Turtur

Betreuer: Dipl. Ing. Michael Przybilla

DLR

Versicherung der selbstständigen Erarbeitung der Studienarbeit

Hiermit versichere ich, dass ich die Studienarbeit selbstständig angefertigt, keine anderen als die angegebenen Hilfsmittel benutzt und die Stellen der Studienarbeit, die im Wortlaut oder im wesentlichen Inhalt aus anderen Werken (auch aus dem Internet) entnommen wurden, mit genauer Quellenangabe kenntlich gemacht habe.

Wolfenbüttel, den _____

Unterschrift des Studenten

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Die Studienarbeit dient der Entwicklung des CITRIC Stroboskop-Kamerasystems für den Rotorversuchstand des deutschen Zentrums für Luft- und Raumfahrt (DLR). Das Kamerasystem wird benötigt um Standbilder des sich rotierenden Rotors aufzunehmen. Dabei soll das System über ein Bedienpanel verfügen, dass zur Steuerung des Objektivs an der Kamera bzw des Schwenk-/Neigekopfs, welcher die Kamera bewegt, dient. Als Komponenten kommen eine Industrie-(Firewire)-Kamera von Sony, die XCD-X710CR, sowie ein Objektiv der Marke Fujinon, das S16x7.3DB-S41, welches über Zoom, Focus und Autoiris verfügt und dem passenden Bedienelement von Fujinon, das CRD-2A, zum Einsatzt. Für das bereits bestehende Projekt muss nun eine Platine entwickelt werden, die die Interaktion der einzelnen Komponenten gewährleistet. Ein besonderes Augenmerk bekommt die Steuerung der Blende (Autoiris) am Objektiv, da hierfür ein FBAS-/BAS-Signal erzeugt werden muss, welches mit einem Microcontroller nachgebildet wird. Dazu muss für den Microcontroller eine enstprechende Software entwickelt werden die dann auf dem Baustein programmiert wird. Das Weiteren muss ein Mikrofonverstärker entwickelt werden, der auch bei erhöhtem Lärmpegel noch ausreichend gute Tonqualität gewährleistet.

2 Ablaufplan

Zu Beginn des Projekts wurde ein Zeitplan in Form eines Ganttdiagrams, siehe Abbildung 2 auf der nächsten Seite, erstellt, um die benötigte Zeit besser planen zu können. Des Weiteren bietet ein Ablaufplan die Möglichkeit, den kritischen Pfad, siehe Abbildung 3 auf Seite 8, im Projekt besser überblicken und gegebenenfalls den Zeitplan weiter aufstocken zu können, um trotzdem zeitnah das Projekts abzuschließen. Es lassen sich mehrere Bereiche im Gantt-Diagramm unterscheiden. Zum einen gibt es zum Anfang einen Bereich der als vorbereitende Tatigkeiten definiert wird. Hierunter fallen z.B. die Sichtung sämtlicher bisheriger Unterlagen über das Projekt sowie das Nachvollziehen der schon vorhandenen Hardware. Einen zweiten Abschnitt bildet die Hardware-Entwicklung, welcher auch den zeitaufwendigsten Teil des Projektes bildet, da jeweils mehrere Prototypen entwickelt und getestet werden müssen, bis ein entgültiges Platinenlayout erstellt werden kann. Die Software-Entwicklung bildet einen weiteren Teil der Zeitplanung. Hierbei handelt es sich um die Entwicklung der Software für die verbaute Hardware, wie beispielsweise Microcontroller usw., die mit den gebauten Prototypen erstmalig getestet und auf Funktionalität geprüft wird. Ein weiterer Teil stellt die Dokumentation dar, welche ständig neben den anderen Tatigkeiten des Projekts überarbeitet und aktualisiert wird. Dies bietet den Vorteil einer zeitnahen Fertigstellung der Arbeit. Der Abschluss bildet den letzten Teil des Projekts, da hier die finale Dokumentation und die Präsentation erstellt wird, wobei der anschließende Vortrag das Projekt (Studienarbeit) beendet.

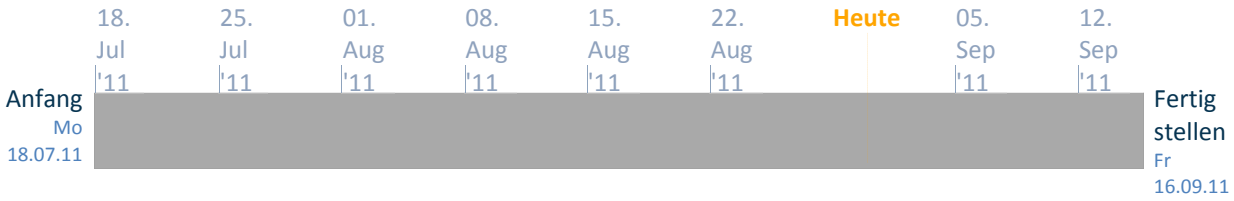
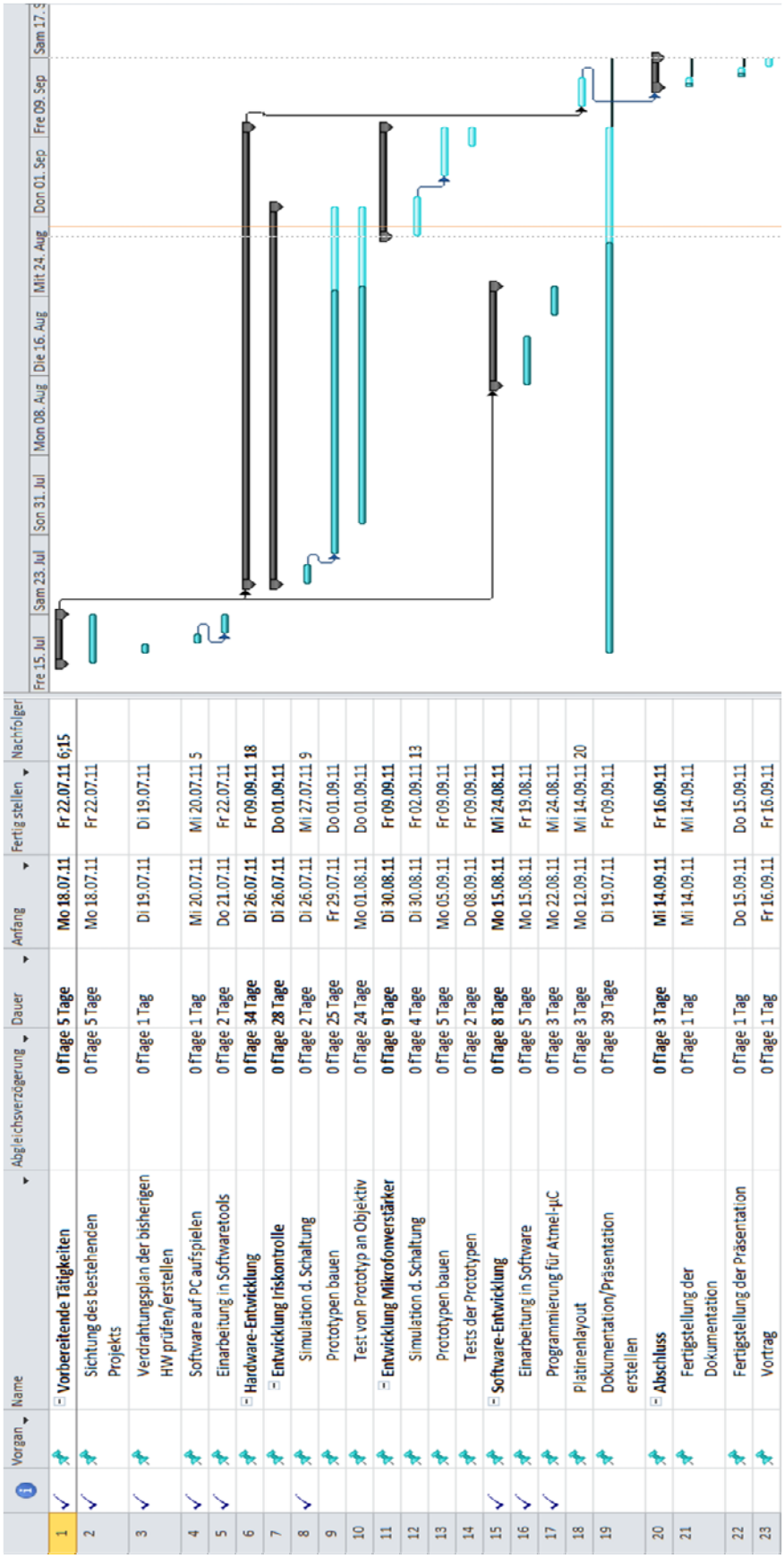


Abbildung 1: Zeitachse des Projekts vom 31.08.2011

Wie die Abbildung 1 verdeutlicht ist ein Zeitraum vom 18.07.2011 bis 16.09.2011, also insgesamt 45 Tage, für die Studienarbeit vorgesehen. Eine spätere Vertigstellung des Projekts hätte zur Folge, dass der Abschließende Vortrag im kommenden Semester während den Vorlesungen statt finden muss, was nicht unbedingt von Vorteil ist. Deshalb ist die Einhaltung des Zeitplans um so wichtiger.



	Vorgan	Name	Abgleichsverzögerung	Dauer	Anfang	Fertig stellen	Nachfolger
1	✓	**Vorbereitende Tätigkeiten**		**0 ffage 5 Tage**	**Mo 18.07.11**	**Fr 22.07.11 6;15**	
2	✓	Sichtung des bestehenden Projekts		0 ffage 5 Tage	Mo 18.07.11	Fr 22.07.11	
3	✓	Verdrahtungsplan der bisherigen HW prüfen/erstellen		0 ffage 1 Tag	Di 19.07.11	Di 19.07.11	
4	✓	Software auf PC aufspielen		0 ffage 1 Tag	Mi 20.07.11	Mi 20.07.11 5	
5	✓	Einarbeitung in Softwaretools		0 ffage 2 Tage	Do 21.07.11	Fr 22.07.11	
6		**Hardware-Entwicklung**		**0 ffage 34 Tage**	**Di 26.07.11**	**Fr 09.09.11 18**	
7	✓	**Entwicklung Iriskontrolle**		**0 ffage 28 Tage**	**Di 26.07.11**	**Do 01.09.11**	
8	✓	Simulation d. Schaltung		0 ffage 2 Tage	Di 26.07.11	Mi 27.07.11 9	
9	✓	Prototypen bauen		0 ffage 25 Tage	Fr 29.07.11	Do 01.09.11	
10	✓	Test von Prototyp an Objektiv		0 ffage 24 Tage	Mo 01.08.11	Do 01.09.11	
11	✓	**Entwicklung Mikrofonverstärker**		**0 ffage 9 Tage**	**Di 30.08.11**	**Fr 09.09.11**	
12	✓	Simulation d. Schaltung		0 ffage 4 Tage	Di 30.08.11	Fr 02.09.11 13	
13	✓	Prototypen bauen		0 ffage 5 Tage	Mo 05.09.11	Fr 09.09.11	
14	✓	Tests der Prototypen		0 ffage 2 Tage	Do 08.09.11	Fr 09.09.11	
15	✓	**Software-Entwicklung**		**0 ffage 8 Tage**	**Mo 15.08.11**	**Mi 24.08.11**	
16	✓	Einarbeitung in Software		0 ffage 5 Tage	Mo 15.08.11	Fr 19.08.11	
17	✓	Programmierung für Atmel-µC		0 ffage 3 Tage	Mo 22.08.11	Mi 24.08.11	
18		Platinenlayout		0 ffage 3 Tage	Mo 12.09.11	Mi 14.09.11 20	
19	✓	Dokumentation/Präsentation erstellen		0 ffage 39 Tage	Di 19.07.11	Fr 09.09.11	
20	✓	**Abschluss**		**0 ffage 3 Tage**	**Mi 14.09.11**	**Fr 16.09.11**	
21	✓	Fertigstellung der Dokumentation		0 ffage 1 Tag	Mi 14.09.11	Mi 14.09.11	
22	✓	Fertigstellung der Präsentation		0 ffage 1 Tag	Do 15.09.11	Do 15.09.11	
23	✓	Vortrag		0 ffage 1 Tag	Fr 16.09.11	Fr 16.09.11	

Fre 15. Jul | Sam 23. Jul | Son 31. Jul | Mon 08. Aug | Die 16. Aug | Mit 24. Aug | Don 01. Sep | Fre 09. Sep | Sam 17. S

Abbildung 2: Ablaufplan (Gantt-Diagramm) des Projekts am 31.8.2011

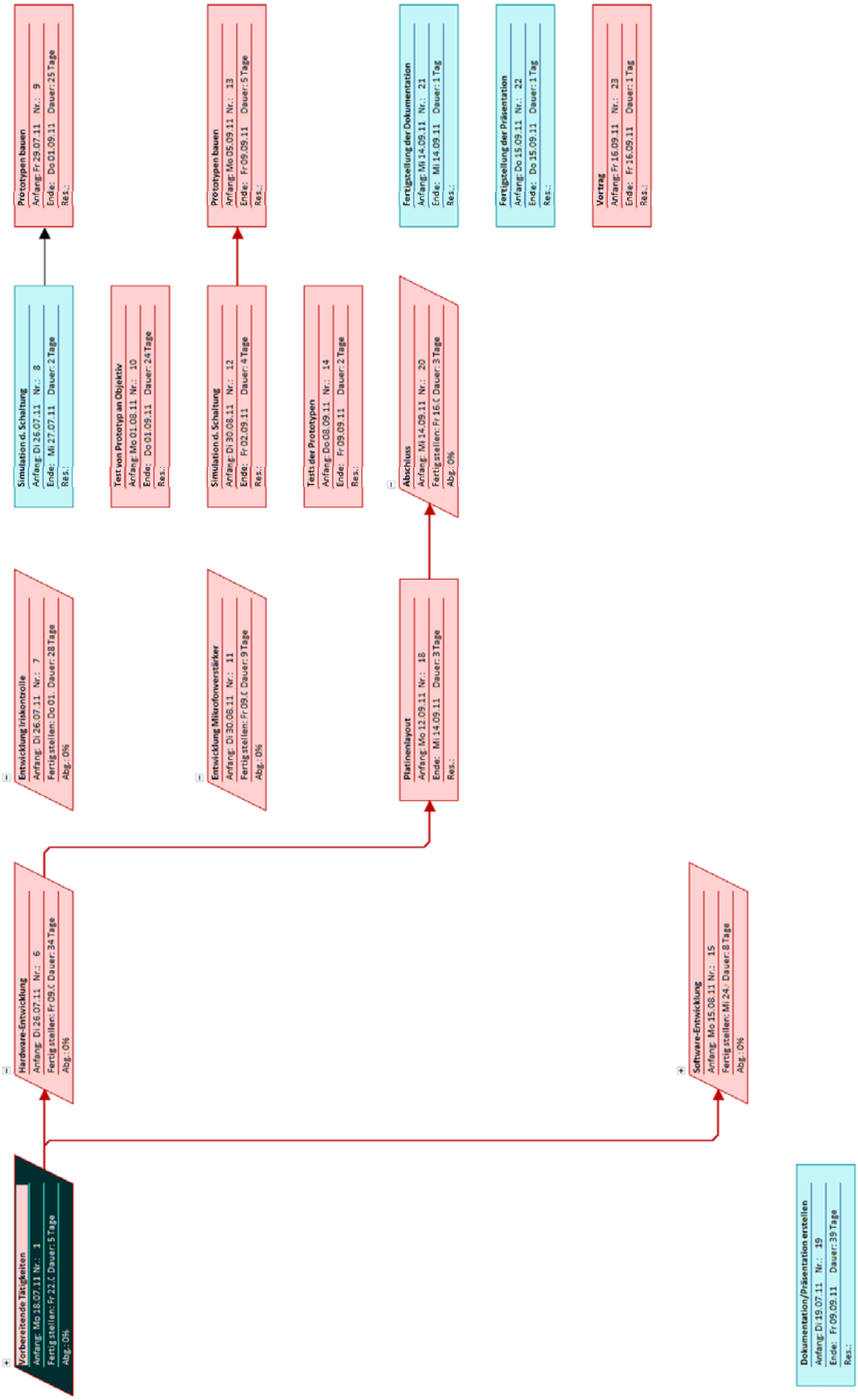


Abbildung 3: Netzplandiagramm des Projekts zum Aufzeigen des kritischen Pfads

3 Grundlagen

3.1 Einarbeitung in OrCad

OrCad ist ein Programm der Firma Cadence Design Systems, welches im wesentlichen Sinn zur Entwicklung elektronischer Schaltungen, zur Schaltplaneingabe und zur Simulation entworfener Schaltungen mittels PSpice dient. Da diese Software sehr komplex ist, wurde ein tutorial aus dem Internet, siehe [1], für die Einarbeitung zur Hilfe genommen.
Wie in jedem schon bestehenden Projekt, ist die Sichtung der bisherigen Unterlagen unerlässlich. Die bisher erstellten Unterlagen, Schaltpläne sowie Simulationsbeispiele und Skizzen wurden mithilfe von OrCad nachvollzogen.

3.2 Vorbereitende Arbeiten

Um die Kamera und den Rest der Peripherie mit der Platine verbinden zu können, müssen aus dem Schwenk-/Neigekopf mehrere Leitungen herausgeführt werden. An diese Leitungen werden dann etwaige Stecker angelötet um die richtige Konnektivität zu gewährleisten. Es werden verschiedene Steckertypen verwendet, wie z.B. BNC, Firewire, usw.. Diese Stecker werden dann für die Verbindung des Objektivs (Zoom, Focus, Iris), der Kamera, der Lampe sowie des Triggereingangs genutzt.

Für die vorherigen Tests am Arbeitsplatz mit der Kamera wird zunächst eine Firewire-Karte in den PC eingebaut, damit die Kamera über den Firewire-BUS mit enstprechnder Software angesprochen werden kann. Nach erfolgreichem Einbau der Karte muss die Software für die Kamera installiert werden. Dies geschieht unter zu Hilfenahme einer Anleitung, siehe [2], und wird bei ausreichender Zeit in einem Extrakapitel behandelt um die Konnektivität an Notebooks zu testen, da es in der Vergangenheit öfters zu Fehlern diesbezüglich kam. Ein Kontakt mit Sony, dem Hersteller der Kamera, hat weiterhin dazu geführt, dass nun auch eine Software inklusive Treiber für 64-Bit Systeme (Windows XP, Windows Vista, Windows 7) verfügbar ist.

3.3 Grundlagen zum Firewireprotokoll (IEEE-1394)

IEEE-1394 ist eine Technik bei der die Daten seriell mit einer geschwindigkeit von bis zu 400 MBit/s (IEEE-1394a) bzw. 800 MBit/s (IEEE-1394b) übertragen werden können und somit Hochgeschwindigkeitsgeräte wie beispielsweise Videokameras oder auch Festplatten mit dieser Technik betrieben werden können.

Diese Technik wurde 1995 zum Industriestandard IEEE-1394 benannt und wird heutzutage als Firewire oder auch i.Link bezeichnet. Die großen Vorteile dieser Technologie sind unter anderem, dass Geräte im laufenden Betrieb des PC angeschlossen und erkannt werden können (Plug & Play), das eine bi-direktionale Datenübertragung möglich ist und das eine Versorgungsspannung (ca. 8 – 40 VDC, ca. 1,5 A) direkt am Bus anliegt. Man unterscheidet bei IEEE-1394 zwischen IEEE-1394a und IEEE-1394b. Die Unterschiede sind in der Übertragungsgeschwindigkeit sowie in einer neuen neunpoligen Steckerbauform begründet. Außerdem wird bei IEEE-1394b ein neues Protokoll und eine andere Signalcodierung verwendet mit dessen Hilfe höhere Übertragungsgeschwindigkeiten möglich sind und selbst sog. Multimode-Glasfaserkabel für die Übertragung verwendet werden können. Eine weiteres Merkmal der Firewire-Technologie ist die isochrone Datenübertragung welche eine quasi-kontinuierliche Übertragungsrate erlaubt um eine unterbrechungsfreie Übertragung von Medien (Musik, Video) zu gewährleisten. Untenstehende Abbildung 4 zeigt die Transaktionen der Pakete auf dem Firewire-Bus innerhalb eines Zyklus.

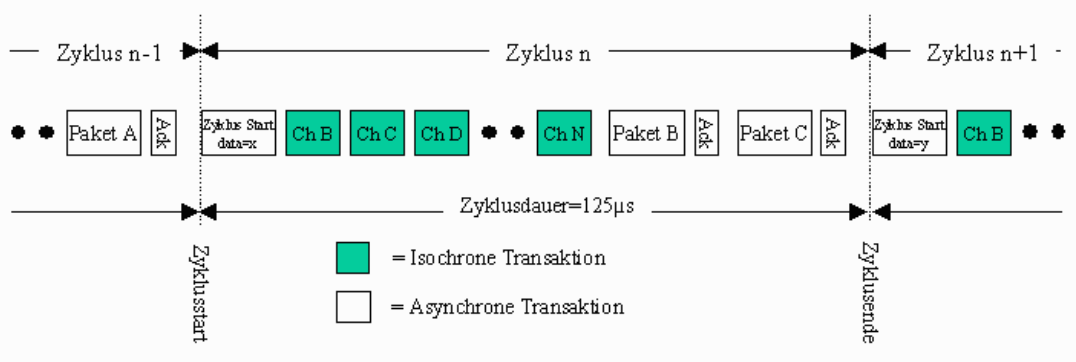


Abbildung 4: Isochrone und Asynchrone Transaktion während eines Zyklus bei Firewire

4 Bearbeitung der Kernaufgabe

4.1 Herangehensweise an das Problem

Die Kernaufgabe dieser Studienarbeit ist die Inbetriebnahme des CITRIC Stroboskop-Kamerasystems. Es ist also wichtig die bisher für das Projekt entwickelte Hardware zu „verstehen“, also wie die ausgewählte Hardware funktioniert. Antworten dazu findet man in den zugehörigen Benutzerhandbüchern sowie in den im Internet erhältlichen Verdrahtungsplänen und Dokumentationen. Auch die bisher in OrCad entwickelten Schaltungsteile für die Iriskontrolle und den Mikrofonverstärker müssen nachvollzogen und simuliert werden um an den richtigen Stellen weiter machen zu können.

4.2 Entwicklung der einzelnen Schaltungskomponenten

Für das Projekt muss eine manuelle Iriskontrolle und ein Mikrofonverstärker entwickelt werden. Die beiden Themen werden jeweils für sich in den folgenden Unterkapiteln behandelt. Die Entwicklung beinhaltet das Entwerfen von Prototypen, die nach dem Entwurf auf Lochrasterplatinen gelötet werden und mit Hilfe eines Oszilloskop auf Funktionalität geprüft werden. Des Weiteren wird die vorher entworfene Software auf die zu programmierende Hardware der Prototypen aufgespielt um diese auf die endgültige Funktion zu prüfen.

4.2.1 Entwicklung der manuellen Iriskontrolle

Die Iriskontrolle soll die Blende und somit die Helligkeit der Kamera steuern. Dies wird ermöglicht, indem eine aus einem Mittelwert entstandene Spannung zwischen 0 V – 1 V an der Iris des Objektivs angelegt wird, ähnlich der Abbildung 5, entnommen aus [3], welche ein FBAS Signal bzw. ein Oszillogramm einer Zeile eines PAL-modulierten Fernsehbildes zeigt.

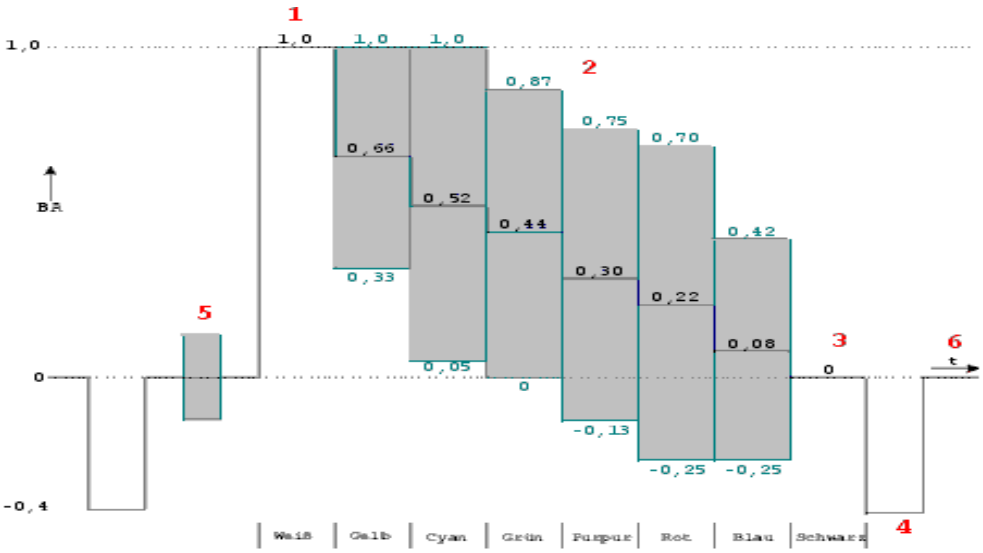


Abbildung 5: Oszillogramm einer Zeile eines PAL-modulierten Fernsehbildes FBAS

Hierbei entspricht 0 V dem Schwarzpegel und 1 V dem Weißpegel.

Um das FBAS- bzw. das BAS-Signal zu erzeugen wird ein Microcontroller entsprechend programmiert. In diesem Fall wurde das Arduino UNO Board, zu sehen in Abbildung 6, ausgewählt, da für dieses Board schon einmal erfolgreich ein FBAS-Signal programmiert und am Fernseher ausgegeben wurde, siehe hierzu auch [4], und es eine günstige Alternative zu anderen Entwicklerboards, beispielsweise von Xilinx, darstellt. Außerdem bietet es die Vorteile einer einfachen Programmierung unter C, das einfache Softwareaufspielen über USB und, dass das Arduino UNO ein Open-Source Projekt ist, was einem ohne Probleme ermöglicht auch Einsicht in die Schaltungspläne des Boards zu nehmen um etwaige Schaltungsteile extrahieren zu können.

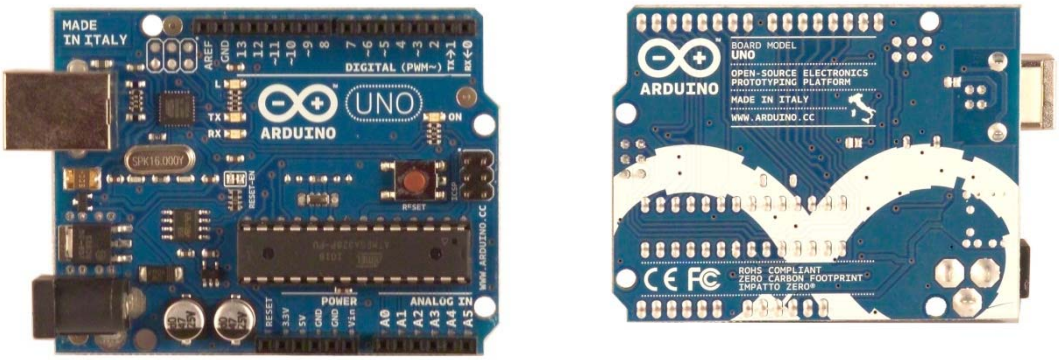


Abbildung 6: Microcontroller Arduino UNO, siehe [5]

Um das Board mit dem Computer über USB zu verbinden, muss die von Arduino bereitgestellten Entwicklungsumgebung, siehe http://arduino.cc/en/Main/Software , heruntergeladen werden. Nach der erfolgreichen Installation der Software und des Boards, siehe für weitere Hilfe auch [10], können nun erste Beispiele mit dem Arduino getestet werden. Dazu muss, wie Abbildung 7 zeigt, unter *Tools -> Board* der Arduino UNO und der unter *Tools -> Serial Port* der, nach der Installation des Boards, entsprechende Port, meisstens Com3, ausgewählt werden.

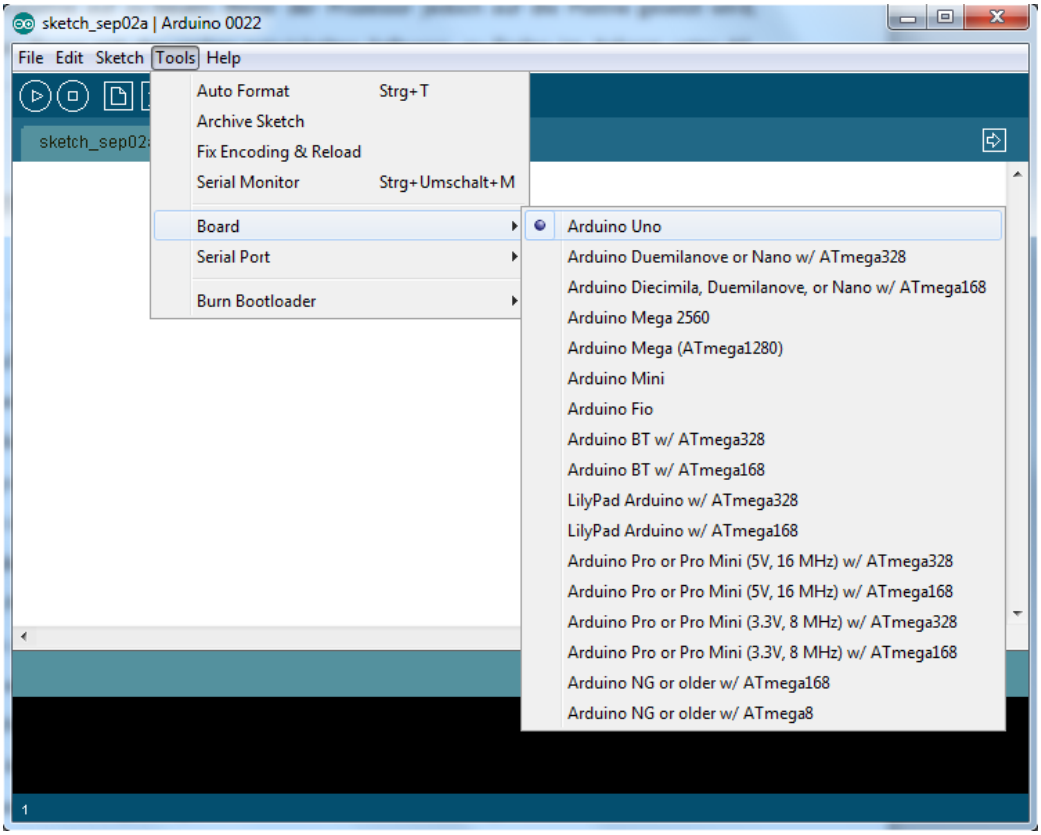


Abbildung 7: Arduino Entwicklungsumgebung mit Auswahlmenü des Boards

Nachdem nun erste Beispiele, wie eine blinkende LED, getestet wurden, konnte die Software, die das FBAS-/BAS-Signal simuliert, auf das Board übertragen werden. Ebenfalls muss nach der Anleitung aus [4] ein entsprechender Aufsatz, in Abbildung 8 dargestellt, für den Arduino UNO gefertigt werden, der das Ausgangssignal über eine BNC-Kupplung an einen Monitor übertragen kann.

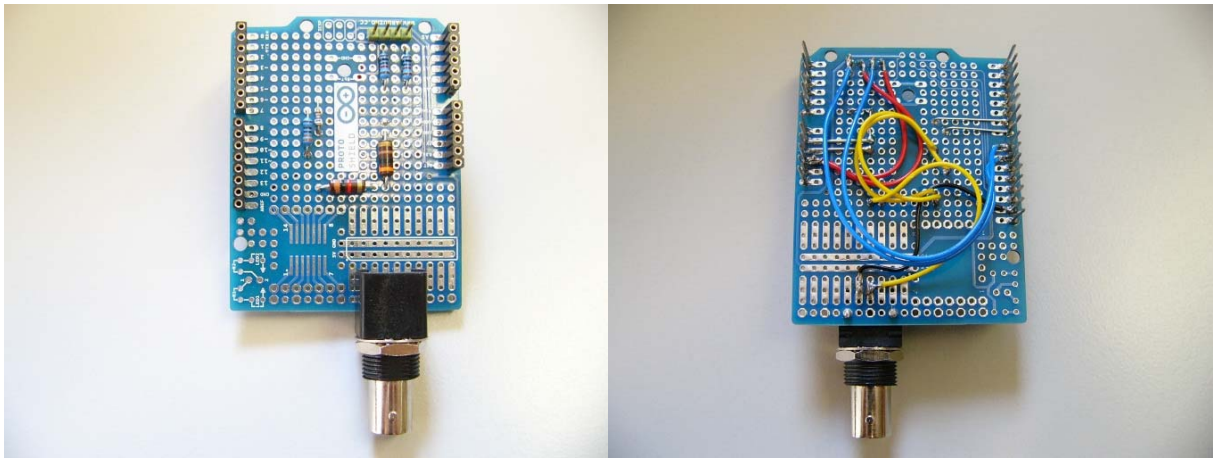


Abbildung 8: Prototyp-Aufsatz für Arduino UNO zum Erzeugen eines Videosignals

Da das Objektiv, bzw. die Blende, ein FBAS-Signal mit den jeweiligen Syncronisationsimpulsen erwartet, ist bei der Programmierung besonders auf die Timingparameter zu achten. Der in Abbildung 9 dargestellte Programmablaufplan zeigt einen Grobentwurf der weiter zu entwickelnden Software.

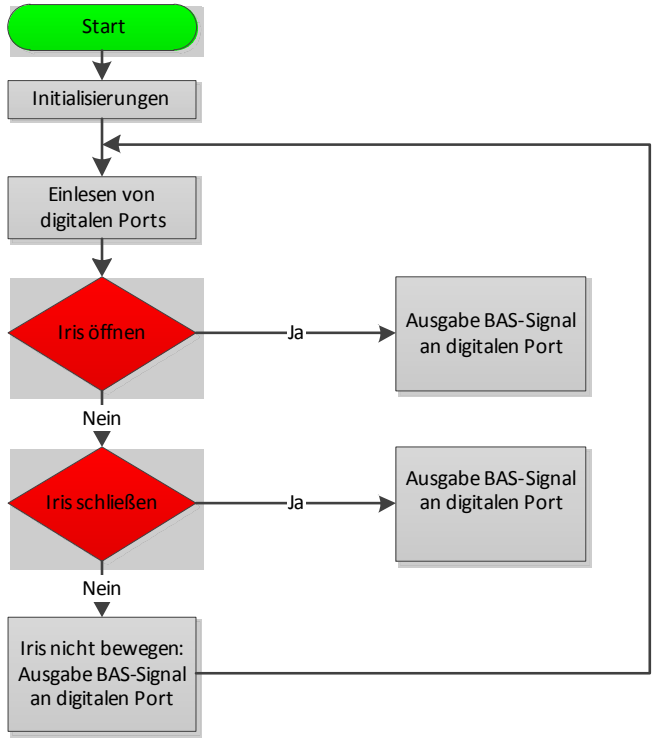


Abbildung 9: Programmablaufplan der Iriskontrolle

Das schon existierende Programm aus [4] wird entsprechend dem Grobentwurf weiter entwickelt, in dem nun eine Unterscheidung entsprechend den Eingangspins statt findet, was über einen Schalter am Bedienpult gesteuert wird. Der fertige Code, im Anhang unter A1 zu finden, muss nun entsprechend auf das Board geladen werden um einen Funktionstest durchführen zu können.

Die folgenden Abbildungen 9, 10 und 11 zeigen die jeweiligen Oszillloskopbilder für die drei verschieden Modi „öffnen“, „schließen“ und „stillstehen“.

Man erkennt den hsync-Impuls, welcher für die horizontale Auslenkung zuständig ist, bei 0 V für 4,7 µs, danach folgt der Schwarzwert für 6 µs bei 248 mV, dann der Grauwert bei 720 mV für 51 µs und ganz kurz der Weißpegel bei 96 mV für 1 µs gefolgt von noch einem Schwarzpegel für 4,7 µs. Jetzt folgt der nächste hsync. – Impuls. Insgesamt ist das Signal ca. 67 µs lang, was etwas vom Standart (64 µs) abweicht, aber die Blende still stehen lässt.

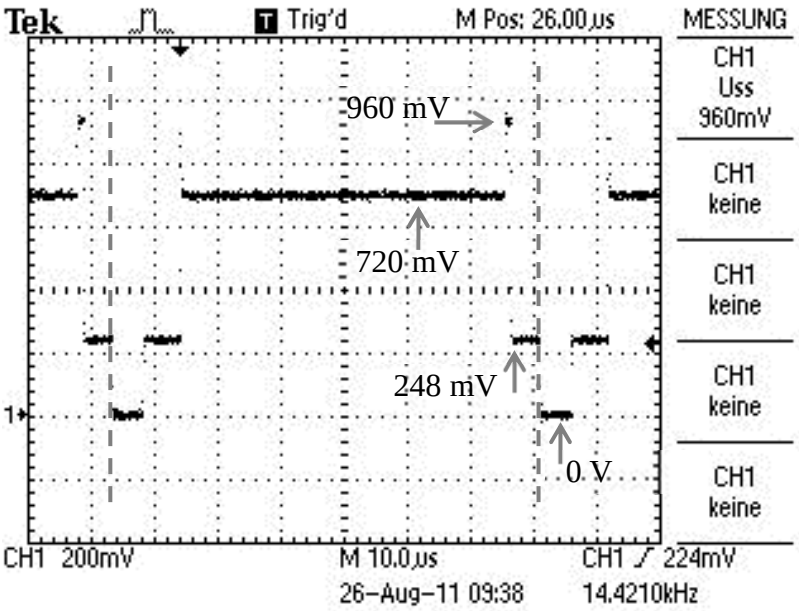


Abbildung 10: Oszilloskopbild wenn die Blende still steht

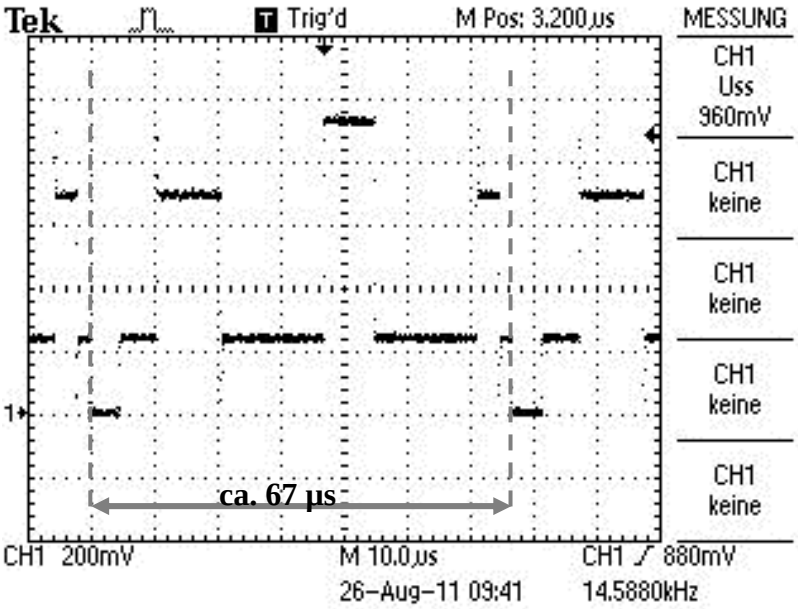


Abbildung 11: Oszilloskopbild wenn die Blende geöffnet wird

Da das Öffnen der Blende einen erhöhten Schwarzpegel erfordert, sieht man in Abbildung 10 das geänderte Timing, so dass Schwarz für 32 µs, Grau für 12 µs und Weiß für 8 µs anliegt.

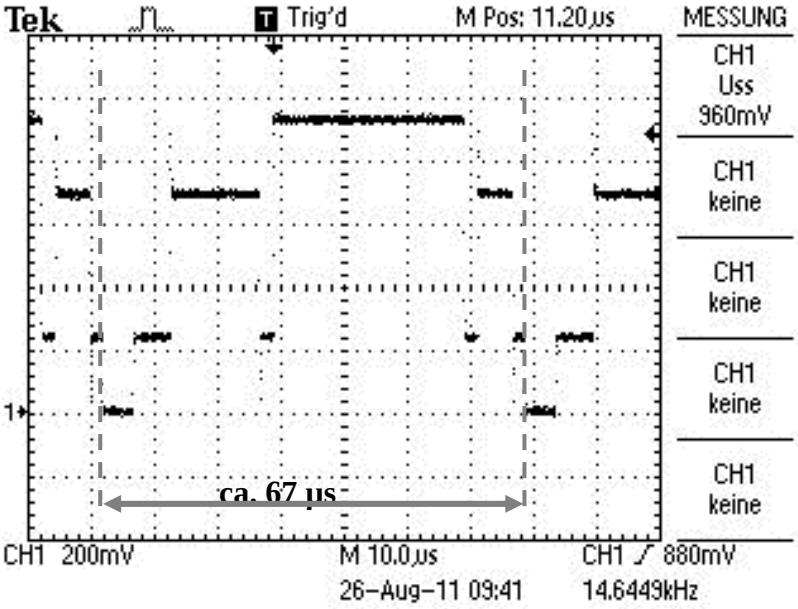


Abbildung 12: Oszilloskopbild wenn die Blende geschlossen wird

Das Gegenteil, also die Blende zu schließen, erfordert einen erhöhten Weißpegel, so erkennt man das in Abbildung 11 der Weißpegel mit 30 µs überwiegt. Der Graupegel liegt insgesamt für 18 µs und der Schwarzpegel für 4 µs an.

Um die Spannungspegel wie in den Abbildungen 9, 10 und 11 auch so als Signal an die Blende übertragen zu können, müssen noch ein paar Schritte beachtet werden. Zum einen werden zwei Widerstände benötigt, 1 kΩ und 330 Ω, die zwischen die Ausgangspins des Arduino UNO Boards und der Leitung, die später zum Objektiv führt, gelötet werden müssen. Diese bilden einen einfachen „2 DAC“, also ein zweifacher digital/analog Wandler, nach. Hinter die oben erwähnten Widerstände müssen unbedingt noch zwei weitere Widerstände, 1 kΩ und 300 Ω, welche als Spannungsteiler dienen, damit eine Ausgangsspannung zwischen 0 V und 1 V entsteht. Um jetzt die Blende mit einem Schalter bedienbar zu machen, werden noch zwei weitere Widerstände, beide 10 kΩ, benötigt. Der Schalter wird nun gemäß der Anleitung zum Arduino UNO Board, vgl. Seite 42ff aus [6], angeschlossen. Die Abbildung 12 zeigt den passenden Verdrahtungsausschnitt.

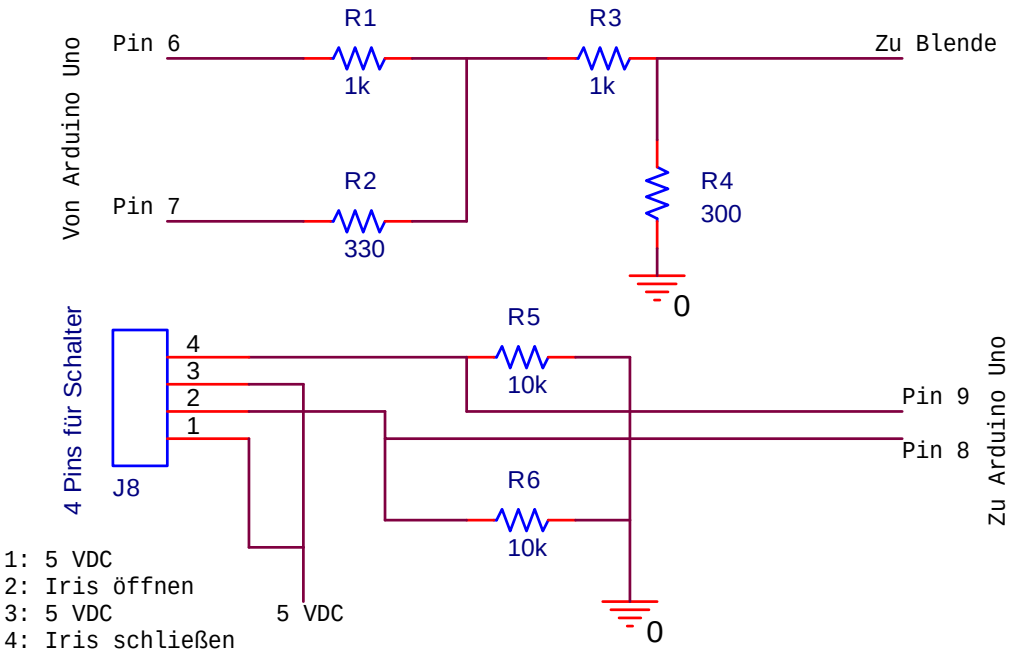


Abbildung 13: Verdrahtungsplan zur Steuerung der Blende mittels Arduino UNO

Pin 6 und Pin 7, siehe Abbildung 7, sind als Ausgänge des Microcontrollers (µC) definiert und die beiden Widerstände, R_1 und R_2, bilden den digital/analog Konverter nach. Die Widerstände R_3 und R_4 sind zum herunter teilen der Spannung nötig. Die Pins 8 und 9 sind als Eingänge des µC definiert an denen, je nach Schalterstellung, erkannt wird, wie die Blende zu regeln ist.

Nun muss ein Prototyp entworfen werden, der den benötigten Teil des Microcontrollers nachbildet. Dazu dient ein Blick in den Schaltungsplan des Arduino UNO, siehe [7], um den Teil zu extrahieren der benötigt wird. Nach einer kurzen Analyse des Schaltplans stehen die benötigten Bauteile fest. Abbildung 13 zeigt genau den Teil, den es nach zu bauen gilt. Dieser beinhaltet einen Prozessor von Atmel, den Atmega328P-PU, einen Quarz der mit 16 MHz taktet, sowie noch ein paar Kondensatoren und einen 1 MΩ Widerstand. Um den Prozessor beim Einschalten einem kurzen Reset zu unterziehen, wird zusätzlich noch eine Power-On Resetschaltung, bestehend aus einem 10 kΩ Widerstand und einem 2,2 µF Kondensator, mit eingeplant, welche direkt an den invertierten Reseteingang vom Atmega328P-PU angeschlossen wird. Somit wird sicher gestellt, dass beim Start des Prozessors keine Fehler wie beispielsweise falsche Registereinträge oder falsche Flags entstehen, da bei jedem anschalten des Prozessors ein kurzer Reset erfolgt.

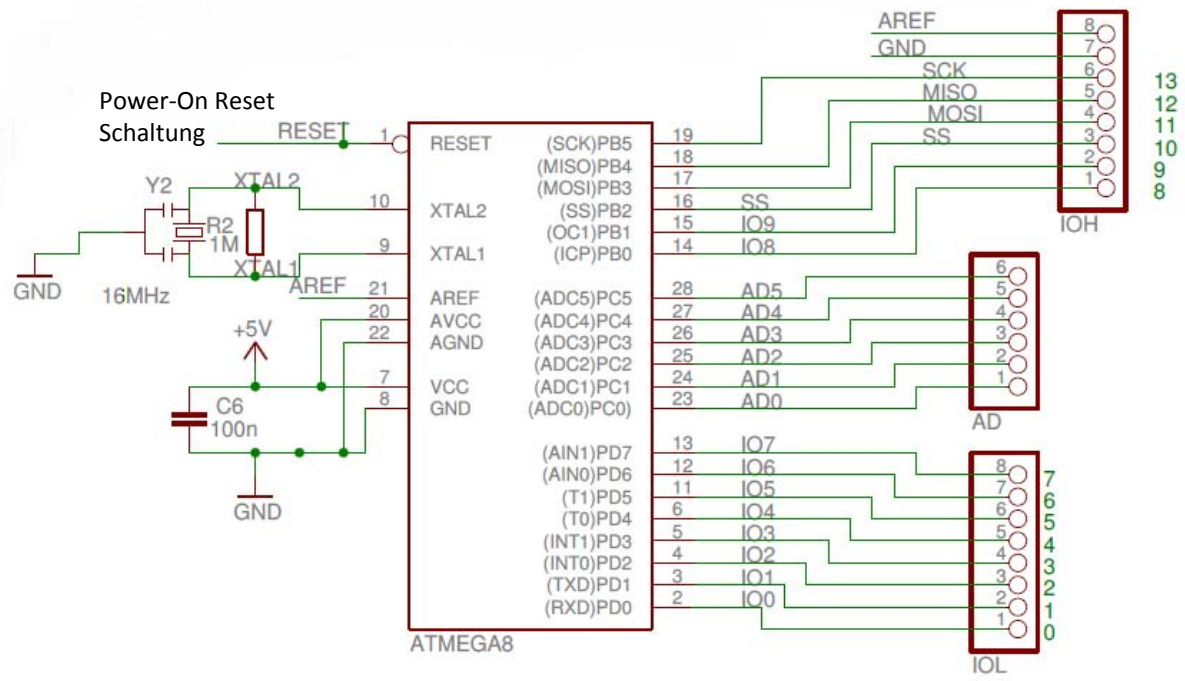


Abbildung 14: Extrahierte Schaltung aus dem Arduino UNO

Als Ein- bzw. Ausgänge dienen lediglich die Pins 12, 13, 14 und 15. Der Rest der I/O-Pins bleibt unbeschaltet. Nun kann damit begonnen werden einen Protoypen auf einer Lochrasterplatine auf zu bauen. Bevor der Prozessor jedoch auf die Platine gesetzt wird, sollte man diesen mit der vorher entwickelten Software, zu finden im Anhang unter A1,

bespielen/brennen. Hierfür muss passend für das Arduino Board ein zusätzlicher Adapter gebaut werden, um einen Bootloader auf den Atmel-Prozessor zu brennen. Der Bootloader, vergleichbar mit dem BIOS eines Computers, ist eine spezielle Software, auch als Startprogramm definierbar, vgl. [8], die den Prozessor startfähig macht und somit notwendig ist, damit die später auf den Prozessor geladene Software auch ausgeführt wird.

Dieser Adapter wird nach Anleitung von [9] auf einer Lochrasterplatine nachgebaut um bei späteren Projekten mit dem Arduino UNO weitere Atmel-Prozessoren mit einem Bootloader bespielen zu können. Somit muss nicht für jedes neue Projekt ein komplettes Arduino Board verbaut werden, sondern es wird dann nur noch der eigentliche Prozessor sowie die speziefischen Hardwarekomponenten für die jeweiligen Projekte benötigt. Um das Arduino Board als AVR ISP (In-System Programmer) benutzen zu können, muss vorerst noch die dazu gehörige Software auf das Board übertragen werden. Diese findet man in der von Arduino bereitgestellten Entwicklungsumgebung unter *File -> Examples -> ArduinoISP* .

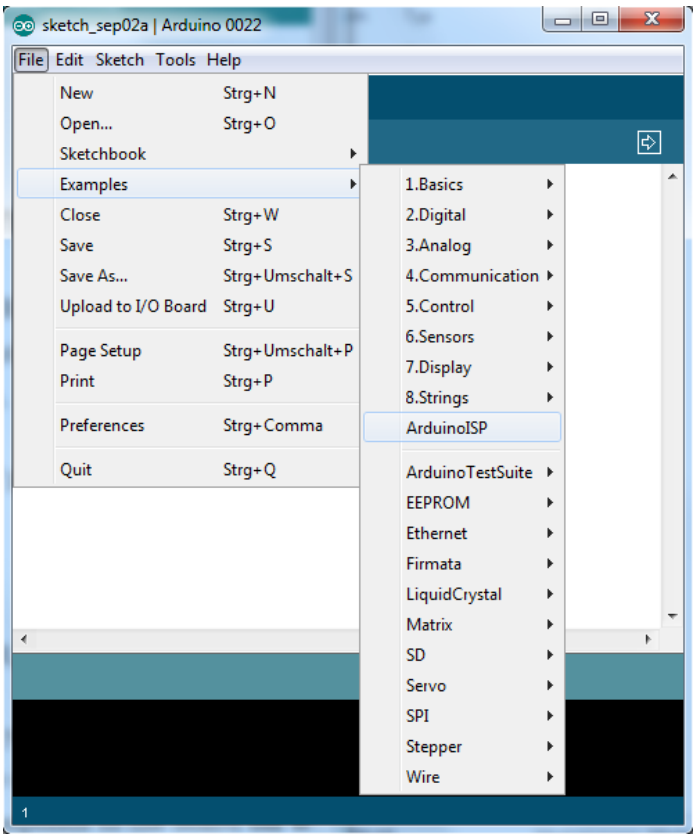


Abbildung 15: Arduino Entwicklungsumgebung mit Auswahl des ArduinoISP-Programms

Nachdem die Software erfolgreich auf das Board übertragen wurde, kann der zu brennende Prozessor Atmega328P-PU in den im Adapter, siehe Abbildung 15, vorgesehenden Sockel eingesetzt werden.

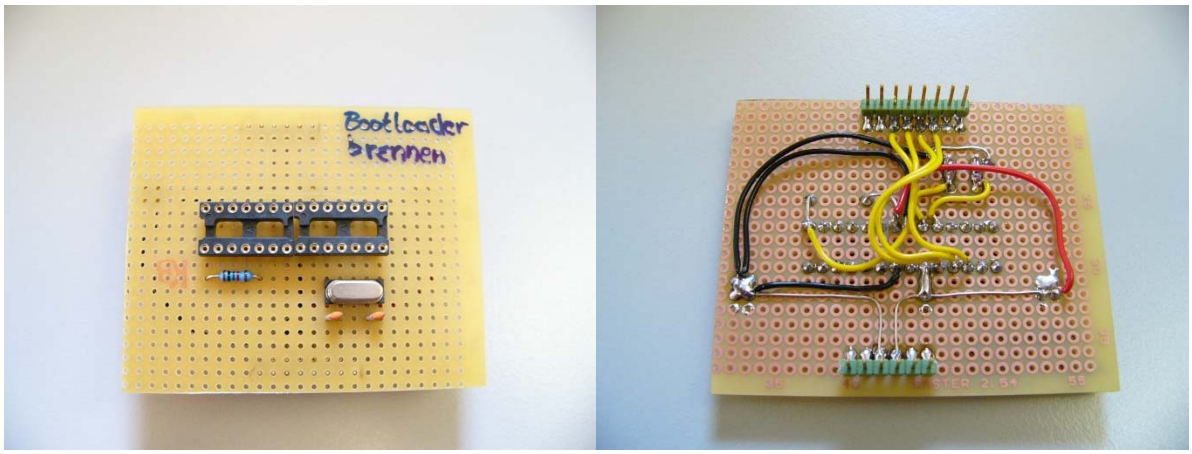


Abbildung 16: Adapter für Arduino UNO zum Brennen von Bootloadern auf Atmega

Nun wird der Brennvorgang über *Tools ->Burn Bootloader ->w/ Arduino as ISP* gestartet.

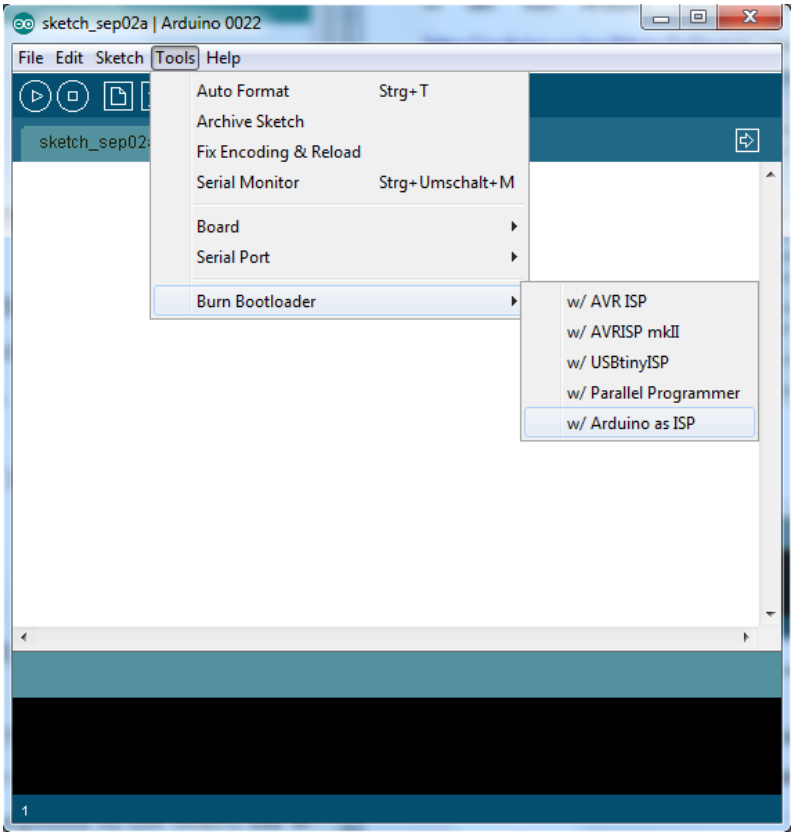


Abbildung 17: Arduino Entwicklungsumgebung mit Auswahlmenü „Bootloader brennen“

Nach erfolgreichem aufspielen des Bootloaders wird der gebrannte Prozessor vom Adapter und der Prozessor des Arduino Boards vorsichtig aus dem Sockel heraus genommen. Anschließend wird der frisch gebrannte Prozessor auf das Arduino Board gesetzt und mit der Software aus A1, im Anhang zu finden, bespielt. Nach dem erfolgreichen Aufspielen der Software kann der Prozessor auf die Prototyp-Platine gesetzt werden und die

Spannungsversorgung eingeschaltet werden. Wie erwartet funktioniert nun die Iris an dem Objektiv über einen manuellen Schalter, so dass die Blende nach belieben geöffnet und geschlossen werden kann.

Der Schaltplan in Abbildung 17 zeigt die gesamte für die Iriskontrolle verbaute Hardware, welche genau so für den Prototypen verbaut wurde. Weiterhin wird der Schaltplan später auch zur Erstellung des Platinenlayouts benutzt.

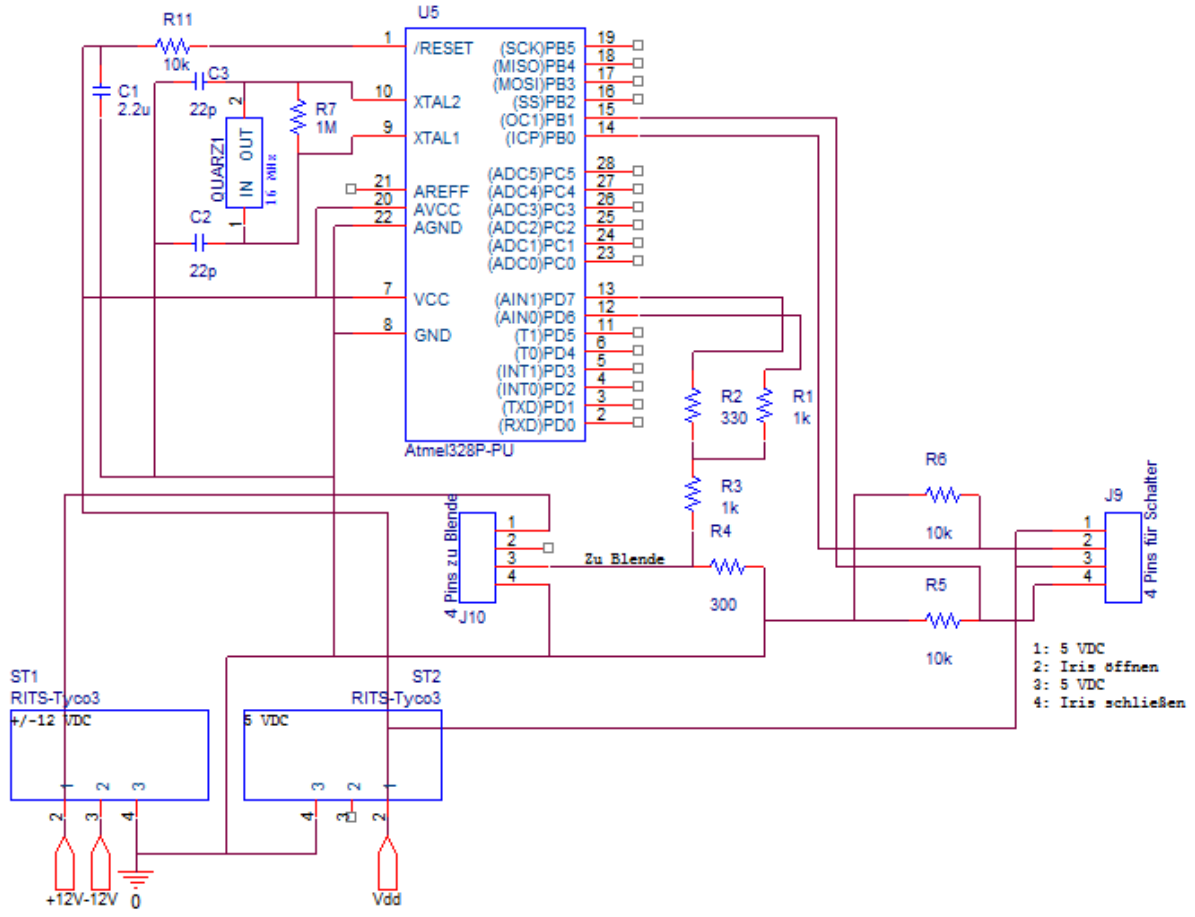


Abbildung 18: Schaltplan für die Erstellung des Prototypen

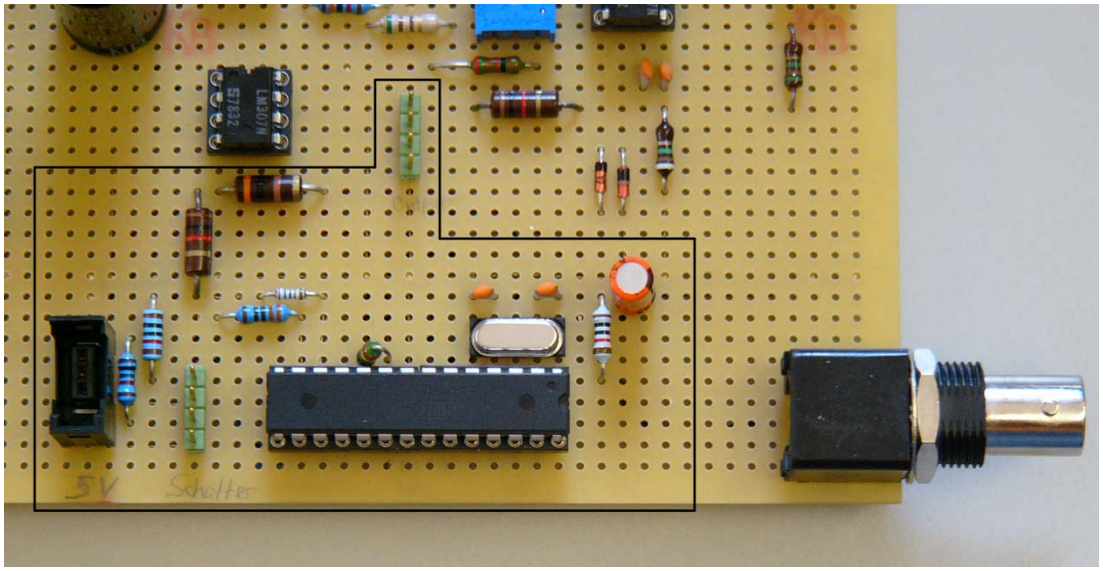


Abbildung 19: Prototyp der Iriskontrolle auf eine Lochrasterplatine

Das oben in Abbildung 18 dargestellte Bild zeigt den in schwarz umrandeten Protoypen der Iriskontrolle. Man erkennt im unteren Teil des Bildes den selbst programmierten Mikrocontroller von Atmel (Atmega328P-PU) sowie zwei Steckerleisten für jeweils vier Pfostenstecker an denen zum einen der Schalter zum bedienen der Iris und zum anderen die Leitung, die zum Objektiv führt, gesteckt wird.

4.2.2 Entwicklung des Mikrofonverstärkers

Als aktiver Verstärker kommt der BB3606BG, ein digital kontrollierbarer Verstärker, von BURR-BROWN zum Einsatz. Dieser gehört zwar schon zu den etwas älteren Modellen von Instrumentenverstärkern, ist jedoch in der Ansteuerung und Inbetriebnahme relativ einfach gestrickt. Desweiteren dient eine nachgeschaltete Operationsverstärkerschaltung dazu unerwünschte tiefe Frequenzen heraus zu filtern und die Ausgangsspannung des Verstärkers um den Faktor 16 herunter zu teilen. Das ist erforderlich, da je nach Anlegen der digitalen Eingangspegel eine Verstärkung von 1 bis 1024 erreicht werden kann. Somit wird maximal eine Verstärkung von 64 erreicht, was mehr als ausreichend ist. Abbildung 19 zeigt den Frequenzgang der reinen Operationsverstärkerschaltung, welche mit einer Wechselspannungsquelle (Amplitude: 16 V) betrieben wurde. Man kann erkennen, dass die über der Frequenz aufgetragenen Spannung ca. 1/16 der Eingangsspannung entspricht . Darüber hinaus liegt die Grenzfrequenz des Filters bei ungefähr 12 Hz, ab der z.B. die störenden Eigenfrequenzen des Windkanals des DNW unterdrückt werden.

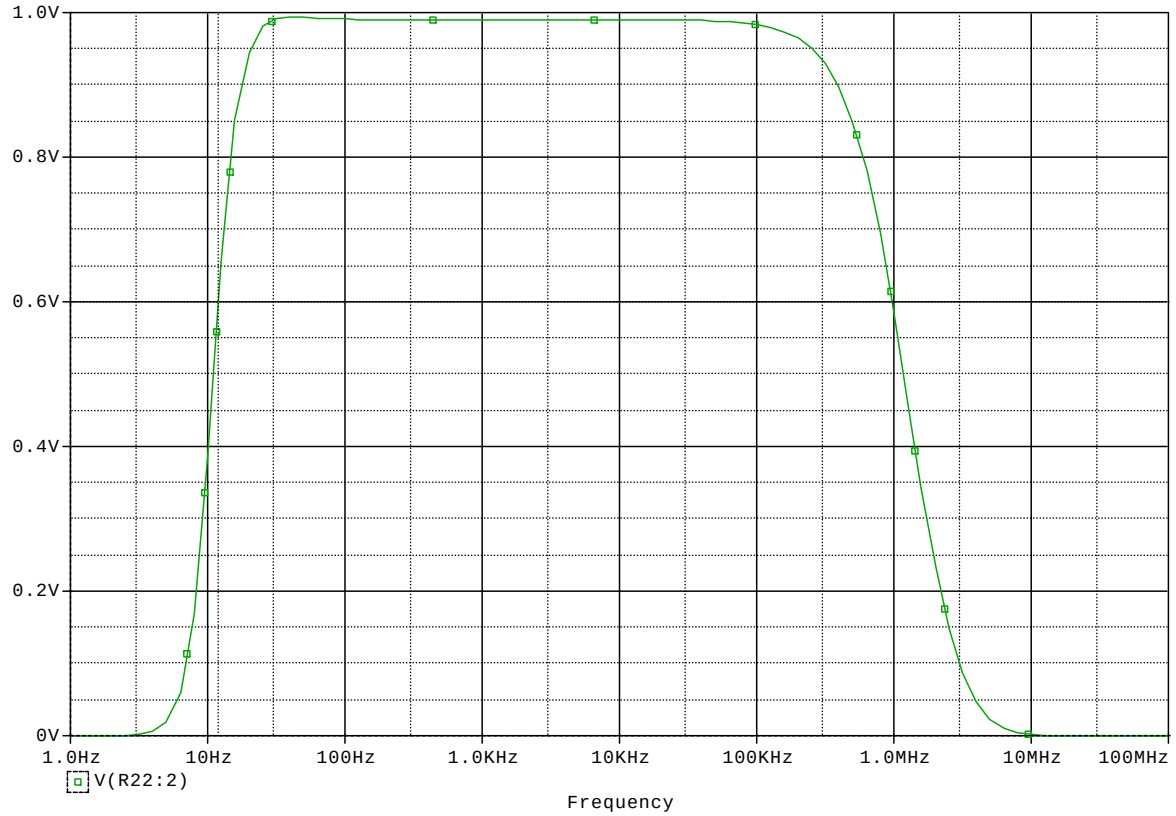


Abbildung 20: Frequenzgang der Operationsverstärkerschaltung (Filter)

Um den Audiverstärker ansteuern zu können und somit seine Verstärkung regeln zu können, muss mit Hilfe eines Mikrokontrollers (µC), in diesem Fall wieder ein Atmel Atmega328P-PU, eine kleine Schaltung entworfen werden, die es ermöglicht an die passenden Eingänge des Verstärkers entsprechende Signale anlegen zu können. Damit der µC das auch leisten kann, muss er mit einer passenden Software bespielt werden.

Der in Abbildung 20 zu sehende Programmablaufplan zeigt grob die für die Programmierung nötigen Maßnahmen um einen reibungslosen Ablauf des Programms zu gewährleisten.

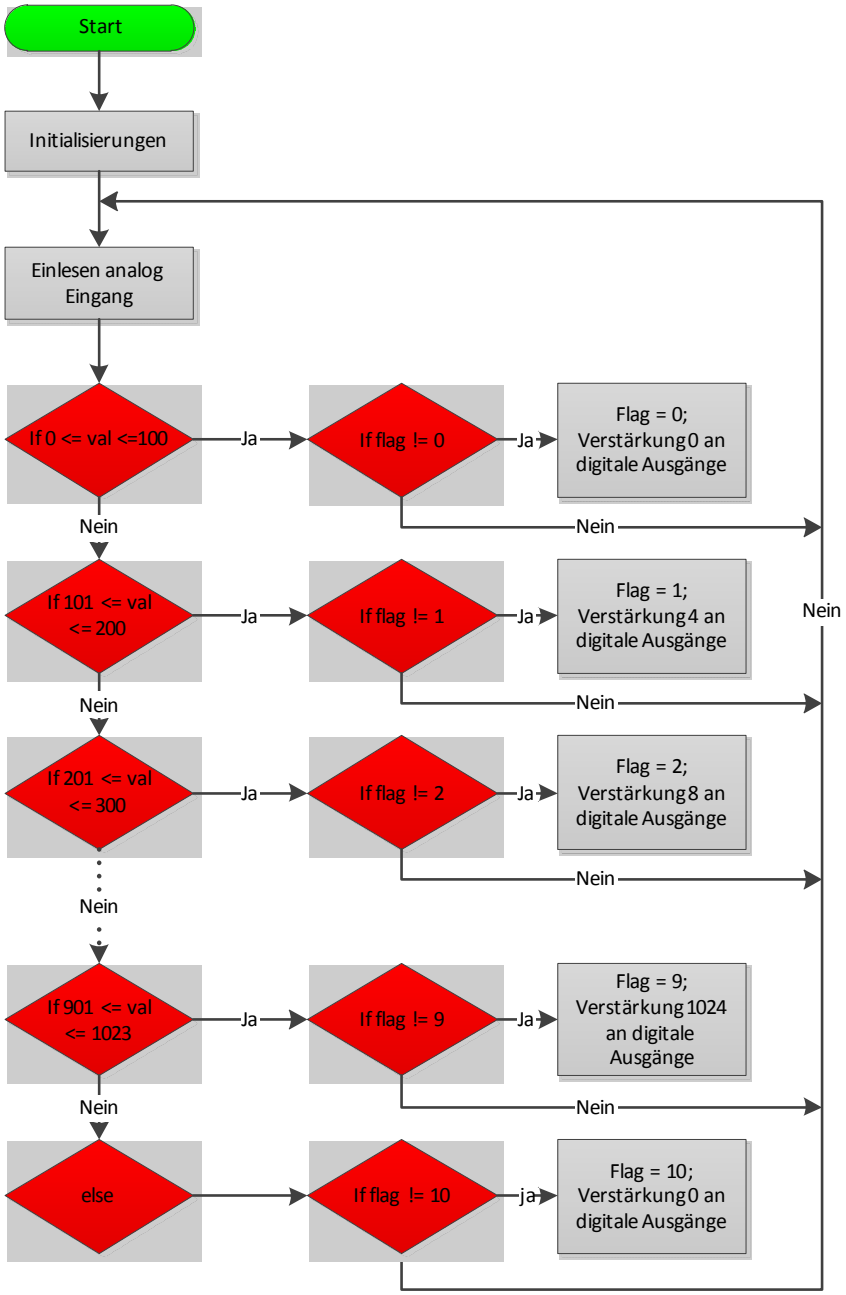


Abbildung 21: Programmablaufplan des µC zur Ansteuerung des Mikrofonverstärkers

4.3 Erstellung des Schaltungsentwurfs der Hauptplatine

Für die Entwicklung der Platine muss ein neues Projekt mit OrCad erstellt werden. Es muss unbedingt auf die genaue Verdrahtung der einzelnen Komponenten geachtet werden um fehler zu vermeiden. Weiter muss darauf geachtet werden, dass für alle sich momentan im Schwenk-/Neigekopf befindlichen Pfostenstecker eine Verbindungsmöglichkeit erstellt wird, damit auch sämtliche Funktion des Kamerasystems gewährleistet werden können.
Abbildung 20 auf der nächsten Seite zeigt den fertigen Schaltungsentwurf welcher zum layouten einer Platine weiter verwendet werden kann. Da das Platinenlayout nicht zum Bestandteil der Studienarbeit gehört, wird an dieser Stelle auf weitere Entwicklungen bezüglich des Layouts verzichtet.

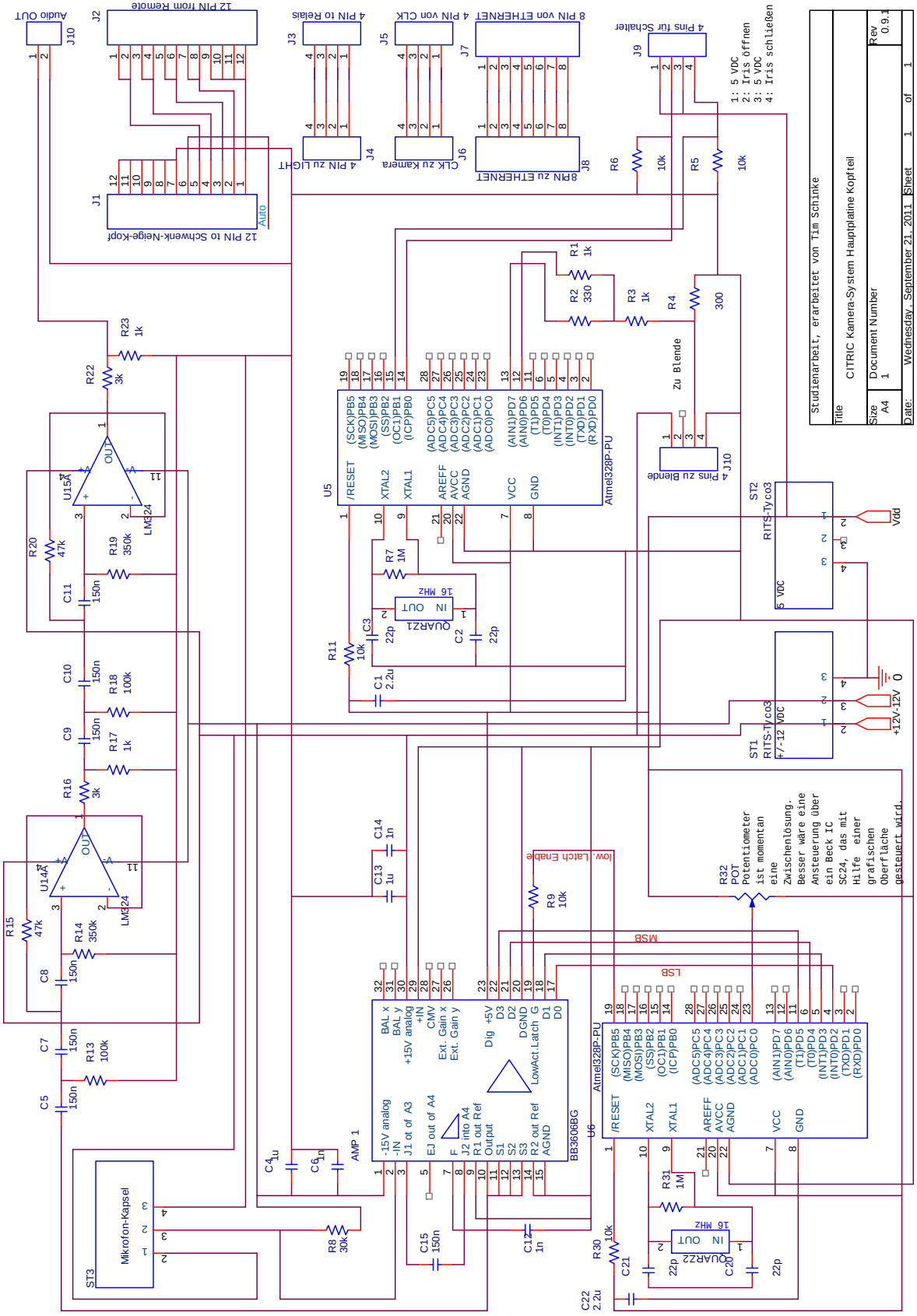


Abbildung 22: Fertiger Schaltungsentwurf

5 Funktionstest

5.1 Betrieb über 6-Pin-Firewirekabel

Sofern der zu benutzende PC über einen Firewire-Anschluss verfügt, kann die Kamera mit dem Desktop-PC über ein sechspoliges Firewirekabel verbunden werden. Andernfalls muss eine zusätzliche Firewire-Karte beschafft und in den Computer eingebaut und falls erforderlich auch installiert werden. In diesem Fall musste eine zusätzliche Firewirekarte in den PC eingebaut werden. Die Karte, vom Herrsteller ARP, wurde über den PCI-Bus intern mit dem Computer verbunden und besitzt die Möglichkeit drei externe Geräte und ein internes Gerät über Firewire zu betreiben. Eine zusätzliche Installation der Karte war nicht nötig.

5.2 Installation der Kamerasoftware

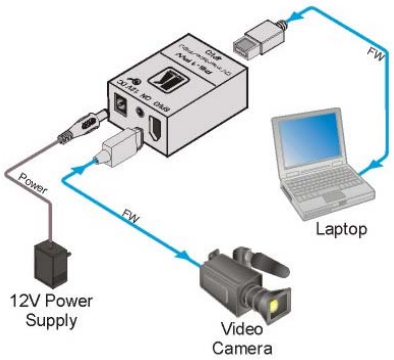
Damit die Kamera (Sony XCD-X710CR) zum leben erweckt werden kann, ist es zwingend erforderlich die passenden Treiber und die Software auf dem mit der Kamera verbundenen PC zu installieren. Für Windows XP in der 32-Bit Variante existiert schon eine passende Anleitung, siehe [4], so dass diese nur Schritt für Schritt bis „*Step 6*“ nacheinander abgearbeitet werden muss. Anschließend wird eine vom DLR selbstentwickelte Software, welche die Kamera bedienbar macht, installiert.

Eine neuere Software inklusive Treibern konnte mit Hilfe von Sony für das Projekt bereitgestellt werden. Diese Software ist in einer 32-Bit sowie in einer 64-Bit Variante verfügbar und hat jedoch den Nachteil, dass die DLR-Software mit den neuen Treibern inkompatibel ist und somit nicht verwendet werden kann.

Nach erfolgreicher Installation von Treiber und Software für die Kamera kann diese an den PC angeschlossen werden und die passende Software gestartet werden. Dies ist entweder die vom DLR geschriebene Software oder, falls die neuere Software von Sony installiert wurde, der „Viewer“. Die Kamera kann nun über den Computer bedient werden.

5.3 Betrieb über 4-Pin-Firewirekabel

Für verschiedene Anwendungsfälle ist es von Vorteil die Kamera mit einem Notebook zu verbinden, wobei die meißten Notebooks einen vierpoligen IEEE 1394a-Stecker (Firewire-Stecker) aufweisen. Bei dieser Art von Steckerbuchse wird keine Versorgungsspannung übertragen, sonder nur die reinen Informationen über vier Adern (1: TPB-, 2: TPB+, 3: TPA-, 4: TPA+).



Da die Kamera über ein sechspoligen Firewire anschluss verfügt, muss ein Adapter die Konnektivität gewährleisten, wobei damit die Spannungsversogung der Kamera noch nicht hergestellt ist. Für diesen Fall, also das auch die Spannungsversorgung an die Kamera geführt wird, muss ein zusätzliches Netzteil verbaut werden, dass den spezifischen Angaben an das Firewireprotokoll gerecht wird. Nach einiger Recherche im Internet wurde ein passender Umsetzer von 4-PIN auf 6-PIN Firewire gefunden der eine passende Spannungsversorgung für die Kamera liefert. Die untenstehnde Tabelle 5.1 zeigt eine Übersicht der verschieden Steckertypen und deren Belegung. Die Bilder der jeweiligen Stecker wurden aus [2] entnommen.

Tabelle 5.1: Übersicht der Firewirestecker und PIN-Belegung

Vierpoliger 1394a-Stecker	sechspoliger 1394a-Stecker	neunpoliger 1394a-Stecker	PIN-Belegung/Bezeichnung
 4 polig	 6 polig	 9 polig	
-	1	8	Vesorgungsspannung +12VDC
-	2	6	Masse
1	3	1	B-
2	4	2	B+
3	5	3	A-
4	6	4	A+
-	2	5	Schirm A-, A+
-	2	9	Schirm B-, B+
-	-	7	Nicht Belegt

6 Zusammenfassung

Die im Rahmen der Studienarbeit entwicklte und hergestellte Platine besitzt nun alle für das Projekt wichtigen Anschlüsse und Funktionen. So wurde ein Microcontroller der Firma Atmel dazu verwendet die eigentlich automatische Iriskontrolle des verwendeten Objektivs manuell bedienbar zu machen. Um die Blende wie gewünscht bedinfähig zu machen, wurde der Controller mit einer zum Teil selbst entwickelten Software bespielt. Mit diesem Hardware/Software Co-Design wurde ein Videosignal (BAS-Signal) erzeugt, mit dem es nun möglich ist die Blende auf- und zufahren zu lassen sowie die Blende in einer gewünschten Position zu halten.

Zusätzlich ist auf der Platine ein Mikrofonverstärker verbaut worden, der es ermöglicht auch unter erhöhtem Lärmpegel noch ausreichend gute Tonqualität wiederzugeben.

Dieser kann mit verschiedenen HIGH/LOW Pegeln an den Eingängen D0 bis D3 in seiner Verstärkung von 4 bis 1024 in Zweierpotenzschritten variiert werden.

7 Ausblick

Nachdem das CITRIC Stroboskop-Kamerasystem für den Rotorversuchstand des DLR erfolgreich getestet wurde, bleibt die Frage, wie das Projekt weiterentwickelt werden kann.

Es würde sich beispielsweise ein Ethernet-Hub im Schwenk-/Neigekopf verbauen lassen, um Netzwerkspeziefische Aufgaben zu lösen, oder auch ein Microcontroller im Bedienpult

implementieren, der Steuerungsaufgaben übernimmt, um etwa einen Autofocus oder eine automatische Iriskontrolle zu realisieren. Um letzteres auch bedienfähig zu machen, könnte man an die bisherigen Bedienelemente Relais schalten um einen Automatikbetrieb zu gewährleisten. Als Microcontroller eignet sich der Embedded Web Controller sc24 der Firma Beck, siehe www.beck-ipc.com/de/products/sc2x/sc24.asp , da er die nötigen Voraussetzungen mitbringt, wie z.B. ausreichende Performance, eine Ethernet-Schnittstelle, genügend I/O-Ports sowie ein Echtzeitmultitasking-Betriebssystem mit verschiedenen Protokollen. Denkbar wäre sogar eine automatische Ausrichtung der Kamera anhand speziefischer Punkte (beispielsweise Fadenkreuz) im Raum was von einem Microcontroller gesteuert wird.

Ein weiteres Projekt könnte auch die Entwicklung eines neuen Treibers für die Sony XCD-X710CR Kamera sein, da von Sony keine weiteren „einzelnen“ Treiber zur Verfügung gestellt werden und der bisherige Treiber unter Windows 7 (64Bit) nicht funktioniert. Dies benötigt jedoch Vorwissen und gute Kentnisse in der Programmierung. Dadurch könnte die bisherige Software des DLR für die Kamera wieder nutzbar gemacht werden.

Denkbar wäre weiterhin einen „Umsetzer“ zu entwickeln, der es ermöglicht, die Kamera (6-Poliger Firewirestecker) mit einem Notebook (4-Poliger Firewirestecker) zu verbinden. Dies könnte vorerst noch mit einer PCMCIA-Firewirekarte und einem zusätzlichen Netzteil getestet werden.

Literaturverzeichnis

[1] Förster, F. M. (19. Juni 2005). Einführung zur Schaltungssimulation am PC mit OrCad PSpice 9.1. Berlin, Berlin, Deutschland. http://www.projektlabor.tu-berlin.de/uploads/media/pspice_tutorial_deutsch.pdf [PDF-Datei], zuletzt geöffnet am 22.07.2011

[2] http://www.kabelstudio.de/DV-Kabel-Firewire-iLINK:_:16.html [html-Datei], zuletzt geöffnet am 27.07.2011

[3] http://de.wikipedia.org/wiki/Fernsehsignal [html-Datei], zuletzt geöffnet am 28.07.2011

[4] http://www.javiervalcarce.eu/wiki/TV_Video_Signal_Generator_with_Arduino [html-Datei], zuletzt geöffnet am 26.08.2011

[5] http://arduino.cc/en/Main/ArduinoBoardUno [html-Datei], zuletzt geöffnet am 26.08.2011

[6] Banzi, M. (Oktober 2008: First Edition). Getting Started with Arduino. O'Reilly Media, Inc, 1005 Gravenstein Highway North, Sebastopol, CA 95470.

[7] Arduino UNO Reference Design http://arduino.cc/en/uploads/Main/arduino-uno-schematic.pdf [PDF-Datei], zuletzt geöffnet am 26.08.2011

[8] Bootloader http://de.wikipedia.org/wiki/Bootloader [html-Datei], zuletzt geöffnet am 02.09.2011

[9] Using an Arduino as an AVR ISP (In-System Programmer)
http://arduino.cc/en/Tutorial/ArduinoISP
[html-Datei], zuletzt geöffnet am 02.09.2011

[10] Getting Started with Arduino
http://arduino.cc/en/Guide/HomePage
[html-Datei], zuletzt geöffnet am 02.09.2011

Anhang

A1 Software Atmel µC für Iriskontrolle

```
/*
  Vertical Bars Pattern PAL TV Signal Generator with Arduino
  Use this code as you want
  2007 Javier Valcarce Garcia, <javier.valcarce@gmail.com>

  Überarbeitet und weiter entwickelt zur Ansteuerung einer AutoIris
  eines Objektivs im Rahmen einer Studienarbeit beim DLR
  08/2011, Tim Schinke
*/

#include <avr/io.h>
#include <util/delay.h>

////////////////////////////////////////////////////////////////////////////////////////////

#define PINA0 6 // LSB, 330 Ohm resistor
#define PINA1 7 // MSB, 1 kOhm resistor
#define PINA2 8 // Iris open
#define PINA3 9 // Iris close

#define LEVEL_SYNC   PORTD &= ~(1 << PINA1); PORTD &= ~(1 << PINA0);
#define LEVEL_BLACK  PORTD &= ~(1 << PINA1); PORTD |=  1 << PINA0;
#define LEVEL_GRAY   PORTD |=  1 << PINA1;  PORTD &= ~(1 << PINA0);
#define LEVEL_WHITE  PORTD |=  1 << PINA1;  PORTD |=  1 << PINA0;

int i = 0;
int irisOpen = 0;
int irisClose = 0;

void setup()
{
  pinMode(PINA2, INPUT);  //PINA2 als Eingang definieren
  pinMode(PINA3, INPUT);  //PINA3 als Eingang definieren
}

////////////////////////////////////////////////////////////////////////////////////////////
inline void vsync_pulse()
{
    // Sync goes low and delay 27.3 microseconds
    LEVEL_SYNC;
    _delay_us(27);
     __asm__("nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t"); // 0.3 microseconds

     // Sync goes high and delay 4.7 microseconds
    LEVEL_BLACK;
    _delay_us(4);

__asm__("nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""n
op\n\t"); // 0.7 microseconds
}

////////////////////////////////////////////////////////////////////////////////////////////
inline void equal_pulse()
{
    // Sync pulse goes low and delay 2.3 microseconds
```

```
    LEVEL_SYNC;
    _delay_us(2);
    __asm__("nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t");
    // Sync pulse goes high and delay 29.7 microseconds
    LEVEL_BLACK;
    _delay_us(29);

__asm__("nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""n
op\n\t");
}
////////////////////////////////////////////////////////////////////////////////////////////
inline void hsync_pulse()
{
    // Front Porch - high for 1.65 microseconds
    LEVEL_BLACK;
    _delay_us(1);

__asm__("nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t");

    // Generate sync pulse 4.7 microseconds
   LEVEL_SYNC;
   _delay_us(4);

__asm__("nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""n
op\n\t");

   // Back Porch 5.6 microseconds.
    LEVEL_BLACK;
    _delay_us(5);

__asm__("nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t""nop\n\t");
}

////////////////////////////////////////////////////////////////////////////////////////////
int main()
{
    register unsigned int line;

    line = 0;
    DDRD = 0xF0; // PORTD,  pin 1 to 7 are outputs

    while(1)
    {
        irisOpen = digitalRead(PINA2);    //PINA2 abfragen
        irisClose = digitalRead(PINA3);   //PINA2 abfragen

        //Abfrage wenn Iris geoeffnet werden soll//////////////////////////////////////////////
              if( irisOpen == HIGH )
        {
           // 1:
                vsync_pulse(); vsync_pulse();
                    // 2:
                    vsync_pulse(); vsync_pulse();
                    // 3:
                    vsync_pulse(); equal_pulse();
                    // 4:
                    equal_pulse(); equal_pulse();
                    // 5:
```

```
            equal_pulse(); equal_pulse();

            // 6-310 (305 lines):
            for( i = 0; i < 305; i++)
            {
                        hsync_pulse();
                        LEVEL_GRAY; _delay_us(10);
                        LEVEL_BLACK; _delay_us(16);
                        LEVEL_WHITE; _delay_us(8);
                        LEVEL_BLACK; _delay_us(16);
                        LEVEL_GRAY; _delay_us(2);
            }

            // 311:
            equal_pulse(); equal_pulse();
            // 312:
            equal_pulse(); equal_pulse();
            // 313:
        equal_pulse(); vsync_pulse();
            // 314:
            vsync_pulse(); vsync_pulse();
            // 315:
            vsync_pulse(); vsync_pulse();
            // 316:
            equal_pulse(); equal_pulse();
            // 317:
            equal_pulse(); equal_pulse();

            // 318-622 (305 lines):
            for( i = 0; i < 305; i++)
            {
                        hsync_pulse();
                        LEVEL_GRAY; _delay_us(10);
                        LEVEL_BLACK; _delay_us(16);
                        LEVEL_WHITE; _delay_us(8);
                        LEVEL_BLACK; _delay_us(16);
                        LEVEL_GRAY; _delay_us(2);
            }

            // 623:
            equal_pulse(); equal_pulse();
            // 624:
            equal_pulse(); equal_pulse();
            // 625:
            equal_pulse(); equal_pulse();
}

//Abfrage wenn Iris geschlossen werden soll//////////////////////////////////////////
else if( irisClose == HIGH )
{
   // 1:
        vsync_pulse(); vsync_pulse();
            // 2:
            vsync_pulse(); vsync_pulse();
            // 3:
            vsync_pulse(); equal_pulse();
            // 4:
            equal_pulse(); equal_pulse();
```

```
            // 5:
            equal_pulse(); equal_pulse();

            // 6-310 (305 lines):
            for( i = 0; i < 305; i++)
            {
                        hsync_pulse();
                        LEVEL_GRAY; _delay_us(14);
                        LEVEL_BLACK; _delay_us(2);
                        LEVEL_WHITE; _delay_us(30);
                        LEVEL_BLACK; _delay_us(2);
                        LEVEL_GRAY; _delay_us(4);
            }

            // 311:
            equal_pulse(); equal_pulse();
            // 312:
            equal_pulse(); equal_pulse();
            // 313:
        equal_pulse(); vsync_pulse();
            // 314:
            vsync_pulse(); vsync_pulse();
            // 315:
            vsync_pulse(); vsync_pulse();
            // 316:
            equal_pulse(); equal_pulse();
            // 317:
            equal_pulse(); equal_pulse();

            // 318-622 (305 lines):
            for( i = 0; i < 305; i++)
            {
                        hsync_pulse();
                        LEVEL_GRAY; _delay_us(14);
                        LEVEL_BLACK; _delay_us(2);
                        LEVEL_WHITE; _delay_us(30);
                        LEVEL_BLACK; _delay_us(2);
                        LEVEL_GRAY; _delay_us(4);
            }

            // 623:
            equal_pulse(); equal_pulse();
            // 624:
            equal_pulse(); equal_pulse();
            // 625:
            equal_pulse(); equal_pulse();
}

//Abfrage wenn Iris still stehen soll//////////////////////////////////////////////
else
{
        // 1:
        vsync_pulse(); vsync_pulse();
            // 2:
            vsync_pulse(); vsync_pulse();
            // 3:
            vsync_pulse(); equal_pulse();
            // 4:
```

```
                equal_pulse(); equal_pulse();
                // 5:
                equal_pulse(); equal_pulse();

                // 6-310 (305 lines):
                for( i = 0; i < 305; i++)
                {
                        hsync_pulse();
                        LEVEL_GRAY; _delay_us(49);
                        LEVEL_WHITE; _delay_us(0);
                        LEVEL_BLACK; _delay_us(4);              }

                // 311:
                equal_pulse(); equal_pulse();
                // 312:
                equal_pulse(); equal_pulse();
                // 313:
            equal_pulse(); vsync_pulse();
                // 314:
                vsync_pulse(); vsync_pulse();
                // 315:
                vsync_pulse(); vsync_pulse();
                // 316:
                equal_pulse(); equal_pulse();
                // 317:
                equal_pulse(); equal_pulse();

                // 318-622 (305 lines):
                for( i = 0; i < 305; i++)
                {
                        hsync_pulse();
                        LEVEL_GRAY; _delay_us(49);
                        LEVEL_WHITE; _delay_us(0);
                        LEVEL_BLACK; _delay_us(4);              }

                // 623:
                equal_pulse(); equal_pulse();
                // 624:
                equal_pulse(); equal_pulse();
                // 625:
                equal_pulse(); equal_pulse();

        }
    }
}
```

A2 Software Atmel µC für Mikrofonverstärker

```
#include <avr/io.h>
#include <util/delay.h>

int analogPin = 0;     // potentiometer wiper (middle terminal) connected to analog pin 3
                       // outside leads to ground and +5V
int val = 0;           // variable to store the value read

int flag = -1;

void setup()
{
 Serial.begin(9600);         //  setup serial

 pinMode(2, OUTPUT);
 pinMode(3, OUTPUT);
 pinMode(4, OUTPUT);
 pinMode(5, OUTPUT);
 pinMode(13, OUTPUT);
}

void loop()
{
    val = analogRead(analogPin);    // read the input pin

    if(0 <= val && 100 >= val)
    {
      if(flag != 0)
      {
        flag = 0;
        digitalWrite(2, LOW);
        digitalWrite(3, LOW);
        digitalWrite(4, LOW);
        digitalWrite(5, LOW);
        digitalWrite(13, LOW);
        _delay_ms(50);
        digitalWrite(13, HIGH);

        Serial.print("Sensor = " );
        Serial.print(val);
        Serial.print("\t Output = 0 0 0 0");
        Serial.println("\t Verstaerkung = KEINE");
      }
    }
    else if(101 <= val && 200 >= val)
    {
      if(flag != 1)
      {
        flag = 1;
        digitalWrite(2, LOW);
        digitalWrite(3, LOW);
        digitalWrite(4, HIGH);
        digitalWrite(5, LOW);
        digitalWrite(13, LOW);
        _delay_ms(50);
        digitalWrite(13, HIGH);
```

```
    Serial.print("Sensor = " );
    Serial.print(val);
    Serial.print("\t Output = 0 1 0 0");
    Serial.println("\t Verstaerkung = 4");
   }
}
else if(201 <= val && 300 >= val)
{
  if(flag != 2)
  {
    flag = 2;
    digitalWrite(2, HIGH);
    digitalWrite(3, LOW);
    digitalWrite(4, HIGH);
    digitalWrite(5, LOW);
    digitalWrite(13, LOW);
    _delay_ms(50);
    digitalWrite(13, HIGH);

    Serial.print("Sensor = " );
    Serial.print(val);
    Serial.print("\t Output = 0 1 0 1");
    Serial.println("\t Verstaerkung = 8");
  }
}
else if(301 <= val && 400 >= val)
{
  if(flag != 3)
  {
    flag = 3;
    digitalWrite(2, LOW);
    digitalWrite(3, HIGH);
    digitalWrite(4, HIGH);
    digitalWrite(5, LOW);
    digitalWrite(13, LOW);
    _delay_ms(50);
    digitalWrite(13, HIGH);

    Serial.print("Sensor = " );
    Serial.print(val);
    Serial.print("\t Output = 0 1 1 0");
    Serial.println("\t Verstaerkung = 16");
  }
}
else if(401 <= val && 500 >= val)
{
  if(flag != 4)
  {
    flag = 4;
    digitalWrite(2, LOW);
    digitalWrite(3, LOW);
    digitalWrite(4, LOW);
    digitalWrite(5, HIGH);
    digitalWrite(13, LOW);
    _delay_ms(50);
    digitalWrite(13, HIGH);
```

```
    Serial.print("Sensor = " );
    Serial.print(val);
    Serial.print("\t Output = 1 0 0 0");
    Serial.println("\t Verstaerkung = 32");
   }
}
else if(501 <= val && 600 >= val)
{
  if(flag != 5)
  {
    flag = 5;
    digitalWrite(2, HIGH);
    digitalWrite(3, LOW);
    digitalWrite(4, LOW);
    digitalWrite(5, HIGH);
    digitalWrite(13, LOW);
    _delay_ms(50);
    digitalWrite(13, HIGH);

    Serial.print("Sensor = " );
    Serial.print(val);
    Serial.print("\t Output = 1 0 0 1");
    Serial.println("\t Verstaerkung = 64");
   }
}
else if(601 <= val && 700 >= val)
{
  if(flag != 6)
  {
    flag = 6;
    digitalWrite(2, LOW);
    digitalWrite(3, HIGH);
    digitalWrite(4, LOW);
    digitalWrite(5, HIGH);
    digitalWrite(13, LOW);
    _delay_ms(50);
    digitalWrite(13, HIGH);

    Serial.print("Sensor = " );
    Serial.print(val);
    Serial.print("\t Output = 1 0 1 0");
    Serial.println("\t Verstaerkung = 128");
   }
}
else if(701 <= val && 800 >= val)
{
  if(flag != 7)
  {
    flag = 7;
    digitalWrite(2, LOW);
    digitalWrite(3, LOW);
    digitalWrite(4, HIGH);
    digitalWrite(5, HIGH);
    digitalWrite(13, LOW);
    _delay_ms(50);
    digitalWrite(13, HIGH);

    Serial.print("Sensor = " );
```

```
    Serial.print(val);
    Serial.print("\t Output = 1 1 0 0");
    Serial.println("\t Verstaerkung = 256");
   }
 }
 else if(801 <= val && 900 >= val)
 {
  if(flag != 8)
  {
    flag = 8;
    digitalWrite(2, HIGH);
    digitalWrite(3, LOW);
    digitalWrite(4, HIGH);
    digitalWrite(5, HIGH);
    digitalWrite(13, LOW);
    _delay_ms(50);
    digitalWrite(13, HIGH);

    Serial.print("Sensor = " );
    Serial.print(val);
    Serial.print("\t Output = 1 1 0 1");
    Serial.println("\t Verstaerkung = 512");
   }
 }
 else if(901 <= val && 1023 >= val)
 {
  if(flag != 9)
  {
    flag = 9;
    digitalWrite(2, HIGH);
    digitalWrite(3, HIGH);
    digitalWrite(4, HIGH);
    digitalWrite(5, HIGH);
    digitalWrite(13, LOW);
    _delay_ms(50);
    digitalWrite(13, HIGH);

    Serial.print("Sensor = " );
    Serial.print(val);
    Serial.print("\t Output = 1 1 1 1");
    Serial.println("\t Verstaerkung = 1024");
   }
 }
 else
 {
  if(flag != 10)
  {
    flag = 10;
    digitalWrite(2, LOW);
    digitalWrite(3, LOW);
    digitalWrite(4, LOW);
    digitalWrite(5, LOW);
    digitalWrite(13, LOW);
    _delay_ms(50);
    digitalWrite(13, HIGH);

    Serial.print("Sensor = " );
    Serial.print(val);
```

```
        Serial.print("\t Output = 0 0 0 0");
        Serial.println("\t Verstaerkung = KEINE");
      }
    }
}
```

A2 CD-Verzeichnis

Studienarbeit_CD

- Berichte
  - Studienarbeit.doc
  - Studienarbeit.pdf
- Zeitplanung
  - Gantt-Diagramm
  - Netzplandiagramm
  - Zeitplan
- Verwendete_Literatur
  - Offline_Webseiten
    - …
    - …
- Software
  - Sony
    - Software/Treiber für x86
    - Software/Treiber für x64
  - DLR
    - Software
    - Treiber für x86