embedded projects
JOURNAL
OPEN SOURCE SOFT-AND HARDWARE PROJECTS

Eine Open-Source Zeitschrift
zum Mitmachen!

# MOUNT SMDrest

[PROJECTS]

- Embedded Linux mit GNUBLIN
- Legal Software Engineering
- USBtin
- Die elektronische Zigarette
- GNUBLIN Installer
- Finite State Machine
- GNUBLIN ADC und PWM


creative commons

firma.embedded-projects.net

DAS HARDWARE FOR YOUR PROJECTS-PORTAL

**Wussten Sie, dass wir eine Firma für kundenspezifische Entwicklungen mit Sitz in Augsburg sind?**

**Wir bieten:**

Hardware, Software, Embedded, Software-Entwicklung, Mikrocontroller, Anwendungsentwicklung, Fachbeiträge/ Literatur, Schaltplan, Webentwicklung, Open-Source, E-Commerce, Platinen-layout, GNU/Linux

**Kommen Sie vorbei!**

embedded projects GmbH
HARDWARE FOR PROJECTS

Holzbachstraße 4, D-86152 Augsburg
Tel +49 (0) 821 279599-0
Fax +49 (0) 821 279599-20
info@embedded-projects.net

Anzeige



# Einleitung

## Ausgabe 02/2012

### Embedded Projects Journal - Ausgabe No. 13

## Einleitung

Trotz schönster Sonnentage haben wir die Ausgabe 13 fertig gebracht.

Viele fleißige Helfer haben wieder mitgearbeitet – Danke!

Wir freuen uns diesmal einen Rechtstext über Urheberrecht, Softwarelizenzen und Open-Source etc. zum ersten Mal im Heft präsentieren zu können. Dieses Thema soll in ein paar weiteren Artikel beleuchtet werden.

Wenn jemand noch spannende Ideen oder Anregungen zu diesem Bereich hat freuen wir uns natürlich über jede E-Mail.

Ziel ist es diese Themen mal mehr aus Bastler oder Entwicklersicht zu betrachten. Ein höchst spannender Versuch.

Noch kurz was in eigener Sache:

Wir stellen bei ElektorLive! 2012 aus. Dies ist ein Seminar- und Ausstellungstag für die Elektronik Entwicklung. Das ganze findet in der Westfalenhalle in Dortmund am 20.Oktober 2012 statt. Selbst haben wir an diesem Tag einen Stand und zusätzlich darf ich ein Seminar über Embedded Linux halten. Wir freuen uns über euer kommen.

http://www.elektor-live.com

Euer embedded projects Team

und Benedikt Sauter

sauter@embedded-projects.net

# wawision.embedded-projects.net

## waWision -
## die Steuerzentrale für Ihre Firma

| | | | |
|---|---|---|---|
| Verwal-tung | Plug & Play | Waren-eingang | Marke-ting |
| FiBu | Produk-tion | automa-tisches Lager | Online-Shops |

### EIN SYSTEM AUS EINER HAND

- keine Installation
- Betriebssystem unabhängig
- Standardhardware Plug & Play
- mitwachsend

### DEMOVERSION

weitere Infos finden Sie auf unserer Internetseite

Holzbachstraße 4, D-86152 Augsburg
Tel +49 (0) 821 279599-0, Fax +49 (0) 821 279599-20
info@embedded-projects.net

embedded projects GmbH
HARDWARE FOR PROJECTS

# GNUBLIN

## Embedded Linux mit Gnublin - Erfahrungen und Projekte

Hubert Högl <Hubert.Hoegl@hs-augsburg.de>

## Erste Erfahrungen mit Gnublin

Dieser Text gibt einige Erfahrungen wieder, die ich als Veranstalter eines Embedded Linux Kurses seit Mitte März mit dem Gnublin Board gemacht habe. Am Schluss beschreibe ich kurz die Versuche, die wir mit dem Gnublin Board gemacht haben.

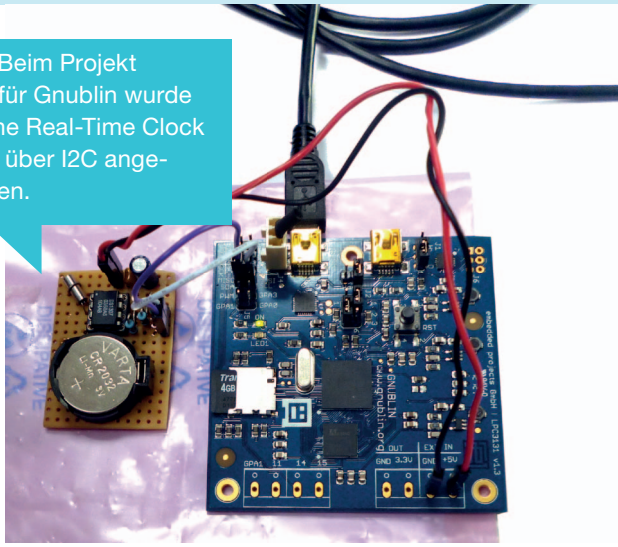


Abb. 1: Beim Projekt Debian für Gnublin wurde auch eine Real-Time Clock DS1307 über I2C angeschlossen.

## Die Hardware

Die Gnublin 1.3 Platine funktioniert meist problemlos. Die Größe und der Funktionsumfang sind gut für eine Vielzahl von Experimenten geeignet. Vor allem die Erweiterbarkeit über die USB OTG Schnittstelle hat sich als sehr praktisch herausgestellt, so dass man gezielt die für ein Experiment notwendigen Peripheriebausteine anschliessen kann, z.B. WLAN, Bluetooth, eine Sound Codec oder ein GPS Modul. Auch die MicroSD Karte als Speicherort für Bootloader, Kernel und Root Filesystem ist eine einfache und praktische Lösung. Durch den geringen Stromverbrauch von unter 100 mA eignet sich das Board auch gut für batteriebetriebene Geräte.

Es gibt trotzdem ein paar Verbesserungspunkte:

- Man wünscht sich gelegentlich an den Schraubanschlüssen eine 5V Spannung für Erweiterungen. Findige Leute haben sich deshalb eine „Spezial-Jumper“ gebaut, der alle drei Pins des „Power-Supply“ Pfostenleiste überbrückt.

- Mit den 8 MByte Hauptspeicher kommt man zwar erstaunlich weit, manche Aufgaben sind aber deswegen nicht möglich. Ein paar Beispiele sind: (1) Der GNU C Compiler läuft mit 8 MByte leider nicht. (2) Manche Linux Distributionen legen am Anfang ein paar Megabyte temporäre Dateien als „tempfs“ ab, das ist bei 8 MByte nicht möglich. (3) Manche einfach erscheinenden Dienstprogramme, z.B. „apt-get“ brauchen so viel Speicher, dass sie in 8 MByte nicht ausführbar sind. (4) Programme aus dem Multimedia-Bereich, z.B. ogg123 oder mpg123 brauchen zum Entkomprimieren viel Hauptspeicher, so dass die erzielbare Datenrate bei 8 MByte deutlich sinkt. (5) Größere Interpreter wie Python passen nur knapp in den freien Speicher, dadurch braucht der Start dieser Interpreter inakzeptabel viel Zeit (5 bis 10 Sekunden). 32 MByte Speicher sind aber bereits in Sicht (Version 1.5). Damit sollten die fünf genannten Beispiele dann kein Problem mehr sein.

  Hinweis: Es gibt jetzt eine 32 MB GNUBLIN Version.

- Fast alle neuen Anwender machen die Erfahrung, dass nach der ersten Sitzung mit dem Board verstärkt Fehler beim Booten auftreten. Das liegt meist daran, dass das „ext2“ Filesystem auf der MicroSD Karte plötzlich Fehler hat, die durch spontanes Abbrechen des Linux Kernels verursacht werden. Seit jeher wird Linux geordnet mit Kommandos wie „shutdown“ oder „halt“ heruntergefahren, nicht einfach durch Ziehen des Steckers abrupt beendet. Hier ist meist der Punkt, an dem man sich mit der Installation von Linux auf dem Entwicklungsrechner anfreunden muss, nur damit schafft man es, die Karte in einem „file system check“ (fsck) wieder zu reparieren. Eine Verbesserung wäre es, die Karte mit dem „ext3“ Filesystem auszustatten, das auf Fehler gutmütiger reagiert. Aber auch damit kann ein fsck nötig sein. Bei wenigen Teilnehmern war die MicroSD Karte gleich am Anfang durch unkontrollierte Änderungen (wahrscheinlich durch wildes Herumklicken) in einem Zustand, dass der Inhalt komplett wiederhergestellt werden musste. Das ist mit dem grafischen „gnublin-installer“ relativ einfach. Es ist ausserdem nützlich, wenn man sich gelegentlich ein „Image“ der Karte macht, z.B. mit dem Werkzeug „dd“, vor allem, wenn man Änderungen an der Karte vorgenommen hat. Dieses Image kann man bei einem späteren „Crash“ zur Rekonstruktion verwenden.

- Eine zweite serielle Schnittstelle (UART) wäre an der Pfostenfeldleiste für Erweiterungen manchmal wünschenswert. Der LPC3131 hat aber leider nur einen UART, der schon für die Konsole verwendet wird.

## Der Kernel- die Kernelversion

Gnublin verwendet den LPC3131 Mikrocontroller von NXP, das ist auch der Grund, warum wir (leider) ziemlich stark auf den Kernel mit der Version 2.6.33 festgelegt sind. Mitarbeiter von NXP haben damals (2010) bei der Anpassung diesen Kernel verwendet und leider bis jetzt keinen Update mehr für eine neuere Kernel-Version gemacht. Für vieles was man mit Gnublin machen kann stört das überhaupt nicht, aber gelegentlich wäre ein neuerer Kernel ganz praktisch, zum Beispiel wenn man:

- neue Peripheriegeräte anschliessen möchte, die es zur Zeit der Version 2.6.33 noch nicht gab. Ein Beispiel: Vor ein paar Tagen habe ich mir einen D-Link Go DWA-123 WiFi Adapter gekauft (USB), der den Treiber rt2800usb verwendet. Diesen Treiber gibt es zwar im Gnublin Kernel, allerdings kennt

sie noch nicht den RT3370 Chip, der in diesem Adapter verwendet wird. Aus diesem Grund kann man beim Gnublin Kernel tendenziell nur ein wenig „reifere“ Peripheriegeräte verwenden, z.B. die Asus WL-167G WiFi Adapter V1 und V2. Damit die Version V3 dieses Adapters funktioniert, musste aber bereits aus dem 2.6.39 Kernel der Treiber staging/rtl8712/ in den Gnublin Kernel 2.6.33 eingebaut werden. Es ist sicher wünschenswert, dass man diese Vorgehensweise nur ganz selten anwendet.

- aus der Weiterentwicklung des Kernels allgemein Nutzen ziehen möchte. Die meisten Neuerungen sind sicher für Gnublin nicht von Bedeutung, allerdings gibt es gelegentlich Verbesserungen in den Frameworks für die Peripherie, z.B. GPIO, PWM und I2C.

Mögliche Auswege aus der Festlegung auf 2.6.33 wären, sich die Mühe zu machen, selber die Anpassung für LPC3131 auf einen neueren Kernel 3.x zu machen (das erfordert aber schon sehr fortgeschrittene Kenntnisse), oder gleich auf einen anderen Controller umzusteigen, der in Linux „Mainline“ unterstützt wird. Ein möglicher Kandidat wäre der LPC3250.

## Den Kernel kompilieren

Mit ein wenig Anleitung schaffen es die Kursteilnehmer relativ schnell, einen neuen Kernel und die dazugehörigen Module zu kompilieren. Auch die Auswahl der gewünschten (einfachen) Kernel-Features durch „make menuconfig“ ist mit ein wenig Übung zu bewältigen. Hier ist vor allem die hohe Zahl und die tiefe Verschachtelung der Menüeinträge am Anfang schwierig. Die Frustration steigt auch durch die vielen Einträge, deren Bedeutung sich nur dem professionellen Kernel-Entwickler erschliessen.

Beim Umgang mit dem Kernel merkt man schnell, dass man sich auch um die *Organisation* von Quelltexten und Daten kümmern muss. Es fängt an mit dem riesigen Quelltextbaum des Kernels, an dessen oberster Stelle auch die Konfigurationsdatei .config sitzt. Nach erfolgreicher Konfiguration/Kompilierung erhält man als Ergebnis einen neuen Kernel arch/arm/boot/zImage und einen Satz an Modulen, die man mit make modules_install INSTALL_MOD_PATH=... an einen geeigneten Ort zusammen mit dem Kernel und der aktuellen Konfigurationsdatei auf dem Entwicklungsrechner ablegt. Von hier muessen die Kerneldateien auf die MicroSD Karte übertragen werden, so dass sie von Gnublin beim nächsten Booten verwendet werden.

Wenn man nun mit mehreren Kernelkonfigurationen Experimente macht und vielleicht sogar auch am Quelltext Änderungen macht, muss man sich bereits einen ordnenden Plan machen, um nicht die Übersicht zu verlieren, welche Features in welcher Version verwendet werden. Auch kann es passieren, dass ein neu komplierter Kernel nicht bootet, so dass man wieder auf den letzten funktionierenden Kernel zurückgehen muss.

Der Anfänger kann sich an diesem Punkt schnell verheddern, so dass nur ein kompletter Neuanfang wieder Ordnung und Übersicht bringt. Hilfreich wäre ein einfaches Kommandozeilenwerkzeug, das den Anfänger bei der Erstellung eines neuen Kernels stärker führt und die Ergebnisse reproduzierbar speichert.

Stark verbesserungsfähig ist die Art und Weise, wie der Kernelcode zur Zeit auf gnublin.org verwaltet wird. Es ist momentan eine Mischung aus Git Repository und klassischen Patches, bei deren Anwendung auch viel schiefgehen kann. Als provisorische Lösung habe ich ein eigenes git Repository mit dem Branch „gnublin“ für den Gnublin Kernel eingerichtet, siehe

http://elk.informatik.fh-augsburg.de/hhwiki/EmbeddedLinux2012

Nachdem man Änderungen am Kernel gemacht hat, stellt sich schnell die Frage, ob alle gewünschten Kernel-Features noch funktionieren, auch die, bei denen man gar keine Änderungen gemacht hat. Eigentlich müsste man nun in der Gnublin Konsole viele Punkte überprüfen, z.B. ob die gewünschten Netzwerktreiber vorhanden sind, ob das sysfs vorhanden ist, ob sich die PWM und ADC Treiber laden lassen, ob ext3 im Kernel aktiviert ist und so weiter. Hier wäre ein kleines Testsystem von Vorteil, das Konsoleneingaben automatisiert ablaufen lassen kann, z.B. mit dem Werkzeug „expect“.

## Das Root Filesystem

Das momentan mit Gnublin ausgelieferte Root Filesystem basiert auf dem ELDK 5.0 (siehe http://www.denx.de). Mittlerweile gibt es die Version 5.2, das zwei für Gnublin schöne Verbesserungen gegenüber 5.0 hat:

- Es gibt kleinere Varianten, die nur noch die wirklich gebrauchten Programme dabei haben, zum Beispiel brauchen wir gar keine grafischen X-Window Programme. Somit schrumpft das Root Filesystem von derzeit über 600 MByte (5.0) auf 100 bis 200 MByte. Das wirkt sich deutlich aus auf alle Operationen, die mit Download, Kopieren, Sichern und Beschreiben des Root Filesystems zu tun haben.

- Die zweite Verbesserung betrifft das Paketmanagement. Die Version 5.2 kommt mit dem ipkg Paketmangager, der Binärpakete aus dem Netz holen und installieren kann. Ein eigenes Paketrepository wäre für Gnublin ideal, da sich fast jeder, der intensiv mit dem Board arbeitet, eigene Zusätze für sein Root Filesystem zusammenbaut, diese aber oft nur auf der lokalen MicroSD Karte gespeichert werden. Diese Zusätze kommen oft aus dem Debian/ARM („armel“) Projekt, man kann die vorkompilierten Programme direkt bei Gnublin übernehmen, oder sie kommen durch selber crosskompilierte Programme. Die Reproduzierbarkeit von Änderungen am Root Filesystem ist also, ähnlich wie schon im Abschnitt über den Kernel angesprochen, mangehaft. Die Anpassung von ELDK 5.2 für Gnublin wird also immer dringender.

- Alternativ gibt es mittlerweile ein für Gnublin angepassstes Debian 6 (siehe Projekte weiter unten). Damit hat man dann den „dpkg“ Paketmanager, der allerdings nur einzelne .deb Pakete installieren kann. Das Holen aus dem Netz und die Auflösung von Abhängigkeiten muss man manuell erledigen. Mit den „apt“ Utilities könnte man das zwar komfortabel erledigen, sie laufen aber leider nicht mit 8 MByte Speicher (mit 32 MByte vielleicht schon).

## Arbeiten auf dem Entwicklungsrechner

Ein deutlicher Wohlfühlfaktor im Umgang mit dem Gnublin Board ist die Fähigkeit, sich auf dem Entwicklungsrechner unter Linux zurechtzufinden. Einige Aufgaben, für die man diesen Rechner verwendet, sind:

1) Betrachten der Gnublin Konsole (Terminalemulation mit picocom o.ä.)

2) Austausch von Dateien mit Gnublin (XYZModem oder scp)

3) File System Check der MicroSD Karte

4) Crosskompilieren von Programmen für Gnublin

5) Test von Web-Anwendungen auf Gnublin mit einem Browser

6) Arbeiten mit dem Kernel
   - Kernelquelltext ändern
   - Quelltext hinzufügen, z.B. ein neues Modul
   - Suchen von Quelltextteilen
   - Änderungen herausfinden (diff)
   - Konfiguieren
   - Kompilieren
   - Kernel u. Module auf MicroSD Karte übertragen
   - Versionsverwaltung mit git

7) Root Filesystem für Gnublin
   - Pakete hinzufügen/löschen
   - ELDK kompilieren
   - Andere Distributionen für Gnublin anpassen

Viele Anfänger sind geneigt, ausschliesslich mit grafischen „Klick-Werkzeugen“ zu arbeiten. Ist das bei 1, 2 und 3 noch in Ordnung, gibt es bei 4, 5, 6 und 7 damit ernste Probleme, da alle verwendeten Programme auf der Kommandozeile angewendet werden. Es sieht frustrierend aus, wenn Anfänger lediglich mit einem grafischen Dateimanager ausgerüstet im Kernel Verzeichnisbaum verloren herumklicken. Um produktiv arbeiten zu können, lohnt es sich daher, zunächst zu lernen, wie man auf einem UNIX-ähnlichen Rechner effektiv auf der Kommandozeile arbeitet. Einige freie Literaturangaben dazu habe ich unter [1] gesammelt.

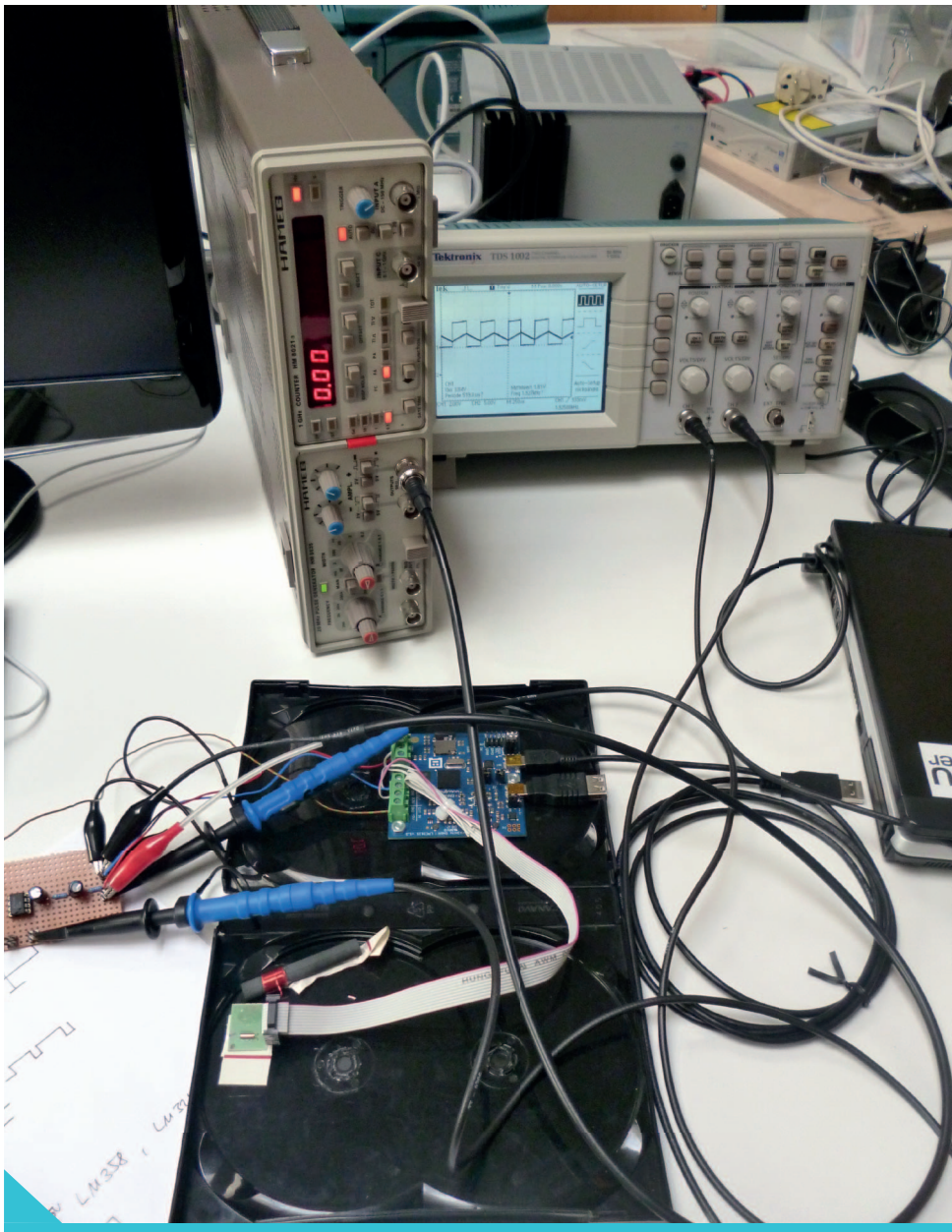



Abb. 2: Auf der linken Seite sieht man das Tiefpassfilter aus zwei RC-Gliedern. Darüber kann man Audio-Signale bis etwa 1 kHz aus einem per Software erzeugten PWM Signal rekonstruieren. Unten links sieht man auch den DCF77 Empfänger der an einen GPIO Pin angeschlossen ist.

Die Dokumentation der Gruppe findet man ebenfalls hier: http://elk.informatik.fh-augsburg.de/hhwiki/EL12-Beschleunigung

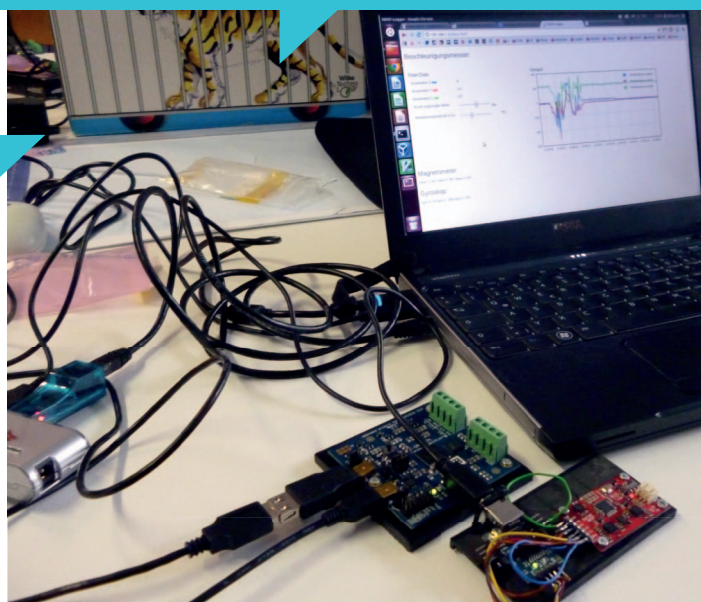


Abb. 3: Auf dem Bild sieht man das „9 DoF“ Beschleunigungssensor-Board von Sparkfun, das über einen USB-UART an Gnublin angeschlossen ist. Auf dem Notebook sieht man die Web-Schnittstelle des Sensors. Zwischen Gnublinund dem Notebook besteht eine Ethernet-Verbindung.

## Die Projekte

1. Debian als Alternative zum ELDK

http://forum.gnublin.org/viewtopic.php?f=4&t=45

2. I2C Real-Time Clock DS1307 zum Setzen der Systemzeit (Teil von 1.)

3. DCF77 Empfänger an GPIO Pin zum Setzen der Systemzeit

4. Sprachausgabe mit Soft-PWM + Tiefpassfilter an GPIO Pin

In einem Kernelmodul wird eine PWM Frequenz von etwa 10 kHz erzeugt. Über eine Gerätedatei kann das PWM Puls- zu Pausenverhältnis geändert werden. Sprachsignale können aus einer Binärdatei mit 8-Bit Amplitudenwerten direkt durch „cat“ auf die Gerätedatei ausgegeben. Die maximale Sprachfrequenz ist etwa 1 kHz.

5. Treiber für NEC Infrarot Protokoll

6. Mehrere LM75 Sensoren (I2C) an Gnublin

7. IMU 9 DoF (Sparkfun) an Gnublin mit interaktivem Web-Interface

http://elk.informatik.fh-augsburg.de/hhwiki/EL12-Beschleunigung

8. Feuchte- und Temperatursensor SHT15 (I2C) an Gnublin

Mit Hilfe des Sensors SHT15 (I2C) von Sensirion soll ein kombiniertes Feuchte- und Temperaturmessgerät entstehen.

9. 1-Wire

Mit Hilfe eines DS2482 I2C auf 1-Wire Adapters soll Gnublin mit dem 1-Wire Bus sprechen.

10. GPS Modul GlobalTop GMS-D1

Das GPS Modul aus dem Shop von Embedded Projects GmbH kostet nur etwa 15 Euro. Man kann es über einen UART oder über die Signale D+ und D- an USB anschliessen. Eine genauere Beschreibung der Projekte wird es im Gnublin Wiki ab Mitte Juli geben.

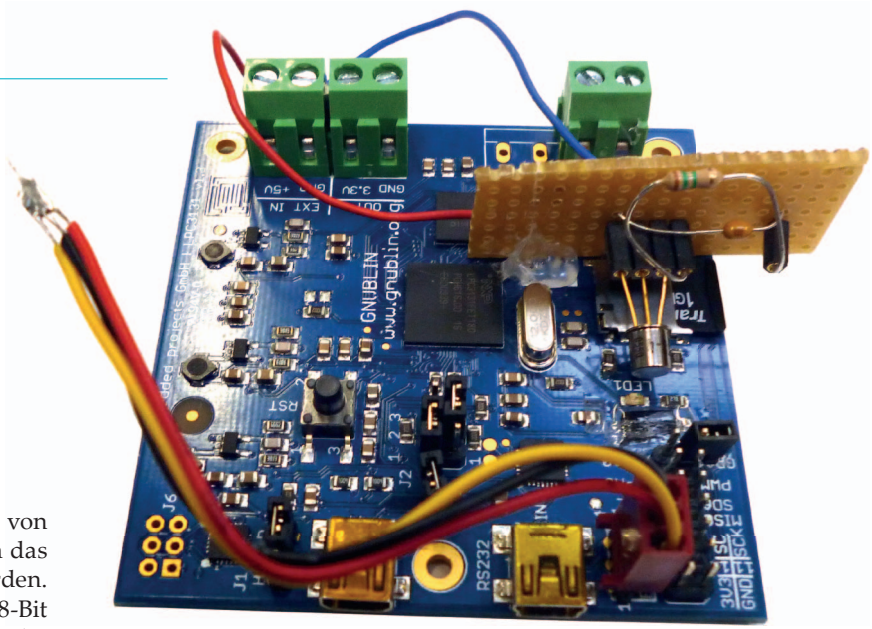



Abb. 4: Die grüne LED auf dem Board beleuchtet einen Fototransistor, der mit seinem Ausgang wieder an einen GPIO Pin angeschlossen ist. Interessant wird sein, ob man daraus mit Hilfe eines Kernel-Moduls einen Oszillator machen kann, der möglichst schnell schwingt. Daraus wird man auch Rückschlüsse ziehen können, wie lang die Interrupt-Latenzzeit ist.

Die Gruppe die den Feuchtigkeitssensor anschliesst hat noch kein Bild. Das Sensor Board ist das von Watterott: http://www.watterott.com/de/Feuchtigkeits-und-Temperatur-Sensor-SHT15-Breakout
Hier findet man eine kurze Beschreibung:
http://elk.informatik.fh-augsburg.de/hhwiki/EW12-Feuchtesensor
Ausserdem experimentiert diese Gruppe mit einem Helligkeitssensor mit I2C Anschluss, siehe
http://www.watterott.com/de/TSL2561-Lichtsensor
Insgesamt soll es eine Wetterstation werden.

## Literatur

[1] H. Högl, Freie Literatur zu GNU/Linux und UNIX

http://elk.informatik.fh-augsburg.de/hhweb/doc/linuxbib.html

Hinweis Kernel 3.3 + Debian:
Mittlerweile ist es möglich, den Kernel in der Version 3.3 zu verwenden. Zudem kann man anstelle des ELDK ein Debian System einsetzen.

# Legal Software Engineering

## Urheberrecht, Patentrecht & Lizenzmodelle

Bernd Suchomski, LL.M [1]

Die großen Softwarefirmen wie Red Hat und Google haben es bereits vorgemacht: Neben der Entwicklung einer intelligenten Software beruht der geschäftliche Erfolg einer Software-Schmiede auch auf dem rechtssicheren Umgang mit dem Produkt. Die wichtigsten rechtlichen Aspekte der Software-Entwicklung bewegen sich regelmäßig innerhalb von drei Eckpunkten:

1. Die Frage, ob und inwieweit ein Code überhaupt schutzfähig ist,

2. unter welchen Bedingungen ein schutzfähiger Code erhältlich ist (Lizenz) und

3. wie sich diese Lizenzbedingungen mit der eigenen Software-Strategie vereinbaren lassen (Lizenzkompatibilität).

Hinter all diesen Punkten befindet sich eine Vielzahl von rechtlichen Herausforderungen bei der Sachverhaltsrecherche und -analyse. Wer sie meistert, dem winkt der geschäftliche Erfolg als Ausgleich für all die Mühe und Aufwendung, die bisher unternommen wurden, um das eigene Business aufzubauen.

Durch die richtige Herangehensweise eröffnet sich die Möglichkeit, eine Idee schneller und besser umzusetzen als die Konkurrenz. So können anschließend die verdiente Reputation und die nötigen Gelder erwirtschaftet werden, um die eigene Erfolgsgeschichte fortzusetzen. Andererseits drohen erhebliche Einbußen und Schadensersatzprozesse, wenn man mit den rechtlichen Aspekten der Softwareentwicklung zu nachlässig umgeht. So können vor allem Mitentwickler Rechte an der Software geltend machen und dadurch den Vertrieb erheblich stören.

## 1. Der rechtliche Schutz von Software

### a. Urheberrecht

Computerprogramme sind sowohl in Form des Objekt- als auch in Form des Quellcodes nach den Sonderregelungen der §§ 69a ff. UrhG geschützt. Der Schutz entsteht bereits in allen 165 Ländern der Berner Übereinkunft (RBÜ) automatisch mit der Formulierung des Codes. Ein Anmelde- oder Registrierungsverfahren wie z.B. beim Patent ist nicht mehr nötig.

Die inhaltlichen Anforderungen, die ein Softwarecode für den Schutz erfüllen muss, sind daneben weitaus geringer als bei Patenten und den übrigen urheberrechtlich geschützten Werken. Nicht schutzfähig sind nur einfachste Trivialprogramme (Stichwort: „Hello World“) und die abstrakte Idee, die hinter dem Algorithmus steht. [2]

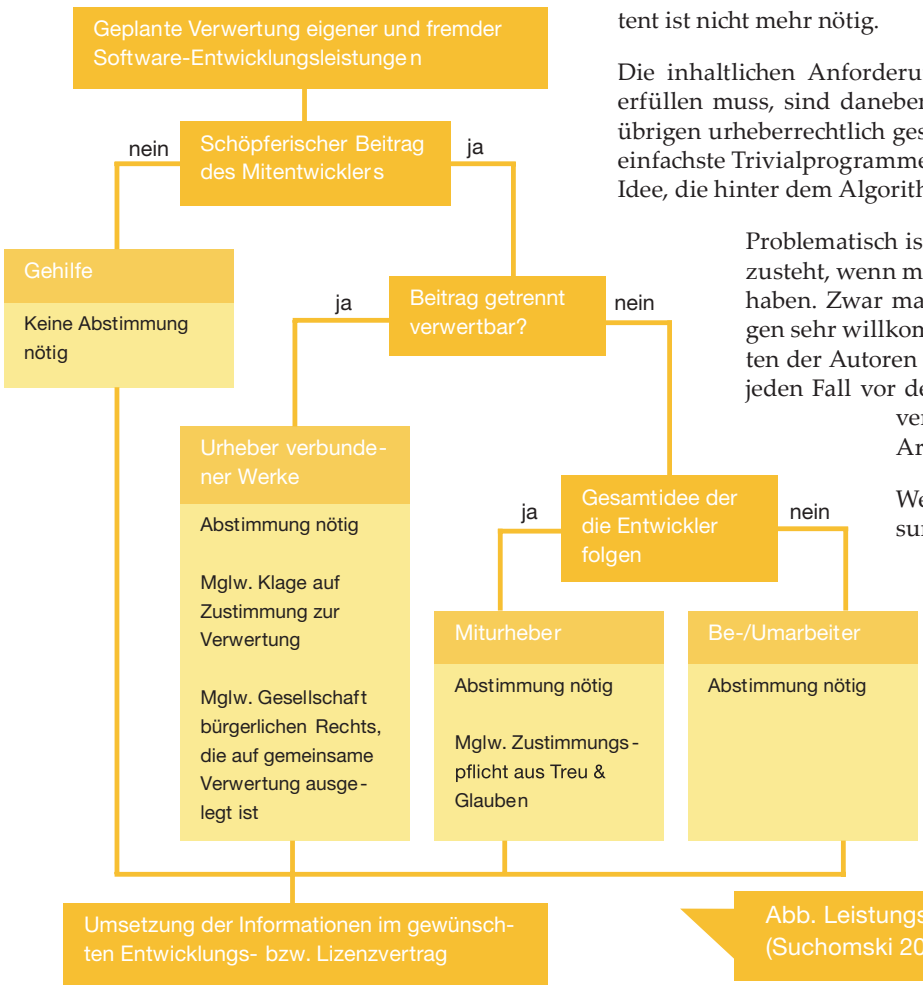



Abb. Leistungsverwertung (Suchomski 2012)

Problematisch ist allerdings die Frage, wem das Urheberrecht zusteht, wenn mehrere Entwickler an einem Projekt gearbeitet haben. Zwar mag die Hilfe von Freunden und Studienkollegen sehr willkommen sein. Sie ist jedoch regelmäßig mit Rechten der Autoren am Endprodukt verbunden. Diese sollten auf jeden Fall vor dem Publishing oder dem Gang zum Investor vertraglich gebündelt werden, damit die harte Arbeit Früchte trägt.

Wenn die Mitentwickler nicht gerade auf Anweisungen, d.h. als Gehilfen tätig geworden sind und keinen schöpferischen Beitrag geleistet haben, ist damit zu rechnen, dass ihnen ein Urheberrecht am Endprodukt zusteht. Soweit das aber der Fall ist, kann die eigene Entwicklung nur noch ungehindert vertrieben werden, wenn sie von der Arbeit Dritter getrennt verwertbar ist. In jedem Fall bleibt damit der wertvolle Beitrag ungenutzt und man muss stattdessen zu einer älteren Vorversion greifen – es sei denn, es lassen sich Hinweise auf eine Zweckgemeinschaft oder eine Bindung der Helfer nach den Grundsätzen von Treu und Glauben finden.

#### b. Patentrecht

Softwarepatente können in Form von Verfahrens- oder Erzeugnispatenten geschützt werden, § 9 PatG. Eine dafür nötige Erfindung muss stets eine neuheitliche Lösung für ein technisches Problem darstellen und gewerblich anwendbar sein, vgl. § 1 PatG.

Mit einem Verfahrenspatent wird ein neues Computerprogramm als Ersatz für die Hardware beansprucht [3] - man denke an Smartphone-Touchscreens, welche die analoge Handy-Tastatur ersetzen. Mit Erzeugnispatenten ist es dagegen möglich, für eine feste Hardware-Software-Kombination[4] Schutz zu erlangen. Auch fest installierte Programme (sog. Embedded-Systeme) fallen unter diese Kategorie. [5] Schließlich können damit auch Patente für Computerprogrammprodukte erteilt werden. Das sind Programme auf Speichermedien, die sich wiederum auf ein patentiertes Verfahren beziehen. [6]

Das Patentrecht besteht dabei unabhängig zum Urheberrecht, vgl. § 69g Abs. 1 UrhG. [7] Doch auch hier gibt es die Möglichkeit, dass Mitentwickler als Miterfinder zählen könnten. In einem solchen Fall ist es wiederum nötig, sich mit allen Miterfindern hinsichtlich der Verwertung eines Patents abzustimmen, vgl. § 6 S. 2 PatG.

Doch der Griff nach Softwarepatenten ist mit Risiken verbunden. Einerseits sind Umfang und Kosten der Patentrecherche abhängig von der Komplexität des Programms. Selbst danach kann die Patentanmeldung vom Deutschen Patent- und Markenamt (DPMA) zurückgewiesen oder im Rahmen eines Einspruchsverfahrens zu Fall gebracht werden. Weiterhin existiert die Möglichkeit, dass ein bereits erteiltes Patent durch einen Nichtigkeitsprozess erfolgreich angegriffen wird.

Schließlich sind Softwarepatente vor allem der Kritik von freien Entwicklern ausgesetzt, da diese die Software-Innovationsmöglichkeiten gefährdet sehen. Denn selbst wenn eine Patentschrift eine Anleitung zur Umsetzung einer technischen Lösungsidee liefern mag, so liegt ihr nicht notwendigerweise ein Quellcode bei, mit dem die Entwickler ihre Software vergleichen könnten. Ob der eigene Quellcode ein Patent verletzt oder nicht, ist oftmals schwer zu erkennen. In solchen Fällen wird selbst ein Gericht zusätzliche Gutachten einholen müssen. Freie Entwickler organisieren sich daher oft in Internet-Registrierungssystemen um Softwarepatenten schon im Entstehen oder wenigstens nachträglich im Rahmen eines Nichtigkeitsverfahrens zu begegnen. [8]

## 2. Bezug und Verwendung geschützter Codes

Urheberrechtlich oder patentrechtlich geschützter Code kann danach unter bestimmten Konditionen (Lizenzen) der Öffentlichkeit bzw. einem ausgewählten Kundenkreis angeboten werden. Ebenso kann ein solcher Softwarecode zur Weiterverarbeitung bezogen werden, um sich einen Teil der Entwicklungsarbeit zu sparen. Hier einige Beispiele:

#### a. Kommerzielle Lizenzen

Kommerzielle Lizenzen bilden den klassischen Kern des Software-Vertriebs. Mit ihnen wird das Programm gegen Entgelt auf Dauer oder auf Zeit überlassen. Der Software-Einsatz kann beim Kunden hinsichtlich der Nutzung im Netzwerk oder auf verschiedenen Computern, bezüglich der gewerblichen Ausrichtung der Verwendung und im Hinblick auf Vervielfältigungs- und Verbreitungshandlungen eingeschränkt sein. Hinzu kommen sog. Audit-Klauseln, die dem Urheber die Kontrolle darüber ermöglichen sollen, ob die gemachten Vorgaben auch durch den Lizenznehmer eingehalten werden. Soweit sie nicht individuell mit den Abnehmern ausgehandelt werden, müssen sie den Grundsätzen der Allgemeinen Geschäftsbedingungen nach §§ 305 ff. BGB entsprechen.

#### b. Open Source

Open Source Software (OSS) stellt ein Gegenmodell zu kommerziellen Lizenzen dar. Solche Lizenzen stellen Quell- und Objektcode frei von jeglichen Lizenzgebühren und zu jedwedem Zweck zur Verfügung. Der Code darf kopiert, bearbeitet und verbreitet werden. Die Benutzung ist trotzdem nicht frei von Auflagen. Grundsätzlich sind bei der Weiterverbreitung den Abnehmern die gleichen Rechte an der ursprünglichen Software zuzugestehen, die man selbst in Anspruch nehmen konnte.

Wurden bestimmte Veränderungen am Code vorgenommen oder Code-Fragmente in eigene Programme überführt, so kann eine OSS-Lizenz verschiedene Folgen anordnen. Möglicherweise ist dann ein Weitervertrieb des veränderten Codes nur getrennt – d.h. Form von Originalprogramm und Patch-Dateien – erlaubt. Falls eine OSS-Lizenz einen Copyleft-Effekt vorsieht, ist es auch möglich, dass das Endprodukt ebenfalls nur noch unter den Bedingungen der OSS-Lizenz vertrieben werden darf. Ob der eigene Entwicklungsbeitrag „infiziert“ wurde, kann anhand der rechtlichen Begutachtung der Programmierschritte und der Systemarchitektur nachvollzogen werden.

Open Source kann aber auch kommerziell vorteilhaft genutzt werden. Der Urheber bzw. Erfinder kann nach dem sogenannten Dual-Licensing-Modell seine Software sowohl unter einer OSS-Lizenz als auch unter einer kommerziellen Lizenz anbieten. Die Verfügbarkeit unter einer unkommerziellen OSS-Lizenz hat den Vorteil, dass sich eine Software schneller verbreitet und somit ein Markt-Standard geschaffen wird. Denjenigen Weiterentwicklern, die anschließend auf diesem Standard aufsetzen aber gleichzeitig kommerziell agieren möchten, kann der Urheber bzw. Erfinder anschließend seine kommerzielle Lizenz verkaufen.

Selbst wenn Weiterentwickler ihren Quellcode lizenzgebührenfrei anbieten müssen, so gilt dies nicht für das Angebot zusätzlicher Dienstleistungen oder Hardware. [9] Beispiele wie Red Hat, SuSE oder Google-Android zeigen, dass es auch möglich ist, sich auf den Nebenmärkten erfolgreich zu bewegen. Die Software kann daneben schneller und effizienter den Kundenwünschen und Hardware-spezifikationen angepasst werden. Hersteller von Embedded-Systemen profitieren von Open Source Software, weil sie eine zeitnahe und störungsfreie

Gerätenutzung ermöglicht. [10] Unabhängig von dem Vertrieb von OSS-Produkten kann ein nicht unwesentlicher Umsatz mit dem Angebot von Support, Beratung und Weiterbildung erzielt werden. [11] Allerdings wurde auch in der Vergangenheit der Versuch unternommen, die freien Weiterentwickler durch das Angebot zusätzlicher Dienstleistungen zu ködern und ihnen in der Folge den Vertrieb von Open Source Software zu erschweren. So hat Oracle die Zertifizierung freier Java-Entwicklungen innerhalb seines Technology Compatibility Kits (TCK) zwar gestattet. Dessen Lizenz sah jedoch teilweise vor, dass die danach zertifizierten Produkte nicht mehr auf Embedded-Systemen zu Einsatz kommen durften, [12] sog. Field of Use Restriction oder Java-Trap. [13]

#### c. Public Domain Software

Das US-amerikanische Recht hat den Begriff der Public Domain Software geprägt. Danach kann der Urheber auf alle Rechte an seiner Software verzichten. Sein Programm kann somit von jeder Person völlig ohne Einschränkungen verwendet werden. [14] In Deutschland gibt es keine vergleichbare Möglichkeit, sich seiner Urheberrechte zu entledigen. Zwar gibt es das rechtliche Phänomen der gemeinfreien Werke. Allerdings entsteht dieser Zustand erst mit Ablauf des urheberrechtlichen Schutzes, d.h. 70 Jahre nachdem der Autor verstorben ist. [15]

#### d. Freeware und Shareware

Bei einer Freeware-Lizenz wird die Software zunächst lizenzgebührenfrei im Binärcode angeboten. Allerdings schränkt eine solche Lizenz Veränderungen an der Software oder deren weiteren Vertrieb ein. Der Quellcode ist dabei ebenfalls nicht erhältlich. [16] Eine Shareware-Lizenz unterscheidet sich davon oftmals nur durch eine zeitliche Begrenzung der Nutzungsrechte. Mit ihnen soll dem Benutzer eine Demonstration des Programms vorgeführt werden, um ihn zum Kauf zu animieren. [17]

#### e. Shared Source Software

Lizenzen nach dem Modell der Shared Source Software gelten mittlerweile als überholt. Sie sind der Open Source Software zwar ähnlich, weil sie den Quelltext zur Verfügung stellen und dessen Bearbeitung erlauben. Allerdings waren solche Befugnisse nur zu Testzwecken erlaubt. [18] Microsoft verteilte solche Lizenzen um sein Betriebssystem Windows CE weiterentwickeln zu lassen. Wollten andere diese Weiterentwicklungen aber verkaufen, bedurfte es einer zusätzlichen Microsoft-Lizenz, die entgeltpflichtig war. [19]

## 3. Lizenzkompatibilität

Hat man danach die Urheberschaft und die Bedingungen der zur Verfügung stehenden Softwarelizenzen geklärt, gilt es diese Elemente zu vereinen, um das Projekt erfolgreich zum Abschluss zu bringen. Einige Lizenzen gestatten dabei, dass der übergebene Quellcode be- oder umgearbeitet werden darf.

Ausgehend von der verfolgten Programm-Architektur muss zunächst geprüft werden, ob überhaupt auf den Quellcode eines Programmes Bezug genommen wird oder ob das Programm nur zu Hilfszwecken, z.B. zum Compiling, ausgeführt wird.

Sollte in den Quellcode derart eingegriffen werden, dass Teile von ihm mit Teilen anderer Quellcodes verbunden werden oder wird ein Programm zur Interaktion mit einem lizenzierten Produkt geschrieben, so sollte untersucht werden, ob die rechtliche Zukunft des neu entstandenen Codes von einer oder mehreren daran beteiligten Lizenzen geregelt wird.

Falls mehrere Lizenzen der beteiligten Codes das Ergebnis einer Be- oder Umarbeitung regeln, ist es ratsam zu klären, ob die Definition des Ergebnisses bestimmt genug ist, um einer AGB-Prüfung standzuhalten. Sollten danach die Bestimmungen der Lizenzen hinsichtlich des entstandenen Ergebnisses identisch sein, steht einer Verwertung der Software nichts mehr entgegen. Weichen sie voneinander ab, besteht die Möglichkeit, dass man eine Lizenzvertrags- oder gar eine Urheberrechtsverletzung beim weiteren Umgang mit dem Entwicklungsergebnis begeht.

Gerade bei Open Source dreht sich dieser Aspekt um die Frage, ob Copyleft-Lizenzen wie die GPLv2 oder GPLv3 an dem Entwicklungsergebnis beteiligt waren. Das Entwicklungsergebnis wird dabei meist als „work based on the program“[20] oder „abgeleitetes Werk“ beschrieben – ein Zustand, der regelmäßig nur über die konkrete Verwendung des Programms bestimmt werden kann. Dieser kann mit dem Aspekt der anderweitigen Verwertbarkeit einer Systemkomponente einhergehen. Damit verschiebt sich die Diskussion auf die technischen Aspekte der einzelnen Komponentengruppen. Für Plugins und Shared Libraries gelten danach andere Standards als bei Middleware und Modul-Verknüpfungen via User Interfaces. Besonders problematisch ist darüber hinaus die Implementierung von Systemmodulen über den Kernel. Um die Eigenentwicklung nicht dem Regime einer bereits existierenden OSS-Lizenz unterwerfen zu müssen, ist es u.a. ratsam, peinlich darauf zu achten, dass die Eigenentwicklung nicht ausschließlich von den verwendeten OSS-Produkten abhängig und stattdessen dynamisch implementiert ist.

## 4. Zusammenfassung

Die heutige Software-Lizenzierung bietet den Entwicklern zahlreiche Verbesserungsmöglichkeiten für das eigene Produkt oder das eigene Unternehmen. Um das eigene Softwareprodukt selbst überhaupt wirksam verwerten zu können, ist es notwendig, sich zuvor der Zustimmung aller Miturheber und Miterfinder zu vergewissern. Flankierend dazu ist es ratsam, seine freien Mitentwickler vertraglich anzuhalten, den gemachten Entwicklungsbeitrag exklusiv zur Verfügung zu stellen. Über einen etwaigen Anteil an den Erlösen können individuelle Vereinbarungen getroffen werden.

Soweit bereits vorhandene Programme bezogen werden, sollten zuvor deren Lizenzbedingungen dahingehend überprüft werden, ob das angestrebte Projekt auch rechtlich realisierbar ist. Spätestens bei Open Source gilt es, sich darüber hinaus der Folgen für die Rechte an der entstandenen Entwicklung bewusst zu werden. In diesem Fall muss eine strategische Entscheidung getroffen werden: Entweder das Dual Licensing nutzen und das Geschäftsmodell effektiv auf Open Source ausrichten oder die eigene Entwicklung so geschickt gestalten, dass man den rechtlichen Copyleft-Effekt umgeht und die Software auch selbst kommerziell verwerten kann.

## Literatur


[1] Der Autor ist Rechtsanwalt und auf das Software- und Contentrecht ausgerichtet. Der Text ist z.T. seiner Veröffentlichung in der MIR-Schriftenreihe entlehnt (Proprietäres Patentrecht beim Einsatz von Open Source Software, tgramedia, Bonn 2011)
E-Mail: suchomski@sonntag-partner.de

[2] Jaeger/Metzger, Open Source Software, 3. Auflage, München 2011, Rn. 120; Spindler, Rechtsfragen bei Open Source, Köln 2004, S. 21 f., Rn. C 2; sehr detailliert mit geschichtlichem Hintergrund: Schiffner, Open Source Software, München 2003, S. 41 ff., m. w. N.

[3] Vgl. Sedlmaier, Patentierbarkeit von Computerprogrammen und ihre Folgeprobleme, München 2004, S. 144, 146; Ohly, CR 2001, 809, 815.

[4] Spindler, Rechtsfragen bei open source, Köln 2003, S. 248, Rn. F 50.

[5] Vgl. Sedlmaier, S. 199, 144, Fn. 17, 18 verweist auf EPA-Beschwerde-Entscheidung 25.07.1997 – T 410/96: „data-processing system comprising means adapted to carry out [...]“

[6] Vgl. Sedlmaier, S. 144 f., 151 ff.

[7] Dogmatisch ergänzend: Sedlmaier, S. 23 ff.

[8] Z. B. System der FFII: http://eupat.ffii.org/purci/index.de.html], erwähnt bei: Jaeger/Metzger, Rn. 288; Sedlmaier, S. 96 a. E.

[9] Dazu: Jaeger/Metzger, Rn. 115.

[10] Jaeger/Metzger, Rn. 20.

[11] Jaeger/Metzger, Rn. 21.

[12] TCK Stand Alone License, Ziffern 1.6, 1.12 und 2.1 (b) (v), Exhibit A II: http://jcp.org/aboutJava/communityprocess/licenses/STANDaloneTCK7Final.docx.

[13] Vgl. Stallman, Free Software – Free Society, FSF 2010, S. 215 ff.

[14] Jaeger/Metzger, Rn. 8; Spindler, S. 17 f., Rn. B 13.

[15] Jaeger/Metzger, Rn. 8; Spindler, S. 17 f., Rn. 13, Fn. 28.

[16] Jaeger/Metzger, Rn. 9; Spindler, S. 18, Rn. 14.

[17] Jaeger/Metzger, Rn. 10.

[18] Spindler, S. 18, Rn. 15.

[19] Jaeger/Metzger, Rn. 11.

[20] Ziffer 2 Abs. 1 GPLv2; Ziffer 0 Abs. 5 GPLv3.


## Weitere Informationen zum Thema Lizenzierung

[1] http://www.sonntag-partner.de/

[2] http://www.groklaw.net/

[3] http://opensource.org/osd.html/

[4] http://www.gnu.org/licenses/licenses

[5] http://gpl-violations.org/


Anzeige

Sicherheit?

Unser neues Open-Source-Projekt

www.picosafe.de

# USBtin

## Ein einfacher USB-nach-CAN Adapter

Thomas Fischl <thomas@fischl.de>

## Einleitung

Ursprünglich für Fahrzeuge entwickelt, findet sich der CAN-Bus mittlerweile in vielen anderen Bereichen wieder. Die Robustheit, das definierte Zugriffsverhalten, die vorgegebene Framestruktur und nicht zuletzt die Verfügbarkeit von Mikrocontrollern mit integrierter CAN-Einheit, machen den Einsatz auch für eigene Projekte interessant. Um die Eigenbauten zu testen, stellt ein PC zum Überwachen und Absetzen von CAN-Nachrichten eine unverzichtbare Hilfe dar. Die Brücke zwischen Rechner und dem CAN-Bus schafft der hier vorgestellte USBtin.

## CAN-Bus

CAN steht für Controller Area Network und ist ein Bussystem, welches ursprünglich für die Vernetzung von Fahrzeugkomponenten konzipiert wurde. Es handelt sich um ein serielles Bussystem, welches Informationen differenziell über zwei Datenleitungen (CAN_HIGH, CAN_LOW) übermittelt. Häufig wird auch eine dritte Leitung, eine Erdungsleitung, mitgeführt.

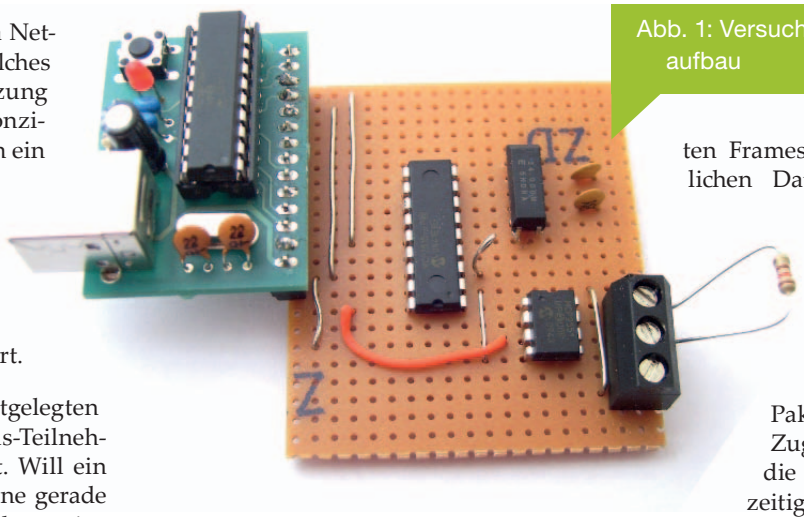


Abb. 1: Versuchsaufbau

Beim CAN-Bus gibt es keine festgelegten Master/Slave-Geräte. Alle Bus-Teilnehmer horchen auf dem Bus mit. Will ein Gerät senden, wartet es, bis eine gerade laufende Übertragung abgeschlossen ist, und versucht dann seine Daten zu übermitteln. Auf dem Bus gibt es zwei mögliche Signalpegel: den rezessiven und den dominanten Pegel. Wie der Name bereits vermuten lässt, überschreibt der dominante Pegel den rezessiven. Wollen gleichzeitig mehrere Bus-Teilnehmer senden, entsteht dadurch eine Priorisierung. Derjenige, der in der Übertragung als erstes einen dominanten Pegel auf den Bus setzt, gewinnt und sendet weiter. Der „Verlierer" erkennt dies und bricht die Übermittlung ab. Erst wenn der Bus wieder frei ist, wird ein erneuter Versuch unternommen. Daten werden innerhalb von sogenannten Frames übertragen. Die eigentlichen Daten werden dabei von Steuerinformationen umrahmt. Zu Beginn eines Frames wird eine Message-ID gesendet. Diese erlaubt den Busteilnehmern zu entscheiden, ob das Paket für sie interessant ist. Zugleich priorisiert diese ID die Nachrichten – bei gleichzeitigem Zugriff gewinnt die CAN-Nachricht mit dem ersten dominaten Bit. Die ID ist entweder 11Bit (Standard) oder 29Bit (Extended) lang. Nach der ID folgen Flags, die Länge des Datenfeldes und schließlich die eigentlichen Nutzdaten. Hiervon können je Frame bis zu acht gesendet werden.

## Schaltung

Die Schaltung (siehe Abb. 2) besteht im Wesentlichen aus dem Mikrocontroller, dem CAN-Controller und dem CAN-Transceiver. Bei dem Mikrocontroller handelt es sich um einen Microchip PIC18F14K50 mit integriertem USB-Client-Modul. Da dieser Chip keine CAN-Peripherie integriert hat, wird die CAN-Funktionalität über einen externen Controller zur Verfügung gestellt: dem MCP2515. Dieser ist über den SPI-Bus an den Mikrocontroller angebunden. Zusätzlich sind drei Steuerleitungen verdrahtet, über die Ereignisse an den PIC gemeldet werden. Um die digitalen Signale in ein CANgerechtes differenzielles Signal zu wandeln, ist zusätzlich ein Transceiver notwendig. Hierfür wurde der MCP2551 gewählt, der Übertragungsraten von bis zu 1MBaud und bis zu 112 Busteilnehmer erlaubt. Alle drei Bausteine sind im DIP-Gehäuse verfügbar und damit mit Bastlerausrüstung ohne Probleme zu löten. Die Spannungsversorgung erfolgt direkt über die +5V-USB-Spannung. Eine galvanische Trennung zwischen dem PC und dem CAN-Bus besteht damit keine! Um den Betriebszustand optisch zu signalisieren, wurde eine LED vorgesehen, die direkt vom PIC gesteuert wird. Der Jumper JP1 aktiviert den Bootloader und hängt an Pin RA3 des Controllers (MCLR ist per Konfigurationsbit deaktiviert). JP2 terminiert die Busleitung mit einem 120 Ohm-Widerstand.

## Aufbau

Ein erster Versuchsaufbau (siehe Abb. 1) wurde mithilfe des PIC Entwicklungsboards USBnub realisiert. Dieses Board besitzt bereits die erforderliche Grundschaltung zum PIC18F14K50, um ohne Zusatzaufwand die USB-Anbindung nutzen zu können. Der MCP2515 war schnell auf eine Lochrasterplatine gelötet und an den Verbinder des USBnub angebunden. Mit diesem Aufbau wurde die erste Schaltungsidee auf Machbarkeit überprüft und eine erste Basis-

Firmware entwickelt. Daraus haben sich einige Schaltungsoptierungen ergeben: So konnte beispielsweise ein Quarz eliminiert werden, indem der CLKOUT-Pin des CAN-Controllers verwendet wird. Mit diesen Verbesserungen konnte der Schaltplan fertig gestellt und das Platinenlayout erstellt werden.

## Bootloader

Um Firmware-Updates direkt über USB ohne extra Programmieradapter zu ermöglichen, wurde ein Bootloader integriert. Der Bootloader sitzt am Anfang des Programmspeichers und wird bei jedem Anstecken des Gerätes zuerst gestartet: Es wird geprüft, ob der Jumper zum Verweilen im Bootloader-Modus gesetzt ist. Ist dies nicht der Fall, wird unmittelbar zum Startpunkt der eigentlichen Applikation gesprungen (in diesem Fall zu Adresse 0x1000 im Flash). Als Bootloader wurde der HID Bootloader aus der Microchip Application Library (MAL v2011-10- 18) verwendet und für den USBtin angepasst. Die Möglichkeit, das EEPROM über den Bootloader zu lesen und zu beschreiben wurde abgeschaltet, um Platz im Flash für die notwendige Initialisierung des CAN-Controllers MCP2515 zu schaffen. Dieser stellt nämlich den Takt für den Mikrocontroller bereit. Nach Anlegen der Versorgungsspannung ist zwar der Taktausgang des CANControllers standardmäßig an, wird aber um den Faktor 8 heruntergeteilt. Um im PIC die USBFunktionalität nutzen zu können, wird ein Taktsignal von 12Mhz genutzt. Über die interne 4fach PLL werden daraus die für die USB-Peripherie nötigen 48Mhz erzeugt. Somit müssen die am CANController anliegenden 24Mhz halbiert werden. Die entsprechende Konfiguration geschieht nachm Initialisierung der SPI-Schnittstelle, nach einem Reset-Kommando und durch Setzen eines Konfigurationregisters im MCP2515. Um den PIC sicher zu starten, ist der sogenannte Fail-Safe Clock Monitor aktiviert. Dieser schaltet automatisch auf den internen Oszillator um, falls die externe Taktversorgung aussetzt. Sobald der Taktausgang des MCP2515 richtig eingestellt ist, wird der PIC per Firmware-Befehl (wieder) auf externe Taktversorgung umgeschaltet. Als Entwicklungsumgebung wurde MPLAB X mit dem C18 Compiler unter Ubuntu eingesetzt. Beide stehen als kostenfreie Downloads auf der Microchip-Webseite zur Verfügung. Der Bootloader wurde über einen Adapter (ICSP-Leitungen auf USB-Buchse und JP1) mithilfe des Programmiergerätes PicKit3 in den PIC geschrieben. PC-seitig wird das Open Source Programm MPHidFlash eingesetzt, um Programmcodes im HEXFormat an den Bootloader zu übermitteln.

## Firmware

Der USBtin sollte möglichst einfach unter den verschiedenen Betriebssystemen zu nutzen sein. Als sinnvolle Möglichkeit wurde das SLCAN-Protokoll identifiziert. Dieses Protokoll hat ihren Ursprung bei CAN-Adaptern für die serielle Schnittstelle und wurde von der Firma Lawicel entwickelt (daher wird es auch Lawicel-Protokoll genannt). Um dieses Protokoll umzusetzen, wurde in die Firmware die USB-Klasse CDC integriert. Damit meldet sich der USBtin als serielle Schnittstelle am PC und kann einfach über ein beliebiges Terminalprogramm bedient werden. Der USB-Stack wurde für den PIC18F14K50 neu geschrieben und unterliegt damit nicht den Einschränkungen zum Verteilen des

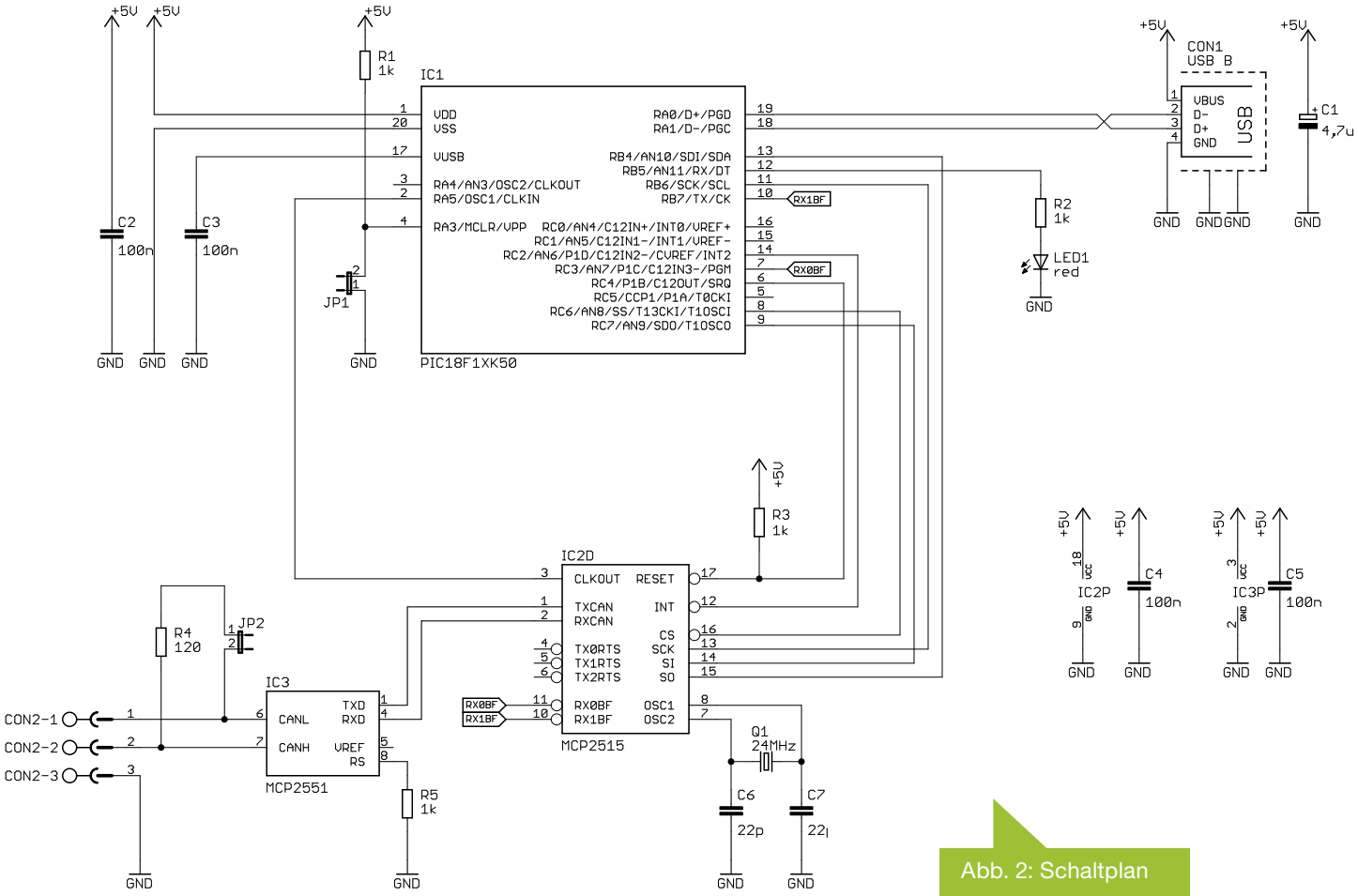



Abb. 2: Schaltplan

Quellcodes, wie dies bei der Microchip-Bibliothek der Fall ist. Die Kommandos werden zeilenweise (abgeschlossen mit CR) an das Gerät übermittelt. Der erste Buchstabe identifiziert das Kommando. Dann folgen eventuelle Parameter. Hier die wichtigsten Befehle:

```
Sx[CR] Setzen der CAN-Bau-
drate

x: 0..8 entspricht 10, 20,
50, 100, 125, 250, 500, 800,
1000 Kbaud

O[CR] CAN-Kanal öffnen

C[CR] CAN-Kanal schließen

tiiildd...[CR] Sendet eine
CAN-Nachricht

iii: Identifier in Hex-
Schreibweise

l: Anzahl Datenbytes in Hex-
Schreibweise

dd: Datenbytes
```

Da das Protokoll teilweise sehr hardwarenah ausgelegt ist und ursprünglich den alternativen CANController SJA1000 von NXP adressierte, konnten nicht alle Kommandos (z.B. manuelles Setzen der Bittiming-Register) exakt umgesetzt werden. Ein Nachteil bei der Nutzung des virtuellen Comports über die Standardtreiber ist die begrenzte Übertragungsgeschwindigkeit zwischen PC und USBtin. Um den Datendurchsatz zu erhöhen, wäre eine modifizierte Firmware denkbar. Der Quellcode ist frei verfügbar – ein Makefile zum Erzeugen und Übertragen der Hex-Datei wird mitgeliefert. Als Compiler wurde der „HI-TECH C PRO for the PIC18 MCU Family (Lite) V9.65“ eingesetzt.

## Software

Prinzipiell kann jedes Terminalprogramm eingesetzt werden, um Kommandos an den USBtin zu senden und empfangene CAN-Nachrichten darzustellen. Abbildung 3 zeigt das Programm HTerm von Tobias Hammer in Aktion. Dieses ist sowohl für Windows als auch für Linux kostenlos verfügbar und bietet die Möglichkeit, Sequenzen in einer eigenen XML-Datei abzuspeichern. Damit sind die wichtigsten Kommandos auf Mausklick abrufbar. Praktisch ist auch die Funktion „Send on enter CR“; damit lassen sich eingetippte Befehle einfach absetzen. Das Bildschirmfoto (Abb. 3) zeigt eine typische Befehlsabfolge: Zuerst wird per „V[CR]“ die Versionsnummer des USBtin abgerufen. Die erfolgreiche Antwort bestätigt, dass die Kommunikation zwischen PC und USBtin funktioniert. Anschließend wird mit „S0[CR]“ die Baudrate auf 10KBaud eingestellt und der CAN-Kanal geöffnet („O[CR]“). Mit „t1233456789[CR]“ wird eine CAN-Nachricht mit der Standard-ID 0x123 und den Datenbytes 0x45, 0x67, 0x89 abgesetzt (die 0x3 nach der ID gibt die Länge der Nutzdaten an).

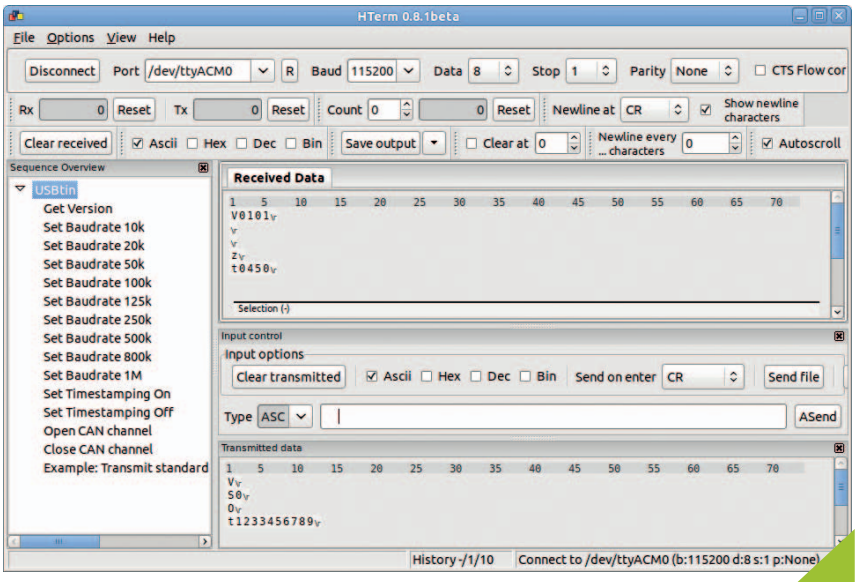



Abb. 3: HTerm in Aktion

## Erweiterungen

Die aktuelle Firmware bietet Erweiterungsmöglichkeiten: gut 30% des Flashspeichers stehen noch zur Verfügung. Eine mögliche Ergänzung betrifft die Filterfunktionen des MCP2515, mit deren Hilfe nur bestimmte CAN-Nachrichten empfangen werden. Denkbar wäre auch eine alternative Firmware, die nicht per virtuellem Comport und per ASCII-Kommandos kommuniziert und damit höhere Übertragungsgeschwindigkeiten ermöglicht. Außerdem könnte die Einführung von zusätzlichen Fifos in der Firmware kurzzeitig erhöhtes Datenaufkommen besser puffern. Möglich wäre auch eine Funktion, um direkt aus der Firmware periodisch CAN-Nachrichten abzusetzen. Sicherlich gibt es noch viele weitere Ideen, um die Firmware um eigene Funktionen zu erweitern. Der Bootloader vereinfacht das Einspielen eigener Firmwarevarianten.

## Bezugsquellen

Alle benötigten Bauteile sind für Bastler im Versandhandel erhältlich. Ein Komplettbausatz inklusive Platine und programmiertem PIC wird angeboten. Bestellmöglichkeit sowie Firmware-Downloads sind auf folgender Seite zu finden:

http://www.fischl.de/usbtin/ □

## Links

[1] Microchip Chiphersteller http://www.microchip.com

[2] MPHidFlash Programm für HID-Bootloader http://code.google.com/p/mphidflash/

[3] USBnub USB-Entwicklungsboard http://www.fundf.net/usbnub/

[4] HTerm Terminalprogramm http://www.der-hammer.info/terminal/

# Die elektronische Zigarette

Tobias Zuber

Ein moderner Mensch nutzt Elektronik um sein Leben unterhaltsamer und effizienter zu gestalten. So sind die rauchlosen Zigaretten schon seit einiger Zeit auf dem Markt. Grund genug um sich ein Modell mal genauer anzuschauen.

## Allgemeiner Aufbau

Im Wesentlichen besteht die E-Zigarette aus drei Teilen, dem Akku mit der Platine, dem Verdampfer und dem Tank für die Flüssigkeit, oft auch Liquid genannt. Das Liquid gibt es in unzähligen Geschmacksrichtungen, mit und ohne Nikotin in unterschiedlicher Konzentration. Im Verdampfer ist eine Heizspirale, die über einen Docht mit Flüssigkeit versorgt wird. Der 650 mAh Litium-Ionen Akku soll bei kontinuierlicher intensiver Nutzung mindestens 6 Stunden halten.

Abb. 1:
Bestandteile der elektronischen Zigarette; Akku mit Platine (li.), Verdampfereinheit (m.), Mundstück mit Tank (re.)

## Innenleben

Das Gehäuse zu öffnen gestaltete sich doch nicht so einfach wie gedacht. Da half nur rohe Gewalt, wie Abb. 4 zu sehen. Neben den oben beschriebenen Komponenten ist das wichtigste natürlich die Platine. Nach der Befreiung aus dem Gehäuse (Abb. 1 li.) erkennt man auf der Oberseite den Taster sowie 2 LED und Widerstände (Abb. 2). Auf der enger bestückten Unterseite liegen ein paar Widerstände, Kondensatoren sowie 3 Halbleiter (Abb. 3, Q1 und Q2 bereits abmontiert). Das Herzstück der Schaltung ist ein(e) MCU mit 8-Pins im TSSOP8 Gehäuse. Die Suche nach der Aufschrift FD2005 ergab erstmal kein passendes Datenblatt. Da, wie fast alles heutzutage, auch die E-Zigarette in China hergestellt wird, wurde ich auf einer chinesischen Seite fündig. Dank Google-translate ist selbst Chinesisch lesbar. Leider gibt es auch hier kein richtiges Datenblatt, z.B. mit Pinbelegung etc., sondern nur eine Funktionsbeschreibung. Der Chip scheint extra für E-Zigaretten gemacht zu sein. Ob ein Standartchip dahinter steckt habe ich nicht herausgefunden.

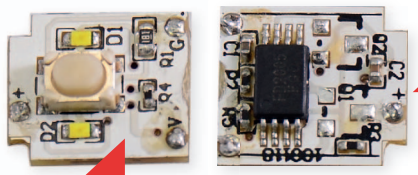
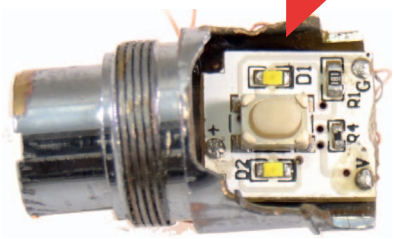


Abb. 2: Taster und LEDs

Abb. 3: Steuerung mit TSSOP8

Abb. 4: Platine im Gehäuse



| Stückliste | |
|---|---|
| Halbleiter | Q1, A18V, SOT23<br>Q2, C633, SOT23<br>FD2005, TSSOP8<br>2 LEDs |
| Widerstände | R1 180Ω; R2 0Ω;<br>R3, R4 4,7k;<br>R5 1M |
| Kondensatoren | C1, C2 |
| Taster | |

Um mir einen groben Überblick der Schaltung zu verschaffen, habe ich versucht einen Schaltplan zu erstellen (siehe Abb. 5). Die zwei Halbleiter im SOT23 Gehäuse ließen sich durch den aufgedruckten SMD-Code nicht so eindeutig identifizieren. Ein kleiner, leider nicht von mir, selbst gebauter Tester half bei einem Bauteil weiter. Anscheinend ist es ein p-MOS FET. Hauptaufgaben des Mikrocontrollers sind die Regelung des Stroms und der Spannung für den Heizdraht, sowie die Ansteuerung der LEDs. Laut Funktionsbeschreibung wird die Spannung bei 3,2V mit konstantem Strom gehalten. Eine weitere Funktion ist die Erkennung von zu niedriger Batteriespannung, die mit blinken der LEDs signalisiert wird, gefolgt von einer Abschaltung. Der Controller hängt direkt an der Versorgungsspannung, wird also nicht erst durch den Taster eingeschaltet. Um die damit verbundene ständige Entladung der Batterie so gering wie möglich zu halten, darf der Mikrocontroller nur sehr wenig Strom verbrauchen oder muss sich in einen Sparmodus schalten. Beim Layout wurde eine doppelseitige Platine benutzt, die auch auf beiden Seiten bestückt ist. Das ist zwar normalerweise aufwändiger und damit teurer, lässt sich aber hier wegen der beschränkten Platinengröße nicht vermeiden. Was sich noch erkennen lässt ist, dass die stromführenden Leiterbahnen von der Spannungsquelle zur Heizwendel etwas breiter angelegt sind.

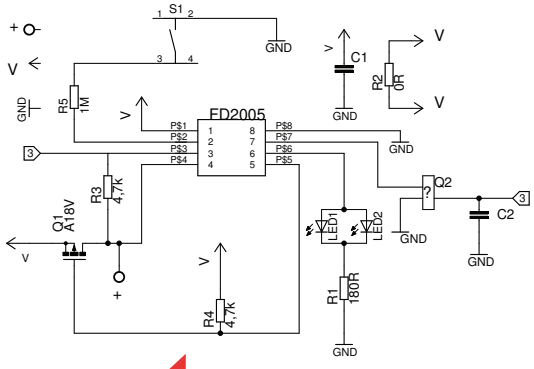



Abb. 5: Schaltung (herausgemessen)

## USB Ladegerät

Die Einheit (siehe Abb. 6) dient im Wesentlichen dazu aus den 5V USB Spannung 4.2 V Ladespannung zu erzeugen. Bei 420mA Ladestrom ist ein normaler USB Anschluss noch ausreichend. Kurz gerechnet: 650mAh durch 420mA, also müsste der Akku mindestens 1,5 Stunden laden, wenn er leer ist und keine Verluste auftreten. Der große Chip auf der Platine ist, anders als vielleicht zu erwarten, kein Mikrocontroller, sondern im Gehäuse befinden lediglich 4 einzelne OPVs. Die Schaltung ist also „nur“ eine Operationsvertärkerschaltung mit ein paar zusätzlichen Bauteilen. Auf der rechten Seite sieht man noch den über zwei Kabel verbundenen praktischen Schraubanschluss.

| Stückliste | |
|---|---|
| Halbleiter | 1AM, SOT23 |
| | 431, SOT23 |
| | UTC324E Y1A58C, |
| | 14-SOP-225 |

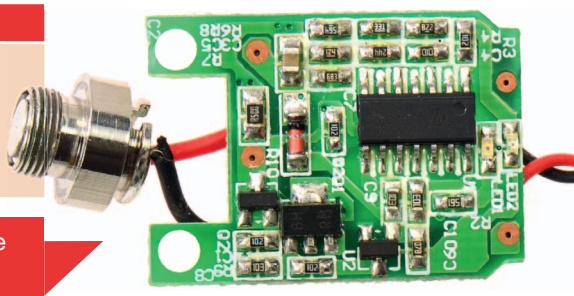


Abb. 6: Platine Ladegerät

## Fazit

Unglaublich, was so alles in einer E-Zigraette steckt, obwohl doch ein Kondensator auch so schön rauchen kann :-).

## Links / Download

[1] http://de.wikipedia.org/wiki/Elektronische_Zigarette

[2] Beschreibung FD2005 http://www.eemcu.com/admin%5Cdown/20086413302691776.rar

[3] Datenblatt UTC324



# GNUBLIN Installer

Michael Schwarz <michael.schwarz91@gmail.com>

Als das GNUBLIN Board herauskam, war ich sofort begeistert davon, da ich nun endlich eine kostengünstige Variante gefunden hatte, um mich in Embedded Linux einzuarbeiten. Nachdem ich ein wenig mit dem Board experimentiert hatte, wollte ich endlich meinen ersten eigenen Kernel und Bootloader kompilieren. Das klappte dank des Wikis auch ganz gut, doch dann kam die größte Schwierigkeit: den Kernel, Bootloader und das Root Filesystem auf eine neue Karte zu bringen.

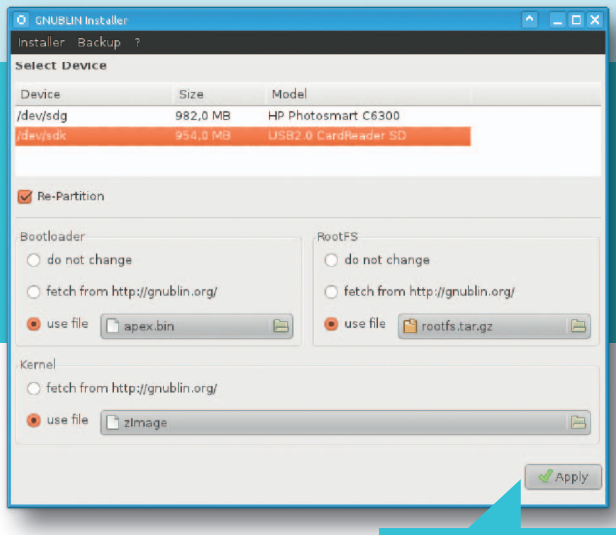



Abb. 1: Oberfläche des Tools

## Erste Versuche

Zuerst wollte ich das für diesen Zweck zur Verfügung gestellte Python Script verwenden. Doch nachdem ich mir den Sourcecode durchgesehen hatte, kam ich zu dem Entschluss, dass ich es doch lieber manuell machen würde. Das funktionierte auch recht gut, war jedoch sehr zeitaufwendig. Darum beschloss ich, mein eigenes, benutzerfreundliches Programm zu entwickeln (siehe Abb. 1).

## Planung

Am Anfang stand, wie bei den meisten Programmen, die Planung und die Überlegung, welche Features das Programm haben sollte und für welche Zielgruppe es geeignet sein sollte. Nach einigen Entwürfen beschloss ich, einen Installer zu schreiben, der für abso-

lute Neulinge auf dem Gebiet Embedded Linux geeignet ist und nur die grundlegenden Funktionen beinhaltet - das Installieren des Grundsystems aus zwei Quellen: den offiziellen GNUBLIN Quellen von http://www.gnublin.org und der eigenen Festplatte. Die nächste Entscheidung war, in welcher Sprache das Programm entstehen sollte. Da doch recht viele plattformabhängige Vorgänge durchgeführt werden, wie der direkte Zugriff auf die Karte oder das Erstellen der Partitionen entschied ich mich gegen Java und für C++. Im Endeffekt wurde das Programm eine Mischung aus C und C++. Low-Level Funktionien wurden in C implementiert und das GUI mit Hilfe von wxWidgets in C++.

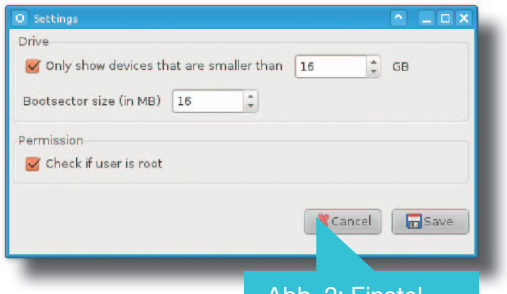



Abb. 2: Einstellungen

## Das Backend

Das Partitionieren der Karte erfolgt mit Hilfe von libparted, der Bibliothek die auch von parted verwendet wird. Die Dokumentation dazu war zwar etwas spärlich, doch Trial-and-Error und das Studieren des parted Quelltexts haben schlussendlich doch noch zum Erfolg geführt. Das Downloaden der Dateien erledigt libcurl und das Entpacken des Root Dateisystems geschieht durch libarchive, welche viele verbreitete Archivformate entpacken kann, darunter tar.gz und tar.bz.

## Das GUI

Das GUI ist in C++ mit dem wxWidgets Framework programmiert, die Oberfläche selbst wurde im wxFormBuilder entworfen. Das macht es einfach, die Elemente der Oberfläche neu anzuordnen und neue Elemente hinzuzufügen.

## Installation

Für Ubuntu gestaltet sich die Installation einfach, hierfür gibt es schon fertige Pakete, die einfach über einen Klick installiert werden können. Auführen kann man den Installer mit Alt+F2 und danach gksu gnublin-installer (Ubuntu) bzw. kdesu gnublin-installer (Kubuntu).

Wer das Programm allerdings selbst kompilieren muss oder möchte, hier eine kurze Anleitung. Als Voraussetzung werden libparted, libcurl, libarchive und libwxwidgets jeweils als Development Version benötigt. Diese kann man mit

```
$ sudo apt-get install libparted0-dev lib-
curl4-openssl-dev libwxgtk2.8-dev libarchive-
dev
```

installieren.

Den Sourcecode kann man von der Projektseite mit

```
$ svn checkout http://gnublin-installer.
googlecode.com/svn/trunk/ gnublin-installer-
read-only
```

downloaden.

Kompiliert wird das Programm mit einem Aufruf von make und kann danach mit

```
$ sudo ./gnublin-installer
```

ausgeführt werden. Wenn das Programm verschoben wird, muss unbedingt die settings.xml im gleichen Ordner liegen, da das Programm sonst nicht funktioniert.

Anzeige

## Das Programm

Das Programm selbst sollte recht selbsterklärend sein. Es wird eine Liste von Geräten angezeigt, auf denen GNUBLIN installiert werden kann (mehr dazu später). Sobald ein Gerät ausgewählt wurde, werden die anderen Punkte aktiviert. Nun wählt man aus, welcher Bootloader, Kernel und welches Root Filesystem verwendet werden soll. Will man nur ein lauffähiges System haben, wählt man einfach überall „fetch from http://gnublin.org“ aus.

Die Option „Re-Partition“ gibt an, ob die Speicherkarte formatiert und die notwendigen Partitionen angelegt werden sollen. Das ist nur dann erforderlich, wenn das GNUBLIN System zum ersten Mal auf dieser Speicherkarte installiert wird.

Der Installationsprozess dauert zwischen einigen Sekunden bis zu mehreren Minuten, je nachdem wie schnell die Internetverbindung ist, ob die Karte partitioniert wird und ob ein Root Filesystem entpackt werden muss. Bei einer erstmaligen Installation, bei der alle Dateien von den GNUBLIN Quellen heruntergeladen werden müssen, kann die Installation schon eine Stunde dauern.

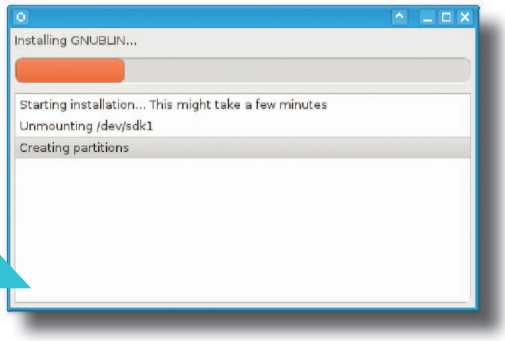



Abb. 3: SD-Karte beschreiben

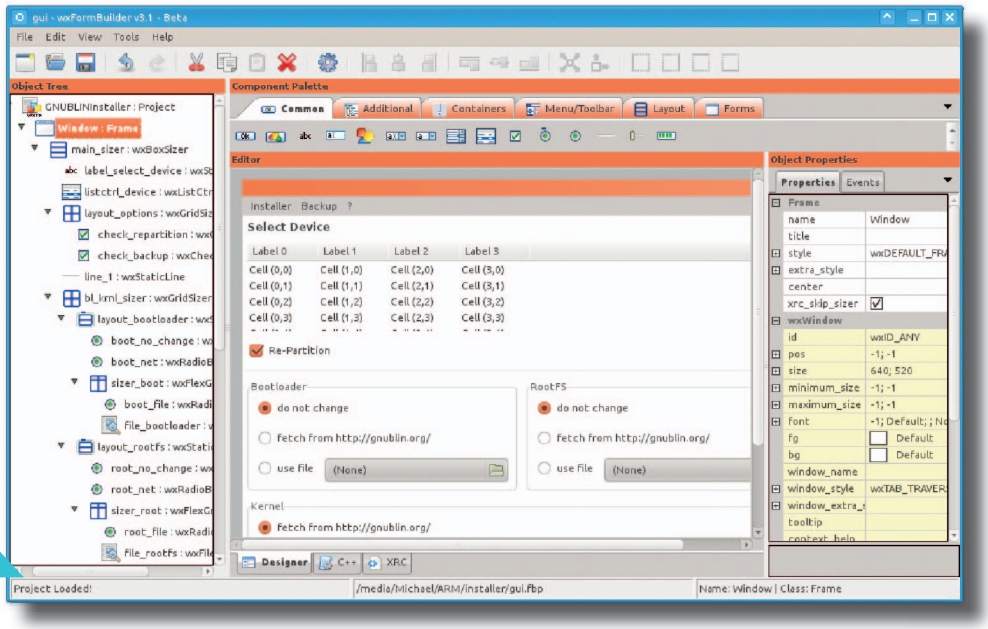



Abb. 4: wxFormBuilder

## Backup

Hat man ein lauffähiges System erzeugt, bietet es sich an, davon ein Backup zu erstellen. Seit Version 1.2 gibt es den Menüpunkt „Backup“ mit dem man sehr einfach ein Backup erstellen und wieder einspielen kann. Das Backup ist ein 1:1 Dump der Speicherkarte, das Einspielen eines Backups sollte also nur auf jene Karte erfolgen, von welcher das Backup erstellt wurde.

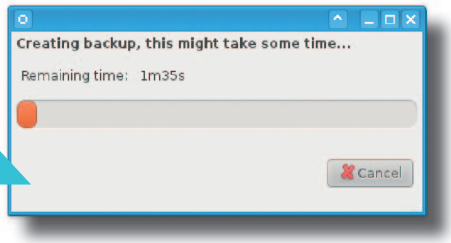



Abb. 5: Backup erstellen

## Einstellungen

Für fortgeschrittene Benutzer gibt es noch die Einstellungen. Dort kann der Filter angepasst werden, welche Geräte angezeigt werden. Standardmäßig werden nur Geräte angezeigt, die eine Kapazität unter 16GB haben, damit man GNUBLIN nicht versehentlich auf eine Festplatte installiert. Weiters kann noch die Größe der Partition verändert werden, auf die der Bootloader installiert wird.

## Zusammenfassung

Der GNUBLIN Installer ist mit dem Ziel entstanden, eine einfache und zuverlässige Möglichkeit zu schaffen das GNUBLIN System zu installieren, um auch Anfängern die Möglichkeit zu geben, sich ihr eigenes System zusammenzustellen. Für die nächste Version ist ein Kommandozeilen Front-End geplant, um den Installer auch in Scripts verwenden zu können.

Wer selber daran weiterarbeiten möchte, kann dies gerne tun, der Quelltext steht unter der GPL v3 und ist auf der Projektseite zum Download verfügbar.

## Links / Download

[1] Projektseite: http://code.google.com/p/gnublin-installer/

[2] Doku zur libparted: http://www.gnu.org/software/parted/api/index.html

[3] Projektseite der libarchive: http://code.google.com/p/libarchive/

[3] Projektseite des wxFormBuilders: http://wxformbuilder.org/

# Implementierung einer Finite State Machine

Philipp Kälin <kaelinphilipp@gmail.com>

## Einleitung

Dieser Artikel hat zum Ziel, eine Möglichkeit zu zeigen wie kleinere bis mittlere endliche Automaten einfach implementiert werden können. Folgende Sachverhalte werden in diesem Artikel beschrieben:

- Grundlagen was ein endlicher Automat ist und wie er funktioniert.
- Notation von Endlichen Automaten in UML1
- Praktisches Beispiel, anhand dessen die Funktionsweise eines Toasters erklärt wird. Dazu wird die Notation in UML verwendet.
- Implementation des Beispiels in C++ auf einem AVR-Controller.

## Allgemeines über Endliche Automaten

Ein endlicher Automat beschreibt eine Maschine oder eine Software welche eine definierte Anzahl Zustände hat, in denen sie sich befinden kann. Wozu braucht man nun so etwas?

Mit endlichen Automaten ist es sehr einfach das Verhalten von Systemen zu beschreiben.

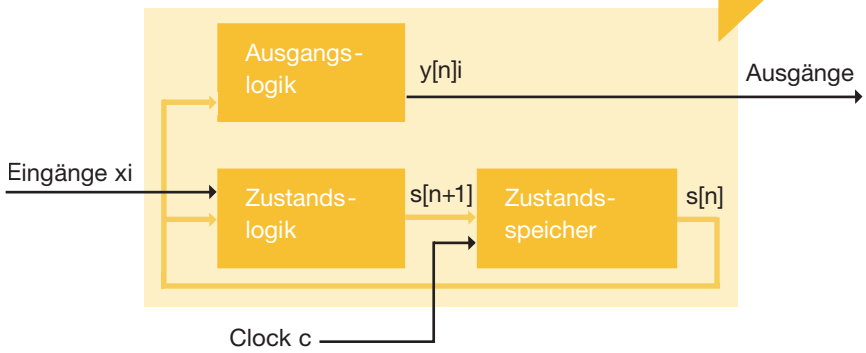
## Funktionsweise eines Moore Automaten

Mit endlichen Automaten kann man auf abstrakter Ebene das Verhalten von Systemen beschreiben, insbesondere auch das von Digitalen Schaltwerken. Endliche Automaten werden jedoch nicht nur in der Digitaltechnik verwendet, sondern werden auch in der Softwareentwicklung angewendet. Wobei in der Softwareentwicklung vor allem der Moore-Automat zur Anwendung kommt. In diesem Kapitel wird das grundsätzliche Verhalten eines solchen Moore-Automaten erklärt, zu einem späteren Zeitpunkt folgt dann die Umsetzung in einer Programmiersprache.

Ein Moore Automat besteht im wesentlichen aus drei Teilen. Der wichtigste ist der Zustandsspeicher. Dieser speichert den aktuellen Zustand s[n]. Der nächste Zustand s[n + 1] wird von der Zustandslogik berechnet. Sobald ein Clock c auftritt wird der Zustand s[n + 1] als aktueller Zustand s[n] übernommen.

Die Ausgangslogik berechnet gleichzeitig aufgrund des neuen Zustandes s[n] den neuen Ausgänge yi. Der neue Zustand s[n + 1] wird jeweils aufgrund des aktuellen Zustandes s[n] und den Eingängen xi berechnet. Dieses Modell kann fast identisch in der Softwareentwicklung angewendet werden. Die Eingänge werden dabei zu Events, welche von verschiedenen Quellen z.B. Taster, Timer oder auch Schnittstellen kommen. Der Clock entspricht einem Verarbeitungsaufruf der State Machine. Dabei ist es nicht zwingend, dass dieser Aufruf in einem festen zeitlichen Abstand erfolgt. Der Aufruf kann auch so oft wie möglich erfolgen. Die Ausgänge können wiederum Hardware, Anzeigen, Kommunikation oder IO sein, oder auch nur Funktionsaufrufe.

Abb. 1: Schematische Darstellung eines Moore-Automaten




## UML Notation von Endlichen Automaten

Die UML ist eine graphische Modellierungssprache zur standardisierten Beschreibung von Softwaresystemen. Der hier beschriebene Teil ist nur ein kleiner Ausschnitt aus der ganzen UML, der jedoch für die Beschreibung von Endlichen Automaten ausreicht.

Auswahl einiger Elemente in UML:

Die Abbildung 2 zeigt die wichtigsten Elemente die zur Beschreibung eines Endlichen Automaten nötig sind. Zur Verdeutlichung soll ein kleines Beispiel helfen. Nach dem Start befindet sich der Zähler im Zustand Start. Dies ist durch den orangen Kreis, welcher mit dem Zustand verbunden ist, signalisiert. Bei Start wird durch die entry-Action gekennzeichnet, dass der Zähler auf eine nicht näher spezifizierte Anzeige ausgegeben wird. Ist die Bedingung der Transition zum Zustand Sekunden zählen erfüllt, also die Start Taste gedrückt, so findet ein Zustandswechsel statt. Im Zustand Sekunden zählen wird zu Beginn die Zählervariable auf 0 zurückgesetzt. Die do-Action zeigt an, dass die Zählervariable bei jedem Tick um 1 erhöht wird und die Anzeige mit dem neuen Wert aktualisiert wird. Aus dem Diagramm ist nicht ersichtlich um welche Periodenzeit

es sich bei der tick-Action handelt. Dies muss gesondert angegeben werden.

Im UML-Diagramm spielt es keine Rolle ob die Beschreibungen in "Deutsch" oder Pseudocode hingeschrieben werden. Grundsätzlich sollte ein UML-Diagramm aber unabhängig von der Programmiersprache sein, wieweit man das jedoch durchziehen möchte ist ebenfalls frei. Ob im Diagramm alle Actions gezeichnet werden, so wie in Abbildung 2, oder ob die nicht verwendeten weggelassen werden wie in Abbildung 3 ist Geschmackssache. Bei der später beschriebenen Implementation wird es jedoch so sein, dass alle Actions vorhanden sein müssen und gegebenenfalls leer bleiben, da es effizienter ist, eine leere Action aufzurufen als zu prüfen, ob diese überhaupt vorhanden ist.

Abb. 2: UML-Elemente eines Endlichen Automaten

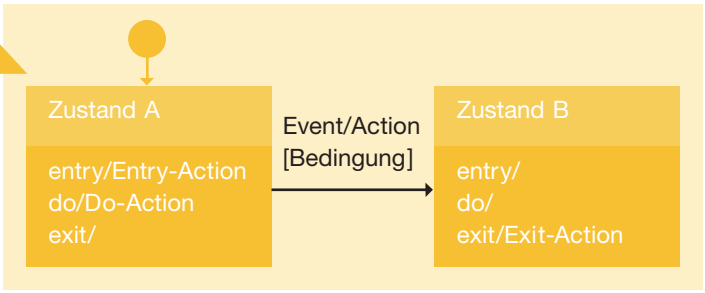
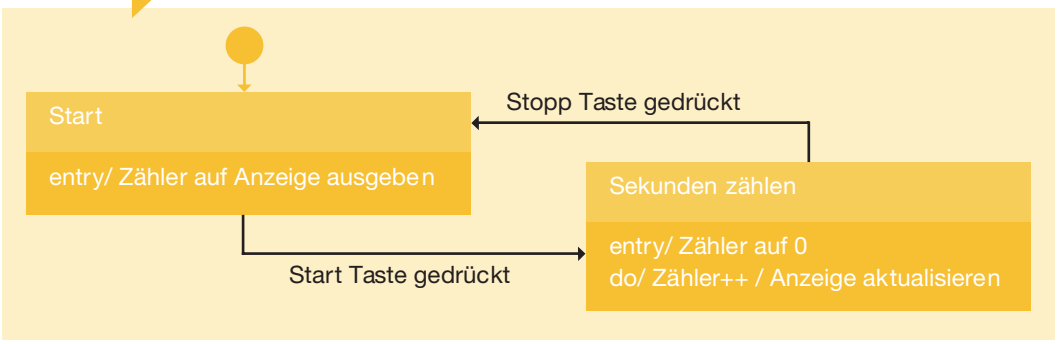



Abb. 3: Beispiel eines einfachen Zählers




## Zustände

Die Zustände werden in einem Rechteck mit abgerundeten (siehe Originaldokument des Autors) Ecken gezeichnet. Der Name sollte beschreiben, was das System in diesem Zustand macht. Was der Zustand macht wird mit drei verschiedenen Action-Attributen dargestellt.

• Die Entry Action beschreibt, was beim Eintreten in einen Zustand gemacht wird. Also dann wenn von einem anderen Zustand in diesen gewechselt wird.

• Die Do-Action beschreibt, was passiert wenn ein Event eintritt, der aktuelle Zustand aber nicht gewechselt wird. Ein Beispiel dafür ist ein Tick Event welcher in bestimmten Zeitabständen auftritt. Daher wird zum Teil auch der Name Tick-Action verwendet.

• Die Exit Action beschreibt, was beim Austreten aus einem Zustand geschieht. Also dann wenn man den aktuellen Zustand verlässt und in einen anderen wechselt.

Der Start eines endlichen Automaten wird mit einem orangen Kreis markiert, welcher durch eine Transition zum Startzustand verbunden ist.

## Transitionen

Ein endlicher Automat macht natürlich nur wenig Sinn wenn er immer im gleichen Zustand bleibt. Übergänge von einem zum nächsten Zustand nennt man Transitionen. Transitionen werden grundsätzlich immer von einem Event ausgelöst. Ein Event kann z.B. sein, dass eine Taste gedrückt wird. Optional an einer Transition ist die [Bedingung] welche zusätzlich erfüllt sein muss, damit eine Transition ausgeführt wird. Optional kann bei einer Transition auch eine zusätzliche \Aktion stehen welche während des Übergangs ausgeführt wird. Für einfachere Zustandsautomaten recht es jedoch aus, nur den Event zu definieren.

## Events

Events sind Ereignisse welche dem endlichen Automaten mitgeteilt werden, damit dieser gegebenenfalls darauf reagieren kann. Dabei wird grundsätzlich zwischen zwei Arten von Events unterschieden:

• User-Events sind Events welche von der Umwelt um das System ausgelöst werden. Beispiele hierfür sind Tasten, Sensoren etc.

• System Events sind Events welche vom System selber generiert werden. Das können z.B. zeitlich gebundene Events sein, Timer Ticks, Fehlermeldungen oder Timeouts etc.

Alle Events die auftreten werden an den Endlichen Automaten übergeben. Ob auf ein Event reagiert wird ist durch die Transitionen des aktuellen Zustandes zu den anderen Zuständen festgelegt.

## Toaster als Beispiel für einen endlichen Automaten

Als anschauliches Beispiel soll ein futuristischer Toaster verwendet werden. Der Toaster besitzt zwei Tasten, eine Start und eine Fertig Taste, mit denen der Toastvorgang gestartet und beendet werden kann. Um die Sache Komfortabler zu machen hat der Toaster eine automatische Brotschublade ähnlich einem CD-Laufwerk. Um zu erkennen ob die Schublade ganz ein- oder ausgefahren ist, sind zusätzlich zwei Endschalter (Endschalter drinnen und Endschalter leer ) vorhanden. Zur Bereitschaftsanzeige ist eine LED Ready vorhanden. Ist die Heizung eingeschaltet leuchtet die LED Heating. Sollte ein Fehler auftreten, z.B. wenn die Brotschublade verklemmt ist, so leuchtet die LED Error.

## Funktionsbeschreibung

Der Startzustand ist Bereit. Die Entry und Exit Action bewirken, dass die LED Ready nur leuchtet wenn sich der Toaster im Zustand Bereit befindet. Sobald man im Zustand Bereit die Start Taste drückt wechselt der Toaster in den Zustand Toast einziehen. Um den Toast einzuziehen muss der Motor für die Schublade drehen, im Beispiel Linkslauf. Wenn alles funktioniert, sollte die Schublade innert nützlicher Zeit eingezogen sein. Wenn die Schublade eingezogen ist wird der Endschalter drinnen gedrückt und der Toaster wechselt in den Zustand Toasten. Im Zustand Toasten muss natürlich die Heizung eingeschaltet werden. Ebenfalls soll die Heating LED leuchten. Um den Toastvorgang zu beenden gibt es zwei Möglichkeiten. Entweder man drückt die Fertig Taste oder die 45 Sekunden Maximaldauer sind abgelaufen. In beiden Fällen wird in den Zustand Toast auswerfen gewechselt. Bein ein- und ausfahren der Schublade kann ein Fehler auftreten, wenn die Schublade verklemmt ist. Ist dies der Fall, so kann die Schublade nicht ganz ein- oder ausgefahren werden, demzufolge spricht der entsprechende Endschalter nicht an. Nach dem verstrichenen Timeout, welches mit after 15s angegeben ist, wird in den Zustand Fehler gewechselt. Beim Toast auswerfen gibt es wieder die zwei Fälle wie beim einziehen. Entweder der Toast wird korrekt ausgeworfen und der Toaster ist wieder im Zustand Bereit, oder es wird in den Fehler Zustand gewechselt.

## UML-Diagramm des Toasters

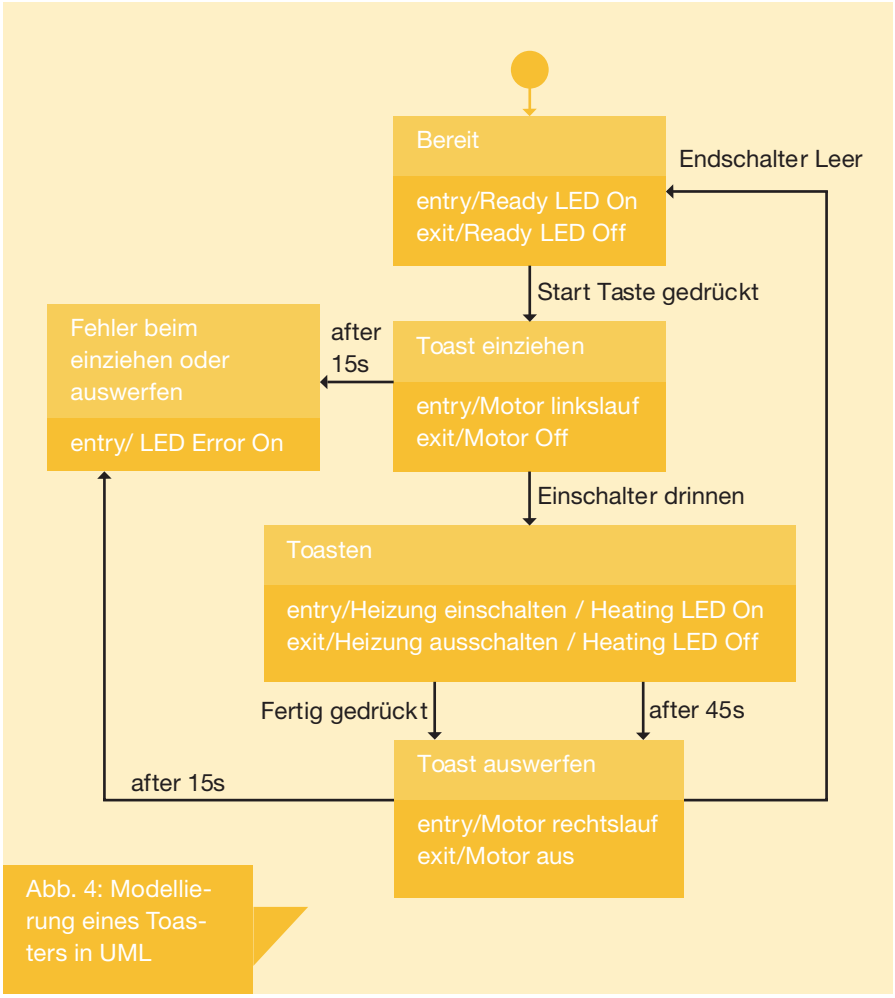



Abb. 4: Modellierung eines Toasters in UML

## Implementierung mittels einer Case Struktur

Für die Implementierung gibt es mehrere Möglichkeiten. Die meist verbreitete ist wahrscheinlich die Case-Struktur. Ein Beispiel einer solchen Case Struktur ist in Listing gezeigt. Eine solche Case Struktur ist gut geeignet um kleinere endliche Automaten abzubilden. Für das Toasterbeispiel gibt es zwei Merkmale, die einem das Leben bei einer solchen Case Struktur schwer machen. Zum einen sind dies die Entry und Exit Actions, wenn in der Case Struktur also in einen Zustand gewechselt wird, so müsste immer abgefragt werden, ob dies der erste Eintritt ist oder nicht. Als zweite Hürde sind die Timeout Bedingungen welche nicht so einfach Realisiert werden können.

```
enum {
 Bereit ,
 ToastEinziehen ,
 ...
 } ToasterState ;

 ToasterState state; // Variable um den aktuellen Zustand zu
speichern

 switch (state) {
 case Bereit :
 ... // Etwas im Zustand machen
 state = ToastEinziehen ; // Aktueller Zustand wechseln
 break;

 case ToastEinziehen :
 ...
 state = Toasten ;
 break;

 ...
 }
```

## Implementierung mittels einer Tabelle

Das UML-Diagramm kann direkt in eine Tabelle übernommen werden welche die gleichen Informationen enthält wie das Diagramm. Die Darstellung mittels Tabellen hat für die Implementation zwei Vorteile, verglichen mit dem UML-Diaramm. Erstens ist nun eine klare Trennung zwischen Transitionen und Actions sichtbar, und zweitens kann eine solche Tabelle ohne

grossen Aufwand in Code umgesetzt werden. Die Variante mit der Case Struktur wie oben gezeigt hat allgemein das Problem, dass die "Inteligenz" zur Umschaltung zwischen den Zuständen verteilt ist und nicht an einem Zentralen Ort. Dieses Problem soll mit der Tabellenversion gelöst werden.

| Aktueller Zustand | Event | Timeout [s] | Nächster Zustand |
|---|---|---|---|
| Bereit | Taster Start | | Toast einziehen |
| Toast einziehen | Endschalter drinnen | | Toasten |
| Toast einziehen | Tick | 15 | Fehler |
| Toasten | Taster Fertig | | Toast auswerfen |
| Toasten | Tick | 45 | Toast auswerfen |
| Toast auswerfen | Endschalter Leer | | Bereit |
| Toast auswerfen | Tick | 15 | Fehler |

Tabelle 1: Transisionen des Toasters in einer Tabelle dargestellt

## Implementation des Toaster Beispiels auf einem AVR

Die Implementation auf einem Atmel ATmega128 in C++ ist so ausgelegt, dass diese gut im AVR-Studio mit dem Simulator nachvollzogen werden kann. Als Sprache kommt C++ zum Einsatz, da erstens die Implementation übersichtlicher und besser nachvollziehbar ist als in C, zweitens sind heute C++ Compiler so gut dass diese durchaus auf auf Embedded Systemen zum Einsatz kommen. Als Compiler für das Beispiel kommt avr-g++ zum Zuge.

| | Entry Action | Exit Action |
|---|---|---|
| Bereit | Ready LED On | Ready LED Off |
| Toast einziehen | Motor Linkslauf | Motor Off |
| Toast auswerfen | Motor Rechtslauf | Motor Off |
| Toasten | Heizung einschalten<br>Heating LED On | Heizung ausschalten<br>Heating LED Off |
| Fehler | LED Error On | |

Tabelle 2: Actions der Zustände in einer Tabelle dargestellt

## Definition der Zustände

Für jeden Zustand wird eine eigene Klasse verwendet. Jeder Zustand wird von der abstrakten Klasse State abgeleitet.

```
class State {
public :
virtual void entryAction () = 0;
virtual void exitAction () = 0;
virtual void tickAcktion () = 0;
};
```

Für jeden Zustand müssen nun die Informationen aus der Tabelle implementiert werden. Als Beispiel soll hier der Zustand Bereit verwendet werden. Dazu wird die Klasse StateBereit von der Klasse State abgeleitet, und die drei Actions implementiert.

```
class StateBereit : public State {
public :
void entryAction ();
void exitAction ();
void tickAcktion ();
};

// ----------------------------------------
---------------------------------
void StateBereit :: entryAction () {
Led :: ready(true );
}

// ----------------------------------------
---------------------------------

void StateBereit :: exitAction () {
Led :: ready( false);
}

// ----------------------------------------
---------------------------------
void StateBereit :: tickAcktion () { }
```

## Definition der Transitionen

Eine Transition verbindet zwei Zustände und muss daher zwei Pointer auf die Zustände haben. Ausserdem ist die Information wichtig, bei welchem Event die Transition ausgelöst wird. Die letzte Angabe wird für das Timeout verwendet und besagt wie lange sich der endliche Automat in einem Zustand befinden darf, bevor die Transition ausgelöst wird.

```
struct Transition {
State* currentState ;
Event event;
uint16 _ t maxTimeBeforeTransition;
State* nextState ;
};
```

## Abbildung der Tabelle in C++

Nun sind alle Einzelteile bekannt um diese in einer Tabelle abzubilden und somit den identischen Informationsgehalt wie in Abbildung 4 zu erhalten. In C++ umgesetzt sieht das folgendermassen aus:

```
Toaster :: Transition fsmApp [] = {
// actualState MaxTimeBeforeTransition
// Event Next State
{& bereit , TasterStart , 0, & einziehen },

{& einziehen , EndschalterDrinnen , 0, & toasten },
{& einziehen , Tick , Seconds (15) , & fehler },

{& toasten , TasterFertig , 0, & auswerfen },
{& toasten , Tick , Seconds (45) , & auswerfen },

{& auswerfen , EndschalterLeer , 0, & bereit },
{& auswerfen , Tick , Seconds (15) , & fehler },
};
```

## Verwendung des endlichen Automaten

Die Verwendung des endlichen Automaten kann in zwei Jobs unterteilt werden. Zum Einen müssen natürlich die auftretenden Events gesendet werden. Da es vorkommen kann, dass mehrere Events auftreten bevor diese abgearbeitet werden, werden die Events in eine Queue eingereiht, deren Länge mit EventQueueSize festgelegt werden kann.

```
1 Toaster :: sendEvent (
Toaster :: EndschalterDrin-
nen );
```

Falls Timeouts verwendet werden, so können Tick Events ganz einfach innerhalb eines Timer-Interrupt gesendet werden.

```
ISR( TIMER0 _ OVF _ vect ) {
Toaster :: sendEvent ( Toas-
ter :: Tick );
}
```

Die Verarbeitung der Events erfolgt am besten im Hauptprogramm in einer Endlosschleife und wird so oft wie möglich aufgerufen.

```
1 Toaster :: process ();
```

## Verarbeitung der Events

Wie oben erwähnt werden die Events in einer Queue gesammelt und mit einem process() verarbeitet. Die Funktion process durchsucht dabei die Tabelle nach einer Zeile, in welcher der aktuelle Zustand und der Event übereinstimmen. Wird eine solche Zeile gefunden, wird ein Wechsel des Zustandes vorgenommen. Der große Vorteil dieser Tabellen Methode gegenüber des bereits genannten Case Konstrukts ist, dass die gesamte Intelligenz des endlichen Automaten in einer Funktion steckt.

```
bool Toaster :: process () {
if ( evQueueIsEmpty ()) {
return false;
}
Event e = evQueue [ evQueueHead ];
evQueueHead = ( evQueueHead + 1) & ( evQueueSize - 1)
for ( uint8 _ t i=0; i< sizeof( fsmApp)/ sizeof( Transition );
i++ ) {
  if ( ( ( currentState == fsmApp[i]. currentState ) && (e ==
fsmApp[i]. event) ) || ( fsmApp[i]. event == 0) )
{
if (e == Tick ) {
fsmApp [i]. currentState -> tickAcktion (); // System Tick
timeInState ++;
if ( ( fsmApp [i]. maxTimeBeforeTransition != 0)
&& ( timeInState < fsmApp[i]. maxTimeBeforeTransition) )
{
fsmApp[i]. currentState -> tickAcktion (); // Normal Tick
} else {
// Force Transition because of an Timeout
fsmApp[i]. currentState -> exitAction ();
currentState = fsmApp[i]. nextState ;
currentState -> entryAction ();
timeInState = 0;
}
} else {
timeInState = 0;
fsmApp [i]. currentState -> exitAction ();
currentState = fsmApp [i]. nextState ;
currentState -> entryAction ();
}
break;
}
}
return true ;
}
```

Ausführlichere Informationen und den Source dieses Artikels sind unter http://www.mikrocontroller.net/articles/Implementierung_einer_Finite_State_Machine zu finden.

## Schlusswort

Die beschriebene Variante ist in einem kompletten Beispiel implementiert welches von [1] heruntergeladen werden kann. Das Beispiel ist so ausgeführt, dass es im Simulator des AVR-Studio 4 nachvollzogen werden kann. Als C++ Compiler wird avr-g++ verwendet, welcher unter anderem in WinAVR vorhanden ist. Die Idee der Tabellenvariante kann natürlich auch in C umgesetzt werden. Dazu wird in der Tabelle jeweils der Pointer auf das Objekt durch drei Funktionspointer ersetzt, welche jeweils direkt auf die Action-Funktionen zeigen. Die gezeigte Möglichkeit bzw. das Beispiel sollte als Denkanstoss verstanden werden und nicht als Referenzimplementation. Es wurden bewusst bestimmte Feinheiten von endlichen Automaten verzichtet, um das Beispiel auf verständlichem Niveau zu halten.

## Links/Lizenzen

[1] www.mikrocontroller.net/articles/Statemachine

[2] Dokumentation: Namensnennung-Weitergabe unter gleichen Bedingungen 2.5 Schweiz (CC BY-SA 2.5)
http://creativecommons.org/licenses/by-sa/2.5/ch/

[3] Programmcode: GNU GENERAL PUBLIC LICENSE Version 2
http://www.gnu.org/licenses/old-licenses/gpl-2.0

[4] Titelbild: Das Tielbild ist von jiangyi 99
http://openclipart.org/people/jiangyi_99/jiangyi_99_toaster.svgz

[5] und steht unter einer Public Domain Lizenz
http://creativecommons.org/licenses/publicdomain/

# Gnublin ADC und PWM

Nils Stec <nils.stec@gmail.com>

Nachdem das Gnublin Board läuft und die ersten Kunden und freiwilligen Helfer das Board in ihren Händen halten, braucht man natürlich auch Möglichkeiten, mit der Aussenwelt zu kommunizieren. Das GPIO-sysfs-Interface sowie SPI funktionieren bereits mit dem Standard-Kernel, der bei der Auslieferung von der SD-Karte bootet, fehlen also noch ADC und PWM. So sind zwei Linux-Kernelmodule entstanden, wovon jeweiles eines den AD-Wandler und eines die PWM-Einheit des LPC3131-Prozessors steuert.

## Einleitung

Der AD-Wandler im LPC3131 hat folgende Eigenschaften: Frei einstellbare Auflösung von 2 Bit bis 10 Bit 4 Channels, von denen 3 auf die Gnublin-Stiftleiste geführt sind von 400k Samples pro Sekunde (@10Bit Auflösung) bis zu 1500kSps/s (@2Bit Auflösung) Analog Input Range von 0V bis 3.3V „single shot"- sowie „continuous conversion"-Mode. Um die verschiedenen Möglichkeiten des ADC optimal zu nutzen, werde ich ihn hier erklären, das Anpassen auf die eigene Anwendung bleibt dem User überlassen. Mehrere Module oder Teile davon im Quellcode hier abzudrucken, würde den Rahmen sprengen. Als Ausgangspunkt für Veränderungen, kann man sich das ADC-Modul anpassen.

## Funktionsbeschreibung

Der ADC des LPC3131 hat 4 gemultiplexte Channels. Bei jedem Durchlauf einer Wandlung, werden alle Channels nacheinander eingelesen, für die die Auflösung festgelegt wurde. Die Ergebnisse dieser Wandlung(en) liegen in den Registern ADC_Rx_REG, wobei x für den jeweiligen Channel von 0 bis 3 steht. Startet man eine Wandlung, geschieht das Multiplexing, das Wandeln und das Übertragen des Ergebnisses/der Ergebnisse in die jeweiligen Register ganz automatisch. Das Ergebnis einer Wandlung ist gültig ab dem Moment, ab dem ein Flag gesetzt wurde bzw. der Interrupt ausgelöst wurde.

## Vorbereitungen am Kernel

Da dem Kernel-Quellcode ein paar Register-Definitionen im Header „arch/arm/mach-lpc313x/include/mach/registers.h" fehlen, müssen wir uns darum kümmern, diese nachträglich hinzuzufügen. Ein User auf mikrocontroller.net („misc") hat einen Patch geschrieben, der das für uns macht. Dieser Patch hat den Dateinamen „adc-registers.patch". Zu finden ist er im mikrocontroller.net-Forum oder auch direkt hier:

```
*** a/arch/arm/mach-lpc313x/include/mach/
registers.h     2011-12-14 14:35:49.560517000
+0100

--- b/arch/arm/mach-lpc313x/include/mach/
registers.h     2011-12-19 13:25:13.630011000
+0100

***************

*** 276,281 ****

--- 276,289 ----

    * ADC _ REG register definitions

    ******************************************
************************/

  #define ADC _ CON _ REG             _ _ REG
(ADC _ PHYS + 0x20)
+ #define ADC _ R0 _ REG              _ _ REG
(ADC _ PHYS + 0x00)
+ #define ADC _ R1 _ REG              _ _ REG
(ADC _ PHYS + 0x04)
+ #define ADC _ R2 _ REG              _ _ REG
(ADC _ PHYS + 0x08)
+ #define ADC _ R3 _ REG              _ _ REG
(ADC _ PHYS + 0x0C)
+ #define ADC _ CSEL _ REG            _ _ REG
(ADC _ PHYS + 0x24)
+ #define ADC _ INT _ ENABLE _ REG     _ _ REG
(ADC _ PHYS + 0x28)
+ #define ADC _ INT _ STATUS _ REG     _ _ REG
(ADC _ PHYS + 0x2C)
+ #define ADC _ INT _ CLEAR _ REG      _ _ REG
(ADC _ PHYS + 0x30)

  /******************************************
************************

   * SYS _ REG register definitions
```

Wer den Patch so einspielen möchte, legt die Datei „adc-registers.patch" im Kernelverzeichnis ab und wendet den Patch an:

```
$ patch -p1 < adc _ registers.patch
```

## Lesen und Schreiben der Register

Der Kernel stellt uns eine recht einfache Methode zur Verfügung, um Register-Inhalte zu verändern bzw. zu lesen. Durch den angewandten Patch haben wir nun die Möglichkeit, auch ohne die jeweilige Registeradresse genau zu kennen, darauf zuzugreifen.

```
ADC _ CON _ REG = 0xA0B1C2D3;              //
write into register

uint32 _ t temp = ADC _ CON _ REG;         //
read from register into „temp“
```

Die Register liegen im Speicherbereich des lpc3131 und sind im Datenblatt immer mit „Base Address“ und „Offset“ angegeben.

Die „Base Address“ ist in unserem Fall 0x13002000. Das Offset ist 0x20. Die Resultierende Adresse für das ADC_CON_REG ist demnach 0x13002020.

Auszug aus dem adc-registers Patch:

```
#define ADC _ CON _ REG                    _ _ REG
(ADC _ PHYS + 0x20)
```

Diese Definition sagt, dass wir Zugriff auf ein Register haben wollen, welches an der Adresse ADC_PHYS+0x20 liegt.

ADC_PHYS ist im Kernel-Quellcode mit 0x13002000 definiert, die Basis-Adresse des AD-Wandlers.

## Clock

Die MCU-Eigene CGU („clock generation unit“) stellt einen Takt für den AD-Wandler bereit. Dieser Takt ist standardmäßig auf 31.25kHz gestellt. Ebenfalls muss der „APB“-Takt („AMBA Peripherial Bus“) aktiviert werden. Dieser Bus verbindet die Peripherie-Geräte wie den ADC mit dem Controller-Kern.

```
cgu _ clk _ en _ dis(CGU _ SB _ ADC _ CLK _ ID, 1);
// enable ADC clock
```

```
cgu _ clk _ en _ dis(CGU _ SB _ ADC _ PCLK _ ID,
1);   // enable bus-clock for ADC
```

Diese zwei Zeilen Code schalten zuerst den ADC-Takt ein, dann den Bus-Takt.

## Einstellen der Channels und der Auflösung

Wie weiter oben schon geschrieben, werden alle Kanäle für die eine Auflösung gesetzt wurde, gesampelt, sobald die Wandlung gestartet wurde. Der zulässige Bereich für die Auflösung liegt zwischen 2 Bit und 10 Bit. Umso niedriger die Auflösung, desto schneller wird die Wandlung von statten gehen. Channels, bei denen 0 im Register für die Auflösung steht, werden von der Wandlung ausgenommen. Dadurch, dass auch mehrere Kanäle mit unterschiedlichen Auflösungen gesampelt werden können, liegt die Dauer für eine AD-Wandlung immer so hoch, wie für den Kanal mit der höchsten Auflösung. Die Formel zum Berechnen der Dauer aus dem LPC3131-User-Manual:

$$samples/s = \frac{F_{CLOCK}}{RESOLUTION \square 1}$$

Bei einer Clock-Frequenz von 31.25kHz und einer Auflösung von 10 Bit, kommt man auf ca. 2.84k Samples/s.

Das zuständige Register ist das ADC_CSEL_RES_REG.

| Bit | Symbol | Beschreibung |
|---|---|---|
| 0-3 | CSEL0 | Auflösung Channel 0 |
| 4-7 | CSEL1 | Auflösung Channel 1 |
| 8-11 | CSEL2 | Auflösung Channel 2 |
| 12-15 | CSEL3 | Auflösung Channel 3 |

Um nun z.B. alle 4 Channel auf 10 Bit Auflösung einzustellen, muss man 0xAAAA in die unteren 16 Bit des Registers schreiben.

Wurden für eine Wandlung mehrere Channels ausgewählt und bei der folgenden Wandlung einer dieser Channels nicht, steht in den Registern für die Wandlungsergebnisse (ADC_Rx_REG) immer noch das Ergebnis der letzten Wandlung.

Eine Überprüfung des ADC_CSEL_RES_REG auf die eingestellten Channels und damit auf die Gültigkeit der Wandlungsergebnisse ist daher sinnvoll.

## Interrupt oder Polling?

Die Frage, die sich wohl jeder stellen wird, ist, ob Aufgabe XY nun besser mit dem ADC-Interrupt oder ohne gelöst wird. Ein „normaler“ Kernel-Treiber für den AD-Wandler, der seine Ergebnisse einfach nur in den Userspace weiterreicht, braucht keinen Interrupt, dort ist er eher hinderlich. Wir können im Interrupt-Handler keine Daten in den Userspace schicken, somit müssten wir im Handler das Ergebnis holen, in einer temporären Variable ablegen und ein Flag setzen, sodass das „Hauptprogramm“ den Inhalt der temporären Variable weitergeben kann. Eine selbstständige Akku-/Versorgungsüberwachung könnte ein Fall für einen Interrupt-Handler sein.

Das LPC3131-User Manual sagt, dass das Ergebnis einer AD-Wandlung gültig ist, sobald das ADC_INT_STATUS-Bit im ADC_INT_STATUS_REG gesetzt ist.

Es lässt allerdings offen, ob dafür das Erzeugen eines Interrupts nötig ist, auch wenn dieser nicht behandelt wird (UM10314 Revision 1.03 – 27 May 2010, Seite 313, Kap. 5.1).

Im ADC_CON_REG gibt es das ADC_STATUS-Bit, welches zum Auslesen des Status des ADCs verwendet werden könnte, wenn der ADC ohne IRQ läuft. Leider zeigt das Board bzw. der LPC3131 ein unerklärliches Verhalten, wenn ich die Abfrage nur über das ADC_STATUS-Bit mache. Das äussert sich darin, dass das ADC-Ergebnis schon vor dem

Beenden der Wandlung abgerufen wird und somit beim ersten Mal 0 ist und alle weiteren Male, das der letzten Wandlung.

Ich aktiviere deshalb den ADC-Interrupt und warte auf das ADC-Interrupt Flag, ohne jedoch einen Handler installiert zu haben. So wird der IRQ gänzlich vom Linux-Kernel ignoriert. Wird das ADC_INT_ENABLE-Bit im ADC_INT_ENABLE_REG gesetzt, dann wird ein ADC Interrupt ausgelöst, sobald die Wandlung fertig ist.

## Zusammenfassung der Register

Nach einem Reset sind sämtliche Bits, die die Einstellungen betreffen auf „0“ gesetzt. Bits, welche nicht in diesen Tabellen auftauchen, sind im Datenblatt als „reserved“ markiert und dürfen nicht mit „1“ beschrieben werden.

Basis-Adresse für alle ADC-Register: 0x13002000.

ADC_Rx_REG
Offset: 0x00, 0x04, 0x08, 0x0C
Beschreibung: ADC-Ergebnisse, 4 Register (ADC_R0_DATA - ADC_R3_DATA)

| Bit | Symbol | R/W | Beschreibung |
|---|---|---|---|
| 0-9 | ADC_Rx_DATA | Read Only | ADC-Ergebnis |

ADC_CON_REG
Offset: 0x20
Beschreibung: ADC Konfiguration

| Bit | Symbol | R/W | Beschreibung |
|---|---|---|---|
| 4 | ADC_STATUS | Read Only | 1 = Wandlung läuft |
| 3 | ADC_START | Read/Write | 1 = Starte Wandlung |
| 2 | ADC_CSCAN | Read/Write | 0 = Single Conv.<br>1 = Continuous Conv. |
| 1 | ADC_ENABLE | Read/Write | 1 = ADC aktiv |

ADC_CSEL_REG
Offset: 0x24
Beschreibung: Channel-/Auflösungskonfiguration

| Bit | Symbol | R/W | Beschreibung |
|---|---|---|---|
| 0-3 | CSEL0 | Read/Write | Auflösung Channel 0 |
| 4-7 | CSEL1 | Read/Write | Auflösung Channel 1 |
| 8-11 | CSEL2 | Read/Write | Auflösung Channel 2 |
| 12-15 | CSEL3 | Read/Write | Auflösung Channel 3 |

ADC_INT_ENABLE_REG
Offset: 0x28
Beschreibung: IRQ Erzeugung aktivieren/deaktivieren

| Bit | Symbol | R/W | Beschreibung |
|---|---|---|---|
| 0 | ADC_INT_ENABLE | Read/Write | 1 = IRQ aktiviert |

ADC_INT_STATUS_REG
Offset: 0x2C
Beschreibung: IRQ Status abfragen

| Bit | Symbol | R/W | Beschreibung |
|---|---|---|---|
| 0 | ADC_INT_STATUS | Read Only | 1 = IRQ anstehend |

ADC_INT_CLEAR_REG
Offset: 0x2C
Beschreibung: IRQ Status löschen

| Bit | Symbol | R/W | Beschreibung |
|---|---|---|---|
| 0 | ADC_INT_CLEAR | Write Only | 1 = ADC_INT_STATUS löschen |

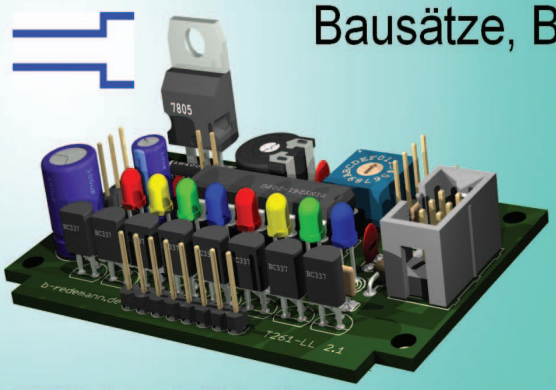
## Beispiel für eine Wandlung – Polling

```
#include <linux/module.h>
#include <asm/io.h>

#define ADC_INT_STATUS   (1 << 0)
#define ADC_INT_ENABLE   1
#define ADC_ENABLE (1 << 1)
#define ADC_START  (1 << 3)
#define ADC_STATUS (1 << 4)
#define ADC_SSCAN  (~(1<<2))              /* single scan */
#define ADC_STOP   (~ADC_START)

int init_module(void) {
   uint16_t r0, r1, r2, r3;

      SYS_ADC_PD = 0;                          /* ADC powerdown mode off */

            cgu_clk_en_dis(CGU_SB_ADC_CLK_ID, 1);   /* enable clock for ADC */
            cgu_clk_en_dis(CGU_SB_ADC_PCLK_ID, 1);
```

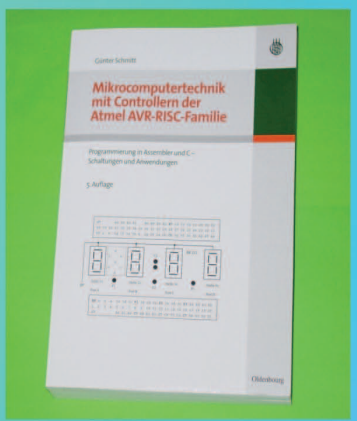
```
            ADC_CON_REG = ADC_CSEL_REG = 0;            /* reset registers to default */
            ADC_INT_ENABLE_REG = ADC_INT_ENABLE;       /* enable interrupt */
            ADC_CON_REG |= ADC_ENABLE;             /* power up ADC */

            ADC_CSEL_REG = 0x0000AAAA;             /* select channel and resolution */
            ADC_CON_REG &= ADC_SSCAN;                  /* single conversion mode */

            ADC_CON_REG |= ADC_START;                  /* start conversion */

            while(!(ADC_INT_STATUS_REG & ADC_INT_STATUS)); /* wait until conversion done */

            r0 = (uint16_t)ADC_R0_REG;
            r1 = (uint16_t)ADC_R1_REG;
            r2 = (uint16_t)ADC_R2_REG;
            r3 = (uint16_t)ADC_R3_REG;

            ADC_CON_REG &= ADC_STOP;

            printk(KERN_INFO „adc testing module: %03x %03x %03x %03x\n“, r0, r1, r2, r3);

            return 0;
}

void cleanup_module(void) {
            ADC_CON_REG &= ~(ADC_ENABLE);          /* disable ADC */

            ADC_CON_REG = 0;                  /* set ADC to default state */
            ADC_CSEL_REG = 0;
            ADC_INT_ENABLE_REG = 0;
            ADC_INT_CLEAR_REG = 0;

      cgu_clk_en_dis(CGU_SB_ADC_CLK_ID, 0);    /* disable clock signals */
      cgu_clk_en_dis(CGU_SB_ADC_PCLK_ID, 0);

      printk(KERN_INFO „adc testing module removed\n“);
      return;

}
```

Anzeige

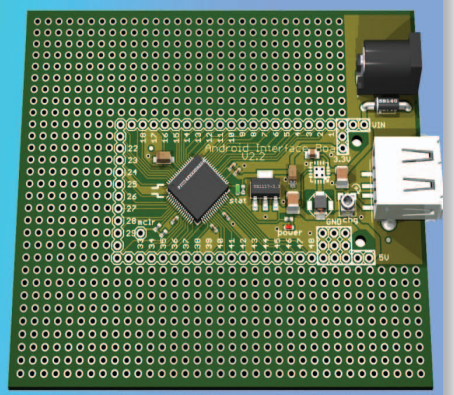
Bausätze, Bücher und Komplettsets



BS1009 - Universallauflicht und Steuerung
Mit dem Attiny861, Trimmer, Drehkodierschalter,
ISP. Ideal für Modellbahn und -bau oder Einsteiger
in die Welt der Elektronik und Mikrocontroller
**15,00 €***




BL9111 - Buch: Mikrocomputertechnik
mit Controllern von Atmel, Günter Schmitt
**39,80 €***



BS1013 - Bausatz Android Interface Board
(IOIO-Clone), 48 Ports frei steuerbar über ein
Androidgerät
**37,00 €***

Weitere Bausätze (u.a.):
BS1007 - DIAMEX All-in-one AVR Programmer 28,50 €
07103 - Bausatz AVR910-USB-Programmer 15,00 €
0B102 - Schrittmotormodul mit dem TMC222 V3.0 16,00 €

* Alle Preise inkl. MwSt.
zzgl. Versandkosten

Ing.-Büro B. Redemann
Mahlower Str. 204
14513 Teltow

Hier im Shop: www.b-redemann.de

## Beispiel – IRQ

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/interrupt.h>
#include <asm/io.h>

#define ADC _ ENABLE (1 << 1)
#define ADC _ START  (1 << 3)
#define ADC _ STATUS (1 << 4)
#define ADC _ CSCAN  (1 << 2)              /* continuous scan */
#define ADC _ SSCAN  (~ADC _ CSCAN) /* single scan */
#define ADC _ STOP   (~ADC _ START)
#define ADC _ INT _ ENABLE  1
#define ADC _ INT _ DISABLE  0
#define ADC _ IRQ _ PENDING  1
#define ADC _ IRQ _ CLEAR   1

#define ADC _ IRQ          9            /* interrupt vector 9 is ADC */

irqreturn _ t adc _ irq _ handler(int irq, void *dev _ id) {
   uint16 _ t r0, r1, r2, r3;
   static uint8 _ t runs;        /* we will report ad values only 8 times to syslog */

   while(ADC _ INT _ STATUS _ REG == ADC _ IRQ _ PENDING) ADC _ INT _ CLEAR _ REG = ADC _ IRQ _ CLEAR;
/* clear IRQ status */

   if(runs < 8) {
     r0 = (uint16 _ t)ADC _ R0 _ REG;
     r1 = (uint16 _ t)ADC _ R1 _ REG;
     r2 = (uint16 _ t)ADC _ R2 _ REG;
     r3 = (uint16 _ t)ADC _ R3 _ REG;



     printk(KERN _ INFO „adc values: $%03x $%03x $%03x $%03x\n", r0, r1, r2, r3);
     /* this is very bad programming practice, NEVER EVER
      * use printk() in interrupt handlers in a „real module"
      */

     runs++;

   }

   return IRQ _ HANDLED;

}


int init _ module(void) {
   int32 _ t retval;

   SYS _ ADC _ PD = 0;                          /* ADC powerdown mode off */

   cgu _ clk _ en _ dis(CGU _ SB _ ADC _ CLK _ ID, 1);     /* enable ADC clock */
   cgu _ clk _ en _ dis(CGU _ SB _ ADC _ PCLK _ ID, 1);

   ADC _ CON _ REG = ADC _ CSEL _ REG = 0;       /* reset registers to default */

   ADC _ CON _ REG |= ADC _ ENABLE;              /* power up ADC */
```

```
   ADC _ CSEL _ REG = 0x0000AAAA;        /* select channel and resolution */
   ADC _ CON _ REG |= ADC _ CSCAN;       /* continuous conversion mode */

   /* add an interrupt handler */
   retval = request _ irq(ADC _ IRQ, adc _ irq _ handler, IRQF _ SHARED, „lpc313x adc irq“, (void *)
(adc _ irq _ handler));

   ADC _ INT _ ENABLE _ REG = ADC _ INT _ ENABLE; /* enable ADC irq */

            printk(KERN _ INFO „adc irq testing module, we will spam to your syslog!!!\n“);

   ADC _ CON _ REG |= ADC _ START;               /* start conversion */

     return retval;

}

void cleanup _ module(void) {
   ADC _ INT _ ENABLE _ REG = ADC _ INT _ DISABLE;       /* disable ADC interrupt */
   ADC _ CON _ REG &= ADC _ STOP;         /* stop ADC */
   ADC _ CON _ REG &= ~(ADC _ ENABLE);           /* disable ADC */

   cgu _ clk _ en _ dis(CGU _ SB _ ADC _ CLK _ ID, 0);     /* disable clock signals */
   cgu _ clk _ en _ dis(CGU _ SB _ ADC _ PCLK _ ID, 0);

   free _ irq(ADC _ IRQ, (void *)(adc _ irq _ handler));        /* remove interrupt handler */

   printk(KERN _ INFO „adc testing module removed\n“);

   return;

}
```

## Hardware

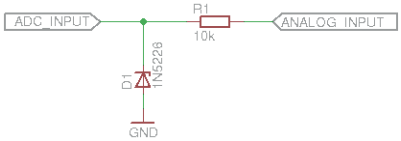
Da die maximale Eingangsspannung des AD-Wandlers bei 3.3V liegt, muss der User dafür sorgen, dass nie mehr angelegt wird. Eine einfache Methode hierfür wäre ein 10k Widerstand um den Strom zu Begrenzen und eine 3,3V Zener Diode.

Eine Auswahl an 3,3V Zener-Dioden:

- 1N476 (0.4W)
- 1N5226, 1N4620 (0.5W)

Auf dem Gnublin-Board sind die 3 ADC-Kanäle auf J5 als GPA0, GPA1 und GPA3 zu finden. Kanal 2 wurde nicht herausgeroutet.

Abb. 1: „disrekter“ DA-Wandler




## Beispielmodule

Im Anhang an diesen Artikel gibt es zwei Beispielmodule zum herunterladen. Das eine zeigt die Verwendung des ADCs im „Polling-Mode“, das andere im „IRQ-Mode“. Ebenfalls habe ich ein Beispielmodul bei den Links angehängt, welches im Interrupt den Wert von AD-Kanal 0 ausliest und diesen als PWM Wert wieder am PWM-Pin ausgibt. Den Makefiles muss die Zielarchitektur, das Cross Compiler-Prefix und das Verzeichnis mit den Kernelquellen mitgegeben werden, hier ein Beispielaufruf:

```
$ ARCH=arm CROSS _ COMPILE=arm-linux-gnueabi-
KDIR=/path/to/kernelsource/ make
```

## PWM

Die PWM-Einheit des LPC3131 ist im Vergleich zum AD-Wandler sehr einfach, die komplette Konfiguration läuft über zwei Register. Die „Grundfunktionen“ sind Pulsweitenmodulation und PDM („Pulse Density Modulation“). Für ein „100%-Signal“ kann der Ausgang fest auf 1 gesetzt werden und die Frequenz kann über 4 verschiedene Vorteiler beinflusst werden

### Funktionsbeschreibung

Im PWM-Modus wird ein 12 Bit-Counter kontinuierlich mit dem PWM-Wert im Register PWM_TMR verglichen. Der Ausgangspin ist solange HIGH, wie der Counter-Wert kleiner ist, als der Wert im Register PWM_TMR, andernfalls ist der Ausgangspin LOW. Solange noch kein PWM_TMR-Wert gesetzt wurde ist der Ausgangspin „tristated", sobald der PWM_TMR-Wert das erste mal gesetzt wird, wird der Ausgang aktiviert. Im PDM-Modus gibt der Wert im PWM_TMR-Register an, wieviele Clock-Cycles vergehen müssen, bevor ein PWM-Puls ausgegeben wird. Die Gesamtzahl der auszugebenden Pulse wird demnach mit dem Wert im PWM_TMR-Register angegeben. In beiden Modi kann das LOOP-Bit im PWM_CNTL-Register (Control Register) dazu verwendet werden um den Ausgangspin nach einer bestimmten Anzahl von Clock-Cycles zu toggeln. Die Anzahl der Clock-Cycles hierfür wird durch die obersten 4 Bits im PWM_TMR-Register vorgegeben.

### Clock, PWM-Frequenz und Duty-Cycle

Die PWM-Einheit verlangt drei verschiedene Clock-Signale von der CGU:

PWM_PCLK – APB Bus Clock

PWM_PCLK_REGS – APB Bus Clock für Zugriff auf Register
PWM_CLK – PWM Clock, wird für die PWM-Ausgabe verwendet

Die PWM_CLK-Frequenz liegt per Default bei 1.465kHz. Durch die verfügbaren Vorteiler von 1, 2, 4 und 8 lässt sich dieser Takt weiter herunterteilen.

Der Duty-Cycle lässt sich mit folgender Formel berechnen:

$$DutyCycle = \frac{TMR}{4095} * 100$$

Bei einem Wert von z.B. 2047 ergibt das ein Duty Cycle von ~49.99%.

Das Abfragen der PWM_CLK-Frequenz lässt sich leicht in einer Zeile erledigen:

```
uint32_t pwm_clk_freq = cgu_get_clk_
freq(CGU_SB_PWM_CLK_ID) / 4095;
```

Beispiel:

```
/* enable clock for PWM */
cgu_clk_en_dis(CGU_SB_PWM_PCLK_ID,
1);
cgu_clk_en_dis(CGU_SB_PWM_PCLK_REGS_
ID, 1);
cgu_clk_en_dis(CGU_SB_PWM_CLK_ID, 1);

PWM_CNTL_REG = 0;        /* PWM Mode */
PWM_TMR_REG = 2047;      /* Duty Cycle
~49.99% */
```

### Zusammenfassung der Register

Nach einem Reset sind sämtliche Bits, die die Einstellungen betreffen auf „0" gesetzt. Bits, welche nicht in diesen Tabellen auftauchen, sind im Datenblatt als „reserved" markiert und dürfen nicht mit „1" beschrieben werden.

Basis-Adresse für alle PWM-Register: 0x13009000.

PWM_TMR_REG
Offset: 0x00
Beschreibung: PWM Timer Register

| Bit | Symbol | R/W | Beschreibung |
|---|---|---|---|
| 0-11 | MR | Read/Write | 12 Bit Timer Wert für PWM |

PWM_CNTL_REG
Offset: 0x04
Beschreibung: PWM Konfiguration

| Bit | Symbol | R/W | Beschreibung |
|---|---|---|---|
| 0-1 | CLK | Read/Write | Vorteiler für PWM_CLK<br>00 = PWM_CLK<br>01 = PWM_CLK / 2<br>10 = PWM_CLK / 4<br>11 = PWM_CLK / 8 |
| 4 | HI | Read/Write | 1 = PWM-Pin dauerhaft HIGH |
| 6 | LOOP | Read/Write | 1 = Loop-Mode ein |
| 7 | PDM | Read/Write | 1 = PDM-Mode ein<br>0 = PWM-Mode ein |

### Hardware

Der PWM-Pin wurde auf die Stiftleiste J5 auf Pin 4 geroutet („PWM"). Über ein RC-Glied lässt sich hiermit ein recht einfacher DA-Wandler aufbauen, der z.B. für die Kontrast-Einstellung von Displays verwendet werden kann. Über einen entsprechenden Operationsverstärker können durch den DA-Wandler auch höhere Ströme getrieben werden, z.B. zum Ansteuern eines Transistors. Zum Steuern von induktiven Lasten wie Motoren, Spulen usw. braucht man am PWM-Pin natürlich einen passenden Leistungstransistor und auch die Freilaufdioden nicht vergessen! Der maximale Strom, der jedem digitalen I/O entnommen werden kann ist im Datenblatt mit 4mA bei den „limiting values" angegeben. Leider steht jedoch bei den „static characteristics" unter „IO pin configured as output" nur „tbd" (to be disclosured). Für Lasten, die mehr als 2mA benötigen würde ich deshalb dringend Transistoren/Treiber vorschlagen. Eine 2mA-Low-Current LED mit entsprechendem Vorwiderstand funktioniert bei mir tadellos direkt am Pin. Um das Board vor induzierten Überspannungen 100%ig zu schützen, sollte der PWM-Pin über einen Optopkoppler galvanisch von der Schaltung getrennt werden - ein Controller im 180-Pin-BGA-Gehäuse ist nicht so leicht gewechselt wie ein DIP-AVR ;)Über ein RC-Glied kann mit der PWM-Einheit natürlich auch eine analoge Spannung erzeugt werden. Dieser „diskrete" DA-Wandler hat jedoch den Nachteil, dass er

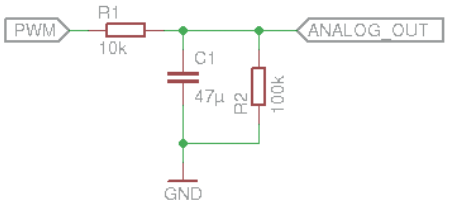



Abb. 2: RC Glied

nicht fähig ist, grosse Ströme zu treiben.

Der Kondensator C1 ist mit 47µF relativ gross. So entsteht eine sehr saubere Gleichspannung am Ausgang, jedoch dauert das „Umschalten“ zwischen zwei PWM-Werten einige Zeit. Für einen Wechsel von 0 auf 100 braucht dieser DA-Wandler, unbelastet, etwa 3 Sekunden. In dieser Zeit sieht man am Oszilloskop schön, wie die Spannung langsam ansteigt. Wenn man schnellere Reaktionszeiten braucht, muss man den Kondensator C1 und die Widerstände verkleinern. Je kleiner dieser jedoch wird, desto unsauberer wird das Ausgangssignal. Mit einem 100nF-Kondensator erreicht man sehr schnelle Reaktionszeiten, doch das Ausgangssignal ist im mittleren Bereich (Duty Cycle von ca. 20% bis 80%) mehr ein Sägezahn als sonst etwas. Die beiden Widerstände R1 und R2 sind natürlich auch für das Timing-Verhalten mitverantwortlich. R1 bestimmt die Zeit, die gebraucht wird um den Kondensator aufzuladen, R2 um ihn zu entladen. Durch den 1:10 Spannungsteiler, der sich aus R1 und R2 bildet ist es nicht möglich komplett an die 100%/3,3V heranzukommen. Der Spannungsbereich liegt bei etwa 0-3V. Braucht man größere Ströme kann man z.B. einen Operationsverstärker als Spannungsfolger einsetzen. Ein LM324 lässt so Ströme bis ca. 20mA zu.

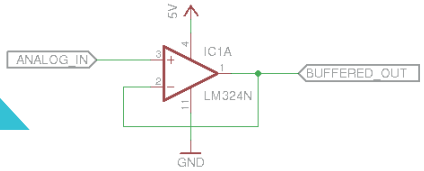



Abb. 3: Operationsverstärker

Und hier noch ein Ausschnitt vom Oszilloskopbild bei einem Duty-Cycle von 25%, die analoge Spannung nur durch das RC-Glied hergestellt, keinen OP-Buffer dahinter. Die analoge Spannung liegt bei etwa 0.75V (Gelb), die Rote Linie ist das PWM-Signal.

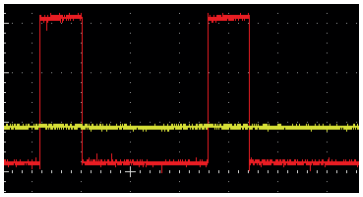


Abb. 4: Duty-Cycle 25%

## Links, Quellen und Danke

Dokumentation:

- Die erste Anlaufstelle für Dokumentation zur Hardware sollten das einfacher „LPC313x Datasheet“ sein, sowie das sehr ausführliche „LPC313x User Manual“.
- Die Linux- und Gnublin-spezifische Dokumentation findet sich auf http://www.gnublin.org.
- Ebenfalls ist das Gnublin-Thema auf mikrocontroller.net sehr informativ.

Quellen:

- Die Quellen zu diesem Artikel stellen im Großen und Ganzen die Kernel-Module von mir und dem User „misc“ dar, „Linux Device Drivers“ und „Linux Kernel Module Programming Guide“.

Danke:

- Bedanken möchte ich mich ganz herzlich beim Gnublin-Team, die mir eines ihrer Boards im Gegenzug für Code und Hilfe kostenlos zur Verfügung gestellt haben sowie beim User „misc“ von mikrocontroller.net, der sich als erster um ADC und PWM gekümmert hat, sowie einen grafischen Installer für die SD-Karte geschrieben hat.
- Ebenso hat er mich schon auf ein paar (böse) Bugs aufmerksam gemacht bzw. geholfen diese zu finden.

Stellenausschreibungen

Auf Jobsuche?

Dann besuchen Sie unseren Online-Stellenmarkt

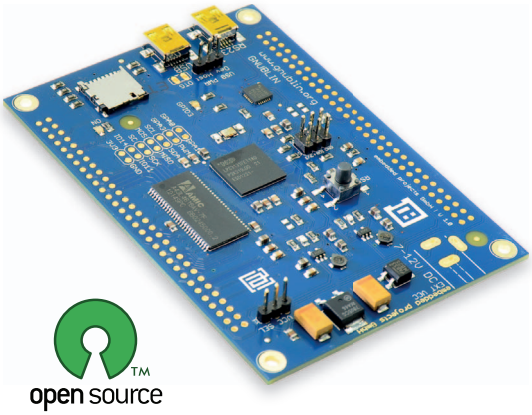
journal.embedded-projects.net/stellenmarkt

# Marktplatz / Neuigkeiten

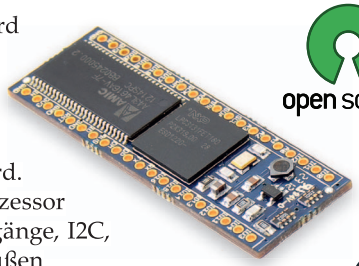
## Die Ecke für Neuigkeiten und verschiedene Produkte

### GNUBLIN Extended

- Nach langer Wartezeit endlich die große GNUBLIN Version mit allen Pins nach außen geführt, perfekt um Aufsteckboards zu entwickeln:
- 32 MB SDRAM (Jetzt neu mit 32 MB RAM!)
- Alle Pins nach außen geführt



### GNUBLIN DIP



Das GNUBLIN embedded GNU/Linux Board gibt es jetzt auch als DIP-Modul zum aufstecken. Auf GNUBLIN in der Version mit dem ARM9 Prozessor LPC3131 von NXP läuft ein vorinstalliertes Linux, welches von einer SD-Karte gebootet wird. Wesentliche Schnittstellen zum Mikroprozessor wie Digitale Ein- Ausgänge, Analoge Eingänge, I2C, SPI oder eine PWM-Leitung sind nach außen geführt:

- ARM9 Prozessor mit 180 MHz (LPC3131)
- 32 MB SDRAM (Jetzt neu mit 32 MB RAM!)
- vorinstalliertes GNU/Linux
- microSD Karten Steckplatz für Bootloader, Kernel, Dateisystem und Swap-Bereich
- USB-Device oder USB-Host Anschluss an Anschlussleisten
- 15 x GPIO, 4 x AD auf Anschlussleiste
- UART,SPI und I2C an Anschlussleisten
- 5V ca. 100mA Stromversorgung (internen Spannungen werden selbst erzeugt)
- 54.0 mm x 18.3 mm
- Standard DIP40 Gehäuse inkl. Präzesions Pin-Leisten

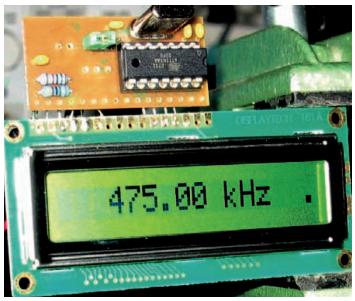
**Haben Sie interessante Produkte oder Neuigkeiten?
Dann senden Sie uns ein Foto, 3-4 Zeilen Text und einen Link mit Betreff „Marktplatz“ an:**

**journal@embedded-projects.net**

**Mehr Infos gibts auf der GNUBLIN Projektseite:**

**http://www.gnublin.org**

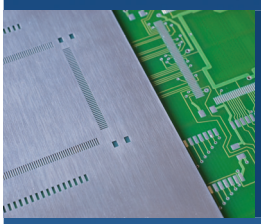
### Zählerplatine für Standard-LCD




Diese kleine Platine verwandelt ein Standard LCD in ein Zählermodul. Je nach eingestecktem uC erhalten Sie: einen sehr genauen Frequenzzähler 0.5Hz..5MHz, gleichbleibende Auflösung von 5 Stellen über den gesamten Bereich, mit eingebauter Abgleichroutine. Verschiedene Vorteilerverhältnisse wählbar; einen Ereigniszähler mit Triggerkontakt/TTL-Triggersignal. Verschiedene Entprellzeiten wählbar. Eine Stoppuhr mit Triggerkontakt/TTL-Triggersignal. Anzeigeformat ss:msms:usus oder hh:mm:ss

**Dies ist ein Produkt präsentiert im Experten-Wegweiser „Find your engineer“: http://www.find-your-engineer.de**

DAS ORIGINAL SEIT 1994
PCB-POOL®
Beta LAYOUT



FREE Stencil
bei jeder PCB Prototyp-Bestellung


Easy-going
17 akzeptierte Layoutformate

PCB-POOL® ist eine eingetragene Marke der Beta LAYOUT GmbH
Alle eingetragenen Warenzeichen sind eingetragene Warenzeichen der jeweiligen Hersteller!

www.pcb-pool.com

Beta LAYOUT
create:electronics

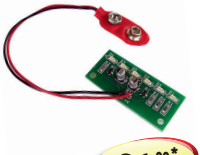
eSTORE®
Beta LAYOUT

Entwickeln, Löten und Bestücken


Arduino Mega (ATMega 1280-16AU)
kompatibel

Reflow-Controller


LED Wechselblinker
SMD-Bausatz




Big Beta-Reflow-Kit


Tool-Kit Extended


* inkl. MwSt. und zzgl. Versandkosten

www.beta-eSTORE.com

Beta LAYOUT
create:electronics

## Interesse an einer Anzeige?

**info@embedded-projects.net**

**Widerstands-Sortiment**

SMD0805, 1%, TK 100, RoHS
**62 Werte** E12-Reihe
**6200** Widerstände

€ 45,-
inkl. 19% MwSt zzgl. Versand

**http://www.FundF.net**

**Kleinrechner mit FPGA**

**www.bomerenzprojekt.de**

→ firma.embedded-projects.net

DAS HARDWARE FOR YOUR PROJECTS-PORTAL

In unserem Online-Shop finden Sie eine große Auswahl verschiedenster Mikrocontrollerboards, Programmer, Debugger u.v.m.
→ shop.embedded-projects.net

Unser Büro in Augsburg besteht aus leidenschaftlichen Entwicklern. Sprechen Sie uns an, wir finden eine Lösung für Ihr Problem.
→ projekte.embedded-projects.net

Speziell für Studenten und Hochschulen, bieten wir diese Ausbildungsinitiative an. Mikrcontrollerboards für den kleinen Geldbeutel.
→ student.embedded-projects.net

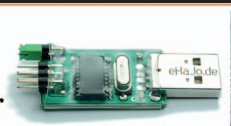
Holzbachstraße 4, D-86152 Augsburg
Tel +49 (0) 821 279599-0
Fax +49 (0) 821 279599-20
info@embedded-projects.net

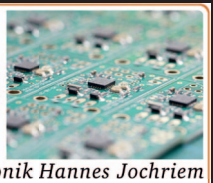
embedded projects GmbH
HARDWARE FOR PROJECTS

**eHaJo**

*Selbstgebastelter Programmieradapter abgeraucht?*

*Macht nix: für 6,90€ gibt's nen neuen!*





*Elektronik Hannes Jochriem*
*Bausätze, Lötübungen uvm.*
*www.eHaJo.de*
*info@ehajo.de*

**FIND** www.f-y-e.de

***your engineer***

Der Experten-Wegweiser
zu Ihrem Elektronikentwickler

- Elektronik- / Softwareentwicklung
- Layout
- Mechatronik
- Bestücker / EMS-Dienstleister
- EMV-Dienstleister

Find-Your-Engineer ist ein persönliches Empfehlungsnetzwerk. Firmen die Elektronik-Experten suchen, wenden sich bitte direkt an:

Markus Kessler
kontakt@find-your-engineer.de

*Mit der besten Empfehlung !*

MIXED MODE

**Software-Entwickler/in gesucht! (München)**

UML
MDA
C++
C# .NET
Embedded Software
Realtime
Java

**www.mixed-mode.de/jobs**

embedded - projects.net
# JOURNAL
OPEN SOURCE SOFT-AND HARDWARE PROJECTS

Bitte frei machen

embedded projects GmbH
Holzbachstraße 4
D - 86152 Augsburg

Name / Firma

Straße / Hausnummer

PLZ / Ort

Email / Telefon / Fax

◯ Ich möchte jede zukünftige Ausgabe erhalten

◯ Wir möchten als Hochschule / Ausbildungsbetrieb jede weitere Ausgabe bekommen. Bitte gewünschte Anzahl der Hefte pro Ausgabe ankreuzen. 5 10

◯ Ich möchten im embedded projects Journal werben oder eine Stellenanzeige aufgeben. Bitte schicken Sie mir Infomaterial, Preisliste etc. zu.

## Werdet aktiv!

Das Motto: Von der Community für die Community !

Das Magazin ist ein Open Source Projekt.

Falls Du Lust hast, Dich an der Zeitschrift durch einen Beitrag zu beteiligen, würden wir uns darüber sehr freuen. Schreibe deine Idee an:

journal@embedded-projects.net

## Regelmäßig

Die Zeitschrift wird über mehrere Kanäle verteilt. Der erste Kanal ist der Download als PDF - Datei. Alle Ausgaben sind auf der Internetseite [1] verfügbar. Diejenigen, die lieber eine Papierversion erhalten möchten, können den zweiten Kanal wählen. Man kann sich dort auf einer Internetseite [2] in eine Liste für die gesponserten Abos oder ein Spendenabo eintragen. Beim Erscheinen einer neuen Ausgabe wird dank Sponsorengeldern an jeden auf der Liste eine Ausgabe des aktuellen Journal versendet. Falls man den Versandtermin verpasst hat, kann man das Heft auch über einen Online - Shop [2] beziehen.

[1] Internetseite (Anmeldeformular gesponserte Abos): http://journal.embedded-projects.net

[2] Online - Shop für Journal:
http://www.embedded-projects.net

## Sponsoren gesucht!

Damit wir weiterhin diese Zeitschrift für jeden frei bereitstellen können, suchen wir dringend Sponsoren für Werbe- und Stellenanzeigen. Bei Interesse meldet Euch bitte unter folgender Telefonnummer: 0821 / 2795990 oder sendet eine E-Mail an die oben genannte Adresse.

## Impressum


embedded projects GmbH
Holzbachstraße 4
D-86152 Augsburg
Telefon: +49(0)821 / 279599-0
Telefax: +49(0)821 / 279599-20

Veröffentlichung: 4x / Jahr
Ausgabenformat: PDF / Print
Auflagen Print: 2500 Stk.
Einzelverkaufspreis: 1 €

Layout / Satz: EP
Druck: flyeralarm GmbH
Titelfoto: Claudia Sauter



Alle Artikel in diesem Journal stehen unter der freien Creativ Commons Lizenz. Die Texte dürfen, wie bekannt von Open Source, modifiziert und in die eigene Arbeit mit aufgenommen werden. Die einzige Bedingung ist, dass der neue Text ebenfalls wieder unter der gleichen Lizenz, unter der dieses Heft steht veröffentlicht werden muss und zusätzlich auf den originalen Autor verwiesen werden muss. Ausgenommen Firmen- und Eigenwerbung.

Dies ist ein Open Source Projekt.

Except where otherwise noted, this work is licensed under http://creativecommons.org/licenses/by/3.0

# BESSER GLEICH ONLINE KALKULIEREN.

## STARRE UND FLEXIBLE LEITERPLATTEN.


ab 12 h Leiterplatten-Express! Pünktlich oder gratis!

LEITON
RECHNEN SIE MIT BESTEM SERVICE

Endlich wird's einfach und Sie bleiben flexibel. Den umständlichen und aufwändigen Kalkulationsprozessen machen wir einen dicken Strich durch die Rechnung. Sie kalkulieren Ihre Leiterplatten online – ganz bequem, ganz schnell. **Einzigartig: Die Online-Kalkulation gilt auch für Serien und flexible Leiterplatten.** Das macht uns weltweit so schnell keiner nach. Einmalig ist zudem der **Leiterplatten-Expressdienst** von LeitOn mit unserer Garantie: Wenn wir bei Expressdiensten den vereinbarten Übergabetermin an den Versender nicht einhalten können, schenken wir Ihnen die bestellten Platinen! Sie möchten mehr darüber wissen? Wir bieten persönliche Beratung am Telefon und einen kompetenten Außendienst. Sie können bei LeitOn immer mit dem besten Service rechnen.

**www.leiton.de** | **kontakt@leiton.de** | Info-Hotline **+49 (0)30 701 73 49 0**