

Projektarbeit

im Studiengang

AIB/CNB

Webserver für ein embedded Board mit AVR-Prozessor

Dokumentation

Referent : 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	iv
Abbildungsverzeichnis	v
Tabellenverzeichnis	vii
Abkürzungsverzeichnis	ix
1 Einleitung	1
2 Grundlagen	3
2.1 AVR Net-IO-Board	3
2.1.1 Technische Daten	3
2.2 Mikrocontroller	4
2.2.1 ATmega32	4
2.2.2 ATmega644P	4
2.2.3 ATmega1284P	4
3 System Architektur	5
4 Tutorial	7
4.1 Einrichten eines neuen Microcotrollers	7
4.2 Konfiguration	9
4.3 HTML Header Compiler	9
4.4 Hexfiles Überspielen	9
4.4.1 AVRDUDE	9

4.4.2	Atmel Studio	9
4.5	Die Website	9
5	Werkzeuge	11
5.1	Das Atmel Studio	11
5.1.1	DeviceProgramming	11
5.2	Projekt Einstellungen	11
5.3	AVRDUDE	12
5.4	HTML Header Compiler	12
6	Der Webserver	13
6.1	Einrichtung	13
6.2	Einbindung der Website	13
7	Die Website	15
8	Kommunikation zwischen Server und Webseite	17
8.1	Kommunikation im Projekt Radig	17
8.2	Erster Ansatz	17
8.3	Polling oder Pushing	18
8.3.1	Polling	18
8.3.2	Pushing	19
8.3.3	Entscheidung	20
8.4	Aufbau der REST-Schnittstelle	20
8.5	Implementierung der REST-Schnittstelle auf dem Server	20
8.6	Implementierung der REST-Schnittstelle auf dem Client	21
9	Zeitlicher Ablauf	23
10	Ausblick	25
11	Fazit	27
	Literaturverzeichnis	29

Abbildungsverzeichnis

Abbildung 1: AVR-NET-IO - Pollin GmbH	3
Abbildung 2: DeviceProgramming	7
Abbildung 3: AVRDUDE Ausgabe	9
Abbildung 4: DeviceProgramming	11

Tabellenverzeichnis

Tabelle 1: Auslesen und setzen von Fuse-Bits mit dem AVRDUDE	8
Tabelle 2: Auslesen und setzen von Fuse-Bits mit dem AVRDUDE	8

Abkürzungsverzeichnis

1 Einleitung

Semesterprojekt and stuff

2 Grundlagen

2.1 AVR Net-IO-Board



Abbildung 1: AVR-NET-IO - Pollin GmbH

2.1.1 Technische Daten

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

2.2 Mikrocontroller

2.2.1 ATmega32

2.2.2 ATmega644P

2.2.3 ATmega1284P

3 System Architektur

- Was soll hier rein??

4 Tutorial

4.1 Einrichten eines neuen Microcotrollers

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 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.

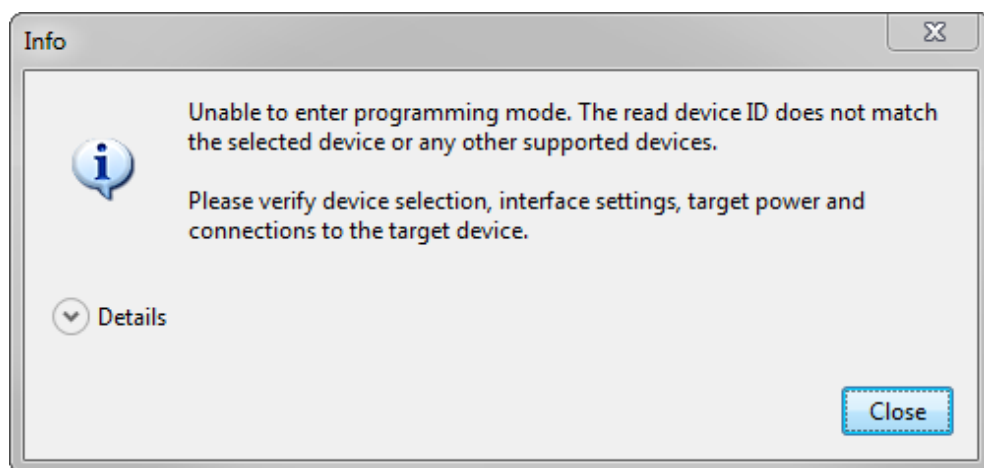


Abbildung 2: DeviceProgramming

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>.

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>

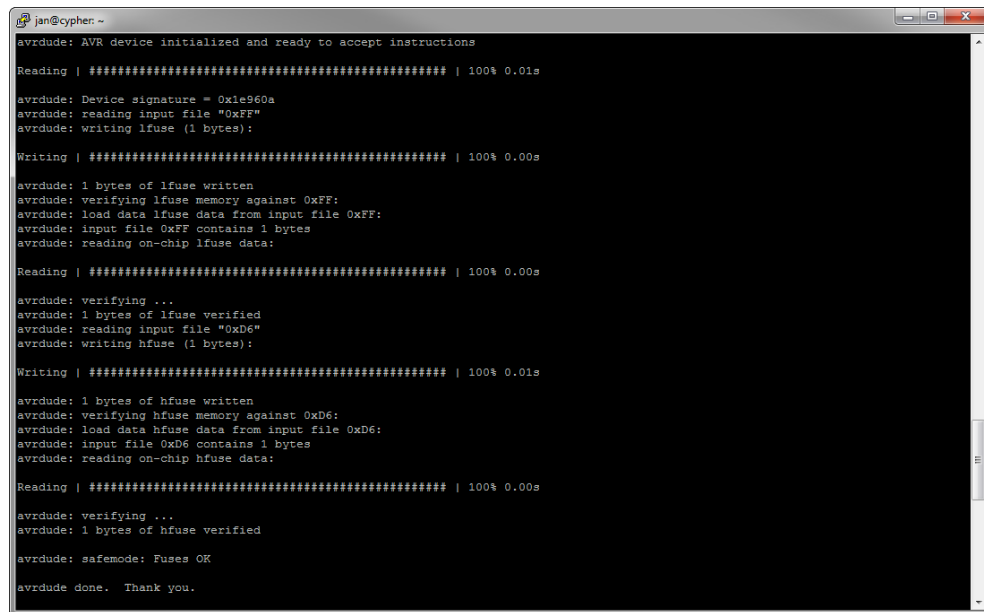
Tabelle 1: Auslesen und setzen von Fuse-Bits mit dem AVRDUDE

Ein Auszug der verwendeten Parameter aus der AVRDUDE Handbuch Seite.

-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>m32 ⇒ ATmega32 m644p ⇒ ATmega644P m1284p ⇒ ATmega1284P</p>
-P port	Use port to identify the device to which the programmer is attached. usb für den AVRISP MKII
-c programmer-id	avrispmkII für den AVRISP MKII
-U memtype:op:filename:filefmt	<p>The memtype field specifies the memory type to operate on.</p> <p>hfuse The high fuse byte. lfuse The low fuse byte.</p> <p>The op field specifies what operation to perform: r read device memory and write to the specified file w 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>Hier die Bytes die gesetzt werden 0xFF bzw 0xD6</p>
-B bitclock	Specify the bit clock period for the JTAG interface or the ISP clock

Tabelle 2: Auslesen und setzen von Fuse-Bits mit dem AVRDUDE

Die Ausgabe von AVRDUDE beim setzen der neuen Fuse-Bit Einstellungen.



```
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 3: AVRDUDE Ausgabe

Anschließend kann der Mikrocontroller zusammen mit dem AV-Net-IO und Atmel-Studio programmiert werden.

4.2 Konfiguration

Einstellen von: IP Mac Ein/Ausgänge

4.3 HTML Header Compiler

Parameter und Funktionsweise

4.4 Hexfiles Überspielen

Zum Übertragen der Hexfiles gibt es verschiedene Möglichkeiten.

4.4.1 AVRDUDE

4.4.2 Atmel Studio

4.5 Die Website

5 Werkzeuge

5.1 Das Atmel Studio

5.1.1 DeviceProgramming

Gerät Auswählen Contoller Auswählen Spannung und Signatur Fuse Bits Hexfile Flas-

hen

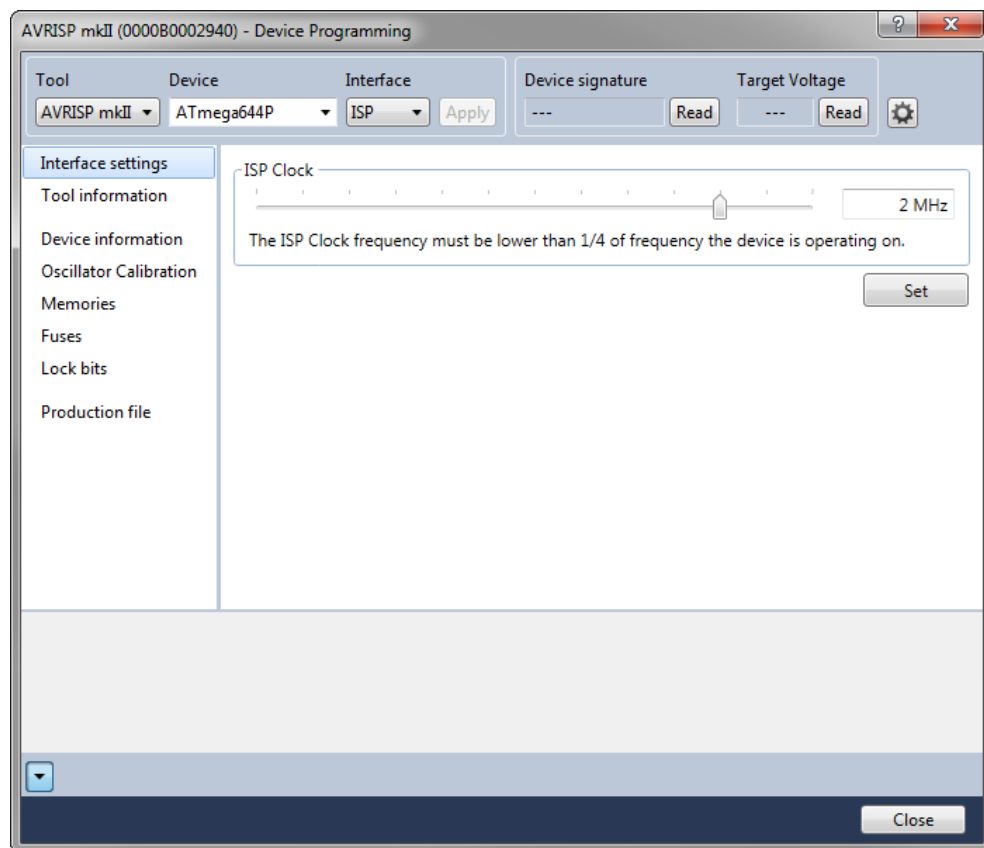


Abbildung 4: DeviceProgramming

5.2 Projekt Einstellungen

Taktfrequenz Programmer Empfolene Tool Settings

5.3 AVRDUDE

5.4 HTML Header Compiler

6 Der Webserver

Als Basis für unser Projekt haben wir die Firmware von Ulrich Radig verwendet. Da diese Vorlage für unseren Anwendungsfall zu umfangreich ist, haben wir uns für die abgespeckte Variante von Günther Menke entschieden. Die Änderungen sind zum einen das entfernte Kamera-Feature und um zusätzlichen Quellcode für einen alternativen Netzwerkcontroller abgespeckt wurde.

6.1 Einrichtung

Die Einstellung des Webserver erfolgt über die `config.h` Datei. Hier können IP-Adresse, Mac-Adresse oder die Ports eingestellt werden.

6.2 Einbindung der Website

Alle für den Betrieb der Website benötigten Dateien sind nicht über ein Dateisystem vorhanden, sondern werden beim zugriff des Benutzers auf den Webserver über die `webpage.h` datei geladen. Das erstellen der `.h` erfolgt über das Beigelegte HTML Header Compiler Werkzeug erstellt. Eine Beispiel zum erstellen der `webpage.h` Datei gibt es im Tutorial Abschnitt. Das Werkzeug wird im Kapitel Werkzeuge detaillierter erklärt. Abschließend ist noch zu erwähnen, das die `webpage.h` nicht für manuelle Bearbeitung gedacht ist. Dies geschieht ausschließlich über die Quell-Dateien und dem anschließenden umwandeln mit dem HTML Header Compiler

7 Die Website

8 Kommunikation zwischen Server und Webseite

8.1 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.

8.2 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 technisches Standart 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);

8.3 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.

8.3.1 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.

8.3.2 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.

8.3.3 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

8.4 Aufbau der REST-Schnittstelle

- 6 URLs
-Hier Zitat aus Präsentation, aber noch anpassen!

8.5 Implementierung der REST-Schnittstelle auf dem Server

Für die Restschnittstelle wurde weitestgehend der bestehende Quellcode von Radig verwendet. Zum schalten der Eingänge sind Platzhalter vorgesehen (z.B. %PortC1) mit diesen kann der direkte Port zusammen mit dem Pin (Hier Port C1) an eine beliebige Stelle im Quellcode platziert werden. 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. Zum setzen der Ausgänge wird über einen Post Aufruf der entsprechende Port und Pin gesetzt. Die Informationen, die zum setzen eines Ports benötigt werden setzen sich zusammen aus einem SET Befehl und dem zu schaltenden Port und Pin. Abschließend muss das Ende der Schaltanweisung mit SUB gekennzeichnet werden, da der Webserver auf diese Steuerzeichen prüft. Ein Beispiel Post zum schalten der Pins C1 und C2 sieht folgendermaßen aus:

```
SET=C1&SET=C2&SUB=Senden
```

Anzumerken ist, dass beim Setzen der Pins die bestehenden Zustände verloren gehen.

8.6 Implementierung der REST-Schnittstelle auf dem Client

- Aufruf der URLs im Hintergrund mit Ajax
- info und pininfo werden zu Beginn einmalig aufgerufen (synchron um zu gewährleisten dass die Daten zur Verfügung stehen für andere Initialisierungen)
- values wird mit `setTimeout(...)` zyklisch asynchron aufgerufen
- synchrones aufrufen von values führt dazu, dass sich die Webseite aufhängt, da der (Single-)JavaScript Thread mit dem laden der Daten beschäftigt ist und nicht für andere Aufgaben zur Verfügung steht. Bei einem asynchronen Aufruf werden die Daten von einem anderen Thread im Hintergrund geladen
- JSON-Text wird mit `JSON.parse(...)` in ein Objekt transformiert
- Informationen werden über entsprechende getter zur Verfügung gestellt
- Funktion `onValueChanged` wird jedes mal aufgerufen wenn neue Daten zur Verfügung stehen
- `onError` wird aufgerufen wenn ein Fehler in der Kommunikation aufgetreten ist
- Über setter werden die entsprechenden URLs asynchron aufgerufen und so die Daten an den Server übermittelt

9 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.

10 Ausblick

Brauchen wir den Ausblick?

11 Fazit

Literaturverzeichnis