

Projektarbeit  
im Studiengang  
AIB/CNB

# Webserver für ein embedded Board mit AVR-Prozessor

Dokumentation

Betreuer : Dr. Jiri Spale

Vorgelegt am : 14.07.2014

Vorgelegt von : Christian Würthner  
Jan-Henrik Preuß  
Ann-Sophie Dietrich  
Marcel Schlipf



## **Abstract**

Die Aufgabe des Semesterprojektes bestand darin, auf einem Pollin-Net-IO-Board, auf welchem als Prozessor ein ATmega644p läuft, einen Webserver aufzusetzen. Als Vorlage für den Webserver gab es eine Version von Ulrich Radig, welche man nutzen, verbessern und ausbauen soll.

Mit Hilfe des Webserver soll es möglich sein, den Status der sich auf dem Board befindenden Pins anzeigen und diese manipulieren zu lassen. Die Anzeige, sowie die Manipulation soll über eine Webseite, auf welcher das Board grafisch dargestellt wird geschehen. Zudem soll man pro Pin auf der Webseite eine Beschreibung und Funktion hinterlegen, welche gespeichert bleibt und abrufbar ist. Die Webseite soll den aktuellen Platinenstand grafisch anzeigen, Änderungen am Board sollen so schnell es möglich ist auf der Webseite grafisch angezeigt werden.



## Inhaltsverzeichnis

Abstract . . . . .	i
Inhaltsverzeichnis . . . . .	vi
Abbildungsverzeichnis . . . . .	vii
Tabellenverzeichnis . . . . .	ix
Abkürzungsverzeichnis . . . . .	xi
1 Einleitung . . . . .	1
1.1 Versionskontrolle . . . . .	1
2 Aufgabenstellung . . . . .	3
3 Team . . . . .	5
3.1 Teammitglieder . . . . .	5
3.1.1 Christian Würthner . . . . .	5
3.1.2 Jan Henrik Preuß . . . . .	5
3.1.3 Ann-Sophie Dietrich . . . . .	5
3.1.4 Marcel Schlipf . . . . .	6
4 Projektplanung . . . . .	7
4.1 Zeitlicher Ablauf . . . . .	7
5 Hardware . . . . .	9
5.1 AVR Net-IO-Board . . . . .	9
5.1.1 Technische Daten . . . . .	9
5.2 Mikrocontroller . . . . .	10

---

5.2.1	ENC28J60 . . . . .	10
5.3	Fuse Bits . . . . .	11
6	Programmieren und Debuggen . . . . .	13
6.1	ISP . . . . .	13
6.2	SPI . . . . .	13
6.3	JTAG . . . . .	13
7	Recherche . . . . .	15
7.1	Ethersex . . . . .	15
7.2	Elektronik 2000 . . . . .	16
7.3	Ulrich Radig . . . . .	17
8	Ausgewählte Lösung . . . . .	19
8.1	Der Webserver . . . . .	19
8.1.1	Änderungen . . . . .	19
8.1.2	Einbindung der Website . . . . .	20
8.2	Die Website . . . . .	20
8.2.1	JavaScript . . . . .	20
9	Technischer Hintergrund . . . . .	25
9.1	Server/Webseite Kommunikation . . . . .	25
9.1.1	Erster Ansatz . . . . .	25
9.1.2	Polling oder Pushing . . . . .	26
9.1.3	Aufbau der Server-Kommunikation . . . . .	28
9.1.4	Implementierung der REST-Schnittstelle auf dem Server . . . . .	29
9.1.5	Erweiterung der POST-Parameter . . . . .	30
9.1.6	Implementierung der REST-Schnittstelle auf dem Client . . . . .	30
9.2	Werkzeuge . . . . .	31
9.2.1	Das Atmel Studio . . . . .	31

---

9.2.2	AVRDUDE . . . . .	33
9.2.3	HTML Header Compiler . . . . .	34
9.2.4	AVRISPMkII . . . . .	36
9.2.5	AVRJTAGICEmkII . . . . .	36
10	Benutzerhandbuch . . . . .	37
10.1	WinAVR Projekt zu Atmel Studio . . . . .	37
10.2	Einen Mikrocontroller austauschen . . . . .	39
10.3	ISP-Programmer anschließen . . . . .	41
10.4	Einrichten eines neuen Mikrocontrollers . . . . .	42
10.5	HTML Header Compiler . . . . .	45
10.6	Konfiguration des Webserver . . . . .	46
10.7	Debuggen über JTAG . . . . .	46
10.8	Die Website . . . . .	48
10.8.1	Der Status Tab . . . . .	48
10.8.2	Der Favoriten Tab . . . . .	48
10.8.3	Der Einstellungs Tab . . . . .	48
10.8.4	Nutzerspezifische Skripte . . . . .	49
11	Rückblick . . . . .	55
11.1	Soll/Ist-Vergleich . . . . .	55
11.2	Verworfenen Varianten . . . . .	56
11.3	Eigenbewertung . . . . .	56
12	Ausblick . . . . .	59
12.1	Struktur des neuen Systems . . . . .	59
12.2	Änderungen an der Webseite/Server-Schnittstelle . . . . .	60
12.3	Änderungen an der Webseite . . . . .	61
12.3.1	Einstellungen . . . . .	61

12.3.2 Implementierung der neuen Schnittstelle . . . . .	61
12.3.3 Scriptfunktionen . . . . .	62
12.3.4 Auswahl des darzustellenden Boards . . . . .	62
12.3.5 Verwaltung der Boards im System . . . . .	63
12.4 Der neue Server . . . . .	63
12.5 Herangehensweise an das Projekt . . . . .	63
12.5.1 Einpflegen des Raspberry Pi . . . . .	63
12.5.2 Betreiben mehrerer Pollin Net-IO Boards . . . . .	63
12.5.3 Verlagern der Skriptfunktionen auf den Server . . . . .	64
13 Fazit . . . . .	65
14 Anhang . . . . .	67
Literaturverzeichnis . . . . .	69



## Abbildungsverzeichnis

Abbildung 1: Zeitplan . . . . .	7
Abbildung 2: AVR-NET-IO - Pollin GmbH . . . . .	9
Abbildung 3: Fuse-Bits im Atmel Studio (ATmega-664P) . . . . .	12
Abbildung 4: Ethersex menuconfig . . . . .	15
Abbildung 5: Ethersex make project . . . . .	16
Abbildung 6: JavaScript um /rest/values abzufragen und in ein Objekt zu parsen	31
Abbildung 7: DeviceProgramming . . . . .	32
Abbildung 8: index.htm . . . . .	34
Abbildung 9: webpage.h . . . . .	35
Abbildung 10: Atmel Studio, Neues Projekt . . . . .	37
Abbildung 11: Atmel Studio, Device Selection . . . . .	37
Abbildung 12: Atmel Studio, Solution Explorer . . . . .	38
Abbildung 13: Atmel Studio, Add Existing Item 1 . . . . .	38
Abbildung 14: Atmel Studio, Add Existing Item 2 . . . . .	38
Abbildung 15: Schraubendreher am Controller . . . . .	39
Abbildung 16: Der gelöste Mikrocontroller . . . . .	40
Abbildung 17: Der Sockel auf dem AVR-Net-IO . . . . .	40
Abbildung 18: Markierung zum Einbau . . . . .	41
Abbildung 19: Schematische Darstellung des ISP Anschlusses . . . . .	41
Abbildung 20: Der Adapter . . . . .	42
Abbildung 21: DeviceProgramming . . . . .	43

Abbildung 22: AVRDUDE Ausgabe . . . . .	44
Abbildung 23: BuildWebpage.sh für Linux . . . . .	45
Abbildung 24: BuildWebpage.bat für Windows . . . . .	46
Abbildung 25: JTAG Pins . . . . .	47
Abbildung 26: Beispielskript: Mehrere Lichter blinken lassen . . . . .	50
Abbildung 27: Beispielskript: Werte werden in digitale Werte umgewandelt . . . . .	51
Abbildung 28: Beispielskript: Lichterlauf . . . . .	52
Abbildung 29: Schematischer Aufbau der Erweiterung . . . . .	59
Abbildung 30: Select Element . . . . .	62
Abbildung 31: Schaltplan AVR-Net-IO Board . . . . .	67
Abbildung 32: Anforderungsdefinition zu Beginn des Projektes . . . . .	68

## Tabellenverzeichnis

Tabelle 2: Mikrocontroller im Vergleich . . . . .	10
Tabelle 3: Die Bedeutung der einzelnen Fuse-Bits (ATmega-664P) . . . . .	11
Tabelle 4: Fuse Einstellungen (ATmega-664P) . . . . .	12
Tabelle 5: Die Pinbelegung für den 6 und 10 poligen Anschluss [mik14] . . . . .	42
Tabelle 6: Auslesen und setzen von Fuse-Bits des ATmega644P mit AVRDUDE	43
Tabelle 7: Auszug AVRDUDE Parameter . . . . .	44
Tabelle 8: Parameter des HTML Header Compiler . . . . .	45
Tabelle 10: JTAG Connections [Atm] . . . . .	47
Tabelle 12: Einige wichtige Skriptkommandos . . . . .	49
Tabelle 13: Soll/Ist Vergleich . . . . .	56



## Abkürzungsverzeichnis

**AJAX** Asynchronous JavaScript and XML

**CMD** Command Control

**DDR** Data Direction Register

**DNS** Domain Name System

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

**HHC** HTML Header Compiler

**HTML** Hyper Text Markup Language

**HTTP** Hyper Text Transfer Protocol

**IDE** Integrated Development Environment

**ISP** In System Programming

**JSON** JavaScript Object Notation

**JTAG** Joint Test Action Group

**NTP** Network Time Protocol

**REST** Representational State Transfer

**RSS** Rich Site Summary

**SOAP** Simple Object Access Protocol

**SPI** Serial Peripheral Interface

**Telnet** Telecommunication Network

**URL** Uniform Resource Locator

**USART** Universal Synchronous and Asynchronous Serial Receiver and Transmitter

**WDT** Watchdog timer

**WOL** Wake on Lan

**XML** Extensible Markup Language



## 1 Einleitung

Durch die immer weiter fortschreitende Vernetzung unserer alltäglichen Elektronik wird das Verlangen nach netzwerkfähigen Mikrocontrollern immer größer.

Wenn zum Beispiel der moderne Kühlschrank erkennen soll, wie gut er gerade gefüllt ist oder die Heizung melden soll wie warm das Wasser ist, muss ein entsprechend ausgestatteter Controller diese Informationen an den Nutzer melden können, welcher dann entsprechend darauf reagieren oder über gespeicherte Funktionen den Controller entsprechend reagieren lassen kann.

Hier greift unser Projekt da mit dem Mikrocontroller und seinen verschiedenen Ein- und Ausgängen unterschiedlichste Anwendungen ermöglicht werden.

### 1.1 Versionskontrolle

Für die Versionskontrolle haben wir für unsere Projekt Dateien auf GitHub. Auf die Funktionsweise von Git wird an dieser Stelle nicht weiter eingegangen, weiterführende Hilfestellung gibt es auf der offiziellen Hilfe-Seite <https://help.github.com/>. Die Projektseite kann unter der Adresse <https://github.com/doofmars/Embedded-Webserver> erreicht werden. Um das Repository zu „clonen“ benötigt man folgende Adresse: <https://github.com/doofmars/Embedded-Webserver.git>. Alternativ können die Projektdaten unter <https://github.com/doofmars/Embedded-Webserver/archive/master.zip> komprimiert heruntergeladen werden.





## 2 Aufgabenstellung

Die Aufgabe des Semesterprojektes bestand darin, einen Webserver auf einem Mikrocontroller aufzusetzen.

Bei der vorgegebenen Hardware handelt es sich um ein Pollin-Net-IO-Board, auf welchem als Prozessor ein ATmega644p läuft. Als Vorlage für den Webserver gab es eine Version von Ulrich Radig, welche das Team nutzen, verbessern und ausbauen soll.

Der Nutzer greift auf das Board über eine Webseite zu, auf welcher das Board grafisch angezeigt wird. Anhand dieser Grafik bekommt der Nutzer einen Überblick und kann den Status der Pins ablesen, sowie diese via Mausklick manipulieren.

Zudem ist es dem Nutzer möglich, pro Pin auf der Webseite eine Beschreibung und Funktion zu hinterlegen, welche gespeichert bleibe und somit bei einem Neustart wieder verfügbar sind. Somit kann sich der Nutzer kleine Skriptfunktionen und Pins, welcher er häufig nutzt als Favoriten setzen und mithilfe der eigenen Beschreibung schneller erfassen, welcher Pin welche Funktion hat.

Die Webseite soll den aktuellen Platinenstand grafisch anzeigen, Änderungen am Board sollen so schnell es möglich ist auf der Webseite grafisch angezeigt werden, genauso sollen Änderungen via Mausklick sofort als Befehl an das Board übertragen und ausgeführt werden.

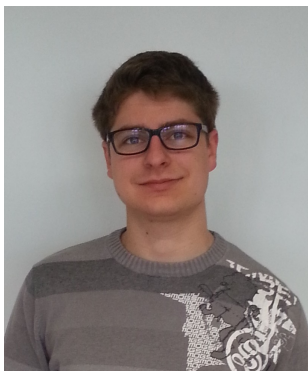
Mit dem ATmega644p ist der gesamte Speicherverbrauch (Also Server und Webseite) auf 64kB begrenzt, was die Herausforderung des Projektes ausmacht. Die Vorlage des Webserver von Ulrich Radig, an welche sich das Team halten und diese weiterentwickeln soll musste daher zunächst ausgemistet werden. Auch bei der Entwicklung der Webseite ist der Speicherverbrauch eine ständige Herausforderung.



## 3 Team

### 3.1 Teammitglieder

#### 3.1.1 Christian Würthner



- Studiengang: Allgemeine Informatik
- Semester: 4
- Vorkenntnisse: Java, HTML, JavaScript, C, C++, SQL
- Posten: Gruppenleiter

#### 3.1.2 Jan Henrik Preuß



- Studiengang: Allgemeine Informatik
- Semester: 4
- Vorkenntnisse: Java, HTML, JavaScript, C, C++, SQL
- Posten: Stellvertretender Gruppenleiter und stellvertretender Schriftführer

#### 3.1.3 Ann-Sophie Dietrich



- Studiengang: Allgemeine Informatik
- Semester: 4
- Vorkenntnisse: Java, C, C++, SQL
- Posten: Schriftführer

### 3.1.4 Marcel Schlipf



- Studiengang: Computer Networking
- Semester: 4
- Vorkenntnisse: Java, HTML, JavaScript, C, jQuery, SQL
- Posten: Verantwortlicher Dokumentation

## 4 Projektplanung

### 4.1 Zeitlicher Ablauf

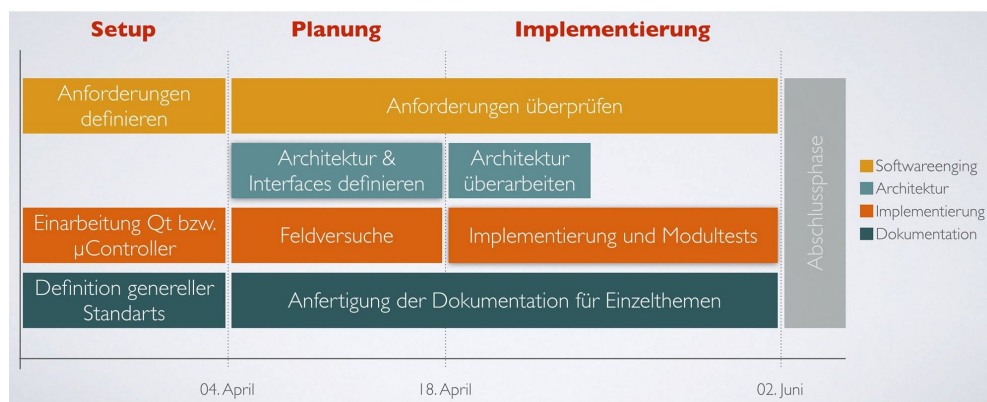


Abbildung 1: Zeitplan

Bei der Projektplanung wurde zu Beginn des Semesterprojektes ein Zeitplan erstellt, mit genauen Terminen und Meilensteinen, an welchem sich das Team orientierte und an dessen zeitlichen Begrenzungen sich das Team hielt. Zudem wurden zu Beginn Anforderungen definiert (siehe Anhang), welche bis zum Ende des Projektes abgearbeitet wurden. Anbei eine tabellarische Auflistung unseres zeitlichen Verlaufs:

26.03.2014	Erstes Einlesen in die Problematik Postenverteilung Festlegen von Standards (Git, Felix, ... ) Erstellung des Zeitplans
02.04.2014	Board in Betrieb genommen GitHub eingerichtet Dokumentation begonnen Problematik von Speicher des ATmega32 Ausweichen auf ATmega644p vorgeschlagen
<b>Meilenstein</b> 04.04.2014	Setup-Phase abgeschlossen Anforderungen sind definiert es wurde sich in die Themengebiete eingelesen Standards wurden definiert und umgesetzt
<b>Meilenstein</b> 18.04.2014	Planungs-Phase abgeschlossen

	Architektur und Interfaces wurden definiert
	Feldversuche (Radig as Is läuft auf Board) abgeschlossen
	Doku wurde begonnen
30.04.2014	Debugger erhalten
	ATmega644p in Betrieb genommen
	Implementierung an Webseite
	Weiterarbeit an Doku
07.05.2014	Board: Pins schaltbar
	Webseite: favorite.js fertig
14.05.2014	Webseite: Übersichtseite fertig
	Seitenleiste verwendbar
	Pins setzen möglich
21.05.2014	Webseite: Feinschliff
	Abänderung der Favoriten zu einer Liste aller Pins
	Pins zu Ein- und Ausgänge schaltbar
28.05.2014	Webserver minimiert
	Skriptfunktionen hinterlegbar
	Favoriten in lokale Datenbank ausgelagert
<b>Meilenstein</b> 02.06.2014	Implementierungsphase abgeschlossen
	Code-Freeze
bis 30.06.2014	Code-Clean-Up
	Feinschliff Dokumentation
	Erstellen von Plaktaten

## 5 Hardware

### 5.1 AVR Net-IO-Board



Abbildung 2: AVR-NET-IO - Pollin GmbH

Auf dem AVR Net-IO-Board sind der Webserver und die Webseite gespeichert. Die verschiedenen Pins lassen sich über die Webseite ansteuern und manipulieren. Die Platine des AVR-Net-IO ist für den einfachen Anschluss der externen Anwendungen gedacht. Der Fokus liegt hier, je nach verwendetem Anschluss-Baord, nicht darauf eine optimale Gestaltung zu haben, sondern die einzelnen Sensoren eher lose an die Platine anzuschließen. Dies geschieht, wie in der Elektronik oft üblich, über die Verwendung von Breadboards, blanke Steckbretter auf welchen Schaltungen konzipiert werden können. Deswegen haben die Analog zu Digital Wandler Schraubverbindungen und auch die im Handel verfügbare Anschlussplatine für den 25 Poligen D-Sub Stecker hat Schraub-Verbindungen. Für die Demonstration haben wir uns spezielle Schaltungen überlegt, welche simpel an die externen Anschlüsse des AVR-Net-IO Angeschlossen werden können. Die Verteilung, welcher Pin an welchem Anschluss ist, kann dem Schaltplan entommen werden (siehe Anhang, Abbildung 32).

#### 5.1.1 Technische Daten

- Betriebsspannung: 9V
- Stromaufnahme: ca. 190 mA
- 14 Digitale Ein/Ausgänge, 4 Digitale Eingänge

- 4 Analoge Eingänge
- ATmega644P Mikrocontroller
- integrierte ISP-Schnittstelle

## 5.2 Mikrocontroller

Das Team hatte zu Beginn einen ATmega32 Prozessor als Vorgabe. Dieser hat jedoch nur 32kB Flash Speicher. Nach ersten Feldversuchen wurde schnell klar, dass ein Speicher von 32kB nicht ausreichen wird, weshalb das Team auf einen ATmega644P ausgewichen ist. Dessen Speicherressourcen liegen bei 64kB und reichen für Webserver und Webseite aus, weshalb das Team nicht auf die größere Variante, den ATmega1284P ausweichen musste.

Anbei ein kleiner Vergleich der wichtigsten Bestandteile der drei Prozessoren. Weitere Informationen würden den Rahmen sprengen und können bei Bedarf dem entsprechenden Datenblatt entnommen werden.

Mikrocontroller	ATmega32	ATmega644P	ATmega1284P
Flash	32kB	64kB	128kB
Pins	44	44	44
Max. Operation Freq.	16MHz	20MHz	20MHz
Max. I/O Pins	32	32	32

Tabelle 2: Mikrocontroller im Vergleich

### 5.2.1 ENC28J60

Der ENC28J60 ist der IEEE 802.3 kompatible Netzwerk Controller des AVR-Net-IO. Er nimmt die externen Anfragen an und reicht diese an den ATmega weiter. Dabei ist er an Port B des ATmegas angeschlossen, Port B kann deswegen nicht für Ein- oder Ausgaben verwendet werden.



### 5.3 Fuse Bits

Die Fuse-Bits sind die grundlegenden Einstellungen, mit denen ein Mikrocontroller arbeitet. Sie müssen geändert werden, wenn ein anderer Taktgeber gewünscht ist oder Schnittstellen deaktiviert oder aktiviert werden sollen.

**Allgemeiner Hinweis:** Dieser Artikel kann die Recherche im Datenblatt nicht ersetzen, besonders bei abweichendem Mikrocontroller sind die Fuse-Bits oft anders gewählt. Mit dem Atmel Studio kann es vorkommen, dass man sich vom Mikrocontroller aussperrt. Ein zurücksetzen der Fuse-Bits kann mit AVRDUDE in diesem Fall versucht werden. Beschrieben wird dieser Vorgang im Benutzerhandbuch 10.4

In Tabelle 3 werden die einzelnen Fuse-Bits zusammen mit ihrer Bedeutung und dem entsprechenden Byte aufgelistet. Der schlussendliche Fuse Wert setzt sich aus zwei Byte zusammen, wobei eine 1 eine deaktivierte Eigenschaft und eine 0 eine aktiviert Eigenschaft bedeutet. Kleinere Mikrocontroller besitzen nur ein High (H) und ein Low (L) Register, größere Mikrocontroller besitzen zusätzlich noch ein Extended (E) Register.

Fuse Name	Bedeutung	Bytes
BODLEVEL	Brown-out Detector trigger level	E-Fuse 0&1
OCDEN	Aktiviert On-Chip Debugging	H-Fuse 7
JTAGEN	Aktiviert das Joint Test Action Group (JTAG) Interface	H-Fuse 6
SPIEN	Aktiviert das In System Programming (ISP) Interface	H-Fuse 5
WDTON	Watchdog timer (WDT) immer an	H-Fuse 4
EESAVE	Schützt den EEPROM während des Lösch-Zyklus	H-Fuse 3
BOOTSZ	Boot Flash Sektor Größe	H-Fuse 1&2
BOOTRST	Boot Reset Vektor	H-Fuse 0
CKDIV8	Teilt den Takt der Uhr intern durch 8	L-Fuse 8
CKOUT	Ausgabe des Takts der Uhr auf Port B1	L-Fuse 7
SUT_CKSEL	Wahl der Takt-Quelle	L-Fuse 0-6

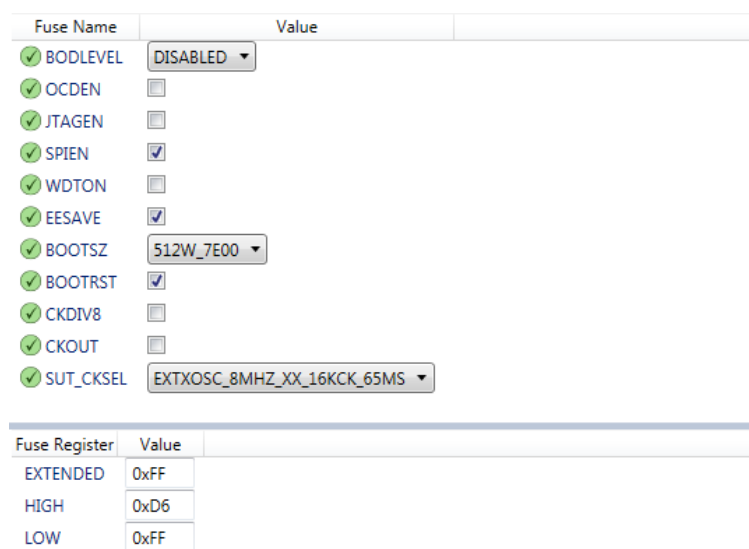
Tabelle 3: Die Bedeutung der einzelnen Fuse-Bits (ATmega-664P)

Wir benötigen für unseren ATmega-664P SPIEN, EESAVE und BOOTRST aktiviert im High Register. Im Low Register muss alles deaktiviert werden, damit der Externe Quarz-Kristall verwendet wird. Da der Brown-out Detector Trigger nicht verwendet wird, kann in dem Extended Register ebenfalls alles deaktiviert werden. Daraus resultiert die Bit Kombination in Tabelle 4. Wenn JTAG verwendet werden soll, muss der entsprechende Bit (H-Fuse 6) gesetzt werden. Ist JTAG aktiviert fungieren auf Port C die Pins 2-5 nicht mehr als IO-Pins, diese werden für JTAG verwendet.

Fues Register	Binär	Hex
Extended-Fuse	1111 1111	FF
High-Fuse	1101 0110	D6
Low-Fuse	1111 1111	FF

Tabelle 4: Fuse Einstellungen (ATmega-664P)

Im Atmel Studio können die Fuse Einstellungen ganz einfach im Device Programming vorgenommen werden (Kapitel 9.2.1.1). In Abbildung 3 sind die festgelegten Einstellungen eingetragen.



Fuse Name	Value
✓ BODLEVEL	DISABLED
✓ OCDEN	<input type="checkbox"/>
✓ JTAGEN	<input type="checkbox"/>
✓ SPIEN	<input checked="" type="checkbox"/>
✓ WDTON	<input type="checkbox"/>
✓ EESAVE	<input checked="" type="checkbox"/>
✓ BOOTSZ	512W_7E00
✓ BOOTRST	<input checked="" type="checkbox"/>
✓ CKDIV8	<input type="checkbox"/>
✓ CKOUT	<input type="checkbox"/>
✓ SUT_CKSEL	EXTXOSC_8MHZ_XX_16KCK_65MS

Fuse Register	Value
EXTENDED	0xFF
HIGH	0xD6
LOW	0xFF

Abbildung 3: Fuse-Bits im Atmel Studio (ATmega-664P)

Weitere Hinweise zu den Fuse-Einstellungen gibt es auf mikrocontroller.net [http://www.mikrocontroller.net/articles/AVR\\_Fuses](http://www.mikrocontroller.net/articles/AVR_Fuses). Eine Website die eine einfache grafische Konfiguration der Fuse-Bits ermöglicht gibt es bei engbedded.com <http://www.engbedded.com/fusecalc/>

## 6 Programmieren und Debuggen

Um einen Mikrocontroller zu programmieren oder zu debuggen, gibt es zwei Interfaces, diese werden hier einmal genauer beleuchtet.

### 6.1 ISP

Ein In System Programming (ISP) fähiger Mikrocontroller kann direkt in der Schaltung programmiert werden, ohne entfernt zu werden. Programmieren kann man entweder mit dem Programmer AVRISPmkII über die SPI Schnittstelle oder mit dem AVRJTAGICEmkII. Der AVRJTAGICEmkII unterstützt zum Programmieren sowohl die die SPI als auch die JTAG Schnittstelle.

### 6.2 SPI

Serial Peripheral Interface (SPI) ist die Schnittstelle, um den Mikrocontroller zu programmieren, über diese Schnittstelle kann nicht debuggt werden. Neben einem Programmer, wie dem AVRISPmkII, hängt an diesem Bus auch der ENC28J60 Netzwerk Controller.

### 6.3 JTAG

Joint Test Action Group (JTAG) ist eine Schnittstelle die neben der Programmierung von Mikrocontrollern auch das Debuggen ermöglicht. Dafür wird allerdings der relativ teure AVRJTAGICEmkII von Atmel benötigt. In Verbindung mit dem Atmel Studio kann dann komfortabel getestet werden. Wie beim Entwickeln von Programmen mit einer anderen Programmiersprache können Breakpoints definiert werden und variabel ausgelesen werden. Für eine umfangreiche Entwicklung wie unseren Webserver, bei dem viele dynamische Ereignisse auftreten können, ist ein solches Programmierwerkzeug sehr hilfreich.



## 7 Recherche

### 7.1 Ethersex

Auf der Projekt-Website (<http://ethersex.de>) bewirbt sich Ethersex als

*„[...] eine Firmware mit Netzwerkunterstützung für 8-bit AVR Mikrocontroller, die durch eine Community entwickelt wird.“ [Eth14]*

Ethersex unterstützt sowohl die von uns verwendeten ATmega Prozessoren, das AVR-Net-IO Board und auch den verwendeten ENC28J60 Netzwerk Controller. Durch einen ausführlichen Quick Start Guide ([http://ethersex.de/index.php/Quick\\_Start\\_Guide/Preparation](http://ethersex.de/index.php/Quick_Start_Guide/Preparation)) auf der Projektseite ist es, auch für Anfänger in diesem Themengebiet, sehr einfach möglich, das Projekt lauffähig zu machen. Fast die gesamte Konfiguration kann über ein Menü vorgenommen werden. So muss nicht wie beim Radig Projekt zum ändern der Mac Adresse in eine Headerdatei geschrieben werden.

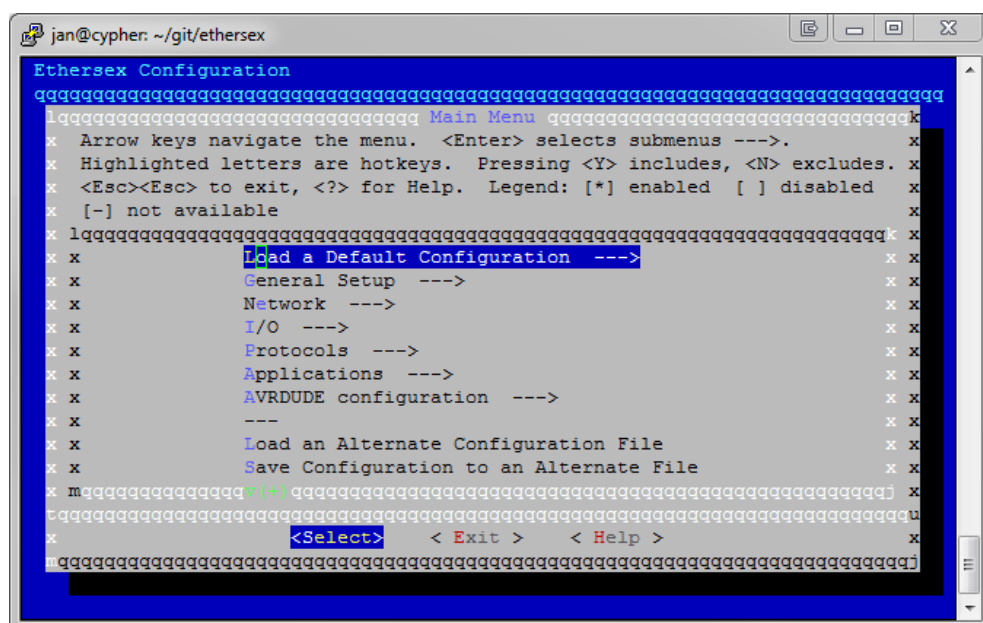
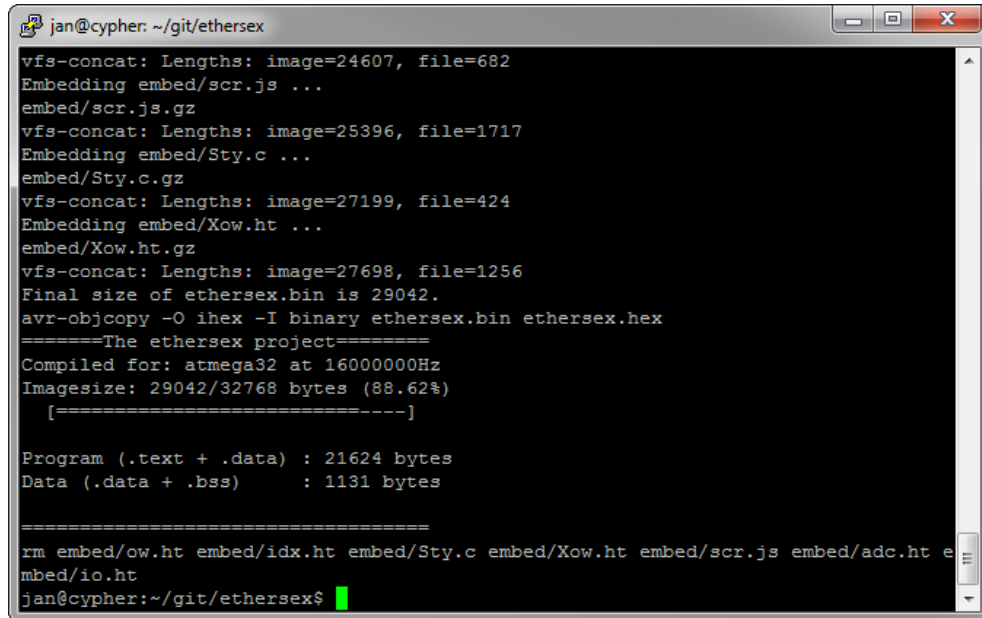


Abbildung 4: Ethersex menuconfig

Im Bild (Abb. 4) sieht man den Startbildschirm des Menüs, das über den Befehl `make menuconfig` erreicht werden kann. Hier können verschiedene Einstellungen getroffen werden. Zum Beispiel, den verwendeten Mikrocontroller, welche Mac Adresse der Netzwerk Controller verwendet oder welche IP Adresse gewünscht ist. Nachdem

die Konfiguration abgeschlossen ist, kann das Hexfile mit dem `make` Befehl erstellt werden. In der Abbildung 5 sieht man, dass am Ende des Make Prozesses die aktuelle Größe des erstellten Binary Datei angezeigt wird.



```

jan@cypher: ~/git/ethersex
vfs-concat: Lengths: image=24607, file=682
Embedding embed/scr.js ...
embed/scr.js.gz
vfs-concat: Lengths: image=25396, file=1717
Embedding embed/Sty.c ...
embed/Sty.c.gz
vfs-concat: Lengths: image=27199, file=424
Embedding embed/Xow.ht ...
embed/Xow.ht.gz
vfs-concat: Lengths: image=27698, file=1256
Final size of ethersex.bin is 29042.
avr-objcopy -O ihex -I binary ethersex.bin ethersex.hex
=====The ethersex project=====
Compiled for: atmega32 at 16000000Hz
Imagesize: 29042/32768 bytes (88.62%)
[=====]

Program (.text + .data) : 21624 bytes
Data (.data + .bss)    : 1131 bytes

=====
rm embed/ow.ht embed/idx.ht embed/Sty.c embed/Xow.ht embed/scr.js embed/adc.ht e
mbed/io.ht
jan@cypher:~/git/ethersex$

```

Abbildung 5: Ethersex make project

Das Einbinden der Website beim Ethersex Projekt wird in der Anleitung folgendermaßen beschrieben:

*„Falls die Option Supply Inline Files aktiviert ist, werden alle Dateien, die unter vfs/embed/ abgelegt sind, automatisch beim Erstellen des Images mit gzip gepackt und an das Ende der Firmware angehängt. Die Dateinamen bleiben dabei unverändert [...]“* [Eth14, [http://www.ethersex.de/index.php/HTTPD\\_\(Deutsch\)](http://www.ethersex.de/index.php/HTTPD_(Deutsch))]

## 7.2 Elektronik 2000

Einen anderen Ansatz verfolgt das Projekt Elektronik 2000. Hier wird nicht nur der Webserver geboten sondern eine erweiterte GUI, um das Board zu programmieren. Dafür wird mit einem grafischem Designer eine Logik entworfen und über einen ISP Programmer auf das Board gebracht.

*„Das E2000-NET-IO basiert auf dem AVR-NET-IO von Pollin. Durch die E2000-Firmware wird aus dem AVR-NET-IO von Pollin ein autak laufendes Logikmodul. Mit diesem Modul können über Netzwerk Schaltvorgänge ausgeführt werden. Außerdem sind Zeitgesteuerte Schaltvorgänge möglich.“* [ele14]

Durch die Netzwerkanbindung des AVR-Net-IO kann dann die programmierte Logik von außen überwacht und gesteuert werden. Dafür gibt es von den Entwicklern eine bereitgestellte Android Applikation. Zusätzlich zu einem Projekt das mit dem AVR-Net-IO arbeitet gibt es mittlerweile eine weitere Version, welche auch mit dem Raspberry Pi zusammenarbeitet und die GPIO Pins des Pis nach außen steuerbar macht.

### 7.3 Ulrich Radig

Ein weiteres Projekt für den AVR-Net-IO ist die AVR Webserver Software (<http://www.ulrichradig.de/home/index.php/software/avr-webserver-software>). Die Software ist ein Webserver für die Mikrocontroller der Baureihen ATmega 32, 64 und 128. Durch die Einfache Bindung der Dateien kann das Projekt einfach an eigene Bedürfnisse angepasst werden. Dabei hat die Software noch weiter Funktionalität:

- **Hyper Text Transfer Protocol (HTTP) Server** Webserver ohne Dateistruktur
- **Sendmail** Senden von E-Mails
- **Weather** Ermitteln von Wetterdaten
- **Network Time Protocol (NTP)** Empfangen von Internetzeit
- **Domain Name System (DNS)** Beantwortung von Anfragen zur Namensauflösung
- **Universal Synchronous and Asynchronous Serial Receiver and Transmitter (USART)** eine Schnittstelle im Mikrocontroller zum Datenaustausch mit PC über die COM-Schnittstelle
- **Telecommunication Network (Telnet)** zeichenorientierter Datenaustausch über eine TCP-Verbindung.
- **Command Control (CMD)** Verwaltung der Telnet-Konsolenbefehle.
- **Wake on Lan (WOL)** Funktionalität um andere Geräte im Netzwerk durch bestimmte Datenpakete aufzuwecken.





## 8 Ausgewählte Lösung

Neben den beiden anderen Projekten haben wir uns für das Projekt von Ulrich Radig entschieden. Die Vorteile des Projektes gegenüber Ethersex oder Elektronik 2000 liegen darin, dass die beiden Projekte zu speziell und umfangreich für unsere Anforderungen sind. Da es zu unserem Aufgabengebiet gehörte eine für den Benutzer möglichst umfangreiche Website zu erstellen kam uns die gut anpassbare Lösung entgegen.

### 8.1 Der Webserver

Zusätzlich von der Ursprungsversion von Ulrich Radig gibt es noch eine etwas vereinfachte Version von Günther Menke. Wir haben uns für die vereinfachte Variante entschieden. Die Unterschiede zwischen beiden Versionen belaufen sich auf das entfernte Kamera-Feature und auf zusätzlichen Quellcode für einen alternativen Netzwerkcontroller.

#### 8.1.1 Änderungen

Obwohl wir die bereits im Funktionsumfang vereinfachte Version von Günther Menke verwenden, gab es trotzdem einiges an Funktionalität, welche wir aus der ursprünglichen Version entfernt haben. Das Problem lag darin, dass wir zum einen für die Dateien der Website möglichst viel freien Speicherplatz benötigen, der von den entsprechenden Funktionen belegt wurde. Zum anderen, dass die entfernten Funktionen nicht für unser Projekt benötigt wurden. Schlussendlich wurden folgende Funktionalitäten aus der Version von Günther Menke entfernt:

- WOL
- Sendmail
- Weather
- NTP
- DNS
- USART
- Telnet

- CMD

### 8.1.2 Einbindung der Website

Die Website, welche hauptsächlich aus verschiedenen .html und .js Dateien besteht, ist mangels Dateisystem für unsere Firmware nicht verwendbar. Die gesamten Dateien müssen in einer C-Headerdatei gebunden werden. Das Erstellen der Headerdatei erfolgt über das beim Projekt beigelegte HTML Header Compiler (HHC) Werkzeug. Eine Beispiel zum Erstellen der webpage.h Datei und eine Erklärung des HHC gibt es im Kapitel Werkzeuge 9.2.3. Eine Anleitung zur Ausführung des HHC gibt es im Benutzerhandbuch 10.5. Abschließend ist noch zu erwähnen, dass die webpage.h nicht für manuelle Bearbeitung gedacht ist. Dies geschieht ausschließlich über die Quell-Dateien und anschließend umwandeln mit dem HHC.

## 8.2 Die Website

Der Aufbau der Webseite ist in mehrere Dateien aufgeteilt. So haben wir mehrere .js und .css Dateien. In der index.html wird alles nur zusammengetragen.

### 8.2.0.1 board.css

Diese Datei behandelt alle Definitionen über die Platine in dem 'Status Tab' der Webseite. Alle Definitionen über die Platine sind direkt in der index.html in einer komplexen div-Verschaltung eingebunden. So können einzelne div-Elemente als klickbar und verlinkbar zu anderen Elementen sein. Eine Veränderung der Verschachtelung oder im CSS-Code kann zu Anzeigefehlern oder ein NichtAnzeigen der Webseite führen. Alle Div-Elemente sind abgemessen und auf die passende Größe ausgerichtet.

### 8.2.1 JavaScript

#### 8.2.1.1 Generell

In rest.js und favorite.js werden die einzelnen Pins über eine ID unterschieden. Diese ID wird vor allem als Parameter für diverse Funktionen wie getValu(id) verwendet. Die ID setzt sich immer aus dem Buchstaben des Ports, sowie aus der Nummer des Pins zusammen. Mögliche IDs sind folglich [A0 bis A7], [C0 bis C7] und [D0 bis D7].

#### 8.2.1.2 rest.js

Die rest.js beinhaltet den gesamten Code zum Abrufen von Daten beim Server. Hierbei bieten die Funktionen loadUrl(...) und loadUrlAsync(...) den Kern der Serverkommunikation. Mit ihnen können beliebige URLs geladen werden, loadUr-

`loadAsync(...)` bietet zusätzlich die Möglichkeit POST-Parameter zu übergeben. Der grundlegende Unterschied zwischen diesen beiden Methoden ist, dass `loadUrl(...)` den Programmablauf blockiert, bis die Daten vollständig geladen sind, wohingegen `loadUrlAsync(...)` die Daten parallel im Hintergrund lädt. Sobald alle Daten erfolgreich geladen sind, wird die Funktion aufgerufen, die `loadUrlAsync(...)` als Parameter übergeben wurde. Weitere Informationen, vor allem zur Parametrisierung der Funktionen, sind der Sourcecode-Dokumentation zu entnehmen.

In der Funktion `initRest()` wird zum ersten Mal die Funktion `startNewRefreshTask()` aufgerufen. Diese ruft die Funktion `refreshValues()` auf, welche sich selbst rekursiv immer wieder nach einer gewissen Pause (der Polling-Frequenz, welche in den Einstellungen festgelegt werden kann) aufruft. Die Polling-Frequenz wird in dem Attribut `pollingFreq` gespeichert. Der so erzeugte Refresh-Task, der vollautomatisch die Werte aktualisiert, kann über `cancelRefreshTask()` wieder gestoppt werden. Die Werte werden danach natürlich nicht mehr aktualisiert, bis mit `startNewRefreshTask()` ein neuer Task gestartet wird. Nach jedem Aktualisieren der Werte wird die Funktion aufgerufen, die in dem Attribut `onValuesChanged` gespeichert wird. In unserer Implementierung ist diese Funktion in der `ui.js` implementiert und organisiert die Aktualisierung der Oberfläche, damit die aktuellen Werte dargestellt werden. Der Wert des Attributes `onValuesChanged` kann mit der Funktion `setOnValuesChanged(...)` manipuliert werden. Die zuletzt empfangenen Werte werden in `cachedValues` gespeichert und sind über diverse Getter wie `getValue(id)` verfügbar.

Das Stoppen eines bestehenden Tasks wird realisiert, indem das Attribut `taskIdCounter` inkrementiert wird. `refreshValues()` überprüft am Anfang, ob der aktuelle Wert von `taskIdCounter` mit dem Wert übereinstimmt, mit der der aktuell ausgeführte Task gestartet wurde. Ist dies nicht der Fall, weil der Task beendet oder ein neuer gestartet wurde, beendet sich der Task automatisch selbst, indem die Funktion `refreshValues()` nicht erneut rekursiv aufgerufen wird.

In der Funktion `initRest()` werden einmalig die REST-URLs `/rest/info` und `/rest/pin-info` geladen. Diese beinhalten nur statische Informationen, welche sich nie ändern. Die empfangenen JSON-Strukturen werden in ein JavaScript Objekt geparkt und in `cachedInfo` und `cachedPinInfo` gespeichert. Der Inhalt dieser Objekte wird über die Getter `getInfo()` und `getPinInfo()` zur Verfügung gestellt. Für `cachedPinInfo` sind zudem Getter wie `getDefaultDescription(id)` vorhanden.

### 8.2.1.3 favorites.js

In der `favorites.js` wird eine Liste aller Pins angelegt.

Beim Hinzufügen eines Pins wird über `addPin(id)` überprüft, ob der Pin bereits in

der Liste existiert. Ist dies nicht der Fall, so wird ein neuer Pin mit Standardwerten in die `favoritelist` geschrieben. Der Nutzer kann für jeden Pin eine Beschreibung, sowie eine Funktion und ob dieser Pin als Favorit markiert werden soll, hinterlegen. Dies geschieht mit den Funktionen `setDescription(id, des)`, `setFunction(id, func)` und `toggleFavorite(id)`.

Bei jeglichen Gettern (z.B. `getFunction()`, `getDescription()`) von Pin wird ebenfalls überprüft, ob der Pin bereits in der Liste existiert. Existiert er, werden die geforderten Werte zurückgegeben. Existiert er nicht, wird er mit Standardwerten in die `favoritelist` geschrieben.

Diese Liste wird über die JavaScript-Datei `db.js`, abgespeichert und bei Neustart aus dieser ausgelesen. Diese `save()` Funktion übergibt die `favoritelist` mithilfe von `JSON.stringify()`.

Zusätzlich kann diese durch Import und Export Funktionen als reiner Fließtext ausgegeben und auf anderen Rechnern eingefügt werden, damit die Daten dort ebenfalls existieren.

Ursprünglich sollte `favorites.js` nur eine Liste der als Favorit markierte Pins speichern, was jedoch nach einigen Überlegungen zu einer Liste aller Pins umgeändert wurde. Der Name `favorite.js` wurde jedoch beibehalten, sowie die Funktion, Pins als Favorite zu setzen. Dies geschieht über ein Flag, welches `true` oder `false` sein kann.

#### 8.2.1.4 db.js

Die `db.js` besteht aus 2 Funktionen:

`getDb(key, defaultValue)` liefert den gespeicherten Wert für den gegebenen `key`, welcher immer eine Zahl oder ein String sein sollte. Ist kein Wert mit dem gegebenen Key vorhanden, wird das übergebene `defaultValue` zurückgegeben.

`putDb(key, value)` speichert das gegebene `value` unter dem gegebenen `key` ab. Wird jetzt `getDb(key, defaultValue)` mit dem gleichen `key` aufgerufen, wird der zuvor abgespeicherte Wert zurückgegeben.

Alle mit `db.js` gespeicherten Werte sind auch nach einem Neuladen der Seite weiterhin verfügbar, solange nicht die IP-Adresse des Pollin Net-IO Boards verändert wird. Die verwendeten Keys müssen natürlich eindeutig sein.

#### 8.2.1.5 ui.js

Die `ui.js` beinhaltet den gesamten Code für das dynamische Gestalten der Webseite. Eine der wichtigsten Funktionen, die generell verwendet wird, ist die `getEl(id)`. Diese Funktion verkürzt die Standardfunktion von JavaScript `getElementById(id)`, um den Sourcecode zu kürzer zu halten, um den daraus resultierenden Speicherbedarf der

Webseite zu minimieren.

`initUi()` sollte nur einmal beim Seitenaufruf geladen werden und initialisiert die Benutzerschnittstellen. `setMain(div)` setzt die Hauptkomponenten von dem Frame 'div'. `addClass(clazz, item)` fügt eine CSS Klasse zu dem gegebenen Item hinzu, während die `removeClass(clazz, item)` die CSS Klasse von dem Item wieder löscht.

`getHighlightElem(id)` markiert die digitalen Pins, oder markiert die Checkboxes der analogen Pins. Folgende Funktionen generieren die dynamische Sidebar: `setSidebarId(id)` listet alle Informationen über die Pins in der Sidebar auf. Die `updateSidebarValues()` hält die Werte der einzelnen Pins auf den aktuellsten Stand.

Die Favoriten Tabelle wird mittels der Funktion `updateFavoritesTable()` erneuert und mit `updateFavoritesTableValues()` werden die Pininformationen der Favoriten auf den aktuellsten Stand gebracht. Ändern sich die Favoriten wird `onFavoritesChanged()` aufgerufen.

text

Das separat aufpoppende Fenster um die Pins / Ports besser zu konfigurieren wird mit `startConfigurePin(id)`. Beim Aufruf wird die ID des Pins benötigt. In der Funktion werden die restlichen Informationen zu dem Pin abgerufen und eingetragen. In dem Fenster sind die Buttons 'OK' und 'zurücksetzen', die dann beim bestätigen `endConfigurePin(reset)` aufruft. `reset` ist der Flag zum zurücksetzen des Pins. Ist er auf `true`, wird der Pin zurückgesetzt, sollte er auf `false` sein, werden die neuen Konfigurationen zum Pin abgespeichert.

Um die `index.html` nicht noch mit weiteren vielen `div`-Elementen zu überfüllen, werden die gleichen `div`-Elemente mit `write(count, string, ids)` generiert. Diese Funktion wird in folgenden verwendet: `writePinCheck(ids)`, `writePinCheckMinus(count)`, `writePinCheckPlus(count)`, `writePinCheckNone(count)`, `writePinMinusBox(count)`, `writePinCheckAnalog(ids)` und `writeAnalogBox(ids)`.



## 9 Technischer Hintergrund

### 9.1 Server/Webseite Kommunikation

Im Projekt Radig ist keine echte Kommunikation zwischen dem Client und dem Server vorhanden.

Die auf der Webseite dargestellten Werte werden vor dem Senden der HTML-Seite im HTML-Code eingefügt, indem Platzhalter im Format „%PORTA0“ ersetzt und so statisch auf der Webseite dargestellt werden. Das Manipulieren der Pins findet über ein HTML-Formular statt. Alle manipulierbaren Pins sind als Input vom Typ Checkbox dargestellt. Diese lassen sich frei manipulieren und erst beim Betätigen des „Senden“-Buttons werden die Informationen per POST-Event an den Server gesendet und so die Seite neu aufgerufen. Der Server filtert die POST Informationen aus dem HTTP-Header und manipuliert die Pins gemäß den Anweisungen. Beim Senden des angeforderten HTML-Dokumentes werden die neuen Werte in den HTML-Code eingefügt, und so die neuen Werte auf der Webseite angezeigt.

Das große Problem bei dieser technisch einfachen Lösung ist, dass geänderte Werte erst beim nächsten Neuladen der Webseite angezeigt werden. Ändert sich ein Pin während die Webseite dargestellt wird, bekommt der Nutzer dies nicht mit. Zudem wird bei jedem Manipulieren eines Pins die gesamte Seite neu geladen und so Unmengen an unnötigen Daten übertragen. Auch zum Darstellen der aktuellen Werte muss die ganze Seite neu vom Server angefordert werden.

#### 9.1.1 Erster Ansatz

Die Kommunikation zwischen Server und Client sollte mit Hilfe einer REST-Schnittstelle stattfinden, die im Hintergrund über Javascript angesprochen werden kann.

Eine REST-Schnittstelle besteht aus einer oder mehreren virtuellen URLs. Beim Aufruf einer solchen URL liefert der Server kein Dokument, das gespeichert ist, sondern erzeugt dynamisch eine Antwort mit den benötigten Informationen und sendet diese als Antwort zurück. Der Server kann beim Aufruf einer URL auch eine Aktion ausführen. Vorteile der REST-Schnittstelle ist die simple Implementierung, sowohl auf dem Client mit JavaScript als auch auf dem Server. Die Inhalte werden mit JSON formatiert, welches einen technischen Standard darstellt und sich in JavaScript direkt in ein Objekt umwandeln lässt. Auf dem Server ist es einfach mit einem Stringformat immer gleiche JSON Strukturen zu erstellen und nur aktuelle Werte einzufügen. Die REST-

Schnittstelle lässt sich leicht um weitere, neue Funktionalitäten erweitern, indem neue virtuelle URLs erstellt werden, die vom Client ansprechbar sind.

Die Anforderungen an eine Lösung in diesem Projekt waren vor allem eine möglichst kompakte Schnittstelle zu schaffen, die wenig Bandbreite verbraucht, um eine hohe Übertragungsgeschwindigkeit zu ermöglichen, trotz des schwachen Servers. Ein besonderes Augenmerk war auf die Übertragung der Messwerte zu legen, da diese nicht wie andere statische Informationen nur einmalig übertragen, sondern kontinuierlich erneuert werden müssen. Die Schnittstelle sollte gut skalierbar sein. Würde später ein Port für eine andere Aufgabe verwendet werden, muss dieser Port ohne Aufwand aus der REST-Schnittstelle ausgeschlossen werden können, damit er von außen nicht manipulierbar ist und so interne Abläufe auf der Platine nicht gestört werden.

Nach den Anforderungen muss die Schnittstelle folgende Aufgaben ermöglichen:

- Abfragen der aktuellen Werte aller verwendbaren Pins
- Abfragen der Konfiguration eines Pins (Eingang oder Ausgang)
- Abfragen von allgemeinen Informationen des Boards (IP, Standard-IP, Mac-Adresse, Serverversion)
- Manipulieren aller als Ausgänge geschaltene Pins
- Manipulieren der Konfiguration eines Pins (als Eingang oder Ausgang setzen)
- Manipulieren von Servereinstellungen (z.B. IP-Adresse);

### 9.1.2 Polling oder Pushing

Die aktuellen Werte der Pins müssen bei jeder Änderung vom Server zum Client übertragen werden, damit diese auf der Webseite immer korrekt dargestellt werden. Hierfür stehen zwei verschiedene Konzepte für die Initialisierung zur Verfügung.

#### 9.1.2.1 Polling

Beim Polling werden die Werte vom Client kontinuierlich angefordert, indem dieser die entsprechende virtuelle Uniform Resource Locator (URL) des Servers aufruft. Dies führt dazu, dass viele unnötige Daten übertragen werden, da sich eventuell nicht bei jedem erneuten Anfordern der Werte diese auch tatsächlich verändert haben und so die gleichen Datensätze oft mehrmals angefordert werden.

Im Vergleich zur Radig-Lösung bietet Polling den Vorteil, dass die Werte kontinuierlich nachgeladen und so immer korrekt dargestellt werden während die Webseite dargestellt wird. Auch das gesendete Datenvolumen wird dahingehend minimiert, dass nur die Nutzdaten übertragen werden und nicht der gesamte HTML-Code der Webseite.



Polling ist technisch sehr einfach zu realisieren, da die Abfrage der Daten einfach zyklisch wiederholt wird.

#### 9.1.2.2 Pushing

Bei Pushing wird im Gegensatz zu Polling die Übertragung der Daten nicht vom Client initialisiert, sondern vom Server. Der Server weiß wann sich die Werte geändert haben und kann dem Client bei jeder Änderung gezielt die neuen Daten übermitteln. Das Übertragen der Daten könnte auf dem Server z.B. durch einen Interrupt ausgelöst werden.

Im direkten Vergleich zu Polling bietet Pushing verschiedene Vorteile. So wird nicht nur das Volumen der übertragenen Daten reduziert indem keine unnötigen Abfragen stattfinden, sondern die neuen Werte gelangen auch genau dann zum Client wenn die Änderung tatsächlich stattgefunden hat, was dazu führt das die Webseite schneller auf Änderungen reagiert.

Die technische Umsetzung von Pushing ist mit diversen Problemen verbunden. Die typische Verbindungsaufbaurichtung ist bei Webanwendungen und Webseiten immer vom Client zum Server. Anders als bei Polling muss bei Pushing die Verbindung vom Server zum Client aufgebaut werden. Dies ist technisch aber nicht möglich, da der Browser bzw. JavaScript keine Möglichkeit haben einen Port des Clientsystems zu öffnen und auf eingehende Verbindungen des Servers zu antworten.

Das Problem lässt sich durch die Benutzung von HTML5 Server-Sent Events umgehen. Hierbei fragt der Client eine virtuelle URL des Servers ab, ähnlich einer REST-Schnittstelle. Der Server überträgt jedoch nicht sofort Daten, sondern schreibt erst bei einem Event (z.B. die Änderung eines Pins) in den geöffneten Stream und pusht so die Daten zum Client. Dieser überwacht den Stream mit Hilfe von JavaScript und empfängt so die neuen Werte und kann sie auf der Webseite anzeigen.

Dieses System ist auf dem Pollin Net-IO Board aber nur schwer umzusetzen, da mehrere Verbindungen verwaltet werden müssen. So ist immer mindestens eine Server-Sent Event Verbindung offen, parallel könnte aber ein Client andere Daten vom Server anfordern. Für das Verwalten mehrerer Verbindungen sind aber viele Ressourcen nötig, da für jede Verbindung auch Daten im RAM hinterlegt werden müssen. Außerdem ist in vielen Situationen ein simples Multitasking nötig, das so auf einem ATmega CPU nicht vorhanden ist. Das Radig Projekt setzt aus diesen Gründen auf HTTP 1.0, bei dem für jede Anfrage eine Verbindung geöffnet und nach erfolgreichem Übertragen der Daten wieder geschlossen wird. So ist auch die Kommunikation mit mehreren Clients problemlos möglich.

### 9.1.2.3 Entscheidung

Pushing ist leider nur schwer umsetzbar auf dem Mikrocontroller. Dies ist bedingt durch die knappen Ressourcen wie Arbeitsspeicher, als auch durch das nur schwer umsetzbare Multitasking, das für einen reibungslosen Ablauf nötig ist. Ohne Multitasking ist nicht gewährleistet, dass alle Dateien übertragen werden können, da der Server, sobald eine Server-Sent Event Verbindung aufgebaut wird, nicht mehr verfügbar ist, bis diese wieder geschlossen wird. Eine Benutzung durch mehrere Clients ist so nicht denkbar und auch die Nutzung von nur einem Client kann zu Problemen führen wenn dieser Daten anfordert, nachdem die Server-Sent Event Verbindung aufgebaut worden ist.

Die Implementierung von Multitasking ist zwar prinzipiell möglich, erfordert aber einen tiefen Eingriff in das Vorlagenprojekt, da neben der eigentlichen Nebenläufigkeit zusätzlich der gesamte Server threadsafe gestaltet werden müsste.

Aus diesen Gründen entschlossen wir uns das technisch unsauberere Polling zu verwenden, da die Implementierung von Pushing den Rahmen des Projektes übersteigen würde.

### 9.1.3 Aufbau der Server-Kommunikation

Bei der Server-Kommunikation gibt es zwei grundsätzliche Kanäle:

- Das Abfragen von Daten beim Server
- Das Manipulieren von Servereinstellungen (z.B. Pinwerte)

Das Manipulieren von Servereinstellungen war bereits im Projekt Radig möglich. Hierfür interpretiert der Server die POST-Parameter jeder Anfrage und setzt ggf. die Pins neu. Für die neuen Anforderungen wie das Manipulieren des DDR haben wir und dazu entschlossen den bereits vorhandenen Code nur leicht zu manipulieren und zu erweitern.

Für das Abfragen von Daten ist im Projekt Radig keine Lösung vorhanden, da hier die Werte statisch in Form von Platzhaltern im HTML-Text eingebunden sind und beim Abfragen der HTML-Datei durch die Werte ersetzt werden. Folglich muss die gesamte Seite erneut geladen werden, um die dargestellten Werte zu aktualisieren. Um dies zu vermeiden haben wir uns entschlossen die Daten dynamisch abzufragen, um sie gesondert von der Webseite laden zu können.

#### 9.1.3.1 Mögliche Techniken für Datenabfrage

Für die dynamische Datenabfrage gibt es verschiedene Standards. Hierzu gehören verschiedene XML-basierte Protokolle wie RSS, als auch das so genannte Representational State Transfer (REST).

XML-Basierte Protokolle wie Rich Site Summary (RSS) oder Simple Object Access Protocol (SOAP) weisen typischerweise einen großen Protokoll-Overhead auf und sind deswegen nur bedingt für den Einsatz auf einem Mikrocontroller geeignet, da das System zu viel unnötige Daten übertragen müsste. Außerdem muss das dynamische HTTP-Abfragen der Daten per JavaScript erfolgen, welches die in XML präsentierten Daten erst wieder parsen müsste, was zusätzlichen Code auf dem Client bedeuten würden.

REST hingegen präsentieren die Daten in JavaScript Object Notation (JSON), welches von JavaScript direkt als Objekt interpretiert werden kann. Dies erspart das aufwendige Parsen der Daten. Außerdem ist der Protokoll-Overhead bei JSON tendenziell kleiner als bei XML-Basierten Lösungen, da weniger lange Tagnamen vorhanden sind, was eine effektivere Übertragung ermöglicht.

Aus diesen Gründen haben wir uns für eine REST Lösung entschieden.

#### 9.1.3.2 Design der REST-Schnittstelle

Bei REST gibt es für jede Abfrage eine separate URL. Bei uns liegen alle URLs im Unterverzeichnis `/rest`. Insgesamt gibt es 3 URLs:

- `/rest/info` liefert generelle, statische Informationen über den Server, wie z.B. die Server-Version
- `/rest/pininfo` liefert generelle, statische Informationen über die einzelnen Pins, wie z.B. den Namen des Pins
- `/rest/values` liefert nur die aktuellen Werte und Data Direction Register (DDR)-Einträge.

Um die dargestellten Werte auf der Webseite zu aktualisieren, muss folglich nur `/rest/values` erneut geladen werden. Aus diesem Grund sollte diese Datei so klein wie möglich gehalten werden um eine optimale Aktualisierungsgeschwindigkeit zu ermöglichen. `/rest/info` und `/rest/pininfo` müssen nur einmalig geladen werden, da die enthaltenen Informationen statisch sind und nie geändert werden. Die Größe dieser Dateien ist deshalb weniger ausschlaggebend.

#### 9.1.4 Implementierung der REST-Schnittstelle auf dem Server

Für die Implementierung der REST-Schnittstelle setzen wir auf dem bestehenden Code aus dem Projekt Radig auf. Typischerweise sind REST-URLs virtuelle URLs. Das bedeutet das unter dieser URL keine Datei vorhanden ist, sondern diese bei Bedarf dynamisch erzeugt werden. Um die Implementierung möglichst einfach zu halten haben wir uns dazu entschlossen unter der gegebenen URL (also z.B. `/rest/values`) eine

mit Platzhaltern versehene Datei abzulegen. Das Vorgehen mit Platzhaltern wurde schon im Projekt Radig verwendet um die Informationen in den statischen Hyper Text Markup Language (HTML)-Code einzufügen. Die Platzhalter werden vor dem Senden der Datei durch zugehörige Werte ersetzt.

Das Schema für so einen Platzhalter ist `%PINXY` und setzt sich zusammen aus dem Aufruf `%PIN`, dem anzusprechendem Port  $X = [A, C \text{ oder } D]$  und dem Pin  $Y = [0-7]$  (z.B. `%PINC1`). Mit diesem Platzhalter kann der direkte Pin ausgelesen (hier Port C und Pin 1) und an eine beliebige Stelle im Quellcode platziert werden. Neu hinzugekommen ist das Ausgeben der Information, ob der konkrete Pin über das DD-Register als Ein- oder Ausgang definiert ist. Das Schema ist für diesen Platzhalter ist `%DDRXY` und setzt sich zusammen aus dem Aufruf `%DDR`, dem anzusprechendem Port  $X = [A, C \text{ oder } D]$  und dem Pin  $Y = [0-7]$  (z.B. `%DDR1`).

#### 9.1.5 Erweiterung der POST-Parameter

Um Manipulationen an dem Board vorzunehmen haben wir uns entschlossen das vorhandenen System zu erweitern. Hierbei werden bei jeder HTTP-Anfrage die POST-Parameter ausgewertet und das Board entsprechend manipuliert.

Das Setzen der Ausgänge wird über einen HTTP-Post Aufruf getätigt. So können die Pins des entsprechenden Ports gesetzt oder umgeschaltet werden. Die Informationen, die zum Setzen eines Ports benötigt werden, setzen sich zusammen aus einem SET Befehl und dem Aufruf `PORTXYZ` zum Setzen oder dem SET Befehl und dem Aufruf `OUTXYZ`. X steht für den anzusprechendem Port  $X = [A, C \text{ oder } D]$ . Y und Z stehen für die Einstellung der Pins in hexadezimaler Schreibweise (00-FF) Abschließend muss das Ende der Schaltanweisung mit SUB gekennzeichnet werden, da der Webserver auf diese Steuerzeichen prüft. Ein Beispiel Post zum Setzen der Pins C0-C3 auf Ein, der Pins C4-C7 auf Aus, dem Umschalten der Pins D0-D3 als Ausgang und der Pins D4-D7 als Eingang sieht folgendermaßen aus:

`SET=PORTC0F&SET=OUTD0F&SUB=Senden`

#### 9.1.6 Implementierung der REST-Schnittstelle auf dem Client

Auf der Webseite müssen die vom Server bereitgestellten REST-URLs im Hintergrund aufgerufen werden können. Hierzu verwendeten wir die bereits in JavaScript enthaltene Asynchronous JavaScript and XML (AJAX) Bibliothek. AJAX erlaubt JavaScript URLs im Hintergrund zu laden, ohne das der Nutzer dies bemerkt oder die Seite neu geladen wird.

```
1 function loadValues() {  
2     var x=new XMLHttpRequest();  
3     x.open("GET", "/rest/values", false);  
4     x.send();  
5  
6     return JSON.parse(x.responseText);  
7 }
```

Abbildung 6: JavaScript um /rest/values abzufragen und in ein Objekt zu parsen

Der in Abbildung 6 beispielhaft gezeigte Quelltext ermöglicht das Laden der URL /rest/values. Hierfür wird ein XMLHttpRequest-Objekt verwendet und der geladene Text (x.responseText) wird mit JSON.parse(...) in ein JavaScript-Objekt verwandelt.

Sämtliche Server-Kommunikation findet ausschließlich in rest.js statt. Hier werden Methoden zum Ansprechen des Servers bereitgestellt, welche überall verwendet werden können ohne Netzwerkcode zu schreiben. Um beliebige Dateien abzufragen gibt es die Methoden loadURL(url) und loadURLAsync(url, postParams, func). loadURL(url) lädt die Datei synchron und gibt den erhaltenen Text zurück. Das synchrone Laden bedeutet, dass der Aufruf dieser Methode das Programm blockiert, bis die Datei vollständig geladen wurde. loadURLAsync(url, postParams, func) hingegen lädt die Datei asynchron im Hintergrund. Sobald das Laden abgeschlossen ist wird die als Parameter übergebene Funktion func aufgerufen. loadURLAsync(url, postParams, func) kann außerdem ein Text als POST-Parameter übergeben werden. Dies ermöglicht das Manipulieren des Boards über die POST-Parameter, welche vom Board interpretiert werden.

rest.js ruft loadURLAsync(...) in einer festen Frequenz kontinuierlich auf. Nach jeder Aktualisierung wird eine Funktion aufgerufen, welche über die Methode setOnValuesChanged(func) gesetzt werden kann.

Alle von rest.js zur Verfügung gestellten Funktionen, sowie deren Verwendung sind der ausführlichen Sourcecode-Dokumentation zu entnehmen.

## 9.2 Werkzeuge

### 9.2.1 Das Atmel Studio

Das Atmel Studio ist als Integrated Development Environment (IDE) das zentrale Werkzeug, um einen Mikrocontroller zu programmieren. Dabei ist das Atmel Studio, welches von Atmel nach einer kostenlosen Registrierung beziehbar ist, stark an die Microsoft Visual IDE angelehnt. Wenn man bereits Erfahrungen mit Visual Studio gesammelt hat, wird man sich schnell im Atmel Studio zurechtfinden. Zu den

reinen Programmierwerkzeugen gibt es im Atmel Studio noch ein paar weitere Funktionen, die das Arbeiten mit den Mikrocontroller erleichtern. Das Atmel Studio kann unter folgender Adresse heruntergeladen werden: <http://www.atmel.com/tools/ATMELSTUDIO.aspx>

#### 9.2.1.1 Device Programming

Eine Kernkomponente beim Atmel Studio ist das „Device Programming Fenster“. Erreicht werden kann es über „Tools → Device Programming“. Hier kann die aktuelle Verbindung mit dem Mikrocontroller getestet werden. Die Einstellungen werden hier aus den Projekt-Einstellungen entnommen, können aber auch frei gewählt werden. Dazu gehören: das momentan angeschlossene Programmierwerkzeug, den verwendeten Mikrocontroller und die Schnittstelle. Durch Drücken der „Read“ Buttons kann die Korrektheit der Verbindung geprüft werden. Wenn es zu keiner Fehlermeldung kommt und die Ziel Spannung bei ungefähr 5V liegt, ist der Mikrocontroller korrekt angeschlossen.

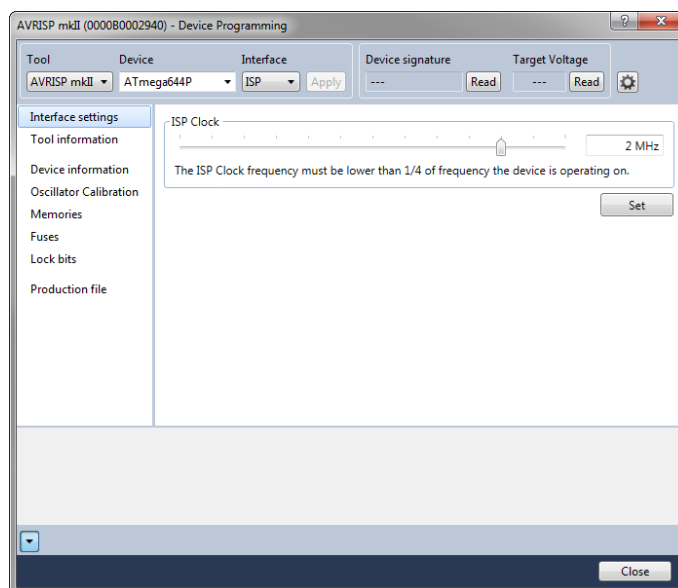


Abbildung 7: DeviceProgramming

Weiter können über den Abschnitt „Fusebits“ die entsprechenden Fuse Einstellungen getroffen werden, dazu mehr im Kapitel Fusebits 5.3.

#### 9.2.1.2 Projekt Einstellungen

Hier kann das Werkzeug zum Programmieren, die Schnittstelle und die Übertragungstakt gewählt werden. Der Übertragungstakt sollte auf ungefähr 1 MHz gesetzt werden, damit er zum einen nicht zu langsam, zum anderen auch nicht zu schnell ist. Der Takt darf nicht 1/8 des Prozessortaktes überschreiten.

### 9.2.2 AVRDUDE

Avrdude ist ein alternatives Werkzeug zum Bearbeiten von Mikrocontrollern. Es ist im Gegensatz zum Atmel Studio lediglich ein Konsolenwerkzeug ohne grafische Oberfläche. Mit Avrdude können unter anderem die Fusebits von einem Mikrocontroller gelesen und gesetzt werden. Weiter ist es möglich eine Sicherung vom aktuellen Stand des Mikrocontrollers herzustellen oder eine Hex Datei auf den Mikrocontroller aufzuspielen. Avrdude unterstützt eine Reihe von ISP Geräten, unter anderem auch den von uns verwendeten Atmel AVRISPmkII. Der Einsatz von Avrdude war notwendig, da die Fusebits von einem neuen Mikrocontroller standardmäßig auf den internen Quarz-Kristall gesetzt sind und nicht auf den externen Kristall des AVR-NET-IO Boards. Eine genaue Anleitung gibt es im Kapitel 10 Benutzerhandbuch.

### 9.2.3 HTML Header Compiler

Zum automatischen Umwandeln der Quell-Dateien (html, js, png etc. ...) haben wir einen speziellen HTML Header Compiler entwickelt, der die Website in eine für den Webserver verständliche Headerdatei umwandelt. Der Compiler durchsucht den Eingabe-Ordner und sammelt die darin enthaltenen Dateien. Daraus entsteht dann die für das Projekt benötigte Header Datei, bestehend einem Array von Buchstaben für jede Datei. Der HHC ist den Projektdateien, mit denen diese Dokumentation ausgeliefert wurde beigelegt, kann aber auch zusammen mit dem Projekt auf Github <https://github.com/doofmars/Embedded-Webserver> bezogen werden.

Eine Beispielumwandlung:

```
1 <HTML>
2   <HEAD>
3     <TITLE>
4       A Small Hello
5     </TITLE>
6   </HEAD>
7 <BODY>
8   <H1>Hi</H1>
9   <P>This is very minimal "hello world" HTML document.</P>
10 </BODY>
11 </HTML>
```

Abbildung 8: index.htm

Das eingegebene Verzeichnis, welches die index.html Datei enthält, wird eingelesen und in die folgende Headerdatei umgewandelt. Abbildung 8 zeigt eine simple „Hallo Welt“ Website, die keine weitere Funktion besitzt. Nachdem die Website mit dem HTML-Header-Compiler umgewandelt wurde, entsteht die Headerdatei aus Abbildung 9. Gut zu erkennen ist das Array aus Buchstaben in hexadezimaler Schreibweise, welches die index.html Datei widerspiegelt. Am Ende der Headerdatei ist ein weiteres Array, bestehend aus Schlüssel-Wert-Paaren für den Webserver. Hier wird hinterlegt, welche Dateien vorhanden sind. Der letzte Eintrag in diesem Array signalisiert das Ende der Suche und ist die Bedingung für den Webserver, um zur Standardausgabe zu wechseln. Bsp: Eine HTML Datei wird angefordert, welche nicht existent ist, resultiert in der Ausgabe von index.html.



```

1 #ifndef _WEBPAGE_H
2 #define _WEBPAGE_H
3
4 //item_0: index.html (145 Bytes, 177 Bytes in file)
5 const PROGMEM char item_0[] = {
6     0x3C, 0x48, 0x54, 0x4D, 0x4C, 0x3E, 0x20, 0x3C, 0x48, 0x45,
7     0x41, 0x44, 0x3E, 0x20, 0x3C, 0x54, 0x49, 0x54, 0x4C, 0x45,
8     0x3E, 0x20, 0x41, 0x20, 0x53, 0x6D, 0x61, 0x6C, 0x6C, 0x20,
9     0x48, 0x65, 0x6C, 0x6C, 0x6F, 0x20, 0x3C, 0x2F, 0x54, 0x49,
10    0x54, 0x4C, 0x45, 0x3E, 0x20, 0x3C, 0x2F, 0x48, 0x45, 0x41,
11    0x44, 0x3E, 0x3C, 0x42, 0x4F, 0x44, 0x59, 0x3E, 0x20, 0x3C,
12    0x48, 0x31, 0x3E, 0x48, 0x69, 0x3C, 0x2F, 0x48, 0x31, 0x3E,
13    0x20, 0x3C, 0x50, 0x3E, 0x54, 0x68, 0x69, 0x73, 0x20, 0x69,
14    0x73, 0x20, 0x76, 0x65, 0x72, 0x79, 0x20, 0x6D, 0x69, 0x6E,
15    0x69, 0x6D, 0x61, 0x6C, 0x20, 0x22, 0x68, 0x65, 0x6C, 0x6C,
16    0x6F, 0x20, 0x77, 0x6F, 0x72, 0x6C, 0x64, 0x22, 0x20, 0x48,
17    0x54, 0x4D, 0x4C, 0x20, 0x64, 0x6F, 0x63, 0x75, 0x6D, 0x65,
18    0x6E, 0x74, 0x2E, 0x3C, 0x2F, 0x50, 0x3E, 0x3C, 0x2F, 0x42,
19    0x4F, 0x44, 0x59, 0x3E, 0x3C, 0x2F, 0x48, 0x54, 0x4D, 0x4C,
20    0x3E, 0x25, 0x45, 0x4E, 0x44};
21
22 //file index (total webpage size: 145 Bytes / 0 Kilobyte)
23 WEBPAGE_ITEM WEBPAGE_TABLE[] = {
24     {"index.html", item_0},
25     {NULL, NULL}
26 };
27
28 #endif //_WEBPAGE_H

```

Abbildung 9: webpage.h

Anzumerken ist, dass standardmäßig der HTML Header Compiler die eingegebenen HTML und JS Dateien optimiert in die Headerdatei speichert. Dazu wird die Formatierung für den Zeilenvorschub, Tabulator oder Wagenrücklauf entfernt. Falls dies für die Entwicklung nicht gewünscht ist, kann man diese Funktion durch den Parameter `-n` oder `-newline` deaktivieren. Außerdem ist die entstandene Headerdatei nicht zur Bearbeitung gedacht. Die Bearbeitung erfolgt ausschließlich im Quelltext (z.B. der `index.html`). Analog kann man dazu einen beliebigen Compiler einer herkömmlichen Programmiersprache sehen. Dieser erzeugt aus dem Quelltext eine Binärdatei, diese ist nicht unbedingt für Menschen lesbar. Bei Änderungen wird der Quelltext angepasst und neu kompiliert. Da die Headerdatei folglich nicht bearbeitet werden muss, haben wir uns für eine Ausgabe als Array aus Buchstaben in hexadezimaler Schreibweise entschieden, da es eine einfachere programmatische Strukturierung erlaubt.

#### 9.2.4 AVRISPmkII

Der AVRISPmkII ist ein einfacher ISP Programmierer. Mit ihm ist es nur möglich Programme auf den Mikrocontroller zu übertragen. Es ist nicht möglich das Programm zu debuggen. Durch den Mangel der Debugfunktion ist der AVRISPmkII relativ günstig und kann auch durch andere baugleiche ISP Programmer ersetzt werden. Für den einfachen Anschluss mit dem AVR-Net-IO kann ein Adapter angefertigt werden, zur Verbindung mit dem Computer nutzt der AVRISPmkII eine USB Verbindung.

#### 9.2.5 AVRJTAGICEmkII

Der AVRJTAGICEmkII ist ein Programmer der neben der ISP Schnittstelle zum Programmieren auch JTAG unterstützt. Über das Atmel Studio ist es einfach möglich mit JTAG das Programm zu debuggen. Der Anschluss von ISP oder JTAG wird im Benutzerhandbuch genauer erklärt.

## 10 Benutzerhandbuch

### 10.1 WinAVR Projekt zu Atmel Studio

Um ein bestehendes Projekt in Atmel Studio zu importieren, muss zuerst ein neues, leeres Projekt angelegt werden. Dafür wählt man im „New Project“ Dialog ein neues GCC C Executable Project aus. Außerdem kann man hier auch den Namen und Speicher-Ort angeben (Abbildung 10). Nach der Auswahl des Projektes erscheint eine Tabelle mit den verschiedenen Mikrocontroller, die unterstützt werden. Hier wählt man den Ziel Controller aus (Abbildung 11)

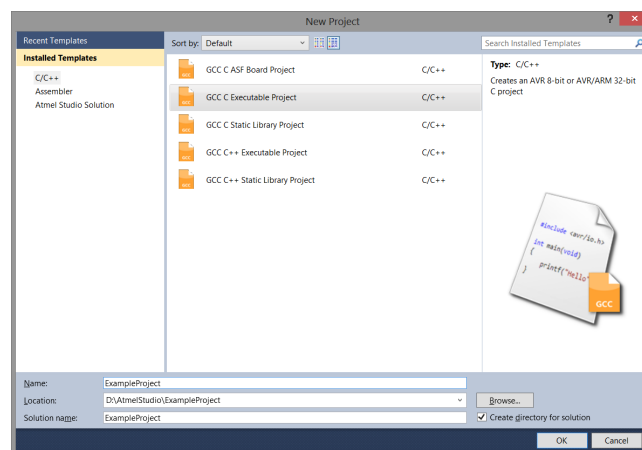


Abbildung 10: Atmel Studio, Neues Projekt

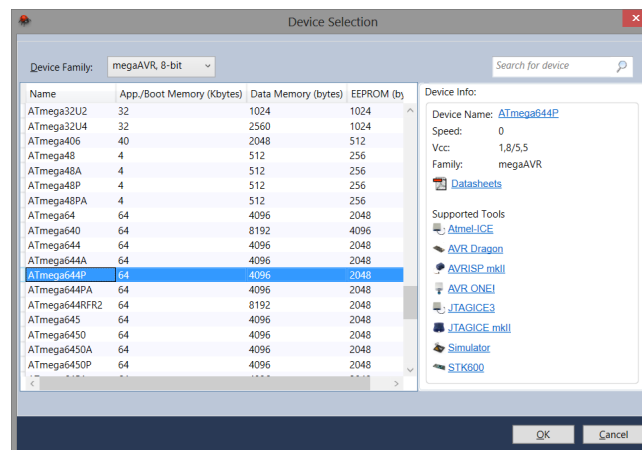


Abbildung 11: Atmel Studio, Device Selection

Nachdem das Projekt erstellt wurde, gibt es im Solution Explorer bereits eine main C Dateien. Diese kann gelöscht werden, da ein neues bestehendes Projekt importiert wird (Abbildung 12). Im nächsten Schritt müssen die neuen Projektdateien ausgewählt und importiert werden. Dies geschieht über einen Rechtsklick in den Solution Explorer (Abbildung 13) → add → Existing Item. Im geöffneten Dateiauswahldialog kann das bestehende Projekt ausgewählt werden (Abbildung 14).

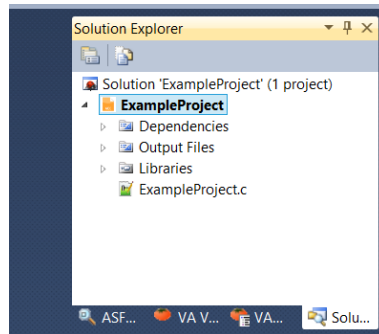


Abbildung 12: Atmel Studio, Solution Explorer

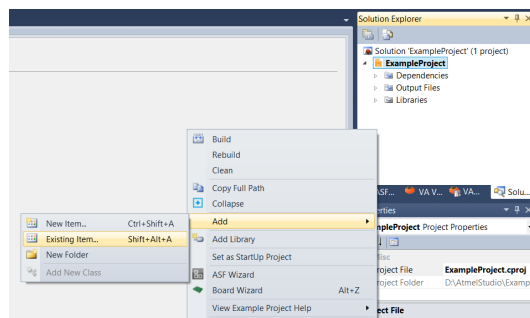


Abbildung 13: Atmel Studio, Add Existing Item 1

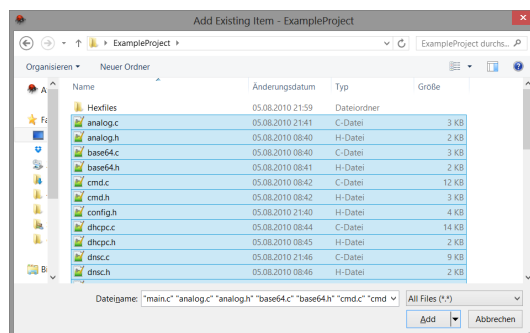


Abbildung 14: Atmel Studio, Add Existing Item 2

## 10.2 Einen Mikrocontroller austauschen

Für unser Projekt sollen alle notwendigen Programmbestandteile sowie die gesamte Webseite auf dem Mikrocontroller gespeichert werden. Der beim AVR-Net-IO mitgelieferte ATmega32 bietet hierfür jedoch nicht ausreichend Speicher. Wir haben uns deswegen für den aus der gleichen Baureihe stammenden ATmega644P entschieden, der mit seinen 64KB Programmspeicher den doppelten Speicherplatz bietet als der kleinere ATmega32.

Für den Wechsel ist es notwendig, den alten Controller vom Sockel zu entfernen und den neuen Controller einbauen zu können. Hierfür gibt es spezielle Werkzeuge, doch wenn man beim Vorgang Vorsicht walten lässt, kann man den Controller auch mit einem kleinen, möglichst breiten, Schlitzschraubendreher entfernen. Dazu zuerst den Mikrocontroller vorsichtig mit dem Schraubendreher als Hebel wie in Abbildung 15 lösen.

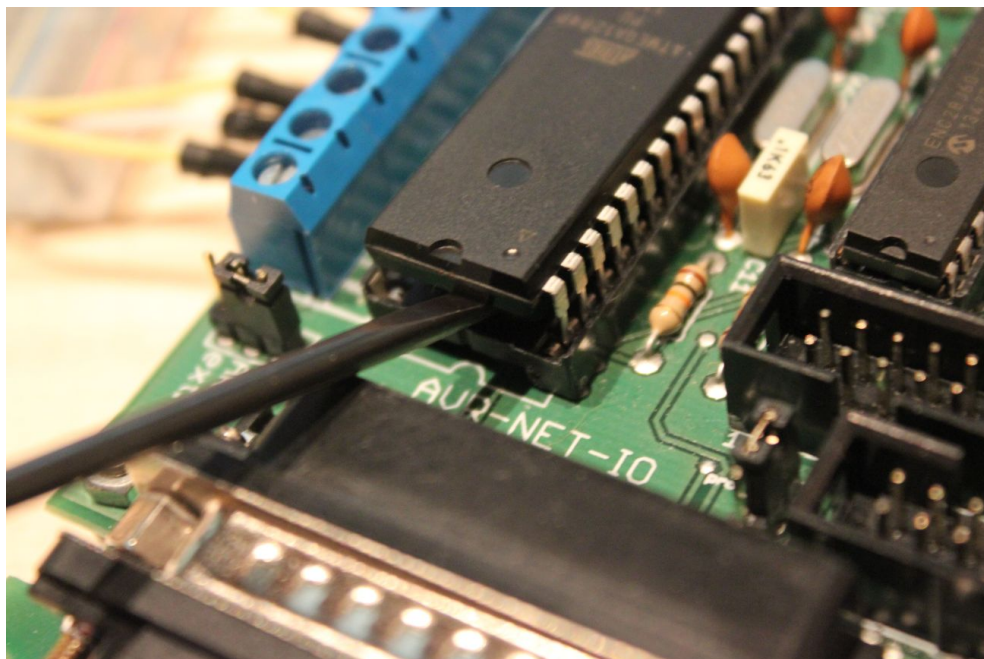


Abbildung 15: Schraubendreher am Controller

Zum einfachen Lösen kann der Hebel auch von der anderen Seite angesetzt werden. Anschließend den gelösten Prozessor abziehen.

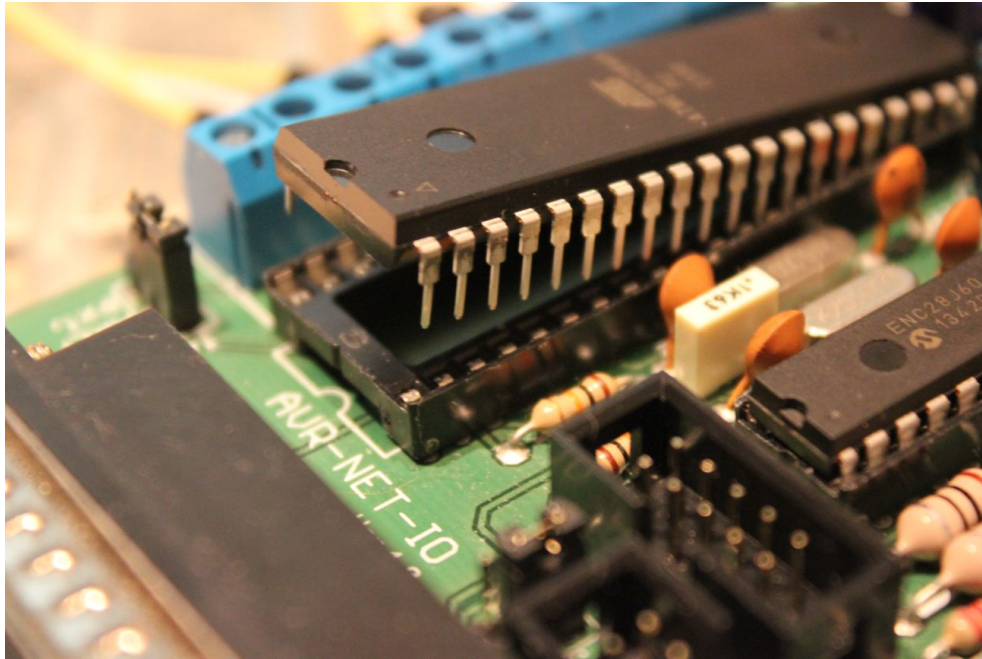


Abbildung 16: Der gelöste Mikrocontroller

Nachdem der Mikrocontroller entfernt wurde, hat man einen guten Blick auf den Sockel (Abb. 17).

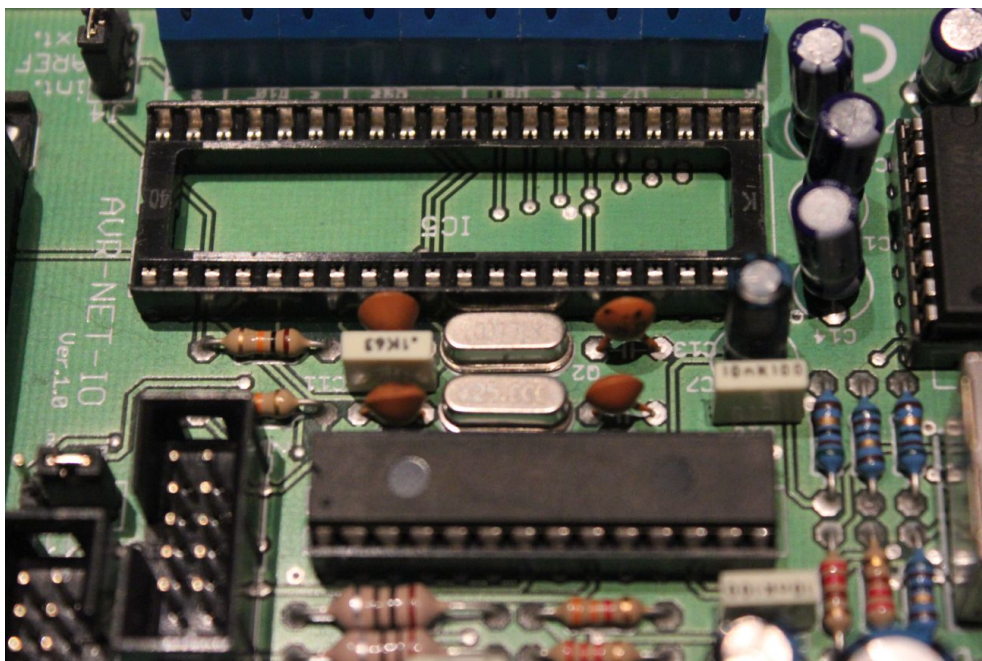


Abbildung 17: Der Sockel auf dem AVR-Net-IO

Beim Einbau ist unbedingt darauf zu achten, den neuen Mikrocontroller entsprechend der D-förmigen Einkerbung in den Sockel zu setzen. (Siehe Abbildung 18)



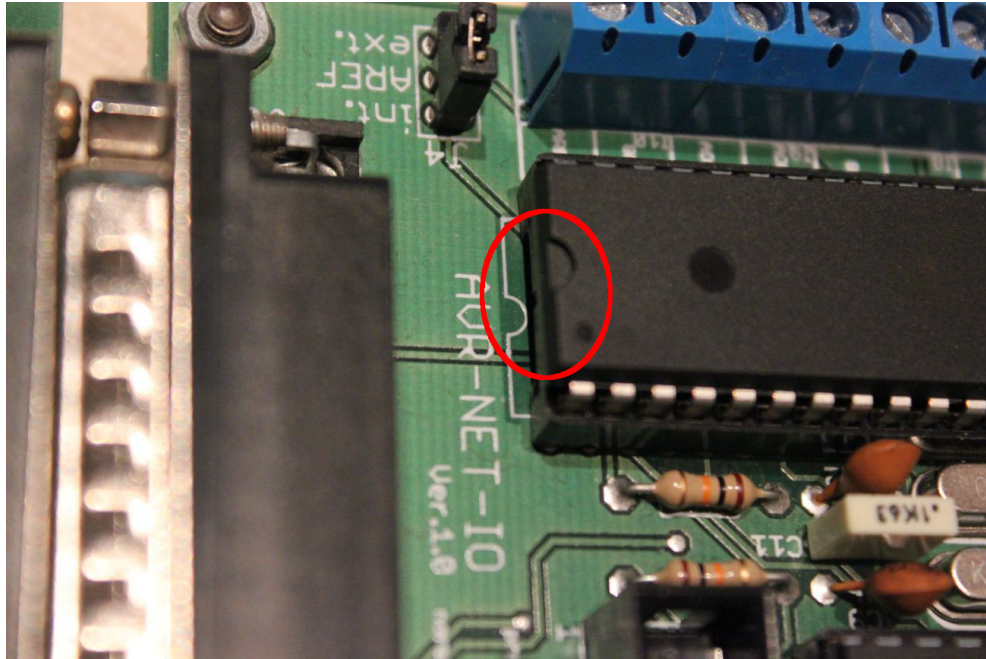


Abbildung 18: Markierung zum Einbau

### 10.3 ISP-Programmer anschließen

Der Anschluss des AVRISPmkII Programmers erfolgt über einen 6 Poligen Stecker, allerdings hat das AVR-Net-IO einen 10 Poligen Stecker. Deswegen wurde hierfür eigens ein Adapter angefertigt, welcher das ISP Signal von den 6 Polen des Programmers auf das AVR-Net-IO bringt.

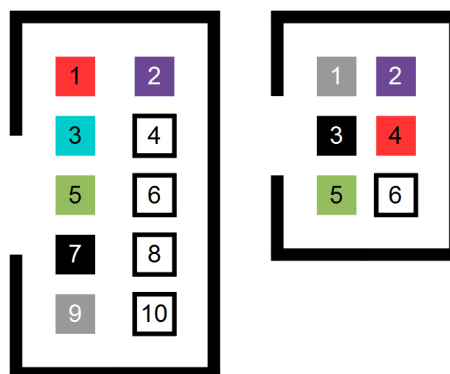


Abbildung 19: Schematische Darstellung des ISP Anschlusses (Pin 1 &amp; 2 ist auf der Platine markiert)

10-poliger Anschluss	6-poliger Anschluss
1 MOSI	1 MISO
2 VCC	2 VCC
3 - (*)	3 SCK
4,6,8,10 GND	4 MOSI
5 RESET	5 RESET
7 SCK	6 GND
9 MISO	

Tabelle 5: Die Pinbelegung für den 6 und 10 poligen Anschluss [mik14]



Abbildung 20: Der Adapter

Mit dem angefertigten Adapter kann der Debugger anschließend ganz einfach mit dem Board verbunden werden. Wenn der Mikrocontroller richtig, wie im Abschnitt 10.4 beschrieben, konfiguriert ist, kann der Programmer auch in Atmel Studio verwendet werden.

#### 10.4 Einrichten eines neuen Mikrocontrollers

Für einen neuen Chip ist es anfangs notwendig die Fuse-Bits richtig zu setzen, damit der Chip ordnungsgemäß arbeitet. Dies ist jedoch im AtmelStudio nicht möglich, da es nicht möglich ist die exakte Geräte-Signatur auszulesen. Das Problem liegt darin, dass standardmäßig die Fuses auf den internen Quarz-Kristall gesetzt sind, und nicht auf den externen Kristall des AVR-NET-IO Boards. Beim Versuch die Fuse-Bits zu setzen, erscheint im Atmel Studio die Fehlermeldung aus Abbildung 21.

Abhilfe schafft hier die alternative Programmiersoftware AVRDUDE, mit ihr ist es möglich die Fuse-Bits zu ändern. Unter Linux kann dieser einfach über die Paketquellen installiert werden, für ein Windows Betriebssystem kann eine ausführbare Kommandozeilen-Anwendung auf der Projekt-Website heruntergeladen werden <http://savannah.nongnu.org/projects/avrdude>. Zusätzlich muss für Windows noch libusb-win32 (<http://sourceforge.net/projects/libusb-win32/>) vorhanden sein, dass der Programmer mit den gewählten Parametern verwendet werden kann. Eine



ausführliche Anleitung gibt es hier: <http://eliaselectronics.com/using-the-avrispmkii-with-avrdude-on-windows/>

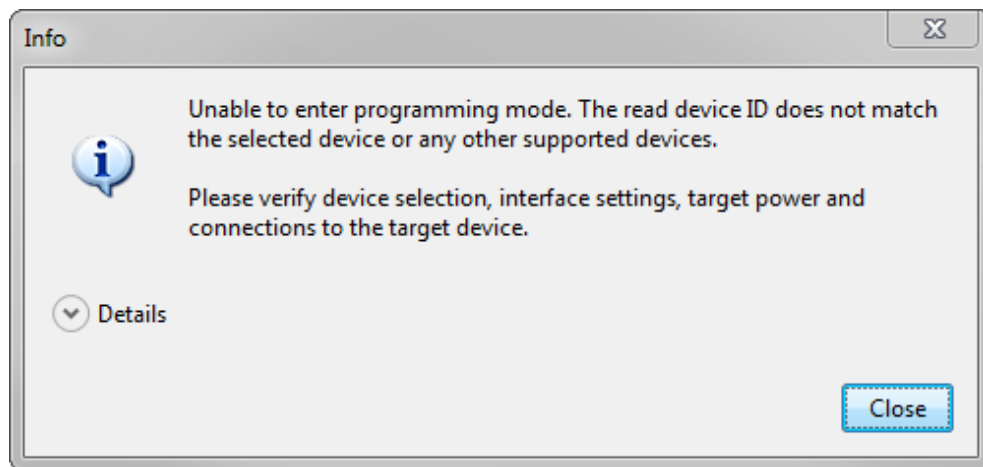


Abbildung 21: DeviceProgramming

Die in folgendem Beispiel angezeigten Befehle sind die von uns verwendeten Fuse Einstellungen. Für eine genauere Beschreibung, wofür die einzelnen Fuse-Bits verwendet werden, ist der Abschnitt 5.3 Fusebits im Kapitel „Hardware“.

Anschließend kann der Mikrocontroller zusammen mit dem AV-Net-IO und AtmelStudio programmiert werden. Der verwendete Mikrocontroller wird jetzt richtig erkannt, da es auch keine Probleme mit der Gerätesignatur gibt.

Auslesen Linux:	<code>sudo avrdude -P usb -p m644p -c avrispmkII -U lfuse:r:-:h -U hfuse:r:-:h -B 22</code>
Setzen Linux:	<code>sudo avrdude -P usb -p m644p -c avrispmkII -U lfuse:w:0xFF:m -U hfuse:w:0xD6:m -B 22</code>
Auslesen Windows:	<code>avrdude.exe -p m644p -c avrispmkII -U lfuse:r:-:h -U hfuse:r:-:h -B 22</code>
Setzen Windows:	<code>avrdude.exe -p m644p -c avrispmkII -U lfuse:w:0xFF:m -U hfuse:w:0xD6:m -B 22</code>

Tabelle 6: Auslesen und setzen von Fuse-Bits des ATmega644P mit AVRDUDE

```

jan@cypher: ~
avrdude: AVR device initialized and ready to accept instructions

Reading | ##### | 100% 0.01s

avrdude: Device signature = 0x1e960a
avrdude: reading input file "0xFF"
avrdude: writing lfuse (1 bytes):

Writing | ##### | 100% 0.00s

avrdude: 1 bytes of lfuse written
avrdude: verifying lfuse memory against 0xFF:
avrdude: load data lfuse data from input file 0xFF:
avrdude: input file 0xFF contains 1 bytes
avrdude: reading on-chip lfuse data:

Reading | ##### | 100% 0.00s

avrdude: verifying ...
avrdude: 1 bytes of lfuse verified
avrdude: reading input file "0xD6"
avrdude: writing hfuse (1 bytes):

Writing | ##### | 100% 0.01s

avrdude: 1 bytes of hfuse written
avrdude: verifying hfuse memory against 0xD6:
avrdude: load data hfuse data from input file 0xD6:
avrdude: input file 0xD6 contains 1 bytes
avrdude: reading on-chip hfuse data:

Reading | ##### | 100% 0.00s

avrdude: verifying ...
avrdude: 1 bytes of hfuse verified

avrdude: safemode: Fuses OK

avrdude done. Thank you.

```

Abbildung 22: AVRDUDE Ausgabe

-p partno	<p>This is the only option that is mandatory for every invocation of avrdude. It specifies the type of the MCU connected to the programmer. These are read from the config file. If avrdude does not know about a part that you have, simply add it to the config file (be sure and submit a patch back to the author so that it can be incorporated for the next version).</p> <p><b>m32 ⇒ ATmega32</b>  <b>m644p ⇒ ATmega644P</b>  <b>m1284p ⇒ ATmega1284P</b></p>
-P port	<p>Use port to identify the device to which the programmer is attached. <b>usb für den AVRISP MKII</b></p>
-c programmer-id	<p><b>avrispmkII für den AVRISP MKII</b></p>
-U memtype:op:filename:filefmt	<p>The memtype field specifies the memory type to operate on.</p> <p><b>hfuse</b> The high fuse byte.  <b>lfuse</b> The low fuse byte.</p> <p>The op field specifies what operation to perform:  <b>r</b> read device memory and write to the specified file  <b>w</b> read data from the specified file and write to the device memory</p> <p>The filename field indicates the name of the file to read or write. The format field is optional and contains the format of the file to read or write.</p> <p><b>Hier die Bytes die gesetzt werden 0xFF bzw 0xD6</b></p>
-B bitclock	<p>Specify the bit clock period for the JTAG interface or the ISP clock</p>

Tabelle 7: Auszug AVRDUDE Parameter

## 10.5 HTML Header Compiler

Da der HTML Header Compiler in Java entwickelt wurde, muss für die Verwendung die Java Laufzeitumgebung ab Version 6 installiert sein. Zum Ausführen des Compilers muss zuerst mit einer Konsole in den Entsprechenden Ordner navigiert werden. Anschließend kann mit folgendem Befehl die Datei ausgegeben werden.

```
java -jar hhc.jar -in <INPUT FOLDER> -out <OUTPUT FILE>
```

Die Angaben in den spitzen Klammern müssen durch den entsprechenden Pfad und die entsprechende Datei ausgetauscht werden. Standardmäßig optimiert der HTML Header Compiler die eingegebenen Dateien, falls dies nicht gewünscht ist gibt es zusätzlich zu den vorgegebenen Optionen weitere Flags die gesetzt werden können. Hier alle Parameter im Überblick:

-in, -input	Der Eingabepfad mit allen für die Website benötigten Dateien. z.B. -in "Webseite"
-out, -output	Die Ausgabe Headerdatei z.B. -out "Webserver/webpage.h"
-v, -verbose	Gibt die Dateiausgabe auf der Konsole aus.
-n, -newline	Behält die Formatierung für den Zeilenvorschub, Tabulator oder Wagenrücklauf in den HTML und JS Dateien (\n and \r \t). Benötigt dadurch abhängig von der Website mehr Speicher, ermöglicht aber ein einfacheres Debuggen von eingebundenem JavaScript Code.

Tabelle 8: Parameter des HTML Header Compiler

Um die Entwicklung zu vereinfachen ist es hilfreich, wenn für den Parameterruf des HTML Header Compilers ein einfaches Shell- oder Batch-Script erstellt wird, das die Dateien aus dem Ordner für die Website als Headerdatei in den Ordner für den Webserver schreibt. Falls eine Änderung an der Website vorgenommen wurde, muss vor dem Programmieren des Mikrocontrollers lediglich der HHC ausgeführt werden.

```

1 #!/bin/sh
2 java -jar "HTML Header Compiler/latest release/hhc.jar" -in "
   Webseite" -out "Webserver/webpage.h"
3
4 pause
```

Abbildung 23: BuildWebpage.sh für Linux

```
1 java -jar "HTML Header Compiler\latest release\hhc.jar" -in Webseite  
  -out Webserver\webpage.h  
2  
3 pause
```

Abbildung 24: BuildWebpage.bat für Windows

## 10.6 Konfiguration des Webservers

Die Einstellung des Webservers erfolgt über die config.h Datei. In der config.h Datei, können die verschiedenen Pins der Ports als Ein- oder Ausgang definiert werden. Dabei gibt es ein paar Eigenheiten zu beachten:

- OUTA steht für den A Port, hier ist zu beachten, dass dieser Port die Analog zu Digital Wandler beherbergt. Mit aktiviertem Wandler ist es nicht möglich, die Pins des Ports funktionierend als Ausgänge zu schalten, da die Spannung nicht gehalten wird.
- OUTB ist nur mit Vorsicht zu genießen. Hier handelt es sich um den Port, der auf dem AVR-Net-IO für die Netzwirkkommunikation genutzt wird. Deswegen wird Port B auch nicht standardmäßig definiert.
- OUTC dieser Port wird von Pollin standardmäßig für die Ausgänge verwendet und ist von uns bereits entsprechend modifiziert. Der gesamte Port wird auf dem AVR-NET-IO über den 25Pin D-Sub Stecker geleitet. Wenn der Fuse-Bit für JTAG geschaltet ist, werden 4 Pins des C Ports für das JTAG Interface verwendet.
- OUTD Liegt auf dem AVR-Net-IO auf dem EXT Anschluss und ist für erweiterte Peripherie geplant, so kann hier ein Cardreader oder ein Erweiterungsboard angeschlossen werden.

Weiter kann die gewünschte IP-Adresse eingestellt werden, unter welcher das Gerät erreicht werden kann. Wichtig ist hier, dass kein anderes Gerät dieselbe Adresse im Netzwerk verwendet. Auch kann die Router IP-Adresse und Netzmaske angegeben werden. Eine weitere wichtige Einstellung ist die verwendete Mac Adresse des Netzwerkcontrollers. Diese wird über die Variablen MYMAC1-6 definiert.

## 10.7 Debuggen über JTAG

Der JTAGICEmkII Debugger von Atmel, den wir für unser Projekt gestellt bekommen haben, unterstützt neben ISP auch JTAG. Allerdings werden für den Anschluss von JTAG andere Pins benötigt als für den Anschluss eines ISP-Programmers.

Pin	Signal	I/O	Description
1	TCK	Output	Test Clock, clock signal from JTAGICE mkII to target JTAG port
2	GND	-	Ground
3	TDO	Input	Test Data Output, data signal from target JTAG port to JTAGICE mkII
4	Vtref	Input	Target reference voltage. Also used to power level converter inputs.
5	TMS	Output	Test Mode Select, mode select signal from JTAGICE mkII to target JTAG port
6	nSRST	Out/-In-put	Open collector output from adapter to the target system reset. This pin is also an input to the adapter so that the reset initiated on the target application board may be reported to the JTAGICE mkII
7	-	-	Not connected
8	nTRST	NC(Output)	Not connected, reserved for compatibility with other equipment (JTAG port reset)
9	TDI	Output	Test Data Input, data signal from JTAGICE mkII to target JTAG port
10	GND	-	Ground

Tabelle 10: JTAG Connections [Atm]

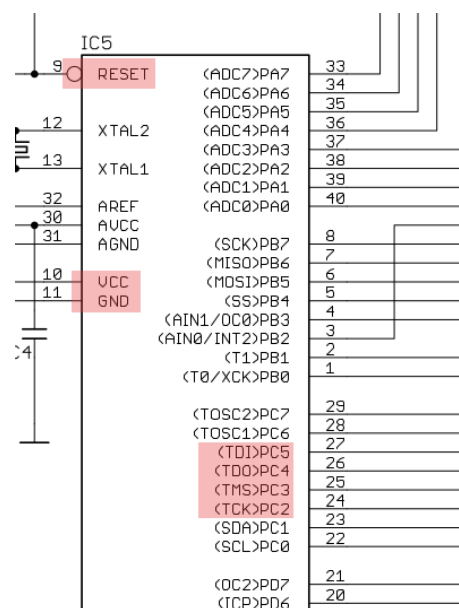


Abbildung 25: JTAG Pins

Die in Tabelle 10 gezeigten Pins entsprechen der Belegung der 10 Anschlüssen des JTAGICEmkII. Diese müssen mit den Pins 2-5 des C Ports, dem Reset Pin, Ground und VCC verbunden werden (Abbildung 25). Die Pins 7 & 8 des JTAGICEmkII werden nicht mit dem Mikrocontroller verbunden.

Damit JTAG funktioniert muss der entsprechende JTAG Fuse-Bit gesetzt sein. Dabei ist zu beachten, dass die Pins 2-5 am Port C des Mikrocontroller nicht für Ein- oder Ausgaben verwendet werden können, sondern bei aktivierten Fuse-Bit gesperrt sind. Als weiterer Hinweis ist zu beachten, dass die verwendete Schnittstelle in den „Projekt-Einstellungen“ umgestellt werden muss. Nach korrekter Verbindung, die im Device Manager überprüft werden kann, kann das Programm auf dem Mikrocontroller debuggt werden.

## 10.8 Die Website

Die Webseite ist in 3 Tabs eingeteilt.

### 10.8.1 Der Status Tab

Im Status Tab ist die Übersicht der einzelnen Pins und Ports des Boards. Fährt der Nutzer mit der Maus über die Pins des Boards, so erscheint auf der rechten Seite des Browser eine Sidebar mit den detaillierten Informationen zu diesem Pin. Hier können die Pins oder Ports als Favoriten gespeichert werden. Genauso können Werte und Skripte der einzelnen Pins oder Ports geändert werden.

Ebenfalls kann der Pin mit einem Mausklick gesetzt werden, oder auf der Sidebar in einer Checkbox gesetzt werden. Dies kann jedoch nicht mit jedem Pin oder Port gemacht werden, da diese Funktion bei manchen nicht möglich sein darf. Diese Pins sind vom System deaktiviert und nicht klickbar.

### 10.8.2 Der Favoriten Tab

Hier werden tabellarisch die ausgewählten Pins oder Ports angezeigt. Auch hier können wieder Pin spezifische Modifikationen vorgenommen werden. Löschen der Favoriten setzt den Pin auf die Standardeinstellungen zurück und ist nicht mehr in Favoritenliste vorhanden. Pins können wieder über den Status Tab hinzugefügt werden.

### 10.8.3 Der Einstellungs Tab

Im Einstellungs-Tab werden die Einstellungen vorgenommen. Hier sind auch die Informationen über das Board, Version der Webseite und Autoren abrufbar.

Die Import/Export Funktion ermöglicht die Datenbank, in der die Benutzereinstellungen und Favoriten untergebracht sind, zu exportieren und auf einem anderen System zu importieren. Die Aktualisierungsrate bestimmt wie schnell sich die Seite aktualisiert, das ist die einzige Einstellungen die vorgenommen werden kann.

#### 10.8.4 Nutzerspezifische Skripte

Nutzerspezifische Skripte können bei den Pins hinterlegt werden, diese werden auch ausgeführt. Diese können genauso ein Verbund aus Pins steuern und manipulieren. Um diese Skripte anpassen zu können werden

##### 10.8.4.1 Erstellen von Skripte

Bei den Pins können auch Skripte hinterlegt werden. Diese Skripte werden in JavaScript geschrieben nur werden die Tags nicht hinzugefügt. Vorrausgesetzt werden kleinere JavaScript Kenntnisse, was aber anhand den Beispielen und den nachfolgenden Skriptkommandos machbar ist.

Name	Beschreibung
getvalues(string id)	Damit wird der Wert des Pins mit der ID abgerufen
setvalues(string id, int value)	Damit wird folgender Wert für den Pin mit der ID gesetzt
getDB(string name, string value)	Damit wird aus der Browserdatenbank ein Wert(Standardwert) für ein bestimmtes Item (name) gesetzt
putDB(string name, int value)	Hiermit wird in die Datenbank das Item (name) mit dem value eingespeichert.

Tabelle 12: Einige wichtige Skriptkommandos

#### 10.8.4.2 Beispielskripte

Die folgenden Beispielskripte können verwendet werden, müssen eventuell noch auf die passenden Pins und Ports angepasst werden.

#### 10.8.4.3 Mehrere Lichter blinken

Hier werden mehrere LED an die Pins D2-D7 angeschlossen

```
1 if (getValue('D2') == "1") {  
2   setValue('D2',0);  
3   setValue('D3',1);  
4   setValue('D4',0);  
5   setValue('D5',1);  
6   setValue('D6',0);  
7   setValue('D7',1);  
8 } else {  
9   setValue('D2',1);  
10  setValue('D3',0);  
11  setValue('D4',1);  
12  setValue('D5',0);  
13  setValue('D6',1);  
14  setValue('D7',0);  
15 }
```

Abbildung 26: Beispielskript: Mehrere Lichter blinken lassen

Durch den Aufruf dieses Skriptes wird zuerst überprüft ob D2 auf 1 gesetzt ist, ist dies nicht der Fall wird D2 auf 1 gesetzt. Von D3-D7 werden Die Werte abwechselnd 0 und 1 gesetzt. Da dieses Skript dauernd erneut aufgerufen wird ist somit beim nächsten Aufruf D2 bei 1 und der Wert wird folglich auf 0 gesetzt. Alle anderen Werte werden auch geändert. Somit entsteht ein wechseln der Lichter.

#### 10.8.4.4 Umwandeln in digitale Werte

Hier werden auch wieder 7 LED verwendet, und ein Sensor angeschlossen an einen analogen Ports.



```
1 setValue( 'D2', 0);
2 setValue( 'D3', 0);
3 setValue( 'D4', 0);
4 setValue( 'D5', 0);
5 setValue( 'D6', 0);
6 setValue( 'D7', 0);
7
8
9 var value = getValue( 'A5' );
10
11 if (value > 869) {
12     setValue( 'D2', 1);
13 }
14 if (value > 719) {
15     setValue( 'D3', 1);
16 }
17 if (value > 569) {
18     setValue( 'D4', 1);
19 }
20 if (value > 419) {
21     setValue( 'D5', 1);
22 }
23 if (value > 269) {
24     setValue( 'D6', 1);
25 }
26 if (value > 119) {
27     setValue( 'D7', 1);
28 }
```

Abbildung 27: Beispielskript: Werte werden in digitale Werte umgewandelt

Je nachdem wie sich der Wert von dem Sensor ändert, ändern sich auch die digitalen Ausgänge zu den LED, ist der gelieferte Wert oberhalb von 869, so wird Anschluss D2 auf 1 gesetzt. Leuchte an D2 leuchtet somit. Da dann aber alle Abfragen wahr sind werden alle Lichter angeschaltet. sollte dann der Wert sich unterhalb von 869 befinden dann wird D2 ausgeschaltet. Somit ändern sich immer die Leuchten sobald die neuen Werte überprüft werden.

#### 10.8.4.5 Lichterlauf

Benötigt werden auch hier wieder 7 LED. Angeschlossen an D2 - D7.

Mittels diesem Skript werden forlaufend die Lichter nacheinander an und wieder abgeschaltet. Somit entsteht ein Lauf des Aufleuchtens.

```
1 var count = getDb("count", "0");
2
3 switch (count) {
4   case "0":
5     setValue('D7',0);
6     setValue('D2',1);
7     console.log(count);
8     count = 1;
9     break;
10  case "1":
11    setValue('D2',0);
12    setValue('D3',1);
13    count = 2;
14    break;
15  case "2":
16    setValue('D3',0);
17    setValue('D4',1);
18    count = 3;
19    break;
20  case "3":
21    setValue('D4',0);
22    setValue('D5',1);
23    count = 4;
24    break;
25  case "4":
26    setValue('D5',0);
27    setValue('D6',1);
28    count = 5;
29    break;
30  case "5":
31    setValue('D6',0);
32    setValue('D7',1);
33    count = 0;
34    break;
35 }
36
37 putDb("count", count);
```

Abbildung 28: Beispielskript: Lichterlauf

Zur Erklärung des Skriptes. In einem Case der Switch-Anweisung wird das vorherige LED ausgeschaltet, die nächste LED angeschalten und den Counter erhöht. Beim nächsten Durchlauf wiederholt sich das immer wieder.

## 10.8.4.6 Countdown mit Button

Für diese Skript wird die 7 Segment Anzeige und ein Button benötigt.

Diese Skript lässt die 7 Segment Anzeige nacheinander durchiterieren, sobald ein button eine bestimmte Zeit gedrückt wird.

```
1 var count = getDb("count", "0");
2 var timer = getDb("timer", "0");
3
4 var set7Digit = function(number){
5     var pins = [[0,0,1,0,0,0,0],
6                 [0,1,1,1,1,1,0],
7                 [0,0,0,1,0,0,1],
8                 [0,0,0,1,1,0,0],
9                 [0,1,0,0,1,1,0],
10                [1,0,0,0,1,0,0],
11                [1,0,0,0,0,0,0],
12                [0,0,1,1,1,1,0],
13                [0,0,0,0,0,0,0],
14                [0,0,0,0,1,0,0]]
15
16     setValue('C0', pins[number][0]);
17     setValue('C1', pins[number][1]);
18     setValue('C2', pins[number][2]);
19     setValue('C3', pins[number][3]);
20     setValue('C4', pins[number][4]);
21     setValue('C5', pins[number][5]);
22     setValue('C6', pins[number][6]);
23 };
24
25 if (getValue('A0') == "1") {
26     if (timer == "1") {
27         setValue('C7', 1);
28         set7Digit(count);
29         if (count > 8) {
30             count = 0;
31         } else{
32             count++;
33         }
34         putDb("count", count);
35     }
36     timer++;
37     putDb("timer", timer);
38 }
39 }
40
41 if (getValue('A0') == "0") {
```

```
42   setValue( 'C7', 0 );  
43   putDb( "timer", "0" );  
44 }
```

Sobald der Button angeschlossen an A0 gedrückt wird, wird ein Timer ausgelöst. Ist dieser Timer bei 8 und der Button noch gedrückt, wird bei den Anschlüssen für die Anzeige durchiteriert. Somit ändert sich nur die Anzeige sollte der Button gedrückt werden.

## 11 Rückblick

### 11.1 Soll/Ist-Vergleich

Der Projektverlauf war positiv und lief im Rahmen nach dem zu Beginn angefertigten Plan (Siehe im Kapitel „Projektplanung“), jedoch gab es einige kleine Änderungen, welche aber keine großen Auswirkungen hatten.

Im folgenden werden die drei aufgetretenen Probleme/Unstimmigkeiten kurz beschrieben und unsere Art, diese zu lösen.

Aufgetretene Probleme	Beschreibung	Änderungen
Anforderung Falsch	Vorgegeben war die Arbeit mit Qt. Qt ist eine C++ IDE und nicht für die Arbeit an einer Webseite geeignet	Webseitenentwicklung mit JavaScript und HTML
Zeitplan nicht eingehalten	Durch die Feldversuche mit dem Board rutschte das Team in die Implementierungsphase. Nachdem Radig-asis lief, versuchten wir die Pins zu manipulieren und stellten fest, dass diese Versuche nicht mehr als „Feldversuche“ zu deklarieren sind.	Meilenstein „Ende der Planungsphase“ wurde 2 Wochen vorgelegt und als „erreicht“ markiert. Die Implementierungsphase wurde dadurch 2 Wochen länger.

Fehlende Ressource: Debugger	Zu Beginn hatten wir einen reinen Programmierer, dh jegliche Arbeit am Board war reines Try and Error. Deshalb haben wir einen Debugger beantragt, auf welchen wir jedoch warten mussten, was zu einem Zeitverlust führte.	Arbeit an Board bis Erhalten des Debuggers try and Error Durch die vielen Arbeitspakete fokussierten wir uns in dieser Wartezeit mehr auf die Webseite und hatten somit kaum Zeitverlust.
---------------------------------	--	--

Tabelle 13: Soll/Ist Vergleich

## 11.2 Verworfenen Varianten

Es wurde als Erweiterung die Ansteuerung des Boards über ein Raspberry Pi als Master in Erwägung gezogen.

Dieser Vorschlag wurde von der Projektverwaltung zunächst zurückgewiesen, da eine serverbasierte Lösung angefordert war.

Die Erweiterung hierfür behielten wir uns im Hinterkopf, konnten sie aber aufgrund von Zeitmangel nichtmehr umsetzen, weshalb man eine genauere Beschreibung hierfür in Kapitel 12 findet.

## 11.3 Eigenbewertung

Christian Würthner

Das Projekt lieferte einen guten Einblick in die Welt der Mikrocontroller und der Webentwicklung. Wir konnten viele neue Techniken lernen und praktisch anwenden. Eine so komplexe Webanwendung mit möglichst wenig Speicherverbrauch zu entwickeln war eine große Herausforderung, es mussten immer Kompromisse zwischen Komfort, Funktionalität und Speicherverbrauch getroffen werden. Auch im Bereich Projektplanung haben wir wertvolle Erfahrungen gesammelt, die wir in das Projekt im 6. Semester mitnehmen werden.

Jan-Henrik Preuß

Trotz anfänglicher Schwierigkeiten hat mir das Projekt sehr viel Spaß gemacht. Vor allem das für Informatiker eher ferne Aufgabengebiete der Elektrotechnik hat mir einiges an Herausforderungen gegeben und mir gezeigt wie man mit wenig Ressourcen ein doch umfangreiches Projekt zustande bekommt. Die Arbeit im Team hat mir gut gefallen und ich habe einiges an praktischen Erfahrungen zur Planung und Umsetzung von einem Semesterprojekt bekommen.

Ann-Sophie Dietrich

Ich empfand das Projekt als lehrreich und hilfreich für das spätere Berufsleben.

Die regelmäßigen Meetings sowohl untereinander, als auch wöchentlichen Meetings mit Herrn Spale, gaben einen guten Einblick in das Berufsleben.

Innerhalb des Projektes habe ich ebenfalls viel gelernt, wie wichtig der Faktor Planung bei einem größeren Projekt ist, wie man Probleme bespricht und gemeinsam nach Lösungen sucht. Auch fachlich hat mich dieses Projekt weitergebracht. Ich hatte zuvor keine Erfahrung mit JavaScript und bekam durch die Arbeit an der Webseite einen guten Einblick.

Die Erfahrungen, welche ich innerhalb des Projektes bekommen habe, werde ich bei späteren Projekten einsetzen.

Marcel Schlipf

Durch das Projekt habe ich sehr viel neue Erkenntnis gewonnen. Angefangen in der Welt der Mikrokontroller bis hin zu Planung. So deckten die wöchentlichen Meetings neue Aufgaben und Probleme auf, die wiederum in Team erledigt, bzw. behoben worden sind. In Team selbst sind wieder gut miteinander ausgekommen.

Ich konnte mich in der Welt der Mikrokontroller noch nicht zurechtfinden, aber mein Wissen durch dieses Projekt erweitern und kann nun etwas mit diesen Geräten anfangen. Die kleinere Erfahrung in der Webseitenentwicklung konnte ich in das Projekt einbringen und ein wenig weiterentwickeln.

Die Erkenntnis aus diesem Projekt und aus der Vorlesung Projektmanagement werde ich auch im 6.Semester im Projekt anwenden.





## 12 Ausblick

In der nächsten Ausbaustufe soll ein Raspberry Pi zwischen den Mikrocontroller und die Webseite geschaltet werden. Dieses System eröffnet eine Vielzahl neuer Möglichkeiten, so können von einer Webseite aus mehrere Pollin Net-IO Boards verwaltet werden und die Favoriten und Skriptfunktionen zentral auf dem Raspberry Pi gespeichert und von allen Clients verwaltet werden. Auch das Pushen von Messwerten könnte über dieses System gelöst werden, so muss die Webseite nicht ständig die Werte pollen. Für das neue System sind einige Änderungen nötig, diese beschränken sich aber größtenteils auf den Server, welcher neu eingerichtet werden muss.

### 12.1 Struktur des neuen Systems

Das neue System besteht folglich aus beliebig vielen Pollin Net-IO Boards, einem Raspberry Pi und einem oder mehreren Clients.

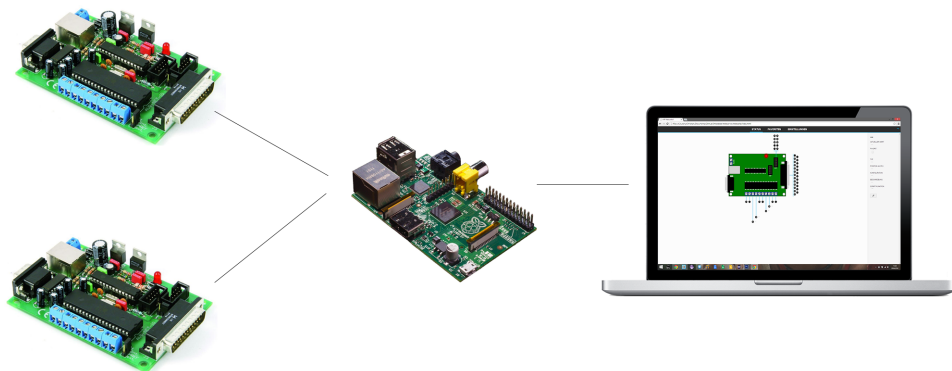


Abbildung 29: Der grobe Aufbau des neuen Systems, links zwei Pollin Net-IO Boards, in der Mitte ein Raspberry Pi und rechts ein Client

Die Clients fragen alle Daten, welche in einer kleinen Datenbank zwischengespeichert werden, von dem zentralen Raspberry Pi ab. So ergeben sich zwei Teilsysteme:

Das erste besteht aus dem Raspberry Pi und den Pollin Net-IO Boards, welche über die von Pollin bereit gestellte Schnittstelle kommunizieren. Die Verwendung der bereits vorhandenen Schnittstelle macht es unnötig am Mikrocontroller irgendwelche

Änderungen vorzunehmen oder eine neue Firmware flashen zu müssen, was die Usability enorm steigert, da das System auch von Laien betrieben werden kann. Sobald neue Werte vorliegen sollten die Pollin Net-IO Boards die Änderungen zum Raspberry Pi pushen, welcher die Werte in einer kleinen Datenbank zwischenspeichert. Zur Verwendung des bestehenden Protokolls muss dieses mit Wireshark analysiert und reverse engineered werden. Hierfür kann die Netzwerkkommunikation des Pollin Net-IO Boards mit der mitgelieferten PC-Software beobachtet werden.

Das zweite System besteht aus dem Raspberry Pi und den Clients. Die Clients fragen über HTTP beim Server die Webseiten-Dateien und Messwerte ab. Die Messwerte sollten nicht wie bei der aktuellen Lösung gepollt sondern nur bei Bedarf mit Hilfe der im Kapitel „Technischer Hintergrund“ erläuterten HTML5 Server-Sent Events Technik zum Client gepusht werden. Dies reduziert den unnötigen Netzwerkverkehr. Ein Client kann immer nur ein Pollin Net-IO Board darstellen, deshalb muss dem Nutzer auf der Webseite die Möglichkeit gegeben werden, das darzustellende Board auszuwählen. Außerdem muss auf der Webseite die zu verwendenden Boards (also welche der Raspberry Pi anspricht und den Clients anbietet) konfiguriert werden können. Ansonsten ist die Webseite ohne große Änderungen übernehmbar.

## 12.2 Änderungen an der Webseite/Server-Schnittstelle

Die Kommunikation zwischen Server und Webseite muss für die neuen Anforderungen entsprechend erweitert werden.

In einem ersten Schritt sollten alle REST-URLs um einen Parameter erweitert werden, der das Pollin Net-IO Board identifiziert, von dem die Informationen angefordert werden. Dies ist nötig, da das System über mehrere Boards verfügen könnte. Der Parameter kann als HTTP-GET Parameter übergeben werden. Als ID eignet sich z.B. die IP-Adresse des betreffenden Boards oder eine künstliche ID in Form einer fortlaufend höheren Zahl. Die aufzurufende URL wäre folglich z.B. `/rest/values?id=192.168.2.6`.

Danach sollte das aktuell über die POST-Parameter stattfindende Setzen von Pins ebenfalls über die REST-Schnittstelle gelöst werden. Hierfür müssen zwei neue URLs eingeführt werden, eine zum Setzen der Pinwerte und eine zum Setzen des DDRs. Natürlich müssen auch die neuen URLs über den Parameter zum identifizieren des betroffenen Boards verfügen.

Zum Schluss muss noch eine URL bereitgestellt werden um eine Liste aller verfügbaren Boards abfragen zu können. Zusätzlich muss noch eine URL zum Hinzufügen eines

neuen Boards und eine zum Entfernen eines vorhandenen Boards angelegt werden.

Sobald an der Webseite die Schnittstelle manipuliert wird, ist die Kommunikation mit dem von uns entwickelten Server nicht mehr möglich. Das System kann nicht mit HTTP-GET Parametern umgehen. Aus diesem Grund sollte zu Beginn der Entwicklung ein Testserver aufgesetzt werden. Dieser kann aus einem lokalen XAMPP-Server bestehen, welcher statt dynamisch Dateien für die REST-Schnittstelle zu erzeugen über statische Dateien verfügt, welche mit Testwerten gefüllt sind. So würde z.B. unter `/rest/pininfo` eine reale Datei liegen, in der anstatt der Platzhalter feste Werte eingetragen sind. Dies ermöglicht die Entwicklung der Webseite bzw. dem Ansprechen des Servers ohne einen funktionsfähigen Raspberry Pi.

## 12.3 Änderungen an der Webseite

### 12.3.1 Einstellungen

Die meisten Änderungen der Webseite finden in den JavaScript Dateien statt. Alle Einstellungen werden mit Hilfe von `db.js` gespeichert. Aktuell werden die Einstellungen lokal gespeichert. Dies kann entweder beibehalten werden oder die Einstellungen können zentral auf dem Raspberry gespeichert werden, was es ermöglichen würde Favoriten etc. zwischen mehreren Clients zu synchronisieren. Hierfür müsste nur der Speicherort geändert werden, an dem `db.js` die Daten ablegt. Anstatt diese lokal zu speichern müssten sie zum Server geschickt werden, welcher sie wiederum an alle anderen Clients weiterleitet.

### 12.3.2 Implementierung der neuen Schnittstelle

Die neue Schnittstelle zwischen Server und Webseite muss natürlich implementiert werden. Alle hierfür nötigen Änderungen finden in der `rest.js` statt. Neben der Implementierung der Server-Sent Events um neue Messwerte zu empfangen müssen auch neue Getter und Setter angelegt werden, um z.B. alle vorhandenen Boards abfragen zu können.

Aktuell wird die Funktion `refreshValues()` dazu verwendet, die Daten zyklisch nachzuladen. Gestartet wird dieser Vorgang von `startNewRefreshTask()`. Diese Funktionen können in Folge der Umstellung auf Server-Sent Events komplett gelöscht werden. Wichtig ist hierbei, dass die Funktion `setOnValuesChanged()` und das Attribut `onValuesChanged` beibehalten wird. Indem `onValuesChanged()` aufgerufen wird, wird `ui.js` darüber informiert, dass sich die Messwerte geändert haben, was zur Aktualisierung der Oberfläche führt.

### 12.3.3 Skriptfunktionen

Aktuell gibt es zwei verschiedene Typen von Skriptfunktionen. Für jeden Pin lässt sich eine Skriptfunktion zum Erstellen des dargestellten Messwertes hinterlegen. Diese Skriptfunktionen müssen nicht geändert werden.

Zusätzlich gibt es eine Skriptfunktion die in den Einstellungen festgelegt werden kann. Sie muss für jedes Board separat verfügbar und anpassbar sein. Aktuell wird sie auf dem Clientsystem als JavaScript ausgeführt, das hat zur Folge dass die Skriptfunktion nur so lange funktioniert, wie das Clientsystem aktiv ist, also die Webseite also angezeigt wird. Diese Skriptfunktion sollte auf dem Raspberry Pi gespeichert und ausgeführt werden.

### 12.3.4 Auswahl des darzustellenden Boards

Da die Webseite immer nur ein Pollin Net-IO Board darstellen kann, muss dem Nutzer eine Möglichkeit gegeben werden, eines aus allen verfügbaren auszuwählen. Hierfür bietet es sich an im div Element mit der ID header ein select anzubieten (z.B. auf der linken Seite, siehe Abbildung 30), mit dem das darzustellende Board ausgewählt werden kann. Für die Positionierung des Elementes kann das div Element mit der ID loading\_loop als Vorlage genutzt werden.

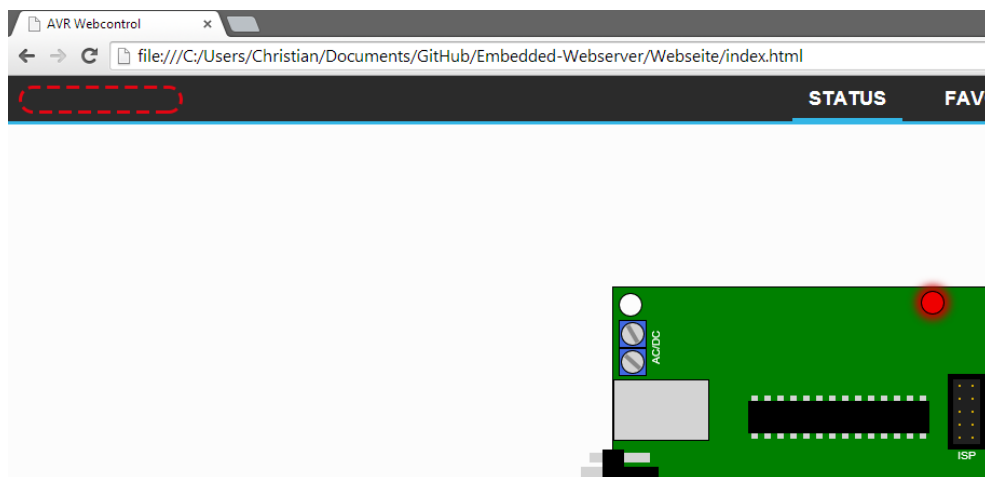


Abbildung 30: Vorgeschlagene Position für ein select-Element um die darzustellende Platine zu wählen

Beim Einfügen des select Elements in der Header ist zu beachten, dass die Höhe des Headers nicht verändert werden sollte. Sollte dies dennoch nötig sein, müssen die Kommentare im CSS-Code beachtet werden, um alle anderen nötigen Änderungen durchzuführen.

### 12.3.5 Verwaltung der Boards im System

Im Einstellungs-Tab der Webseite muss eine Möglichkeit ergänzt werden, um die neue Pollin Net-IO Boards zum System hinzuzufügen und vorhandene zu editieren oder zu löschen. Für jedes Board sollte eine Skriptfunktion und ein Name hinterlegbar sein. Der Name ermöglicht es dem Benutzer die Boards leichter voneinander zu unterscheiden.

## 12.4 Der neue Server

Beim Server, welcher auf dem Raspberry Pi betrieben werden sollte, gibt es zwei grundsätzliche Lösungsmöglichkeiten. Zum einen lässt sich der Server als C/C++/Java-Programm realisieren, das einen Webserver zur Verfügung stellt oder man realisiert den Server als PHP-Programm auf einem XAMPP-Server.

Bei beiden Alternativen muss der Server mit den einzelnen Pollin Net-IO Boards kommunizieren, welche Daten über die REST-Schnittstelle bzw. die Server-Sent Events bereitstellen sowie die Webseite hosten.

## 12.5 Herangehensweise an das Projekt

### 12.5.1 Einpflegen des Raspberry Pi

Im ersten Schritt sollte der Raspberry Pi in das bestehende System eingepflegt werden. Hierfür sollte die Kommunikation zwischen Server und Webseite vorerst so belassen werden wie sie aktuell ist, inklusive Polling. Der Server muss die Daten von dem Pollin Net-IO Board empfangen (per Polling oder Pushing) und in eine kleine Datenbank (oder ein Array etc.) hinterlegen. Bei jeder Abfrage der Webseite werden die aktuellsten Daten weitergeleitet.

Zu beachten ist, dass später die Skripte auf dem Server ausgeführt werden sollen. Aus diesem Grund würde sich PHP als Programmiersprache anbieten, da PHP-Code, der als Text vorliegt, direkt mit `eval()` ausgeführt werden kann.

### 12.5.2 Betreiben mehrerer Pollin Net-IO Boards

Anschließend sollten mehrerer Pollin Net-IO Boards betrieben werden können. Hierfür muss die Server/Webseiten-Schnittstelle entsprechend dem Kapitel 12.2 überarbeitet werden. Die direkte Implementierung der HTML5 Server-Sent Events ist empfehlenswert. Außerdem muss die Oberfläche der Webseite um eine Auswahlmöglichkeit für das zu verwendende Board sowie eine Konfigurationsmöglichkeit für die einzelnen Boards im System erweitert werden (siehe Kapitel 12.3.4 und 12.3.5).

### 12.5.3 Verlagern der Skriptfunktionen auf den Server

Im letzten Schritt sollten die Skriptfunktionen, die nach jedem Neuladen der Werte ausgeführt wird (aktuell im Settings-Tab einstellbar), auf dem Server gespeichert und ausgeführt werden. Die Skriptfunktionen müssen natürlich von der Webseite aus für jedes Board einzeln anpassbar sein. Wenn die Serverstruktur auf Basis des XAMPP-Servers gewählt wurde, bietet sich PHP als Skriptsprache an, da der Code nach dem Empfang neuer Werte direkt mit `eval()` ausgeführt werden kann. Wichtig ist hierbei, dass über einfache Getter und Setter Funktionen der Zugriff auf alle aktuellen Messwerte möglich ist und die Skriptfunktion auch Pins von Boards sowie deren DDR manipulieren kann.

## 13 Fazit

Die Welt der Mikrocontroller steckt voller Möglichkeiten, jedoch enthält sie, gerade für Laien, noch einige Startschwierigkeiten.

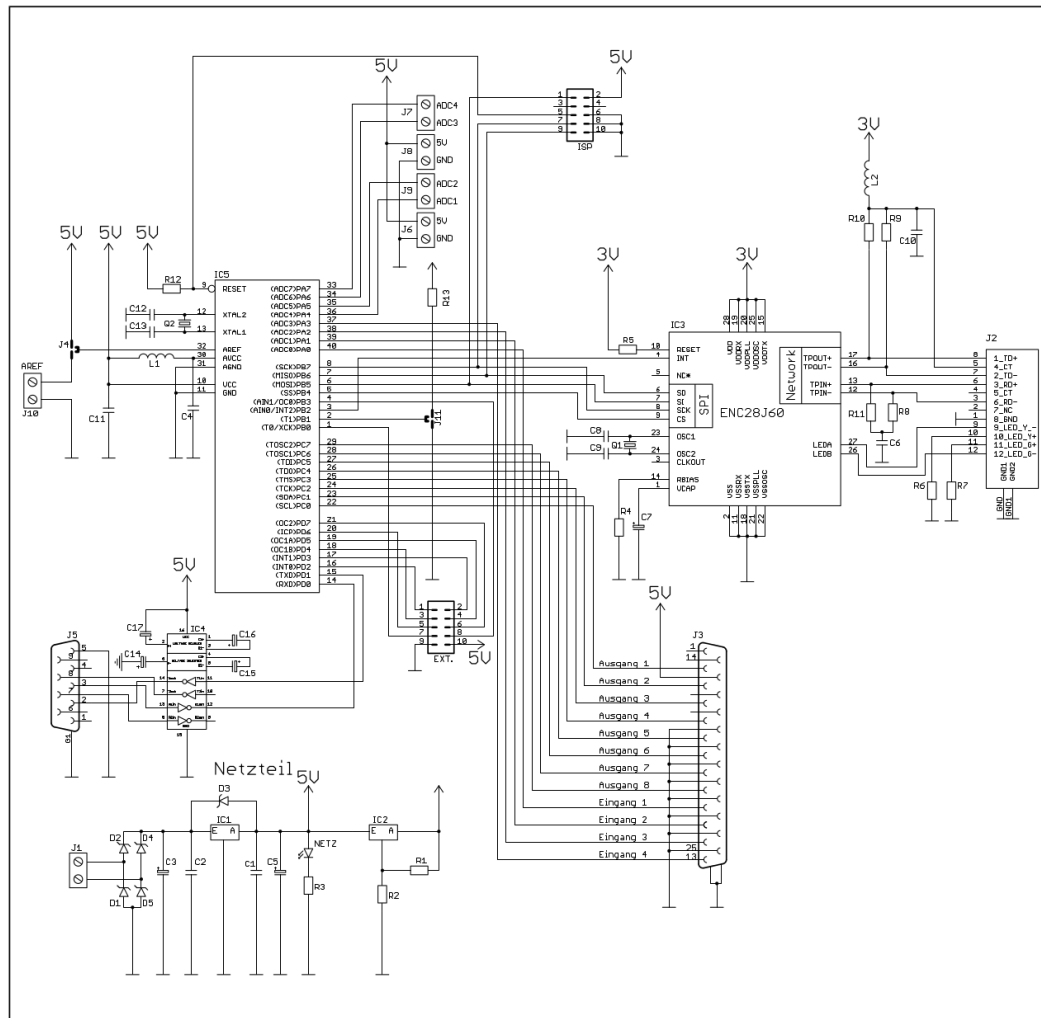
Im Gegensatz zum Arbeiten mit Computern, bei welchen der Speicherplatz für einfache Programme schier unbegrenzt scheint, kommt es bei den Mikrocontrollern mit ihrem geringen Speicher auf jedes Byte an. So bestand in unserem Projekt die Schwierigkeit nicht nur darin, den Server mit weiteren Funktionen auszustatten, sondern auch bei der Programmierung möglichst auf Effizienz zu achten und den bestehenden Webserver von nicht benötigten Funktionen zu befreien, sowie beim Hinzufügen neuer Funktionen streng auf den Speicherverbrauch zu achten.

Als eine weitere Herausforderung bei Mikrocontrollern kam noch die ganze elektronische Seite hinzu. Als Informatiker haben wir durch das Studium kaum Berührung mit diesem Thema gehabt und mussten uns vielerorts in die Thematik einarbeiten. Doch hat sich das Projekt im Endeffekt als handhabbarer erwiesen als anfangs gedacht. Die Hauptaufgaben bestanden hier im Beschaffen von Bauteilen, dem Erstellen von Platinen zum Testen der Funktionen oder dem Programmieren und Debuggen des Mikrocontrollers.





## 14 Anhang



Schaltplan für das AVR-NET-IO-Board

Abbildung 31: Schaltplan AVR-Net-IO Board

Themenbereich	Funktionalität	Status	Zuständig	Priorität	Beschreibung
Webseite	Nicht Funktional	Wartend	Nicht festgelegt	• • • • •	Die Webseite muss auf Chrome ab Version 33.0, Firefox ab Version 28.0 und Safari ab Version 7.0 lauffähig sein
	Funktional	Wartend	Nicht festgelegt	• • • • •	Beim Aufrufen der Webseite muss der Benutzer die IP-Adresse der Platine eingeben oder kann eine zuvor eingegeben IP-Adresse aus einer Verlaufsliste auswählen (auf Client gesichert)
	Funktional	Wartend	Nicht festgelegt	• • • • •	Die Webseite beinhaltet eine grafische Übersicht über die Platine
	Funktional	Wartend	Nicht festgelegt	• • • • •	In der grafischen Übersicht werden die aktuellen Werte der Analogeingänge sowie die Pinbelegung angezeigt
	Nicht Funktional	Wartend	Nicht festgelegt	• • • • •	Die Pins werden nicht als Ports organisiert sondern durchnummeriert und in der Übersicht gemäß ihrer Position auf der Platine dargestellt.
	Funktional	Wartend	Nicht festgelegt	• • • • •	Der Nutzer kann durch klicken auf Pins in der Übersicht diese setzen und löschen
	Funktional	Wartend	Nicht festgelegt	• • • • •	Der Nutzer kann die Pins/Analogeingänge der Übersicht in eine Favoritenliste aufnehmen
	Funktional	Wartend	Nicht festgelegt	• • • • •	Der Nutzer kann die Pins/Analogeingänge in der Favoritenliste wieder entfernen
	Funktional	Wartend	Nicht festgelegt	• • • • •	Der Nutzer kann für jeden Eintrag (Pin/Analogeingang) eine Beschreibung hinterlegen
	Funktional	Wartend	Nicht festgelegt	• • • • •	In der Favoritenliste werden die aktuellen Werte der dargestellten Pins/Analogeingänge angezeigt
	Funktional	Wartend	Nicht festgelegt	• • • • •	Der Benutzer kann ein kurzes Script für jeden Favoriteneintrag hinterlegen, der den vom Board gelieferten Messwert umwandelt. Dieser modifizierte Wert wird dann in der Favoritenliste als Messwert angezeigt
	Nicht Funktional	Wartend	Nicht festgelegt	• • • • •	Die dargestellten Werte in Favoritenliste/Übersicht sollen mindestens in einem 3s Takt aktualisiert werden
	Funktional	Wartend	Nicht festgelegt	• • • • •	Der Nutzer kann die Taktfrequenz der Aktualisierungen frei einstellen. Der Bereich liegt zwischen der schnellst möglichen Frequenz (zB. 3 Sekunden, statisch festgelegt) und 60 Sekunden.
	Funktional	Wartend	Nicht festgelegt	• • • • •	Der Nutzer kann die Aktualisierung der Werte pausieren
	Nicht Funktional	Wartend	Nicht festgelegt	• • • • •	Die Webseite soll keine veralteten Elemente wie <table>, <frame> oder <iframe> besitzen
	Nicht Funktional	Wartend	Nicht festgelegt	• • • • •	Zum aufrufen der Webseite muss keine Internetverbindung vorhanden sein, es werden keine Daten von externen Servern geladen
	Funktional	Wartend	Nicht festgelegt	• • • • •	Die aktuellen Favoriten und deren Beschreibungen sowie Scriptfunktionen werden auf dem Client gesichert und beim erneuten Laden der Webseite wiederhergestellt. Die Belegung der Favoritenliste etc. ist für jede Platine (Unterscheidung anhand IP-Adresse) separat gesichert
	Funktional	Wartend	Nicht festgelegt	• • • • •	Die Webseite enthält folgende Statusinformationen der Platine: Version des Servers, IP-Adresse, Mac-Adresse
	Funktional	Wartend	Nicht festgelegt	• • • • •	Der Nutzer kann auf der Webseite die IP-Adresse der Platine ändern
Server	Nicht Funktional	Wartend	Nicht festgelegt	• • • • •	Die Webseite darf maximal 55kb groß sein (bei Verwendung des ATmega644) bzw. 1119kb (bei Verwendung des ATmega1284)
	Nicht Funktional	Wartend	Nicht festgelegt	• • • • •	Es sollen nur websichere Schriften verwendet werden, um zu garantieren das die Darstellung der Webseite auf allen Plattformen identisch ist
	Nicht Funktional	Wartend	Nicht festgelegt	• • • • •	Die Serversoftware soll auf dem Pollin Net-IO Board mit einem ATmega644 oder ATmega1284 lauffähig sein
	Funktional	Wartend	Nicht festgelegt	• • • • •	Die Serversoftware bietet einen HTTP-Server mit einer REST-Schnittstelle für die Kommunikation mit Clients
	Nicht Funktional	Wartend	Nicht festgelegt	• • • • •	Die Server-Software darf maximal 9kb Speicher im Program Memory verbrauchen (ohne Webseite)
	Funktional	Wartend	Nicht festgelegt	• • • • •	Der Server bietet folgende Informationen über die REST-Schnittstelle: Pinbelegung, Werte der Analogeingänge, aktuelle IP-Adresse, Mac-Adresse, Version des Servers
	Funktional	Wartend	Nicht festgelegt	• • • • •	Über die REST-Schnittstelle können folgende Dinge manipuliert werden: IP-Adresse, Pinbelegung, hat der Client Schreibrechte
	Nicht Funktional	Wartend	Nicht festgelegt	• • • • •	Die Serversoftware soll auf dem Webserver Projekt von Ulrich Radio aufbauen
	Nicht Funktional	Wartend	Nicht festgelegt	• • • • •	Der HTTP-Server soll mit HTTP 1.0 kommunizieren
	Funktional	Wartend	Nicht festgelegt	• • • • •	Der Server soll drei Clients parallel mit Daten versorgen können.
	Nicht Funktional	Wartend	Nicht festgelegt	• • • • •	Nur ein Client hat gleichzeitig schreibrechte

Abbildung 32: Anforderungsdefinition zu Beginn des Projektes

## Literaturverzeichnis

- [Atm] ATMEL (Hrsg.): *JTAGICE mkII Quick Start Guide*. Atmel
- [ele14] *E2000-NET-IO*. <http://www.elektronik2000.de/content.php?id=49>.  
Version: 16:23:40, 09.06.2014
- [Eth14] *Ethersex*. [ethersex.de](http://ethersex.de). Version: 12:49:27, 09.06.2014
- [mik14] *AVR In System Programmer*. [http://www.mikrocontroller.net/articles/AVR\\_In\\_System\\_Programmer](http://www.mikrocontroller.net/articles/AVR_In_System_Programmer). Version: 14:05:46, 25.06.2014