Studienarbeit

Fernüberwachung und Fernsteuerung über Internet (Android Applikation)

Studiengang Informationstechnik

an der Dualen Hochschule Baden-Württemberg Stuttgart

****

Betreuer: Klaus Hartmann

Name: Hans-Joachim Krauch

Matrikelnummer:

Name: Waldemar Siebert

Matrikelnummer: 0568072

Studiengang: Informationstechnik

Vertiefung: Ingenieursinformatik

Anfertigungszeitraum: 18.10.11 – 11.06.12

**Ehrenwörtliche Erklärung**

gemäß § 5 (2) der „Studien- und Prüfungsordnung DHBW Technik“ vom 18. Mai 2009.

Wir haben die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ort, Datum Unterschrift

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ort, Datum Unterschrift

**Abstract**

Deutsch

English

Inhaltsverzeichnis

[Abstract 2](#_Toc326742618)

[1 Auswahl des Mikrokontrollers 2](#_Toc326742619)

[2 AVR-NET-IO 2](#_Toc326742620)

[3 Firmware 2](#_Toc326742621)

[4 Kommunikationskonzept 2](#_Toc326742622)

[5 Stand der Dinge 2](#_Toc326742623)

[6 App-Design 2](#_Toc326742624)

[7 Android App 2](#_Toc326742625)

[7.1 Android 2](#_Toc326742626)

[7.2 Android SDK 2](#_Toc326742627)

[7.3 App Architektur 2](#_Toc326742628)

[7.4 Ergebnis der App Implementation 2](#_Toc326742629)

[7.4.1 GUI 2](#_Toc326742630)

[7.4.2 Ablauf 2](#_Toc326742631)

[7.4.3 Besonderheiten 2](#_Toc326742632)

[7.5 Bedienungsanleitung der App 2](#_Toc326742633)

[8 Kommunikationsprotokoll 2](#_Toc326742634)

[8.1 SCPI 2](#_Toc326742635)

[8.2 Eigene Befehle 2](#_Toc326742636)

[9 TODOs: 2](#_Toc326742637)

[9.1 Datenverwaltung 2](#_Toc326742638)

[9.2 SPI-Protokoll 2](#_Toc326742639)

[9.3 Die erste Doku soll so ca. 5 Seiten umfassen 2](#_Toc326742640)

[9.4 Aufgabenstellung 2](#_Toc326742641)

[10 Literaturverzeichnis 2](#_Toc326742642)

**Abbildungsverzeichnis**

[Abbildung 1: AVR-NET-IO von Pollin 2](#_Toc326742704)

[Abbildung 2: Client-Proxy-Server Konzept 2](#_Toc326742705)

[Abbildung 3: Konzept des App Designs 2](#_Toc326742706)

**Tabellenverzeichnis**

# Einleitung

## Einführung

## Motivation

## Aufgabenstellung

# Stand der Technik

## Was ist Android

## Was ist ein AVR

## Fazit/Entscheidung

Für bestimmte Komponenten oder Firmware usw.

# Entwicklungsumgebung

Eclipse CDT/IDE für JAVA und C. Mit Android SDK und AVR-SDK und avrdude.

# Auswahl des Mikrokontrollers

Für die Umsetzung eines Mikrokontrollers, der mit einem Ethernet Controller kommuniziert kamen mehrere Möglichkeiten in Frage. Zum einen war es möglich alles neu zu entwickeln und hierfür einen Mikrokontroller nach Vorgaben auszuwählen, oder aber auf bereits bestehende Projekte aufzubauen. Die Vorgaben hierfür waren, einen Mikrokontroller mit 40 Beinen zu nutzen, der mit einem Ethernet Controller kommunizieren soll und genügend Platz für einen Webserver aufweist. Durch eine Recherche sind zwei Projekte aufgefallen. Bei dem ersten Projekt handelt es sich um einen Webserver von Ulrich Radig. Dies ist gut dokumentiert und enthält einen Schaltplan mit einer Stückliste der benötigten Bauteile. Beim zweiten Projekt handelt es sich um den kommerziellen Bausatz AVR-NET-IO von Pollin, welcher inklusive aller Bauteile und einer kleinen Software bestellt werden kann. Eine Bestellung als Fertigmodul ist auch möglich, kostet aber entsprechend mehr.

Die Auswahl fiel deshalb auf den AVR-NET-IO von Pollin als Bausatz. Der Hauptgrund für die Verwendung des AVR-NET-IO ist der gute Support durch die frei verwendbare Ethersex-Firmware. Zudem ist der Bausatz günstig zu haben und lässt sich mit den nötigen Lötkenntnissen gut zusammenbauen. Für die Kommunikation zwischen dem Mikrokontroller und dem Ethernet Controller wird SPI verwendet. Des Weiteren besitzt das NET-IO eine ISP-Schnittstelle, die es erlaubt den verwendeten Controller direkt auf dem Board zu programmieren.

# AVR-NET-IO

Der AVR-NET-IO, wie in Abbildung 1 zu sehen, besteht aus einer Ethernet-Platine mit einem Sockel für einen 40 Pin Mikrokontroller, einem Netzwerkcontroller ENC28J60 und einem Seriell Controller MAX232. Auf der Platine befinden sich 8 digitale Ausgänge, 4 digitale Eingänge und 4 ADC-Eingänge, die über TCP/IP abgerufen/geschaltet werden können. Weiterhin ist eine RJ45 Netzwerkbuchse, ein 9-poliger Sub-D Anschluss für RS232, eine ISP-Schnittstelle und alle weiteren benötigten Komponenten vorhanden. Die Platine muss mit 9V Betriebsspannung versorgt werden und hat eine Stromaufnahme von etwa 190mA. Die digitalen Ein- und Ausgänge sind mit 0V und 5V Pegeln als Richtwert angegeben. Die Platine hat mit den verwendeten Komponenten die Maße 108mm Länge, 76mm Breite und 22mm Höhe.

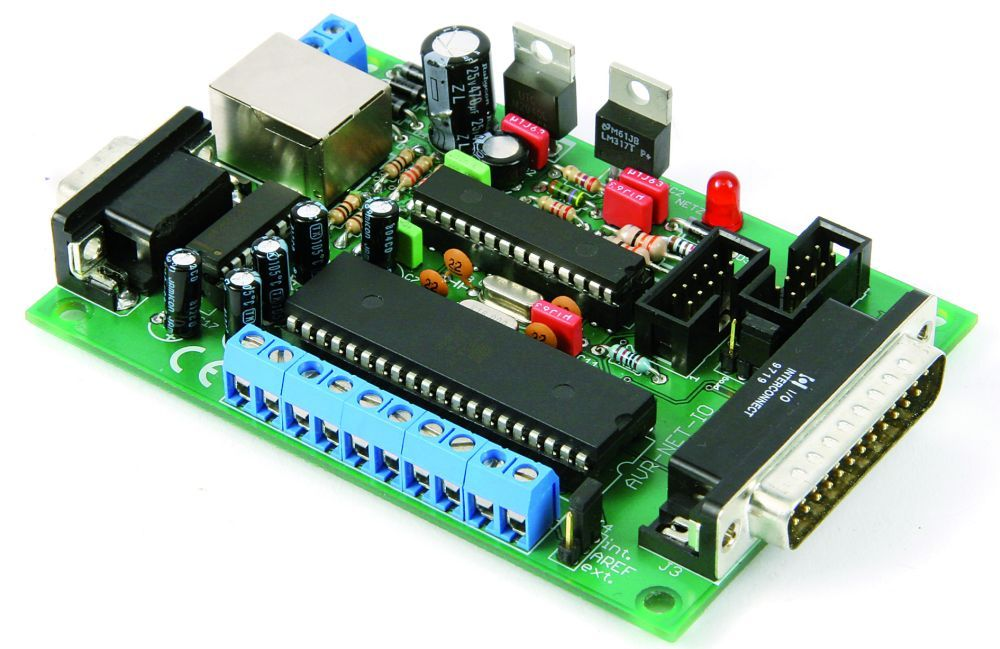


Abbildung 1: AVR-NET-IO von Pollin

# Firmware

Die Recherche nach einer geeigneten Firmware für den Mikrokontroller brachte das „Ethersex“-Projekt als eine sehr interessante Möglichkeit hervor. Ethersex ist eine Firmware, welche für die Verwendung auf AVR 8-Bit Mikrokontroller mit Netzwerkanschluss zugeschnitten ist. Das Ethersex-Projekt unterstützt mehrere Boards, darunter das AVR-NET-IO von Pollin. Besonders zeichnet sich die Firmware durch einen bereits implementierten TCP/IP-Stack (IPv4 & IPv6) und zahlreiche weitere Features, wie z.B. ein HTTP-Server, aus.

Auf dem Mikrokontroller sorgt Ethersex für die Kommunikation mit dem Ethernet-Controller und die Verarbeitung der IP-Pakete. Ethersex erlaubt es außerdem TCP-Befehle (ECMD) an den Mikrokontroller zu schicken. So kann beispielsweise der Hostname des Mikrokontrollers an einem PC abgefragt werden.

# Kommunikationskonzept

Das Kommunikationskonzept besteht aus mehreren Komponenten. Es muss eine Kommunikation zwischen einem Android Smartphone und dem Mikrokontroller aufgebaut werden. Hierfür wird per Android-App ein bestimmter TCP-Befehl an den am Netz angeschlossenen Ethersex-Controller gesendet. Der Ethersex-Controller reagiert auf den Befehl und empfängt die übergebenen Parameter. Ohne diese zu verarbeiten, leitet er diese an den über SPI angeschlossenen Steuercontroller weiter.

Im Steuercontroller wird der Befehl verarbeitet und der Rückgabewert anschließend wieder über die SPI-Schnittstelle dem Ethersex-Controller mitgeteilt. Dieser packt diesen in ein bzw. mehrere IP-Pakete und sendet diese dem Android-App als Antwort zurück.

Der Ethersex-Controller verhält sich, wie Abbildung 2 zu sehen, im Grunde wie ein Proxy-Server: Er empfängt einen Request und leitet diesen an den Steuercontroller weiter. Die Antwort vom Steuercontroller wird ebenso an die Android-App weitergeleitet.



Abbildung 2: Client-Proxy-Server Konzept

# Stand der Dinge

**Ethersex**:

Das AVR NET-IO wurde erfolgreich aufgebaut und mit einem ATmega644 bestückt (ausreichend Speicherkapazität). Die Ethersex-Firmware wurde angepasst, kompiliert und auf den Mikrokontroller übertragen. Mit dem Webbrowser ist es möglich den Webserver des Ethersex-Controllers aufzurufen. ECMD-Befehle (Ethersex-Commands) konnten erfolgreich über TCP an den Mikrokontroller gesendet werden.

**Android-App:**

Das Design ist im folgen Kapitel zu sehen. Jedoch funktioniert die Kommunikation über TCP noch nicht.

# App-Design



Abbildung 3: Konzept des App Designs

Abbildung 3 zeigt das App Design für Android. Auf dem ersten Tab sind die Messwerte. Auf dem zweiten Tab sind die Schalter und auf dem letzten Tab sieht man das Impressum. Die genaue Auslegung des Designs kann sich noch ändern.

# Android App

Ein Ziel der Studienarbeit war es, durch eine Applikation auf einem Android Mobiltelefon, Steuerbefehle an den Mikrokontroller zu senden, Daten vom Mikrokontroller abzufragen und in der Applikation darzustellen.

## Android Betriebssystem

Android ist ein Betriebssystem für mobile Geräte, das im Oktober 2008 in Version 1.0, veröffentlicht wurde. Das Besondere daran ist, dass das Betriebssystem als freie Software angeboten wird. Dadurch darf der Quellcode von jedem eingesehen, verändert und weiterverbreitet werden. Dabei muss beachtet werden, welchen Freie-Software-Lizenzen die einzelnen Komponenten der Android Plattform unterliegen.

Um eine Applikation für das Android Betriebssystem zu entwickeln, benötigt man sowohl ein Java-SDK, als auch ein Android-SDK.

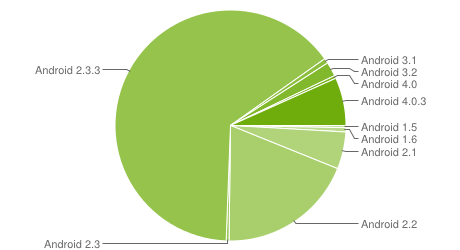


Abbildung 4: Verteilung der aktiven Android Versionen[[1]](#footnote-1)

Zur Auswahl der Version des Android-SDK kam eine Statistik der momentan aktiv genutzten Geräte zu gute. Die Erfassung wurde nach 14 Tagen am 01.06.2012 abgeschlossen und ergab die in Abbildung 4 dargestellte Verteilung. Prozentual gesehen haben Android Geräte mit der Version 1.6 oder darunter einen Anteil von unter 1%. Die Version 2.1 besitzt 5,2%, Version 2.2 19,1% und Version 2.3.X besitzt 65% Anteil. Die reine Tablet Version 3.X besitzt 2,7% und die neue gemischte Version 4.0.X besitzt schon 7,1% des Gesamtanteils der aktiven Android Geräte.[[2]](#footnote-2)

Da die Android-API Versionen abwärtskompatibel sind, kann eine mit der SDK für Android Version 1.6 entwickelte Applikation auch in höheren Versionen ausgeführt werden. Durch diese Tatsache und durch die Verteilungsstatistik wurde die Applikation in dieser Studienarbeit für die Version 2.1 entwickelt. Hierdurch werden die meisten Geräte abgedeckt und es stehen mehr Neuerungen als in Version 1.6 zur Verfügung.

### Verbreitung



Abbildung 5: Marktanteil der Smartphone Plattformen (BITKOM e.V., 2012)

Zur Ermittlung des Markanteils der einzelnen Betriebssysteme hat die BITKOM die Firma comScore beauftragt. Durch eine Onlineumfrage mit über 5000 Handy Besitzern ist eine Verteilung wie in Abbildung 5 herausgekommen. Diese Abbildung zeigt den Anteil aktiver Smartphone in Deutschland im Quartal 1 von 2012 und in Klammern von Quartal 1 von 2011. Android zeigt hier mit 40% für Q1 2012 die Marktführung auf und durch das Wachstum von 23% in einem Jahr, ein gutes Zeichen für die weitere Verbreitung. (BITKOM e.V., 2012)

Durch die große Verbreitung des Android Betriebssystems, steht die Applikation einem breiten Anwenderkreis nichts entgegen.

## Android SDK

Um Applikationen für das Android Betriebssystem zu schreiben, benötigt man das von Google frei zum Download angebotene SDK(Software Development Kit), das nicht nur den notwendigen Compiler mitbringt, sondern auch Werkzeuge zum Debuggen und Emulieren eines Android Mobiltelefons/Smartphones. Aktuell steht die Version X.X zur Verfügung.

## App Architektur

1. Architektur einer App

Wenn man die Eclipse IDE zum Schreiben von Android Applikationen nutzt, werden bei der Erstellung eines Android Projekts alle notwendigen Dateien angelegt. Die wichtigsten Dateien sind: AndroidManifest.xml, strings.xml, Layout->main.xml und MainActivity.java.

1. Lifecycle einer App

In der Dokumentation auf developers.android.com wird der Lifecycle einer Applikation aufgezeigt. In diesem Bild wird aufgezeigt in welcher Reihenfolge die überschreibbaren Methoden aufgerufen werden. So kann z.B. bei onStop() alle Daten gespeichert werden, die evtl. beim nächsten Aufruf gebraucht werden.

## Ergebnis der App Implementation

Während der Implementierung kam es zu zahlreichen Änderungen der Architektur der Applikation, wobei das Hauptaugenmerk darauf lag das in der ersten Dokumentation vorgestellte Design umzusetzen.

### GUI

1. GUI der App zeigen und Elemente erklären

### Ablauf

1. Ablauf der Logik der App aufzeigen

### Besonderheiten

1. Auf Besonderheiten eingehen: z.B: wie Messages geschickt werden, wie Schalter nachgefragt werden ob diese geschaltet wurden. TCP Verbindung als Singleton usw.

## Bedienungsanleitung der App

Installation durch ausführen der apk.

Dann beim Start zu finden bei allen Apps.

Beim Start: Popup um Adresse des AVR-Servers einzugeben: IP oder DynDNS möglich.

Automatisch werden Schalterzustände abgefragt.

Als erstes taucht der erste Tab auf. Durch den Knopfdruck auf aktualisieren werden aktuelle Werte geholt.

Auf den anderen Tabs passieren andere Sachen.

## Evaluierung der Android Applikation

### Testfall 1:

### Testfall 2:

## Anleitung zur Nutzung der Applikation

### Voraussetzungen

Android Gerät mit mind. Android Version 1.6 und WLAN oder Internetverbindung.

### Installation

Apk Downloaden und installieren.

### Nutzung des Programms

Screens und was zutun ist!

#### Verbinden

#### Messwerte abfragen

#### Schalter abfragen und Schalten

#### Impressum

#### Beenden

Beenden durch Drücken der zurück Taste auf dem Android Gerät.

# Kommunikationsprotokoll

Um einen Datenaustausch zwischen den Kommunikationspartnern Android Applikation und Proxykontroller zu gewährleisten musste ein Kommunikationsprotokoll eingeführt werden. Hierfür orientierte man sich an bekannten Protokollen, die üblicherweise eingesetzt werden.

## SCPI

SCPI steht für Standard Communication Protocol Interface und beschreibt dabei ein Kommunikationsprotokoll, dessen Aufbau wie in Abbildung X aussieht.

SCPI wird z.B. bei Messgeräten eingesetzt um beispielsweise die Werte eines Multimeters abzufragen.

Was ist SCPI, woher kommt es?

## Eigene Befehle

Für die Zwecke dieser Studienarbeit wurde ein eigenes Protokoll erstellt, das sich an SCPI orientiert. Wie auch bei SCPI wird zuerst das genannt, welche Klasse angesprochen werden soll. Daraufhin kommt der Befehl, welcher den Zweck des Aufrufs verdeutlicht und in den Klammern steht dann die ID des Zielobjekts, um genau eines Anzusprechen.

Befehle mit einem Fragezeichen vor der Klammer haben einen Rückgabewert, Befehle ohne Fragezeichen dagegen keinen.

Die Tabelle X zeigt welche Befehle sich an wen richten und welche Antworten auf diese Befehle vom Proxykontroller zurückgegeben werden.

|  |  |  |
| --- | --- | --- |
| Zweck des Befehls | Befehl an Proxykontroller | Antwort des Proxykontroller |
| *Quellen (X = 1,…,20)* | | |
| Name der Quelle X | SOURCE:NAME?(@X) | z.B. „Tempfuehler1“ |
| Wert der Quelle X | SOURCE:VALUE?(@X) | z.B. „24.2“ |
| Einheit für Wert der Quelle X | SOURCE:UNIT?(@X) | z.B. „°C“ |
| *Schalter (Y = 1,…,8)* | | |
| Name des Schalter Y | SWITCH:NAME?(@Y) | z.B. „Schalter1“ |
| Wert des Schalter Y | SWITCH:VALUE?(@Y) | „1“(=AN) oder „0“(=AUS) |
| Schalter Y anschalten | SWITCH:ON(@Y) | Keine Antwort |
| Schalter Y ausschalten | SWITCH:OFF(@Y) | Keine Antwort |
| *Statusbyte* | | |
| Statusbyte abfragen | \*STB? | „!“ für keine neuen Messwerte und Quellnamen  „A“ wenn neue Messwerte  „a“ wenn neue Quellnamen  „O“ wenn neue Messwerte und Quellnamen |
| Statusbyte löschen | \*CLS | Keine Antwort |

Die Tabelle X zeigt die Besonderheiten des Protokolls. Durch die Befehle kann bei jeder Messquelle und jedem Schalter die jeweiligen Namen und Werte abgefragt werden. So kann auch jeder einzelne Schalter durch einen Befehl angeschaltet oder ausgeschaltet werden. Ein besonderes Augenmerk kann man hier auf den Befehl \*STB? werfen. Durch diesen Befehl ist es möglich statt bei den Messquellen alle Namen, Werte und Einheiten einzeln abzufragen, nur bei Veränderung von Namen oder Messwerten, nur alle Namen oder alle Messwerte zu holen. Hierdurch wird die Anzahl der ausgetauschten Nachrichten vermindert.

Unsere reduzierte Fassung des SCPI Protokolls zur Kommunikation zwischen Applikation und Proxykontroller.

Mit Befehl und Antwort

# TODOs:

## Datenverwaltung

Wie die Daten geholt werden, ob Zwischenspeichern auf dem Controller und wann von dem andern Controller ler geholt werden will. Nur bei Zugriff, oder mit Aktualisierungstaste.

Bei Zugriff und wenn auf Aktualisierungs-Taste gedrückt wird.

→ Kein Zwischenspeichern auf dem Controller

## SPI-Protokoll

festlegen, zum Steuercontroller bzw. Android-App (SCPI?)

Von dir frei wählbar, muss allerdings noch net in die erste Doku m.M. nach

## Die erste Doku soll so ca. 5 Seiten umfassen

Es soll folgendes enthalten sein:

Ein Konzept mit 1-2 Möglichkeiten für bestimmte Entscheidungen, die wir getroffen haben.

1. Hardwareauswahl
2. Wie die Oberfläche der App/Webserver gestaltet werden soll
3. Wie die Kommunikation gestaltet werden soll

Es soll grob wiedergeben, was bisher geleistet wurde.

Im 6. Semester soll eine technische Beschreibung folgen.

# Ergebnis/Fazit des Projekts

Zusammenfassen was es kann und:

Was noch möglich ist.

## Aufgabenstellung

Thema: Fernüberwachung eines Mikrokontrollers

Beispielvorgehen:

1. Kontroller + AVR Architektur + Code verstehen
2. Kontroller wählen und kaufen (40Pin)
3. TCP/IP Stack implementieren 🡪 AVR Webserver
4. Android App: feste Felderanzahl, Messwerte sind irrelevant
5. Daten kommen von einem zweite Kontroller über SPI
6. Soll über das Internet ansprechbar sein (Auslesen und Schalter)
7. Schalter soll dargestellt werden, erst wenn auch umgeschaltet wurde soll die Stellung verändert werden

Ziel der Dokumentation:

* Ist eher für die Wartung 🡪 Modulbeschreibung, Aufbau, usw.
* Bedienungsanleitung
* Installationsanleitung

Quellen:

* AVLFreaks
* U. Radig
* <http://www.androidpit.de/de/android/wiki/view/Android_Anf%C3%A4nger_Workshop>
* [http://www.ethersex.de](http://www.ethersex.de/)
* <http://www.mikrocontroller.net/articles/AVR_Net-IO_Bausatz_von_Pollin>
* Bild:http://www.pollin.de/shop/dt/MTQ5OTgxOTk-/Bausaetze\_Module/Bausaetze/Bausatz\_AVR\_NET\_IO.html

**Literaturverzeichnis**

BITKOM e.V. (29. 05 2012). *Wettkampf der Smartphone-Plattformen*. Abgerufen am 06. 06 2012 von BITKOM: http://www.bitkom.org/72321\_72316.aspx

Google Inc. (kein Datum). *Platform Versions | Android Developers*. Abgerufen am 05. Juni 2012 von Android Developers: http://developer.android.com/resources/dashboard/platform-versions.html

1. Vgl. (Google Inc.) [↑](#footnote-ref-1)
2. Vgl. (Google Inc.) [↑](#footnote-ref-2)