

Projektarbeit  
im Studiengang  
AIB/CNB

# Webserver für ein embedded Board mit AVR-Prozessor

Dokumentation

Betreuer : Dr. Jiri Spale

Vorgelegt am : 30.07.2014

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



## **Abstract**

[Englisches Abstract (100-120 Worte)]

[Deutsches Abstract (100-120 Worte)]



## Inhaltsverzeichnis

Abstract . . . . .	i
Inhaltsverzeichnis . . . . .	v
Abbildungsverzeichnis . . . . .	vii
Tabellenverzeichnis . . . . .	ix
Abkürzungsverzeichnis . . . . .	xi
1 Aufgabenstellung . . . . .	1
2 Team . . . . .	3
2.1 Teammitglieder . . . . .	3
3 Projektplanung . . . . .	5
3.1 Zeitlicher Ablauf . . . . .	5
4 Hardware . . . . .	7
4.1 AVR Net-IO-Board . . . . .	7
4.1.1 Technische Daten . . . . .	7
4.2 Mikrocontroller . . . . .	8
4.2.1 ATmega32 . . . . .	8
4.2.2 ATmega644P . . . . .	8
4.2.3 ATmega1284P . . . . .	8
4.2.4 ENC28J60 . . . . .	8
4.3 Fuse Bits . . . . .	8
5 Programmieren und Debuggen . . . . .	9

---

5.1	ISP . . . . .	9
5.2	SPI . . . . .	9
5.3	JTAG . . . . .	9
6	Recherche . . . . .	11
6.1	Andere Lösungsmöglichkeiten . . . . .	11
6.1.1	Ethersex . . . . .	11
6.1.2	Elektronik 2000 . . . . .	12
7	Ausgewählte Lösung . . . . .	15
7.1	Der Webserver . . . . .	15
7.1.1	Änderungen . . . . .	15
7.1.2	Einbindung der Website . . . . .	16
7.2	Die Website . . . . .	16
8	Technischer Hintergrund . . . . .	17
8.1	Vorgeschlagene Lösung . . . . .	17
8.1.1	Kommunikation im Projekt Radig . . . . .	17
8.1.2	Erster Ansatz . . . . .	17
8.1.3	Polling oder Pushing . . . . .	18
8.1.4	Aufbau der Server-Kommunikation . . . . .	20
8.1.5	Implementierung der REST-Schnittstelle auf dem Server . . . . .	21
8.1.6	Erweiterung der POST-Parameter . . . . .	22
8.1.7	Implementierung der REST-Schnittstelle auf dem Client . . . . .	23
8.2	Werkzeuge . . . . .	24
8.2.1	Das Atmel Studio . . . . .	24
8.2.2	AVRDUDE . . . . .	24
8.2.3	HTML Header Compiler . . . . .	25
8.2.4	AVRISPMkII . . . . .	27

---

8.2.5	AVRJTAGICEmkII . . . . .	27
9	Benutzerhandbuch . . . . .	29
9.1	Einen Mikrocontroller austauschen . . . . .	29
9.2	ISP-Programmer anschließen . . . . .	31
9.3	Einrichten eines neuen Mikrocontrollers . . . . .	32
9.4	HTML Header Compiler . . . . .	35
9.4.1	Konfiguration des Webserver . . . . .	36
9.5	Debuggen über JTAG . . . . .	37
9.6	Hexfiles Überspielen . . . . .	37
9.6.1	Atmel Studio . . . . .	37
9.7	Die Website . . . . .	37
10	Rückblick . . . . .	39
10.1	Soll/Ist-Vergleich . . . . .	39
10.2	Verworfenen Varianten . . . . .	39
10.3	Eigenbewertung . . . . .	39
11	Ausblick . . . . .	41
12	Fazit . . . . .	43
	Literaturverzeichnis . . . . .	45





## Abbildungsverzeichnis

Abbildung 1: AVR-NET-IO - Pollin GmbH . . . . .	7
Abbildung 2: Ethersex menuconfig . . . . .	11
Abbildung 3: Ethersex make project . . . . .	12
Abbildung 4: JavaScript um /rest/valor abzufragen und in ein Objekt zu parsen	23
Abbildung 5: DeviceProgramming . . . . .	24
Abbildung 6: index.htm . . . . .	25
Abbildung 7: webpage.h . . . . .	26
Abbildung 8: Schraubendreher am Controller . . . . .	29
Abbildung 9: Der gelöste Mikrocontroller . . . . .	30
Abbildung 10: Der Sockel auf dem AVR-Net-IO . . . . .	30
Abbildung 11: Markierung zum Einbau . . . . .	31
Abbildung 12: Schematische Darstellung des ISP Anschlusses . . . . .	31
Abbildung 13: Der Adapter . . . . .	32
Abbildung 14: DeviceProgramming . . . . .	33
Abbildung 15: AVRDUDE Ausgabe . . . . .	34
Abbildung 16: BuildWebpage.sh für Linux . . . . .	36
Abbildung 17: BuildWebpage.bat für Windows . . . . .	36



## Tabellenverzeichnis

Tabelle 1: Die Pinbelegung für den 6 und 10 poligen Anschluss [mik14] . . . . .	32
Tabelle 2: Auslesen und setzen von Fuse-Bits des ATmega644P mit AVRDUDE	33
Tabelle 3: Auslesen und setzen von Fuse-Bits mit dem AVRDUDE . . . . .	34
Tabelle 4: Parameter des HTML Header Compiler . . . . .	35



## **Abkürzungsverzeichnis**

**ISP** In System Programming

**SPI** Serial Peripheral Interface

**JTAG** Joint Test Action Group

**HHC** HTML Header Compiler



## **1 Aufgabenstellung**





## **2 Team**

### **2.1 Teammitglieder**

Jan Henrik Preuß

Christian Würthner

Ann-Sophie Dietrich

Marcel Schlipf



## 3 Projektplanung

### 3.1 Zeitlicher Ablauf

Zu Beginn des Projektes mussten wir feststellen, dass einige Teammitglieder noch sehr unerfahren in der Welt der Microcontroller waren. Somit war es zunächst notwendig, sich mit den Grundlagen zu beschäftigen und sich in die Problematik einzulesen. Nach der ersten Gruppenbesprechung wurden Posten verteilt und ein grober Zeitplan erstellt. Schnell stellte sich heraus, dass wir ohne eine erste Besprechung und ohne die Platine nicht wissen, ob unsere Ideen und Vorschläge überhaupt umsetzbar sind, geschweige denn den Anforderungen entsprechen.

Nach der ersten Besprechung, in welcher wir die Platine überreicht bekamen, begannen die ersten Einarbeitungen mit dem Controller. Standardmäßig war eine Software beigelegt, mit welcher sich bereits die Ein- und Ausgänge steuern ließen.

Eine weitere Problematik lag darin, dass wir zwar einen **In-System-Programmer** (ISP) zum Anschluss der Platine an den PC hatten, doch war bei diesem Entwicklungswerkzeug die falsche Pinbelegung vorhanden. Nach einiger Recherche fanden wir jedoch einige Anleitungen im Internet, welche hierbei für Klärung sorgten.

Die Standard-Ausführung des Controllers reichte jedoch nicht für ausreichendes testen, weshalb wir noch weiteres Zubehör anschaffen wollen.

Beim AVR-NET-IO sind die Digitalen ein und Ausgänge nur über den 25-Pin seriellen Eingang zu erreichen. Deswegen wurde ein Bausatz angefordert, den wir auch umgehend von Herrn Schellhammer erhalten haben. Mit diesem Bausatz können die digitalen Ausgänge direkt mit den Klemmen belegt werden.

Nachdem für den ISP Programmierer der Richtige Adapter gelötet wurde, konnten erste Tests mit dem Board gefahren werden. Zuerst wurde Testweise die Ethersex Firmware auf den Microcontroller aufgespielt und in betrieb genommen. Für das Radig Projekt gab es allerdings noch ein paar Probleme, bevor die Software in Betrieb genommen werden konnte.



## 4 Hardware

### 4.1 AVR Net-IO-Board



Abbildung 1: AVR-NET-IO - Pollin GmbH

#### 4.1.1 Technische Daten

- Betriebsspannung 9V
- Stromaufnahme ca. 190 mA
- 8 Digitale Ausgänge, 4 Digitale Eingänge
- 4 Analoge Eingänge
- ATmega32 Mikrocontroller
- integrierte ISP-Schnittstelle

## **4.2 Mikrocontroller**

### 4.2.1 ATmega32

### 4.2.2 ATmega644P

### 4.2.3 ATmega1284P

### 4.2.4 ENC28J60

## **4.3 Fuse Bits**

## 5 Programmieren und Debuggen

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

### 5.1 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 SPI als auch die JTAG Schnittstelle.

### 5.2 SPI

### 5.3 JTAG





## 6 Recherche

### 6.1 Andere Lösungsmöglichkeiten

Neben dem vorgegebenen Projekt von Ulrich Radig gibt es noch einige andere Möglichkeiten einen Webserver oder Scripte auf das AVR-Net-IO zu bekommen.

#### 6.1.1 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]*

Sie 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 Projekt Seite ist es auch sehr einfach möglich selbst für Laien einen lauffähiges Projekt hinzugekommen. Fast die gesamte Konfiguration kann über einen Menü vorgenommen werden. So muss nicht wie beim Radig Projekt zum ändern der Mac Adresse in eine Headerdatei geschrieben werden.

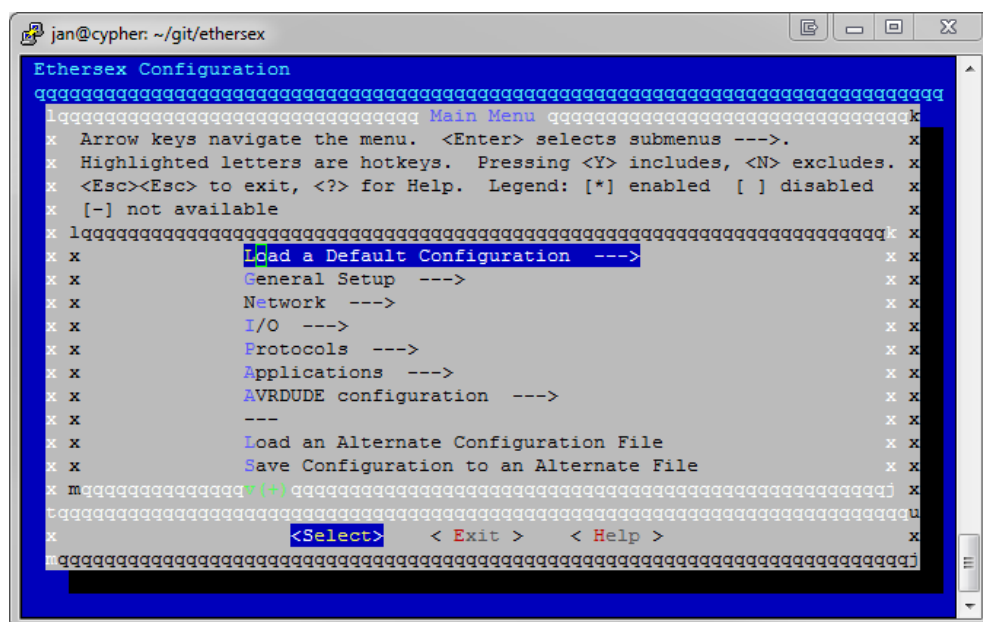
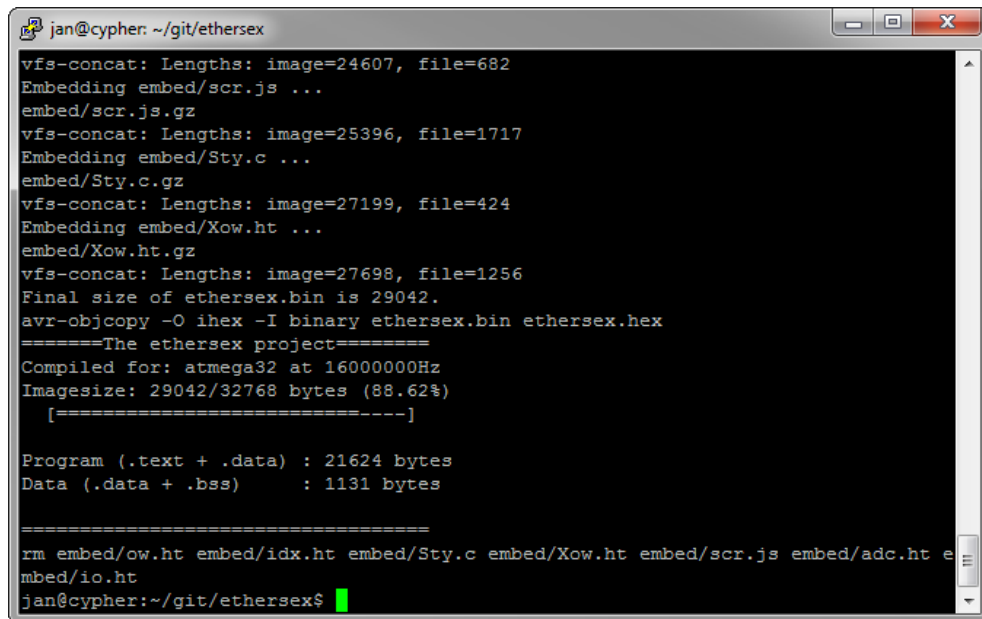


Abbildung 2: Ethersex menuconfig

Im Bild (Abb. 2) sieht man den Startbildschirm des Menüs, das über den Befehl `make menuconfig` erreicht werden kann. Hier können verschiedenste Einstellungen getroffen werden. Zum Beispiel, den verwendeten Mikrokontroller, 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 3 sieht man das am ende des Make Prozesses die aktuelle Größe des erstellten Binary Datei angezeigt ist.



```

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 3: 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))]

### 6.1.2 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 lau-*

*fendes 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 die auch mit dem Raspberry Pi zusammenarbeitet und die GPIO Pins des Pis nach außen steuerbar macht.



## 7 Ausgewählte Lösung

Neben den Beiden anderen Projekten haben wir uns für das Projekt von Ullrich Radig Entscheiden. Die Vorteile des Projektes gegenüber Ethersex oder Elektronik 2000 liegen darin, das die beiden Projekte zu speziell und umfangreich für unsere Anforderungen sind. Zum einen war es unsere Aufgabe eine möglichst umfangreiche Website zu erstellen, die Struktur von Ethersex ist für eine auf die Website fokussierte Programmierung schlicht zu umfangreich und schlechter anpassbar.

### 7.1 Der Webserver

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

#### 7.1.1 Änderungen

Zu der bereits vereinfachten Version von Günther Menke mussten wir für unser Projekt noch Funktionalität von der ursprünglichen Version entfernen. Das Problem lag darin, dass wir für die Dateien der Website möglichst viel freien Speicherplatz benötigen, der von den entsprechenden Funktionen belegt wurde. Schlussendlich wurde folgende Funktionalität aus der Version von Günther Menke entfernt:

- **(WOL) Wake on Lan** Funktionalität um andere Geräte im Netzwerk durch bestimmte Datenpakete aufzuwecken.
- **Sendmail** Senden von E-Mails.
- **Weather** Ermitteln von Wetterdaten.
- **(NTP) Network Time Protocol** Empfangen von Internetzeit
- **(DNS) Domain Name System** Beantwortung von Anfragen zur Namensauflösung.

- **(USART) Universal Synchronous and Asynchronous Serial Receiver and Transmitter** eine Schnittstelle im Mikrocontroller zum Daten Austausch mit PC über die COM-Schnittstelle.
- **(Telnet) Telecommunication Network** zeichenorientierten Datenaustausch über eine TCP-Verbindung.
- **(CMD) Command Control** Verwaltung der Telnet Konsolen Befehle.

#### 7.1.2 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 8.2.3. Eine Anleitung zur Ausführung des HHC gibt es im Benutzerhandbuch 9.4. Abschließend ist noch zu erwähnen, das die webpage.h nicht für manuelle Bearbeitung gedacht ist. Dies geschieht ausschließlich über die Quell-Dateien und anschließend umwandeln mit dem HHC.

### 7.2 Die Website

## 8 Technischer Hintergrund

### 8.1 Vorgeschlagene Lösung

#### 8.1.1 Kommunikation im Projekt Radig

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 SSendenButtons 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, das geänderte Werte erst beim nächsten neu laden 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.

#### 8.1.2 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 werden sondern kontinuierlich erneuert werden müssen. Die Schnittstelle sollte gut skalierbar sein. Würde später ein Port für eine andere Aufgabe zu 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 geschalteter Pins
- Manipulieren der Konfiguration eines Pins (als Eingang oder Ausgang setzen)
- Manipulieren von Servereinstellungen (z.B. IP-Adresse);

### 8.1.3 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 zur Verfügung wie die Übertragung der Daten initialisiert werden.

#### 8.1.3.1 Polling

Bei Polling werden vom Client kontinuierlich die Werte erneut angefordert, indem dieser die entsprechende virtuelle 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 zu der 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 werden kann.

#### 8.1.3.2 Pushing

Bei Pushing wird im Gegensatz zu Polling 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 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, dass die Webseite schneller auf Änderungen reagiert.

Die technische Umsetzung von Pushing ist mit diversen Problemen verbunden. Die typische Verbindungsaufbauichtung ist bei Webanwendungen und Webseiten immer vom Client zum Server. Anders als bei Polling müssen bei Pushing Daten vom Server zum Client gelangen. Hierfür muss eine Verbindung vom Server zum Client aufgebaut werden. Dies ist technisch aber nicht möglich, da der Browser bzw. JavaScript keine Möglichkeit hat, einen Port des Client-Systems 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.

Um HTML5 Server-Sent Events auf dem Pollin Net-IO Board zu implementieren würde es als einen tendenziell größeren CPU erfordern mit dem auch Multitasking möglich ist sowohl auch eine grundlegende Umgestaltung des Radig-Projektes um mehrere HTTP Verbindungen parallel zu ermöglichen.

#### 8.1.3.3 Entscheidung

Da die technische Umsetzung vom Pushing nur schwer möglich ist werden wir auf Polling setzen.

TODO: Hier noch Argumentation mit Übertragungszeit und Screenshot aus Chrome Network Log

#### 8.1.4 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 uns 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.

##### 8.1.4.1 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 REST.

XML-Basierte Protokolle wie RSS oder SOAP weisen typischerweise einen großen Protokoll-Overhead auf und sind deswegen nur bedingt für den Einsatz auf einem Microcontroller 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 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.

#### 8.1.4.2 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/valus` liefert nur die aktuellen Werte und DDR-Einträge.

Um die dargestellten Werte auf der Webseite zu aktualisieren muss folglich nur `/rest/valus` 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.

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

Für die Implementierung der REST-Schnittstelle setzen wir auf dem besethenden 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 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 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 DDRegister 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 oder D] und dem Pin Y = [0-7] (z.B. %DDRD1 ).

Damit das System für uns möglichst flexibel Arbeitet, haben wir uns dafür entschieden, diese Dynamischen Angaben in eine Separate Datei auszulagern und über die REST Schnittstelle abzufragen. Konkreter liegt in dem /Rest Verzeichnis eine values Datei die die Platzhalter enthält. Zusätzlich gibt es noch eine info und pininfo Datei, die statische Informationen zu den Pins und Ports Enthält.

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 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

#### 8.1.6 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.

TODO

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

Auf der Webseite müssen die vom Server bereitgestellten REST-URLs im Hintergrund aufgerufen werden können. Hierzu verwenden wir die bereits in JavaScript enthaltene 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 4: JavaScript um `/rest/valus` abzufragen und in ein Objekt zu parsen

Der hier beispielhaft gezeigte Quelltext ermöglicht das laden der URL `/rest/valus`. 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 bereit gestellt, 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, das 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 `setOnValueChanged(func)` gesetzt werden kann.

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

## 8.2 Werkzeuge

### 8.2.1 Das Atmel Studio

#### 8.2.1.1 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 Microcontroller bestimmt werden, es können die Fuse Bit Einstellung geändert werden oder einzelne Hex Dateien auf den Mikrocontroller aufgespielt werden.

Übersicht k Gerät Auswählen Contoller Auswählen Spannung und Signatur Fuse Bits Hexfile Flashen

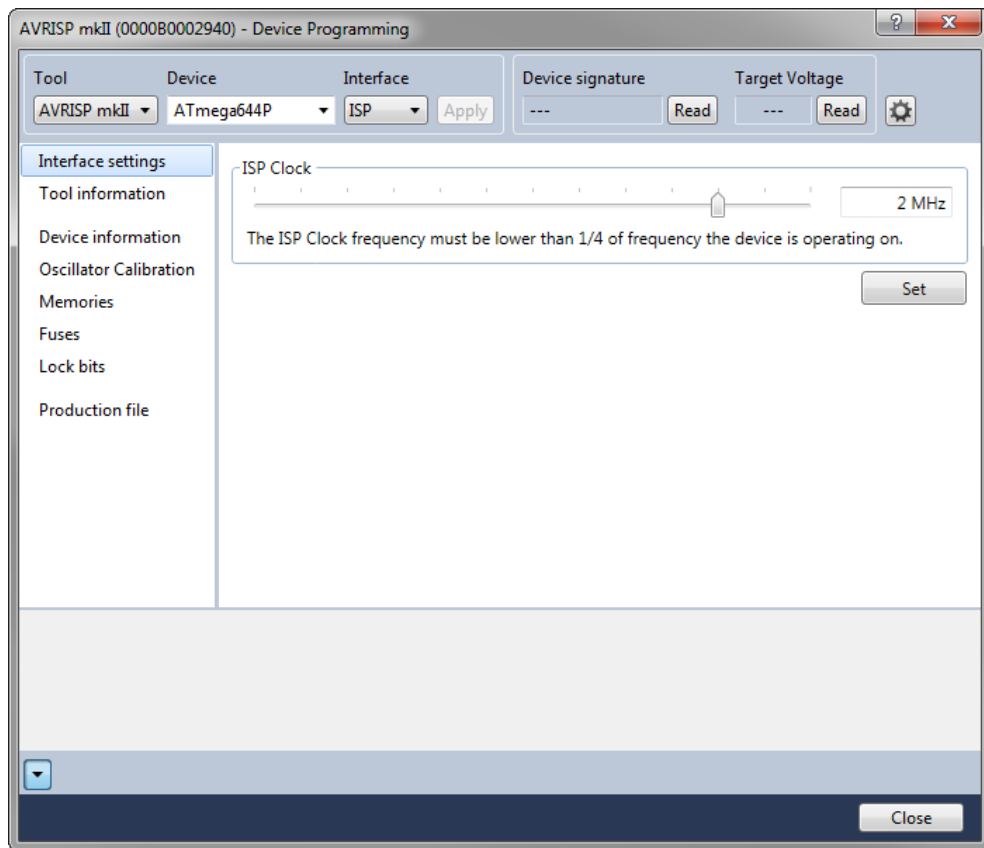


Abbildung 5: DeviceProgramming

#### 8.2.1.2 Projekt Einstellungen

Taktfrequenz Programmer Empfolene Tool Settings

### 8.2.2 AVRDUDE

Avrdude ist ein Alternatives Werkzeug zum Bearbeiten von Mikrocontrollern. Es ist im Gegensatz zum Atmel Studio lediglich ein Konsolen Werkzeug ohne grafische Ober-

fläche. Mit Avrdude können unter anderem die Fusebits von einem Mikrocontroller gelesen und gesetzt werden. Weiter ist es möglich eine Sicherung von dem Aktuellen stand des Mikrocontrollers herzustellen oder ein 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 Microcontroller Standartmäßig 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 9 Benutzerhandbuch.

### 8.2.3 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 Beispiel Umwandlung:

```
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 6: index.htm

Das eingegebene Verzeichnis, welches die index.html Datei enthält wird eingelesen und in die folgende Headerdatei umgewandelt. Abbildung 6 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 7. 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 die 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 7: webpage.h

Anzumerken ist das 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 Compiliert. 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.

#### 8.2.4 AVRISPmkII

#### 8.2.5 AVRJTAGICEmkII



## 9 Benutzerhandbuch

### 9.1 Einen Mikrocontroller austauschen

Für unser Projekt sollen alle notwendigen Programmbestandteile sowie die gesamte Website auf dem Microcotnroller 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 ATmeag32.

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 8 lösen.

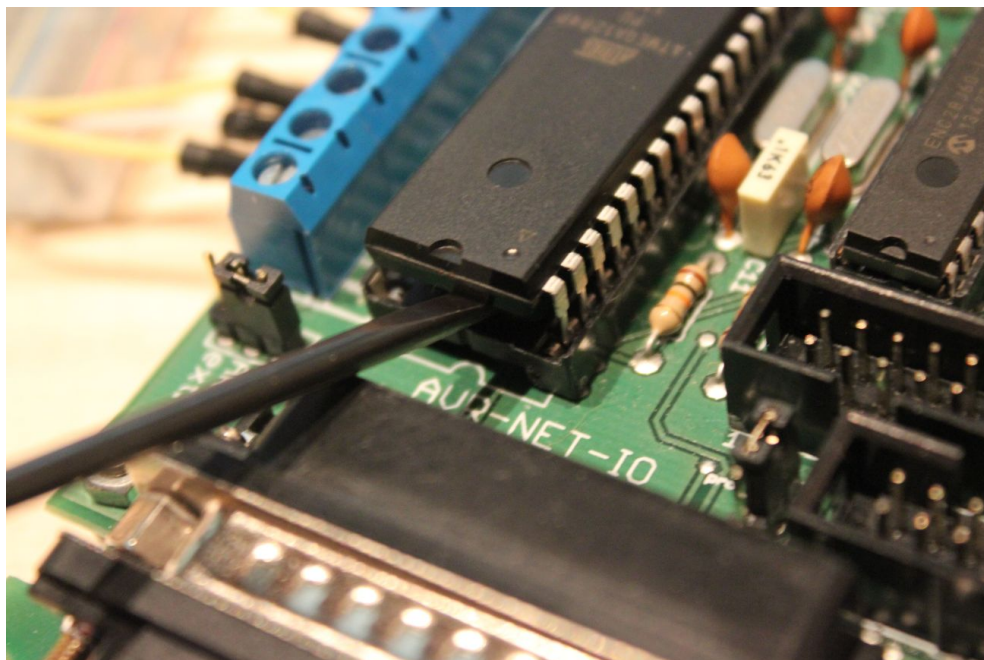


Abbildung 8: Schraubendreher am Controller

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

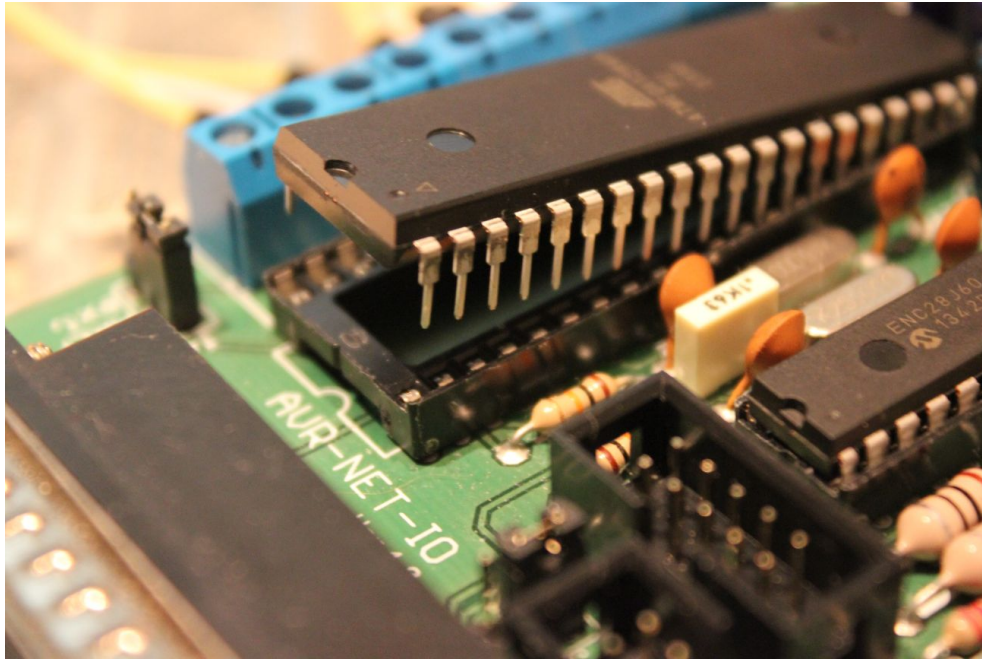


Abbildung 9: Der gelöste Mikrocontroller

Nachdem der Mikrocontroller entfernt wurde hat man einen guten Blick auf den Sockel (Abb. 10).

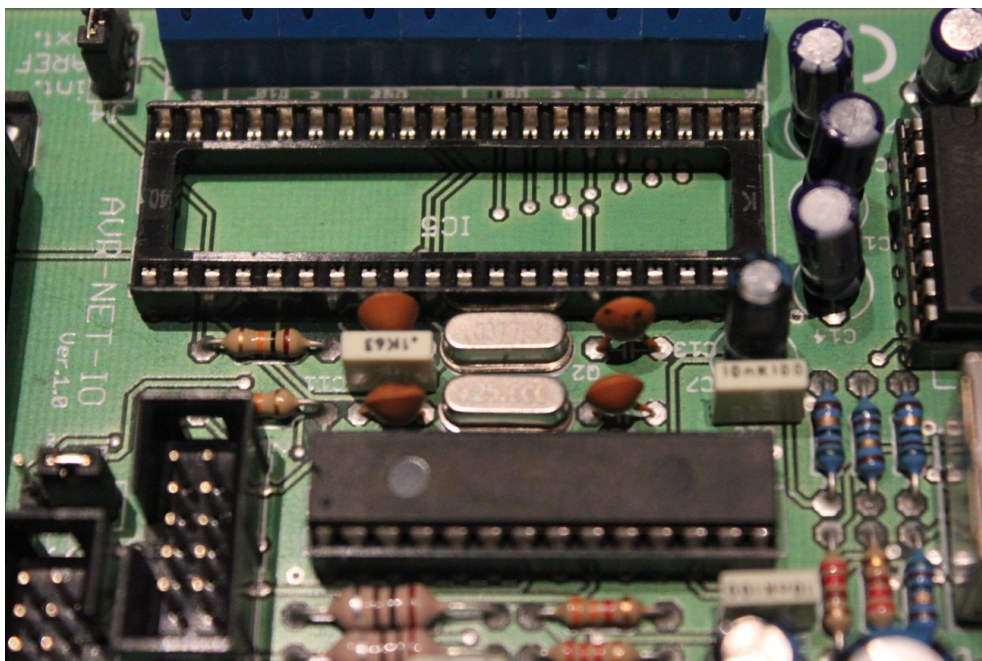


Abbildung 10: Der Sockel auf dem AVR-Net-IO

Beim Einbau ist unbedingt darauf zu achten den neuen Mikrocontroller entsprechend der D-Förmige Einkerbung in den Sockel zu setzen. (Siehe Abbildung 11)

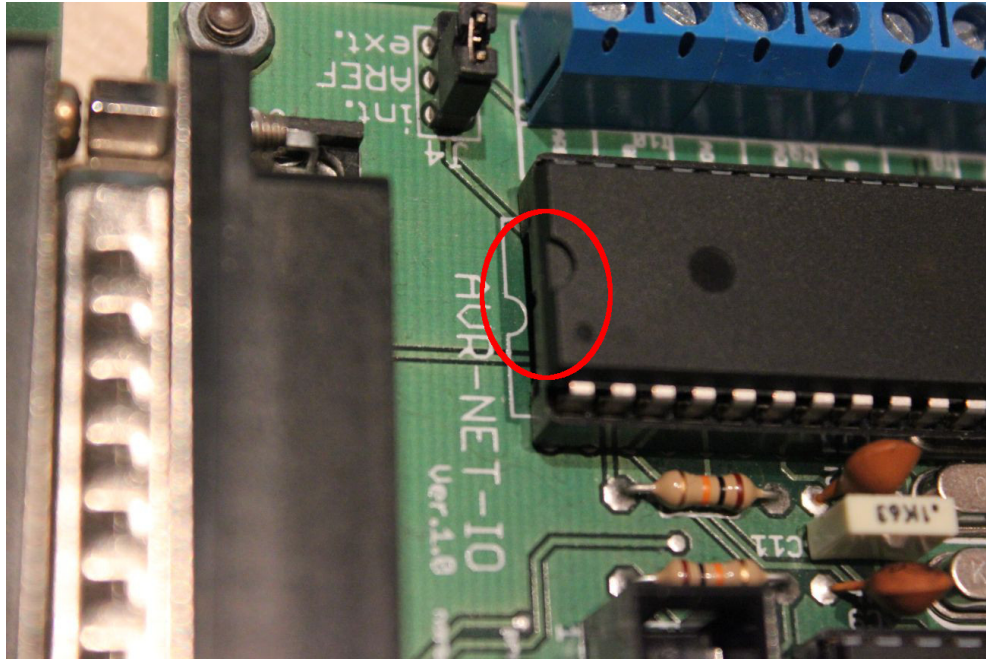


Abbildung 11: Markierung zum Einbau

## 9.2 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 der das ISP Signal von den 6 Polen des Programmers auf das AVR-Net-IO bringt.

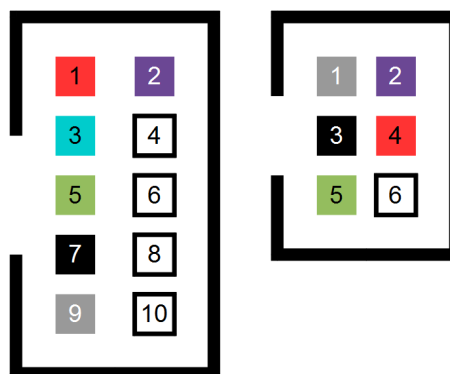


Abbildung 12: Schematische Darstellung des ISP Anschlusses (Pin 1 &amp; 2 ist auf der Platine markiert)

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



Abbildung 13: Der Adapter

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

### 9.3 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, das Standartmäßig 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 wird man im Atmel Studio mit folgender Fehlermeldung begrüßt.

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, das der Programmer mit den gewählten Parametern verwendet werden kann. Eine



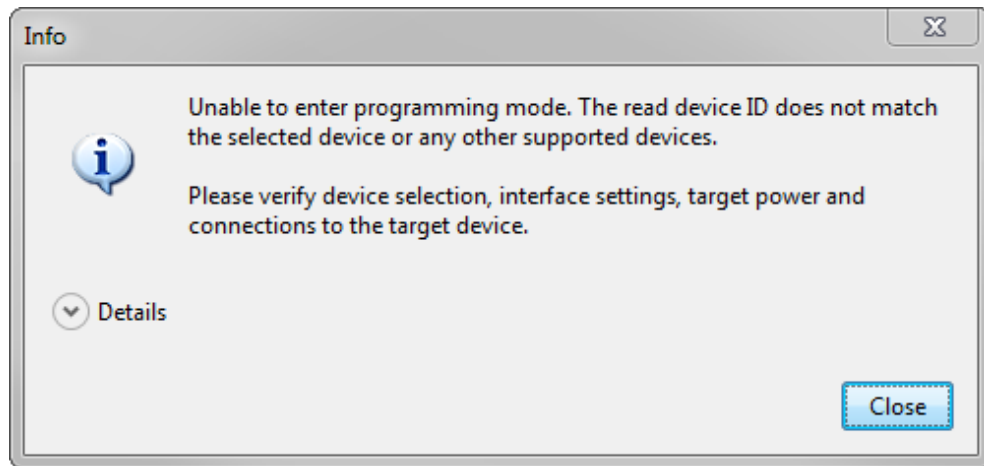


Abbildung 14: DeviceProgramming

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

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 4.3 Fusebits im Kapitel Hardware.

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

Ein Auszug der verwendeten Parameter aus der AVRDUDE Handbuch Seite:

```

jan@cyphen: ~
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 15: 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	Use port to identify the device to which the programmer is attached. <b>usb für den AVRISP MKII</b>
-c programmer-id	<b>avrispmkII für den AVRISP MKII</b>
-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	Specify the bit clock period for the JTAG interface or



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.

#### 9.4 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 ausgegebenen Dateien auf der Console 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 4: 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-Skript erstellt, 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 16: BuildWebpage.sh für Linux

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

Abbildung 17: BuildWebpage.bat für Windows

#### 9.4.1 Konfiguration des Webserver

Die Einstellung des Webserver erfolgt über die config.h Datei. In der config.h Datei, können Zum einen 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 du beachten, das 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 Netzwerkcommunication genutzt wird. Deswegen wird Port B auch nicht Standardmäßig definiert.
- OUTC dieser Port wird von Polin standardmäßig für die Ausgänge verwendet und ist von uns bereits so 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.
- OUTC Der C Port 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 der das Gerät erreicht werden kann. Wichtig ist hier, das kein anderes Gerät die selbe Adresse im Netzwerk verwendet. Auch kann die Router IP-Adresse und Netzmaske angegeben werden. Eine weitere Wichtige Einstellung ist die verwendete Mac Adresse des Netzwerk Controllers. Diese wird über die Variablen MYMAC1-6 Definiert.

## **9.5 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.

## **9.6 Hexfiles Überspielen**

Zum übertragen der Hexfiles gibt es verschiedene Möglichkeiten.

### **9.6.1 Atmel Studio**

## **9.7 Die Website**



## **10 Rückblick**

### **10.1 Soll/Ist-Vergleich**

### **10.2 Verworfenene Varianten**

### **10.3 Eigenbewertung**

Jan-Henrik Preuß

Christian Würthner

Ann-Sophie Dietrich

Marcel Schlipf



## 11 Ausblick





## 12 Fazit

Die Welt der Mikrocontroller steckt voller Möglichkeiten ist aber auch mit einigen Schwierigkeiten behaftet. Anders als beim arbeiten mit Computern bei denen der Speicherplatz für einfache Programme schier unbegrenzt ist kommt es bei den Mikrocontroller auf jedes Byte an. So bestand in unserem Projekt nicht nur die Schwierigkeit 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. Als eine weitere Herausforderung bei Mikrocontrollern kommt 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 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.



## Literaturverzeichnis

- [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