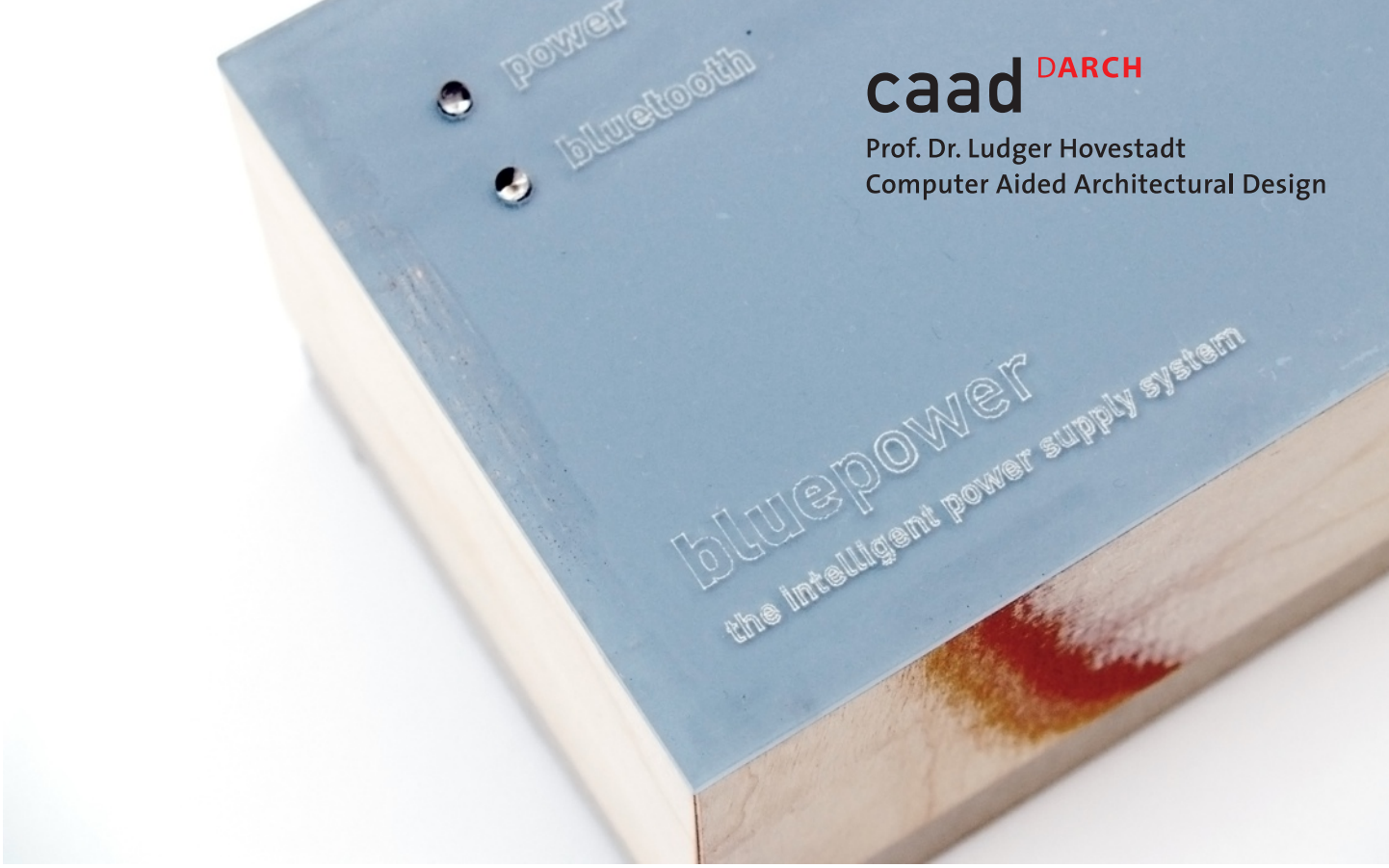


caad DARCH
Prof. Dr. Ludger Hovestadt
Computer Aided Architectural Design

bluepower - the intelligent power supply system

elective course student project

© by Yves Seiler, August 2009, yves@yoveo.com

Motivation

Im Web stolperte ich in den letzten paar Jahren immer wieder über verschiedenste Projekte aus der Sparte der Aktionskunst. Sie machten sich in irgend einer Weise Elektronik zu Nutze um beispielsweise interaktive Installationen zu ermöglichen. Beim Lesen der Projektbeschriebe fiel dabei ein Wort besonders oft: *Arduino*.

Ich habe schon früher in verschiedenen Sprachen programmiert, vermisste dabei aber immer den Link in die reale, physische Welt. Diesen Link bietet *Arduino* auf einfache und günstige Art und Weise. Schon bevor das Wahlfach *Physical Computing* angekündigt wurde hatte ich mein *Arduino* Board zu Hause und erste Erfahrungen damit gemacht. Allerdings fehlte mit neben dem Studium die Zeit, mich richtig damit zu beschäftigen. Das Wahlfach bot mir dann endlich den nötigen Rahmen, mich ernsthaft mit der Materie zu beschäftigen und vermittelte gleichzeitig das nötige Wissen, welches mir gerade im elektrotechnischen Bereich fehlte.

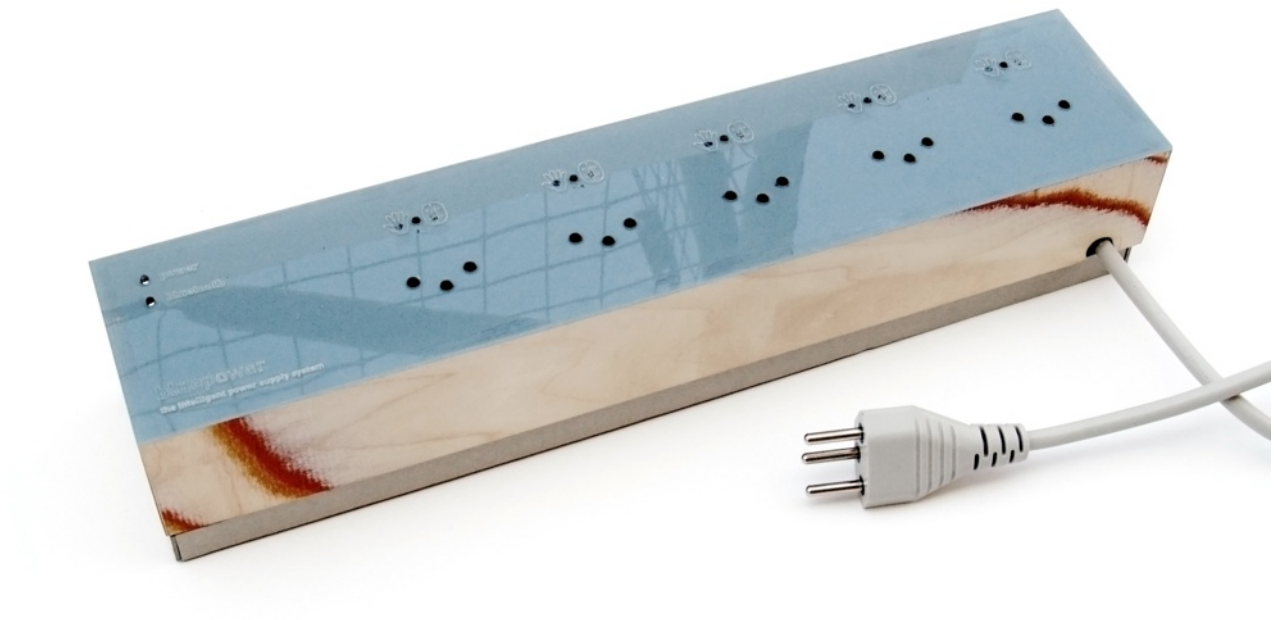
Zielsetzung

Ich entschied mich für ein – zumindest aus damaliger Sicht – einfaches Projekt. Der aktuelle Strom-Diskurs und die beschränkte Verfügbarkeit dieser Ressource inspirierte mich dazu, eine intelligente Lösung für die Stand-By-Problematik zu finden. Die Idee ist, dass nur dann Strom an Geräte abgegeben wird, wenn der Benutzer tatsächlich vor Ort ist. Neben dem offensichtlichen Vorteil, dass so Stand-By-Strom gespart wird, ermöglicht die automatische Stromschaltung aber auch noch ganz andere Dinge. So ist es zum Beispiel auch möglich, dass nur dann Licht brennt, wenn der Benutzer zu Hause ist. Und fliesst kein Strom können auch keine Kurzschlüsse entstehen: Kabelbrände und Feuer können in Abwesenheit verhindert werden. So wird schliesslich auch die Lebensdauer der Geräte verlängert.

Die Frage stellte sich dann aber, wie man auf einfache, unkomplizierte Art feststellen könnte, ob der Benutzer zu Hause ist. Dabei kamen verschiedene Funkvarianten in Frage, vor allem Xbee und Bluetooth. Ich entschied mich

für Bluetooth, da somit kein zusätzlicher Sender herumgetragen werden muss. Fast alle besitzen heute ein Bluetooth-fähiges Mobile, welches für diesen Zweck gebraucht werden kann.

Um nun den Stromzufuhr zu kontrollieren, entschied ich mich dazu, eine Steckerleiste mit Bluetooth-Aktivierung zu bauen. Die Stromzufuhr sollte hierbei sowohl manuell wie auch über Bluetooth gesteuert werden können.



Hardware

Das Endprodukt besteht im Wesentlichen aus den folgenden Komponenten (Preise ohne Versandkosten, Zollgebühren, Fehlkäufe, etc.):

#	Beschreibung	Preis ca.
1	Arduino Mini: Prozessor	36
1	Bluegiga WT32: Bluetooth Chip	75
5	Transistoren: verantwortlich für den Stromzufuhr der Relais	10
5	Relais: verantwortlich für das Schalten des Stroms (240V)	20
2	8bit Shiftregister: stellen mehr digitale Pins zu Verfügung	2
1	5V Netzteil für den Arduino Mini Prozessor	25
1	Spannungskonvertor 5V => 3.3V für Bluetooth Chip	2
12	LEDs für die Statusanzeige des Interface	10
4	Infrarot Dioden für die Interaktion mit dem Interface	10
n/a	div. Widerstände, Platinen, Kabel, Holz & Plexiglas für das Gehäuse	70
	TOTAL CHF:	260

Ich habe das Gehäuse relativ früh in der Entwicklung gebaut, was mir im folgenden etwas Probleme beschert hat alle Komponenten einzupassen. Würde ich nun einen zweiten Prototypen bauen, würde ich sehr viel weniger Kabel benutzen und dafür eine grosse Platine löten. Das Problem an Kabeln ist einerseits, dass sie sehr viel Platz benötigen, andererseits brachen mir die Kabel bzw. der Draht an der Lötstelle mehrere Male. Hier hat sich Heissleim als sehr praktisch erwiesen um Lötstellen zu sichern. Kabel machen die Entwicklung auch sehr schnell unübersichtlich. Geholfen hat da nur, die Kabel schön zu bündeln.

Was rückblickend sehr viel Zeit in Anspruch nahm war das Beschaffen der Komponenten. Man bekommt zwar viele Komponenten über Pusterla, Distrelec und Konsorten. Anderes (wie in diesem Fall der Bluetooth-Chip) kann man aber nur im Ausland bestellen, was durch Steuern und die horrenden Bearbeitungsgebühren des Zolls sehr teuer werden kann. Abgesehen von den genannten kleinen Problemen war die Entwicklung auf der Hardware Seite aber ziemlich problemlos. Die Arduino-Plattform macht das Entwickeln wirklich sehr einfach und verzeiht dem Laien auch einiges.

Im Folgenden gehe ich auf einige Hardware-Fragestellungen ein, die ich während der Entwicklung zu lösen hatte.

Prozessor
Ich wollte zuerst mit einem 3.3V Chip arbeiten, weil aber die NewSoftSerial-Library (siehe Software) nur mit 5V Chips bzw. 16 Mhz läuft, habe ich mich schliesslich für den Arduino Mini entschieden. Der Vorteil eines 3.3V Chips wäre ausserdem gewesen, dass ich mir keine Gedanken über die Stromversorgung des Bluetooth Chips machen hätte müssen, da dieser ebenfalls mit 3.3V läuft.

Bluetooth
Was den Bluetooth-Chip anbelangt hatte ich nicht so viel Auswahl. Ich schaute mir die verschiedenen Chips auf Sparkfun.com an und entschied mich für den WT32 mit Breakout-Board, was das Löten sehr vereinfacht (keine SMD-Verbindungen). Der Chip lässt sich sehr einfach mit Kommandos über die serielle Schnittstelle steuern. Das einzige Problem war hier eigentlich, den Chip richtig zu konfigurieren (Bluetooth Code of Device, baud rate, Kommunikationsmasken, Stromsparmodi etc.).

Stromversorgung DC
Ich wollte die DC-Stromversorgung für den Prozessor, den Bluetooth-Chip etc. im Gehäuse haben um ein möglichst kompaktes Design zu ermöglichen. Da ich das Gehäuse schon gebaut hatte, bevor ich ein Netzteil organisiert hatte, stand ich vor dem Problem, ein möglichst kleines Netzteil zu finden. Bei Pusterla wurde ich nach etwas suchen aber fündig. Ich musste das Netzteil dann nur noch aus dem eingeschweissten Gehäuse befreien; mit dem Dremel und einer Trennscheibe ging das ganz gut.

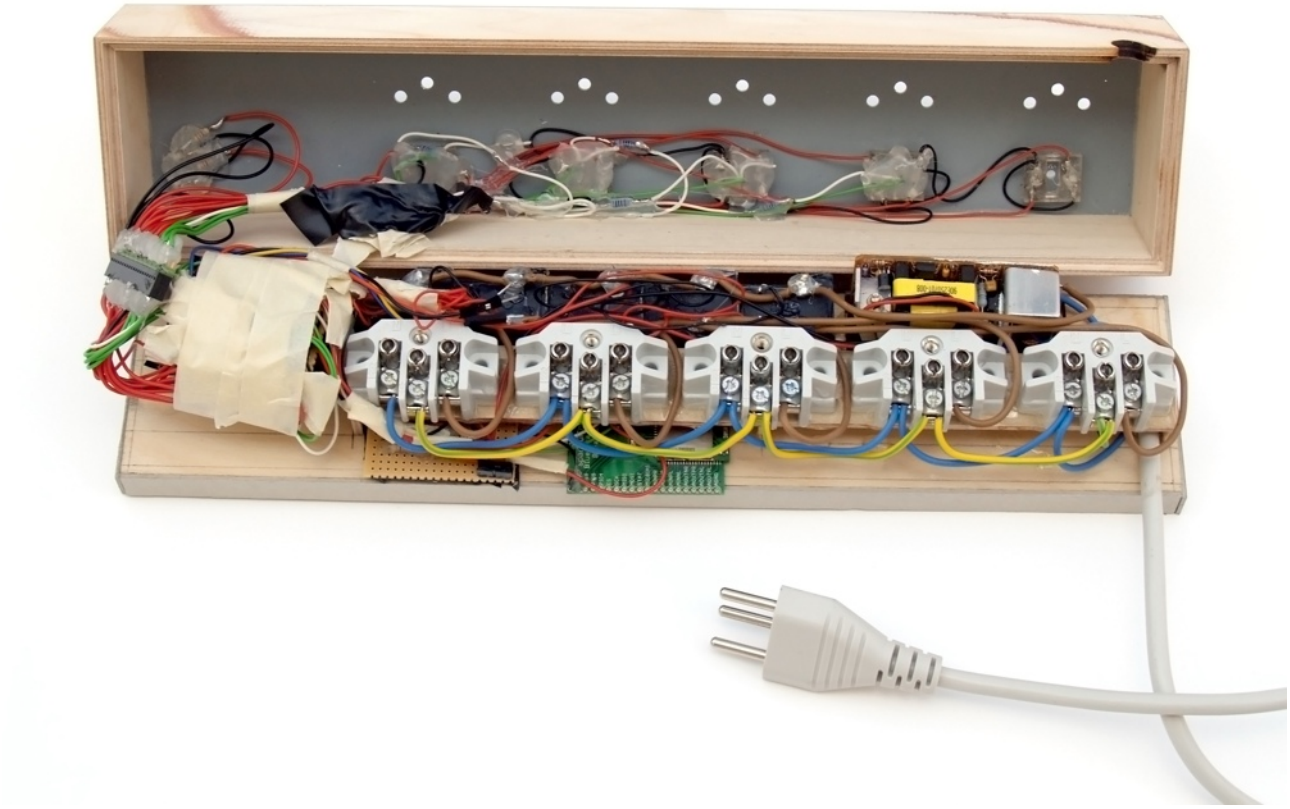
Der Vorteil an einem 3.3V Chip wäre gewesen, dass ich den Strom direkt für den Prozessor und den Bluetooth-Chip hätte verwenden können. So musste ich zusätzlich noch einen Voltkonverter einbauen, welcher mir die 5V in 3.3V wandelt.

Relais
Um später den Strom zu schalten, musste ich die entsprechenden Relais organisieren. Diese mussten mit einer Spannung von 240V zurechtkommen. Wie sich herausstellte, ist die zu Verfügung stehende Leistung des Arduino zu schwach um die Relais direkt über die Pins zu schalten, weshalb ich zusätzlich noch Transistoren kaufen musste um die Relais direkt über das Netzteil zu speisen.

Shift Register
Für die Steuerung aller Komponenten brauche ich relativ viele Pins; mehr Pins als der Arduino Mini bereitstellt (14 digitale Pins). Um dieses Problem zu beheben, baute ich zusätzlich zwei 8bit Shift Register ein. Es handelt sich dabei um Chips, welche eine Konvertierung von Seriell zu Parallel erlauben. Sie können in Reihe geschaltet werden und ermöglichen so theoretisch unendlich viele Pins.

Infrarot-Dioden
Für die Interaktion mit dem Interface suchte ich nach einer einfachen, knopflosen Variante. Ich entschied mich schliesslich für eine Lösung mit Infrarotsensoren. Das funktioniert eigentlich ziemlich gut – solange noch ein

gewisses Mass an Tages oder Kunstlicht vorhanden ist. Hier wäre es intelligent gewesen, zusätzlich noch Infrarotdioden mit einzubauen. Auch musste ich einen Algorithmus programmieren, damit ein Lichtwechsel (Kunstlicht) nicht als Schaltung interpretiert wird. Da ich übersehen hatte, dass der Arduino Mini nur 4 analoge Inputs besitzt (statt den erwarteten 5), steuert nun eine der 4 Dioden zwei Relais.



Software

Das Programmieren verlief eigentlich ohne Probleme. Wer Processing kennt, fühlt sich in der Arduino-Umgebung sofort zu Hause. Die Sprache ist sehr einfach aufgebaut und der Code ist schnell auf den Chip geladen. Das Debugging ist so sehr schell und effizient machbar. Probleme entstehen erst durch Fehler in der Hardwarekonfiguration (lose Verbindungen/Kabelbruch, Denkfehler in der Verdrahtung, etc.), welche das Fehlersuchen sehr aufwändig machen können. Hier hilft nur – genau wie beim Programmieren von Software – das Isolieren der verschiedenen Prozesse und genaues Beobachten und Messen.

Bluetooth
Konkret hatte ich vor allem mit der Kommunikation zum Bluetooth-Chip Probleme. Die Probleme waren am Anfang vor allem darauf zurückzuführen, dass ich den Bluetooth-Chip an die Standardpins des Arduino für die serielle Kommunikation gehängt hatte. Debugging-Nachrichten, die ich mir an den Computer gesendet hatte um den Verlauf des Programms zu verfolgen, wurden immer auch an den Bluetooth-Chip gesendet, was zu vielen Fehlermeldungen des Chips geführt hat. Das Verwenden der NewSoftSerial-Library, welche es ermöglicht, auch andere Pins als Serielle Schnittstelle zu verwenden, hat aber dieses Problem beseitigt. Aus mir unerfindlichen Gründen hatte ich dann aber eine Reihe von Timingproblemen – Nachrichten kamen unvollständig oder gar nicht an – welche ich nur mit viel ausprobieren lösen könnte.

Der Chip lässt sich sehr einfach über Kommandos steuern. Mit dem Befehl

```
INQUIRY 8
```

zum Beispiel wird dem Chip mitgeteilt, dass er nach Geräten mit Bluetooth in der Umgebung suchen soll (für 8*1.28 Sekunden, wobei dies die empfohlene Dauer ist um alle Geräte zu finden). Mit Hilfe von solchen einfachen Befehlen lassen sich sämtliche Funktionen des Chips steuern. Der Chip behandelt Verbindungsanfragen in meinem Fall eigenständig, sodass ich – solange der Chip richtig konfiguriert ist – nur noch die ausgegebenen Bluetooth-Adressen *parsen* muss. Das Arbeiten mit Text ist in Arduino eine etwas diffizile Sache, weil es Strings als Datentyp und entsprechende Methoden eigentlich nicht gibt. Ein String ist in Arduino einfach ein Array von Chars. Dies führt sehr schnell dazu – wenn man beispielsweise ein Array von Strings haben möchte – dass man mit mehrdimensionalen Arrays arbeitet. Das ist nicht wirklich ein Problem, verkompliziert das Ganze aber gehörig. Ein weiteres Problem ist, dass man Strings (aufgrund der beschriebenen Problematik) nicht direkt miteinander vergleichen kann. Wenn ich also prüfen möchte, ob ein Gerät aus der Umgebung schon bekannt ist, muss ich Char für Char (also Buchstabe für Buchstabe) miteinander vergleichen:

```
if (activeDevice[0] == device_adress[0]){
if (activeDevice[1] == device_adress[1]){
if (activeDevice[2] == device_adress[2]){
if (activeDevice[3] == device_adress[3]){
if (activeDevice[4] == device_adress[4]){
if (activeDevice[5] == device_adress[5]){
if (activeDevice[6] == device_adress[6]){
if (activeDevice[7] == device_adress[7]){
if (activeDevice[8] == device_adress[8]){
if (activeDevice[9] == device_adress[9]){
if (activeDevice[10] == device_adress[10]){
if (activeDevice[11] == device_adress[11]){
if (activeDevice[12] == device_adress[12]){
if (activeDevice[13] == device_adress[13]){
if (activeDevice[14] == device_adress[14]){
if (activeDevice[15] == device_adress[15]){
if (activeDevice[16] == device_adress[16]){
```

Es gibt externe Libraries für Arduino, welche solche Funktionen beinhalten, ich versuchte in dieser Arbeit aber, auf möglichst keine externen Ressourcen zurückzugreifen. Hier wünschte ich mir, dass solche Funktionen noch in Arduino integriert werden. Aber wenn man betrachtet, wie schnell sich Arduino entwickelt, habe ich da auch keine Bedenken.

Shift Register
Die Shift Register lassen sich eigentlich sehr einfach steuern. Im Prinzip sendet man einen binären Informationsstrang an den Chip, welcher die Pins definiert, die Strom führen.

```
// ein Beispiel:
// LED1 an:     1
// LED1 aus:    0
// LED1 aus:    0
// LED1 an:     1
// führt zu:    1001
```

Der Strang muss aber zuerst noch ins Dezimalsystem umgewandelt werden. Weil Arduino dafür keine Methode bereitstellt, habe ich eine kleine Funktion geschrieben, welche spezifisch 16 bit (da ich zwei 8bit Shift Register verwende) ins Dezimalsystem umwandelt:

```
  for (i = 0; i < 16; i++){
    power = round(pow(2,i));
    settingsValue = settingsValue + long(int(settings[15-i])*power);
  }

// 1001 wird somit zu 9
```

Wenn ich also im Programm den Status einer LED ändern möchte, muss ich nur die entsprechende 0 zu einer 1 machen – oder vice versa – und dann den neuen Informationsstrang wieder ins Dezimalsystem umwandeln. Schliesslich wird der Informationsstrang folgendermassen an den Chip gesendet, wobei das Senden in zwei Schritten erfolgt, da immer nur 8 bit aufs Mal gesendet werden können:

```
// settingsValue: pins to be activated
void shiftOutData(){

  digitalWrite(dataPin, 0);
  digitalWrite(clockPin, 0);

  digitalWrite(latchPin, LOW);
  shiftOut(dataPin, clockPin, LSBFIRST, settingsValue);
  shiftOut(dataPin, clockPin, LSBFIRST, settingsValue >> 8);
  digitalWrite(latchPin, HIGH);

}
```

Programmstruktur
Damit das Programm einigermassen übersichtlich bleibt, habe ich versucht so viel Code wie möglich in Funktionen unterzubringen. Insgesamt ist der Code nahezu 1000 Zeilen lang, weshalb es sehr wichtig ist, dass man den Überblick behält. So besteht der Hauptcode meines Programms nur aus folgenden Zeilen:

```
void loop(){ // start loop

  if (mySerial.overflow()){          // if serial buffer overflow
  Serial.println("overflow");
  mySerial.flush();                  // delete buffer content
  }

  while (mySerial.available()){
    parseSerial();                   // parse serial data
  }

  switchPower();                     // check interface
  sendInquiry();                     // send bluetooth commands
  setRelaisState();                  // switch relais

} // end loop
```

Schlussbemerkungen

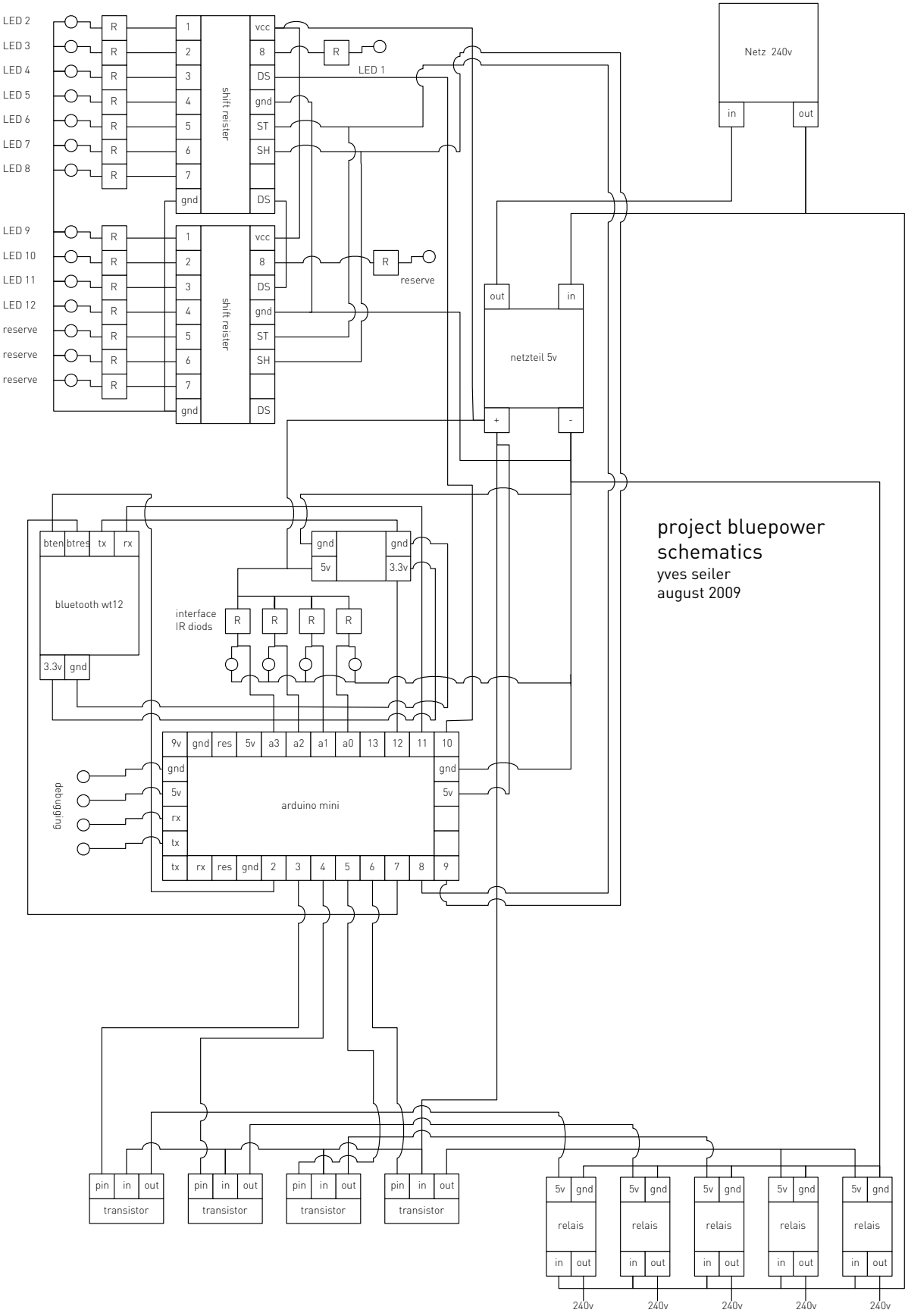
An diesem Projekt habe ich solange gearbeitet wie an keinem anderen bis jetzt. Es ist nun über ein Jahr her, dass ich mit diesem Projekt begonnen habe. Das liegt einerseits daran, dass ich neben dem Entwurf immer nur etwas Zeit entbehren konnte, andererseits steckt hinter dem Projekt aber wirklich eine Menge Arbeit. Da ich vor allem was die elektrotechnische Seite anbelangt wirklich so gut wie kein Know-How besass, musste ich mir das alles mühsam erarbeiten. Zwischendurch kam ich tatsächlich an meine Grenzen. Aufgrund der grossen Anzahl von Fehlerquellen kann die Entwicklung eines solchen Projektes sehr frustrierend sein, diese Erfahrung musste ich

mehrere Male machen. Doch dann kommt auch immer wieder Freude auf, wenn man dann sieht, dass es funktioniert.
Im Nachhinein kann ich deshalb wirklich sagen, dass sich die investierte Zeit gelohnt hat. Einerseits habe ich sehr viel gelernt was *Physical Computing* anbelangt. Denn das war eigentlich auch das Hauptziel, weil ich wirklich weitere ähnliche Projekte in Angriff nehmen möchte. Es ist aber auch ein gutes Gefühl, ein Thema von Null auf 100 zu bearbeiten und am Schluss tatsächlich ein fertiges Produkt in den Händen halten zu können.

Auch sehe ich nun umso mehr das Potential der Arduino-Plattform, denn die Möglichkeiten sind wirklich gewaltig, vor allem, wenn man berücksichtig, wie niedrig die Investitionen sind.

Links zum Thema:

Link	Beschreibung
http://www.arduino.cc	Arduino Seite
http://arduiniana.org/libraries/NewSoftSerial/	NewSoftSerial Library
http://www.arduino.cc/playground/Code/BitMath	Tutorial Bit Mathematics
http://bluetooth-pentest.narod.ru/software/bluetooth_class_of_device-service_generator.html	Bluetooth CoD Generator
http://www.sparkfun.com/datasheets/Wireless/Bluetooth/WT32_Datasheet-1.pdf	Bluetooth Chip WT32 datasheet
http://www.sparkfun.com/datasheets/Wireless/Bluetooth/iWRAP3_User_Guide.pdf	Bluetooth Chip WT32 protocol
http://www.easycalculation.com/binary-converter.php	Binary Konverter
http://www.conrad.ch	Elektronik Komponenten
http://www.pusterla.ch	Elektronik Komponenten
http://www.distrelec.ch	Elektronik Komponenten
http://www.sparkfun.com	Elektronik Komponenten


LED 2
LED 3
LED 4
LED 5
LED 6
LED 7
LED 8
LED 9
LED 10
LED 11
LED 12
reserve
reserve
reserve
R
shift reister
vcc
DS
gnd
ST
SH
LED 1
reserve
Netz 240v
in
out
netzteil 5v
+
-
project bluepower
schematics
yves seiler
august 2009
bten
btres
tx
rx
bluetooth wt12
3.3v
gnd
5v
interface
IR diods
9v
res
a3
a2
a1
a0
13
12
11
10
arduino mini
debugging
pin
transistor
relais
240v