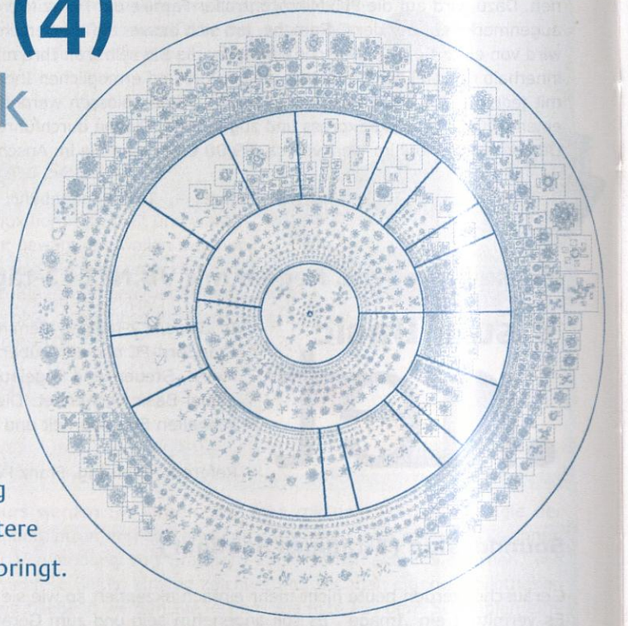
Embedded Linux leicht gemacht (4)

Ein Kernel-Überblick



Von Benedikt Sauter [1]

GNU/Linux bietet Schnittstellen für Geräte und Anwendungen an, die man von modernen Desktop-Computern oder Servern kennt. Auch im Embedded-Bereich nimmt Linux dem Programmierer viel Arbeit ab: für Netzwerk, USB, Bluetooth und mehr muss man nicht mehr aufwendig eigene C-Lösungen entwickeln. Und es gibt noch weitere Vorzüge, die solch ein modernes Betriebssystem mitbringt.

In den letzten Folgen der Serie haben wir die Toolchain, den Kernel und Bootloader sowie ein Standard-Dateisystem zum Laufen gebracht. In unserer Artikelreihe soll dem Entwickler aber auch gezeigt werden, was „unter der Oberfläche" passiert, zum Beispiel bei der Geräte- und Prozessverwaltung. Wir wollen uns in dieser Folge zuerst einmal anschauen, was das Betriebssystem im Vergleich zur klassischen, direkt auf der Hardware laufenden Firmware („Bare Metal") an Mehrwert bietet.

Was ist überhaupt ein Betriebssystem? Im Wesentlichen ist das eine Software-Sammlung, die man benötigt, um eigene Anwendungen parallel mit anderen ablaufen lassen zu können. Der Übergang von „normaler" Firmware zu einem echten Betriebssystem ist fließend. Da geht es los mit einem sauber konstruierten Hardware-Layer, der immer wieder benutzte Funktionen (zum Beispiel zur Speicherverwaltung) mitbringt. Und auch ein kleiner Scheduler, der einzelnen Programmen zyklisch Rechenzeit zuteilt, ist schon sehr „betriebssystem-like".

Typisches Betriebssystem

Natürlich gibt es zum Thema eine Vielzahl von Büchern [2]. Uns soll hier ein grober Überblick über die wesentlichen Komponenten eines Betriebssystems genügen:

- Geräteverwaltung
- Prozessverwaltung
- Speicherverwaltung
- Dateiverwaltung
- Benutzer- und Rechteverwaltung

Im Embedded-Bereich ist wohl die Geräteverwaltung besonders bedeutsam. Das Betriebssystem muss die Peripherie und Hardware „abbilden", so dass von der Anwendung aus ein einfacher Zugriff möglich ist. Dieser wird durch Gerätetreiber gesteuert. Das Betriebssystem regelt auch alles Nötige, wenn gleichzeitig mehrere Programme auf SD-Karten, Netzwerkschnittstellen usw. zugreifen wollen.
Eine Anwendung ist in einem Betriebssystem (sofern man kein aufwändigeres Programmiermodell wählt) ein einzelner, selbständig ausführbarer Prozess. Ein Prozess hat einen eigenen virtuellen Adressraum, wo er keinen anderen Prozess stören kann. Zentral werden Prozesse in der Prozesstabelle [3] verwaltet, dort gehört zu jedem ein Prozesskontrollblock („Process Control Block"). In diesem findet man die typischen Speicherbereiche, wo Registerinhalte und andere wichtige Variablen zwischengespeichert werden:

- Befehlszähler
- CPU-Register
- Stack-Pointer
- Zustand geöffneter Dateien
- Zustand des Prozesses

etc.

Starten wir unser Programm „Hello World", dann legt der Kernel beim Laden der Anwendung einen neuen Prozess an. Es wird immer vom aktuellen Prozess aus ein neuer abgeleitet, wobei auch die Benutzerrechte zuerst einmal vererbt werden. Mit dem Befehl

```
ps -ejH
```

kann man sich die dadurch entstehende, baumartige Struktur anzeigen lassen. Man sieht, dass der erste vom Kernel aufgerufene Prozess immer „/sbin/init" ist. Dieses Programm wird beim Start als allererstes vom Kernel aufgerufen.

Als Entwickler interessieren uns besonders die folgenden Features des Betriebssystems:

- Schnittstelle für Gerätetreiber
- Dateisystem für Programme und Daten
- Ausführen und Starten von Programmen
- Gemeinsame Nutzung von Bibliotheken durch verschiedene Programme
- Zugriff auf Ein- und Ausgabeschnittstellen
- Auslagern von Arbeitsspeicher auf einen nichtflüchtigen Speicher
- Speicherverwaltung für Anwendungen bzw. Prozesse
- Prozessverwaltung

Beginnen wir einmal mit der Geräteverwaltung.

Konfiguration des Kernels

Abhängig von der Anwendung werden verschiedene Schnittstellen und Peripherieblöcke des eingesetzten Prozessors benötigt. Der Linux-Kernel bietet durch seinen modularen Aufbau eine sehr feine Einstellmöglichkeit dessen an, was an Treibern und Mechanismen in den Kernel mit aufgenommen werden soll. Andere Teile kann man später dynamisch nachladen. Wie das alles funktioniert, schaut man sich am besten direkt im Quelltext des Kernels an.
Dafür wechseln wir in den Quelltextbaum des Kernels...

```
cd ElektorLinuxBoardDownload_20120509

cd linux-2.6.33-lpc3131x
```

...und rufen dort das Menü des Kernels auf:

```
make menuconfig
```

Direkt danach öffnet sich das Hauptmenü (**Bild 1**). Sollte das Menü nicht erscheinen, kann es sein, dass die Bibliothek für das Anzeigen solcher Konsolen-Menüs fehlt. Diese kann man einfach auf unserem Ubuntu-System nachinstallieren:

```
sudo apt-get install libncurses5-dev
```

Vom Hauptmenü des Kernels aus kann man alle Features und Gerätetreiber frei ein- und ausschalten. Für den Neuling ist es leider nicht immer ganz eindeutig, wo etwas im Menü zu finden ist, aber mit etwas Übung kennt man sich schnell aus. Anstelle das Menü zu verwenden, könnte man auch direkt im Ordner des Quelltextes (linux-2.6.33-lpc3131x) die Konfigurationsdatei „.config" (der Punkt bedeutet, dass es sich um eine versteckte Datei handelt) aufsuchen und mit einem Editor editieren. Aber bequemer geht alles

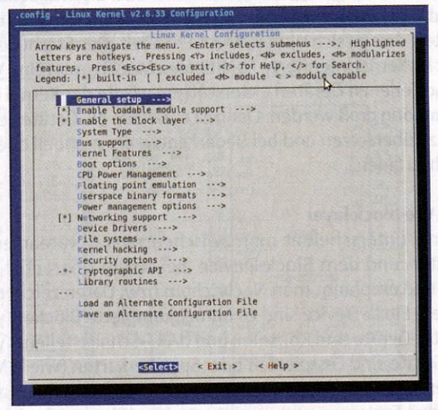


Bild 1. Hauptmenü des Kernels.

mit diesem Menü. Bevor man mit dem Bearbeiten der Konfiguration beginnt, sollte man sich aber eine Sicherheitskopie der aktuellen Konfiguration anlegen.

```
cp .config config_backup
```

Tipp: Viele Linux-Einsteiger fragen immer nach einer guten Dokumentation. Die beste Dokumentation für den Kernel ist im Kernel selbst zu finden. Zum einen kann man im Menü jederzeit auf „Hilfe" klicken und Informationen zum jeweiligen Item abrufen, oder man stöbert einmal im Ordner „Documentation" (direkt im Quelltextverzeichnis) nach dem passenden Thema.

In Bild 1 sieht man, dass die Menüpunkte nach Gruppen sortiert sind. Es geht los mit allgemeinen Einstellungen zum Kernel, und dann weiter über die Gerätetreiber und Dateisysteme bis hin zu Sonderfunktionen wie Verschlüsselung und Sicherheit.

General Setup
Im Menüpunkt „General Setup" findet man grundlegende Kernel-Eigenschaften. So z.B. ob es einen Auslagerungsspeicher gibt, wie der Speicher verwaltet wird, wie die interne Kommunikation umgesetzt wird. Dazu kommen die Eigenschaften des Kernel-Images selbst, Optimierungen für kleine Systeme, Statistik, Tools und vieles mehr. Wenn man sich mit dem Linux-Kernel beschäftigt, findet man nach und nach immer mehr nützliche Features. Es gibt wohl auch nur wenige Menschen, die alle Funktionen kennen und verstehen!

Enable loadable module support
Das Gute an Linux ist, dass man den Kernel zur Laufzeit ganz einfach mit Modulen erweitern kann. Im Kernel-Menü erkennt man,

ob ein Feature bzw. Treiber fest im Kernel oder als Modul übersetzt ist; und zwar an der Markierung vor dem Menüeintrag. Ein „M“ bedeutet, dass es sich um ein Modul handelt. Das Sternchen steht für fest im Kernel integrierte Funktionen. Möchte man eine Vielzahl an verschiedenen Geräten unterstützen, würde der Kernel irgendwann unnötig groß werden. Dann ist es besser, die Gerätetreiber als Modul zu übersetzen und bei Bedarf entweder manuell oder automatisch zu laden.

Enable the block layer
Im Kernel unterscheidet man zwischen zwei Gerätearten: dem Character- und dem Block-Device. Bei einem Character-Device sendet und empfängt man Nachrichten bzw. Daten zeichenweise. Bei einem Block-Device sind es immer komplette Blöcke. Typische Character-Geräte sind Konsolen und UART-Schnittstellen. Typische Block-Geräte sind Festplatten oder Speicherkarten (wie MMC und SD). Benötigt man keine Block-Devices (kommt am ehesten bei Embedded-Systemen vor), kann man etwas Platz im Speicher sparen, wenn man den „Block Layer“ deaktiviert.

System Type
Einige Features des Kernels hängen direkt vom Prozessor-Typ ab. So kann man hier z.B. gezielt Caches oder Speicherverwaltungseinheiten ein- und ausschalten. In diesem Unterpunkt stellt man (sofern schon vorhanden) den vorliegenden Prozessor-Typ ein. Fehlt der eigene Prozessor, muss man sich ein Patch bzw. entsprechendes Board-Support-Package suchen (oder selbst schreiben).

Bus support
Typische PC-Busse kann man hier aktivieren. In unserem Fall steht nur die Option PCMCIA zur Verfügung.

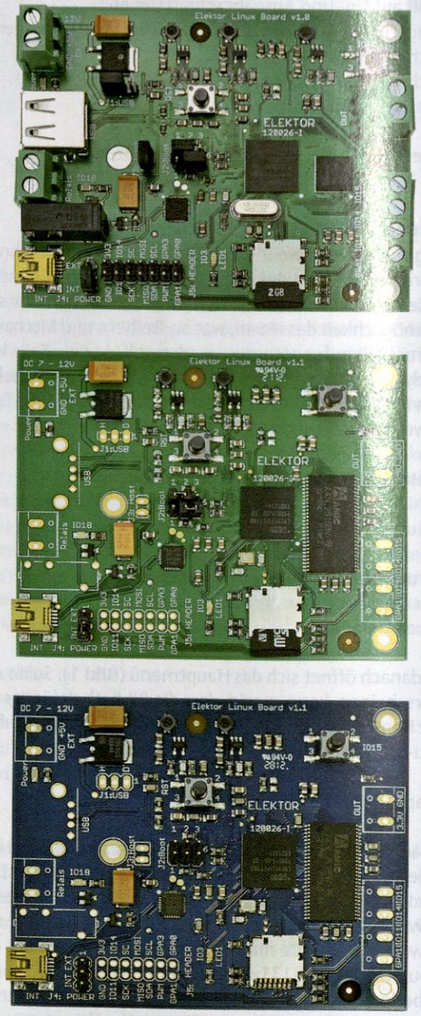
Kernel Features
Sehr kernel-spezifisch: Man kann hier einstellen, wie die Binärschnittstelle für Programme aussieht oder auch solche Optionen wie die Anzeige der Auslastung anhand der Blinkfrequenz einer LED.

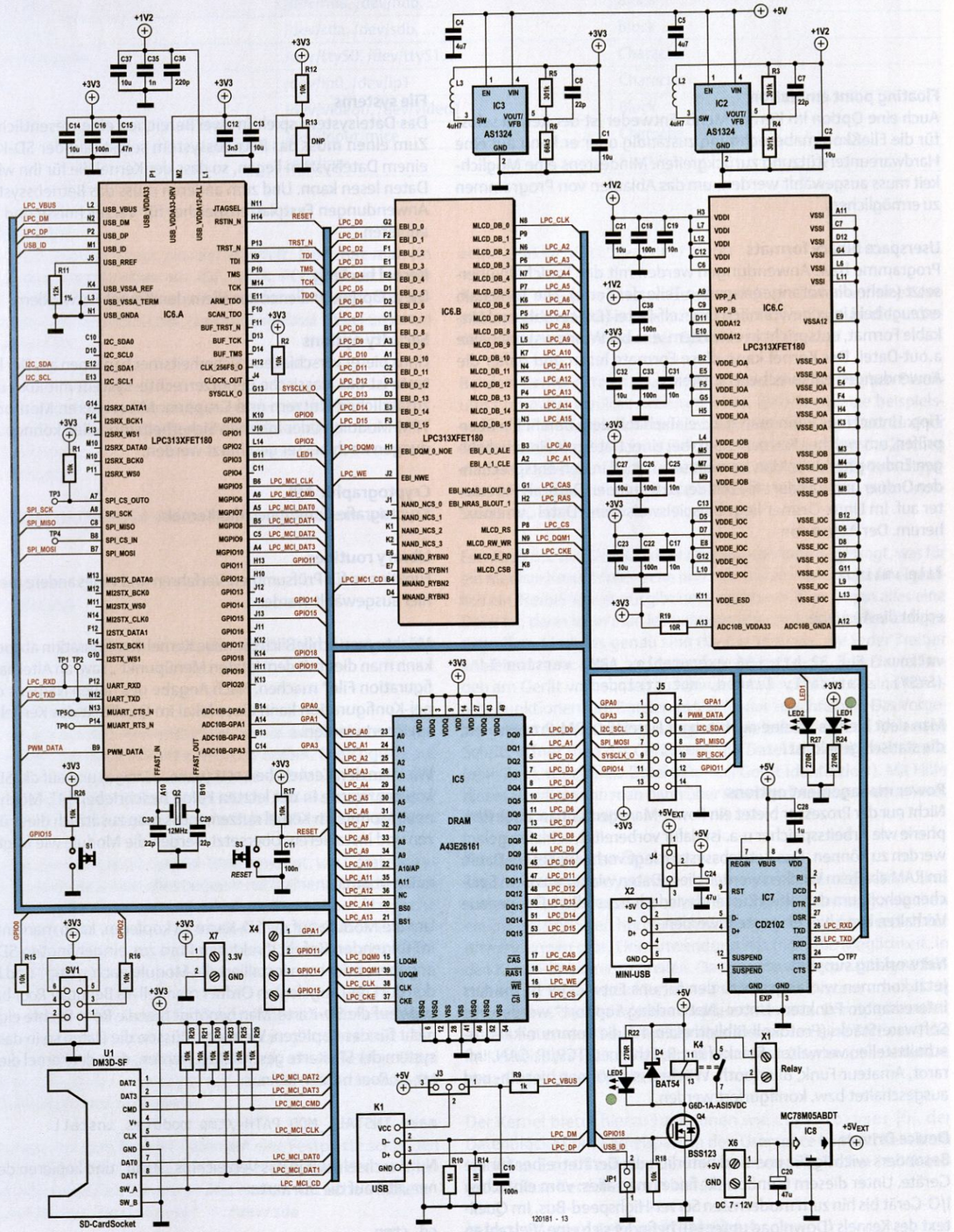
Boot options
Auf welche Weise der Kernel gestartet wird, hängt beim Elektor-Linux-Board im Wesentlichen vom Bootloader ab. Primäre Schnittstelle vom Bootloader zum Kernel ist die sogenannte „Kernel Commandline“. In dieser Zeile kann man dem Kernel Parameter übergeben, die später von diesem oder anderen Programmen ausgewertet werden können. In diesem Unterpunkt könnte man einstellen, dass eine andere, eigene Commandline eingesetzt werden soll. Oder man lässt den Kernel so übersetzen, dass man ihn direkt aus einem Flashspeicher starten lassen kann.

CPU Power Management
Moderne Prozessoren bieten immer mehr Unterstützung für ein gutes Power-Management an. Damit diese Features von Anwendungen genutzt werden können, muss sie der Kernel entsprechend implementiert haben und anbieten.

Neue Version des Elektor-Linux-Boards

Der Ansturm auf das Elektor-Linux-Board war wirklich gewaltig! Aufgrund der zahlreichen Bestellungen musste die nächste Produktions-Charge schneller aufgesetzt werden als erwartet. Leider hat der DRAM-Speicherchip im BGA-Gehäuse eine Lieferzeit von rund 20 Wochen. Daher fiel die Entscheidung, das Layout auf die Bauform TSOP54 umzustellen. Der neue Arbeitsspeicher ist genauso groß, benötigt aber anstatt 1,8 V eine Stromversorgung von 3,3 V. Das obere Bild zeigt die ursprüngliche Board-Version mit der BGA-Bauform des Arbeitsspeichers. Die Board-Version im mittleren Bild ist bereits auf einen DRAM-Chip im TSOP54-Gehäuse umgestellt. Ab Anfang Juli wird die Board-Version 1.1-blau ausgeliefert, bei welcher der nicht mehr benötigte 1,8-V-Spannungsregler entfallen ist. Diese aktuellste Version des Elektor-Linux-Boards ist leicht an der blauen Platinenfarbe zu erkennen. Den dazugehörigen Schaltplan haben wir hier abgebildet.





Floating point emulation
Auch eine Option im Kernel-Menü: Entweder ist der Kernel selbst für die Fließkommaberechnung zuständig oder er kann auf eine Hardwareunterstützung zurückgreifen. Mindestens eine Möglichkeit muss ausgewählt werden, um das Ablaufen von Programmen zu ermöglichen.

Userspace binary formats
Programme bzw. Anwendungen werden mit der Toolchain übersetzt (siehe die vorangegangenen Teile der Serie). Die Toolchain erzeugt bei Linux gewöhnlich eine .elf-Datei (Executable and Linkable Format, entspricht in etwa dem .exe bei Windows) oder eine a.out-Datei. Der Kernel kann diese Formate lesen und daher die Anwendungen entsprechend starten.

Tipp: Unter Linux kann man ganz einfach mit dem Befehl file überprüfen, um welches Format es sich bei einer Datei mit einer beliebigen Endung handelt. Man bewegt sich mit cd in den entsprechenden Ordner und ruft dort file mit der zu testenden Datei als Parameter auf. Im Linux-Ordner liegt beispielsweise eine Datei „vmlinux" herum. Der Aufruf von

```
file vmlinux
```

ergibt die Ausgabe:

```
vmlinux: ELF 32-bit LSB executable, ARM, version 1
(SYSV), statically linked, not stripped
```

Man sieht hier: Es ist eine ausführbare Datei für ARM-Prozessoren, die statisch gelinkt ist.

Power management options
Nicht nur der Prozessor bietet ein Power-Management an, auch Peripherie wie Arbeitsspeicher u.a. ist dafür vorbereitet, schlafen gelegt werden zu können. Das Betriebssystem legt vorher wichtige Daten im RAM ab. Beim Wecken werden diese Daten wieder aus dem Speicher geholt, um den alten Zustand wiederherzustellen. Das genaue Verhalten kann hier eingestellt werden.

Networking support
Jetzt kommen wir langsam zu den für uns Entwickler besonders interessanten Punkten. Unter „Networking Support" werden alle Software-Stacks (Protokoll-Bibliotheken) für die Kommunikationsschnittstellen verwaltet. Stacks für z.B. Ethernet, TCP/IP, CAN, Infrarot, Amateur-Funk, Bluetooth, Wireless etc. können hier ein- und ausgeschaltet bzw. konfiguriert werden.

Device Drivers
Besonders wichtig für uns sind natürlich die Gerätetreiber für die Geräte. Unter diesem Menüpunkt findet man alles: vom einfachen I/O-Gerät bis hin zum modernsten Server-Highspeed-Bus. Im Quelltext des Kernels (Download unter [4]) befindet sich eine Vielzahl an USB-, I2C- und SPI-Geräten, auf die wir zurückgreifen können.

File systems
Das Dateisystem spielt in zwei Bereichen eine wesentliche Rolle. Zum einen muss das Betriebssystem selbst (auf der SD-Karte) in einem Dateisystem liegen, so dass der Kernel die für ihn wichtigen Daten lesen kann. Und zum anderen muss das Betriebssystem den Anwendungen Festplattenspeicher für Verzeichnisse und Dateien anbieten.

Kernel hacking
Diese Optionen dienen vor allem den Kernel-Entwicklern.

Security options
Linux bietet verschiedene Sicherheitsmechanismen an. Der bekannteste ist das klassische Benutzerrechte-System mit *Root* und den zusätzlichen Nutzern und Gruppen. Mit weiteren Methoden wie TPM-Modulen oder internen Sicherheitsmodellen können Anwendungen noch besser geschützt werden.

Cryptographic API
Kryptografie-Funktionen des Kernels.

Library routines
Funktionen für Prüfsummen-Verfahren und vieles anderes mehr kann hier ausgewählt werden.

Möchte man schließlich die neue Kernel-Konfiguration abspeichern, kann man dies mit dem letzten Menüpunkt „Save an Alternate Configuration File" machen. Nach Angabe des neuen Namens der Kernel-Konfiguration kann diese lokal im Quelltext des Kernels abgespeichert werden.

Wie man den Kernel übersetzt (`make zImage`) und auf die SD-Karte kopiert, wurde in der letzten Folge beschrieben [4]. Möchte man neue Module im Kernel nutzen, muss man zusätzlich diese übersetzen und installieren. Übersetzt werden die Module wie folgt:

```
make modules
```

Um die Module auf die SD-Karte zu kopieren, kann man entweder im folgenden Befehl direkt den Pfad zur eingehängten SD-Karte angeben, oder man installiert die Module nach „/tmp" und kopiert den dort neu angelegten Ordner manuell als Benutzer *Root* bzw. mit sudo auf die SD-Karte. Man benötigt hier die Root-Rechte eigentlich nicht für das Kopieren. Vielmehr müssen die Daten so in das Dateisystem der SD-Karte geschrieben werden, dass der Kernel diese später als *Root* nutzen kann.

```
make INSTALL_MOD_PATH=/tmp modules_install
```

Nun wechseln wir in das Verzeichnis „/tmp" und kopieren den Ordner „lib" auf die SD-Karte:

```
cd /tmp
sudo cp -R lib /media/86b3be7-00f3-45e0-832e-1f48c2c3065e
```


Tabelle 1. Typische Gerätedateien im /dev/-Verzeichnis

Geräteklasse	Datei in /dev/	Character- oder Blockdevice
IDE-Festplatte	/dev/hda, /dev/hdb, ...	Block
SCSI-Festplatte	/dev/sda, /dev/sdb, ...	Block
RS232-Schnittstelle	/dev/ttyS0, /dev/ttyS1, ...	Character
Drucker	/dev/lp0, /dev/lp1	Character
Kamera	/dev/video0, /dev/video1	Block
Maus	/dev/input/mice	Character
RAM	/dev/ram	Block

Schnittstellen
Ursprünglich stammt Linux aus der PC-Welt. Dort hat man im Wesentlichen Schnittstellen wie IDE, SATA, PCI, AGP etc. In der Embedded-Welt liegt der Fokus auf den üblichen Mikrocontroller-Schnittstellen und natürlich USB. Zusammengefasst ist bei unserem Board vorhanden:

- USB (als Host und Device)
- UART (aktuell als Systemkonsole, kann aber auch als Schnittstelle für weitere Geräte verwendet werden)
- I2C (als Master)
- SPI (als Master)

Parallel bietet der Prozessor zudem Hardwarefunktionen wie:

- GPIO
- A/D-Wandler
- PWM-Ausgang

Nach und nach werden wir uns diese Schnittstellen genauer anschauen.

Gerätedateien
Um die Geräte aus den Anwendungen heraus ansprechen zu können, benötigt man eine Schnittstelle zum Treiber. Die Zugriffe auf Treiber werden unter den verschiedenen Betriebssystemen auf unterschiedliche Weise gelöst. Linux ist ein UNIX-Dateisystem, und unter UNIX ist das Konzept ganz einfach: Jedes Gerät wird als Datei behandelt.
Die Idee ist, dass man keine Spezial-Tools benötigt, um Geräte anzusteuern. Sondern dass man alles bequem mit kleinen, einfachen Programmen (von der Konsole aus) erledigen kann, über klassische Datei-Operationen. Wie das am Beispiel eines GPIO-Pins funktioniert, haben wir bereits im zweiten Artikel [5] anhand einer blinkenden LED gezeigt.

Im Wesentlichen gibt es zwei Befehle, mit denen man bereits viel erreichen kann: `cat` und `echo`. Der Befehl `cat` gibt den Inhalt einer Datei aus. Und mit `echo` kann man Daten in eine beliebige Datei schreiben; oder, genauer gesagt, man kann die Ausgabe von `echo` in eine beliebige Datei umlenken.

Möchte man nun zum Beispiel Daten auf eine Festplatte schreiben (direkt, nicht über das Dateisystem) oder Daten an der seriellen Schnittstelle ausgeben, kann man dies einfach mit `echo` machen:

```
echo „Daten auf Festplatte" > /dev/sda
```

oder

```
echo „Hallo Welt" > /dev/ttyS0
```

Achtung: Wenn man testweise einmal direkt Daten auf eine Festplatte oder Speicherkarte schreiben möchte, sollte man dies unbedingt mit einem unbenutzten Speichermedium tun!

Bei den Gerätedateien muss man eigentlich nur zwischen Block- und Character-Devices unterscheiden. Eine Festplatte beispielsweise wird immer blockweise angesprochen, im Gegensatz zu einer RS232-Schnittstelle, welche ein klassisches Terminal ist, das Zeichen einzeln empfangen und verarbeiten kann. In der **Tabelle 1** sind typische Gerätedateien aufgelistet, wie sie in einem UNIX- oder Linux-System zu finden sind.

Eventuell hat sich der eine oder andere Leser bereits gefragt, was für ein Mechanismus hinter `echo` und `cat` steckt – wieso damit eigentlich ein Treiber etwas ausgibt bzw. annimmt. Nun, wenn alles eine Datei ist, dann kann man lesend oder schreibend auf diese Dateien zugreifen. Und das genau sind die Funktionen, die jeder Treiber anbieten muss: `Read`, `Write` und als dritte `Ioctl` (um Einstellungen am Gerät vornehmen zu können). `Read` und `Write` sind klassische Funktionen, um Dateien zu lesen oder zu schreiben. Das Vorgehen ist bei einem Gerät genau gleich wie bei einer Datei. Im ersten Schritt öffnet man das Gerät oder die Datei und erhält ein *Handle* (eine Zahl, welche die Datei oder das Gerät identifiziert). Mit Hilfe dieses *Handles* kann man dann über `Read` und `Write` mit dem Gerät oder der Datei arbeiten.

Kernel- und User-Space
Klassische Betriebssysteme – zu denen auch Linux gehört – sind meist so aufgebaut, dass es eine besondere Trennung zwischen einem Kernel-Speicherbereich und einem Speicherbereich für Anwendungen gibt. Eine Anwendung hat nicht die Möglichkeit, in den Kernel-Speicher zu greifen. Das ist eine weitere wichtige Information für den Entwickler, der sehr nahe am System arbeitet. In Linux gibt es den Kernelspace und den Userspace. Prinzipiell handelt es sich bei einem Gerätetreiber aber immer um eine Software, die im Kernelspace läuft. Wie kommt man dann überhaupt an empfangene Daten einer Schnittstelle?

Der Kernel bietet hierzu Funktionen wie `copy_to_user` [6], der Datenblöcke vom Kernelspace in den Userspace kopiert. Bei einem lesenden Zugriff auf eine Gerätedatei muss ein Treiber im Hintergrund diese Kernelfunktion aufrufen.

Integration eines Gerätetreibers
Jetzt wissen wir, wie ein Treiber im Kernel prinzipiell arbeitet. Doch

```
root@gnublin:~# cat /proc/devices
Character devices:
  1 mem
  4 /dev/vc/0
  4 tty
  4 ttyS
  5 /dev/tty
  5 /dev/console
  5 /dev/ptmx
  7 vcs
 10 misc
 13 input
 21 sg
 89 i2c
128 ptm
136 pts
153 spi
180 usb
189 usb_device
252 usbmon
253 lpc313x_pwm
254 lpc313x_adc

Block devices:
  1 ramdisk
259 blkext
  7 loop
  8 sd
 65 sd
 66 sd
 67 sd
 68 sd
 69 sd
 70 sd
 71 sd
128 sd
129 sd
130 sd
131 sd
132 sd
133 sd
134 sd
135 sd
179 mmc
```

Bild 2. Liste der verfügbaren Treiber.

stellt sich jetzt noch die Frage, woher der Kernel die Information hat, dass z.B. mit `/dev/ttyS0` der Treiber für die serielle Schnittstelle angesprochen wird?
Hierfür gibt es sogenannte Major- und Minor-Nummern. Dies sind eindeutige Nummern, die in der Kernel-Dokumentation hinterlegt sind. Diese müssen dann auch im jeweiligen Treiber des Gerätes stehen.
Welche Treiber mit welchen Major-Nummern aktuell verfügbar sind, kann man mit dem folgenden Aufruf ermitteln.

```
cat /proc/devices
```

Als Ausgabe erhält man eine Liste der Gerätetreiber, sortiert nach Character- und Blockdevices (**Bild 2**).

Anschließend kann man einmal den Ordner „/dev“ aufsuchen und `ls -l` eingeben, um den Inhalt dieses Ordners anzuzeigen (**Bild 3**). Der Ordner enthält eine Fülle von Dateien, die unsere Geräte symbolisieren. In der Ausgabe sieht man bei jeder Gerätedatei verschiedene Informationen. Ganz am Anfang steht ein „c“ oder „b“, was bedeutet, dass es sich um ein Block- oder Character-Device handelt. Anschließend sieht man die klassischen Rechte (dazu in einer späteren Folge der Serie mehr). Dann folgt der Benutzer, dem diese Datei gehört. Gleich dahinter sieht man zwei Zahlen, die durch ein Komma getrennt sind. Genau das sind die Major- und Minor-Nummern. Die Major-Nummern sind 1,4,5,8,89,253 und 254; die Minor-Nummern 0,1,2,3,9,12,24 und 64.

Greift man jetzt aus dem Userspace auf eine solche Datei zu, kann das Betriebssystem anhand der hinterlegten Nummer und der Information aus „/proc/devices“ genau feststellen, welcher Treiber gemeint ist.
Solche Gerätedateien kann man einfach mit dem Tool „mknode“ [7] anlegen. Das werden wir später noch gemeinsam machen.

```
root@gnublin:/ cd /dev
root@gnublin:/dev# ls -l
total 1
crw------- 1 root root   5,  1 Sep 27 06:02 console
prw------- 1 root root       0 Sep 27 06:02 initctl
srw-rw-rw- 1 root root       0 Sep 27 06:02 log
crw-r--r-- 1 root root 254,  0 Sep 26 03:57 lpc313x_adc
crw-r--r-- 1 root root 253,  0 Sep 26 04:21 lpc313x_pwm
lrwxrwxrwx 1 root root      12 Jan  1  1970 mtab -> /proc/mounts
crw-r--r-- 1 root root   1,  3 Dec 22  2011 null
crw-r--r-- 1 root root  89,  1 Sep 26 03:06 pca9555
crw-rw-rw- 1 root root   5,  2 Jan  1  1970 ptmx
drwxr-xr-x 2 root root       0 Jan  1  1970 pts
crw-r--r-- 1 root root   1,  9 Sep 26 04:28 random
brw-r--r-- 1 root root   8,  0 Sep 27 05:52 sda
brw-r--r-- 1 root root   8,  1 Sep 27 05:52 sda1
crw-rw-rw- 1 root root   5,  0 Jan  1  1970 tty
-rw-r--r-- 1 root root      24 Jan  1  1970 tty0
crw------- 1 root root   4, 64 Sep 27 06:03 ttyS0
crw-r--r-- 1 root root   1,  9 Sep 26 04:30 urandom
```

Bild 3. Diese Dateien symbolisieren unsere Geräte.

USB-Seriell-Wandler integrieren

Es ist nun Zeit, die Theorie in die Praxis umzusetzen. Vom Linux-Board aus wollen wir einen einfachen USB/Seriell-Wandler (per USB) ansteuern (siehe **Bild 4**). Im Wesentlichen sind die folgenden Schritte zu machen:

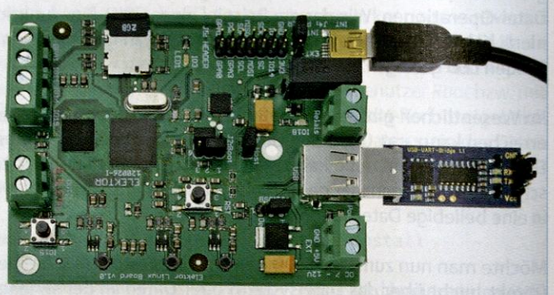


Bild 4. USB/Seriell-Wandler CP2102 an unserem Elektor-Linux-Board.

- Aktivierung des Treibers im Kernel
- Übersetzen des Kernels
- Kopieren des Kernels auf die SD-Karte
- Prüfen, ob der Treiber verfügbar ist
- Gerätedatei anlegen
- Das Gerät testen

Aktivierung des Treibers im Kernel
Wir wechseln zuerst in den Quelltextbaum von Linux:

```
cd ElektorLinuxBoardDownload_20120509/

cd linux-2.6.33-lpc313x/

. ./set.h

make menuconfig
```

Im ersten Schritt (**Bild 5**) wählt man den Menüpunkt „Device Drivers“ aus. Anschließend gibt es weiter unten einen Punkt „USB-support“ (**Bild 6**). In der Standardeinstellung ist hier ein „M“ anstelle wie im Screenshot ein „*“ zu finden. Mit der Leertaste kann man zwischen den drei Optionen: „M“ (als Modul), „*“ (fest in den Kernel) oder „nichts“ wählen. Wir wählen das „*“, um das Hantieren mit Modulen zu vermeiden.

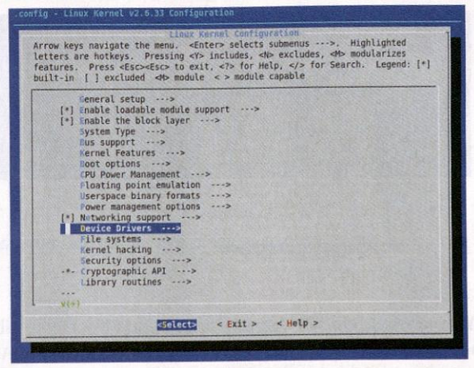


Bild 5. Menüpunkt „Device Drivers“.

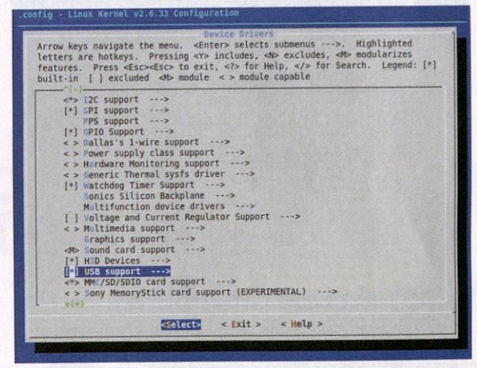


Bild 6. Menüpunkt „USB support“.

Der nächste Menüpunkt ist der „USB Serial Converter support“; auch hier muss man aus dem „M“ ein „*“ machen (**Bild 7** und **Bild 8**). Mit der Enter-Taste kommt man in das Untermenü zur Auswahl des passenden Treibers (**Bild 9**). Viele Leser verwenden vermutlich Chips von FTDI oder den Controller CP210x von Silabs, daher setzen wir bei beiden Treibern ein „*“. Um die Änderung zu übernehmen, geht man überall auf „Exit“, bis man schließlich auf der letzten Seite das Speichern der neuen Konfigurationsdatei bestätigen muss.

Übersetzen des Kernels
Jetzt kann man den Übersetzungsvorgang starten:

```
make zImage
```

Als Ergebnis erhält man die Ausgabe in **Bild 10**.

Kopieren des Kernels auf die SD-Karte
Die SD-Karte aus dem Board steckt man nun in einen SD-Karten-Leser und schließt diesen an den PC an. Nun wird der Kernel auf die SD-Karte kopiert.

```
sudo cp arch/arm/boot/zImage /
media/386b3be7-00f3-45e0-832e-1f48c2c3065e/
```

Nach dem Kopieren sollte man das Dateisystem manuell aushängen, um sicherzustellen, dass alle Blöcke sicher auf die SD-Karte übertragen wurden [4].

```
umount /media/386b3be7-00f3-45e0-832e-1f48c2c3065e
```

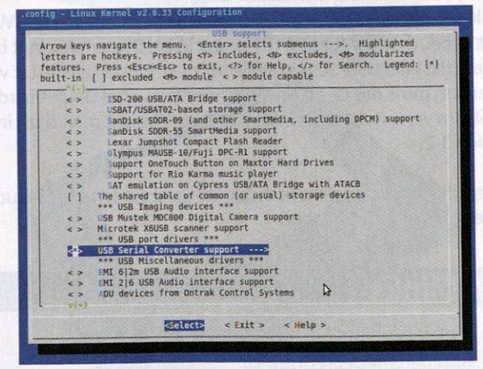


Bild 7. Auswahl „USB Serial Converter support“ als Bestandteil des Kernels.

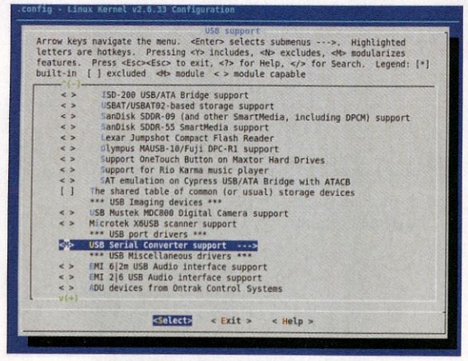


Bild 8. Auswahl „USB Serial Converter support“ als nachladbares Modul.

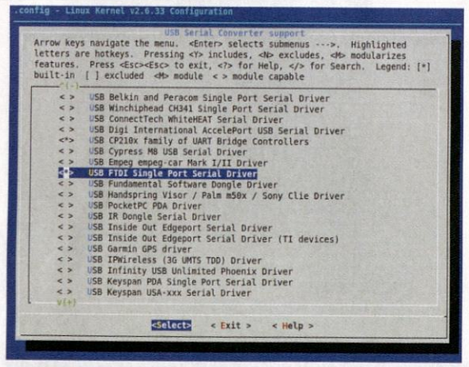


Bild 9. Hier wählt man den verwendeten USB/Seriell-Adapter aus.

Prüfen, ob der Treiber verfügbar ist

Jetzt kann man das Elektor-Linux-Board wie gewohnt starten. Wichtig ist dabei, dass der Jumper JP1 gesteckt ist! Dieser erzwingt beim Booten, dass die USB-Schnittstelle im Host-Modus gestartet wird. Außerdem muss das Gerät am USB-Anschluss vom Linux-Board aus mit Strom versorgt werden. Den Jumper 3 bringt man dazu in die Position 3-2.

Während des Bootens sieht man nun die erste Änderung. Es tauchen die beiden neuen Treiber auf:

```
cp210x: v0.09:Silicon Labs CP210x RS232 serial
adaptor driver

USB Serial support registered for FTDI USB Serial
Device
```

Gerätedatei anlegen

Nach dem Aufruf von `cat /proc/devices` sieht man jetzt einen neuen Eintrag: „188 ttyUSB“.
Für diese Major-Nummer muss man jetzt eine Gerätedatei anlegen:

```
mknod /dev/ttyUSB0 c 188 0
```

Das Gerät testen

Steckt man den USB/Seriell-Adapter jetzt an das Board an, dann erscheint folgende Ausgabe:

```
usb 1-1: New USB device found, idVendor=10c4,
idProduct=ea60

usb 1-1: New USB device strings: Mfr=1, Product=2,
SerialNumber=3

usb 1-1: Product: CP2102 USB to UART Bridge
Controller

usb 1-1: Manufacturer: Silicon Labs

usb 1-1: SerialNumber: 0001

cp210x 1-1:1.0: cp210x converter detected

usb 1-1: reset full speed USB device using lpc-ehci
and address 2

usb 1-1: cp210x converter now attached to ttyUSB0
```

In der letzten Zeile sieht man, dass das Anlegen der Gerätedatei geklappt hat. Der Kernel weist dem frisch erkannten Gerät die korrekte Gerätedatei zu.

Unser Linux-Betriebssystem bringt ein kleines Terminalprogramm namens „microcom“ mit. Mit Hilfe einer Brücke zwischen RX und

Weblinks

[1] sauter@embedded-projects.net
[2] http://en.wikipedia.org/wiki/Operating_Systems:_Design_and_Implementation
[3] http://de.wikipedia.org/wiki/Prozesstabelle
[4] www.elektor.de/120180
[5] www.elektor.de/120146
[6] www.gnugeneration.com/mirrors/kernel-api/r4299.html
[7] http://linuxwiki.de/mknod


```
  CC      arch/arm/boot/compressed/misc.o
In file included from arch/arm/boot/compressed/../../../../lib/decompress_inflate.c:10:0,
                 from arch/arm/boot/compressed/misc.c:248:
arch/arm/boot/compressed/../../../../lib/zlib_inflate/inflate.c: In function 'zlib_fixedtables':
arch/arm/boot/compressed/../../../../lib/zlib_inflate/inflate.c:100:1: warning: the frame size of 2176 bytes is larger than 1024 bytes
  SHIPPED arch/arm/boot/compressed/lib1funcs.S
  AS      arch/arm/boot/compressed/lib1funcs.o
  LD      arch/arm/boot/compressed/vmlinux
  OBJCOPY arch/arm/boot/zImage
  Kernel: arch/arm/boot/zImage is ready
```

Bild 10. Ausgabe beim Übersetzen des (erweiterten) Kernels.

TX (Jumper) am USB/Seriell-Adapter kann man den Treiber hiermit testen. Hierzu rufen wir das Terminalprogramm mit folgenden Parametern auf:

```
microcom -s 9600 /dev/ttyUSB0
```

Gibt man jetzt Buchstaben ein, so werden diese zum Adapter geschickt und von dort zum Board zurückgesandt. Wenn der Treiber funktioniert, dann müssten diese Buchstaben jetzt in der Ausgabe erscheinen. Unterbricht man die Verbindung zwischen RX und TX am USB/Seriell-Adapter, dann erscheinen die eingegebenen Buchstaben nicht mehr. So sehen wir, dass der Gerätetreiber funktioniert. Mit STRG-x beendet man microcom wieder.

Auf die gleiche Art und Weise können wir jetzt LAN- und WLAN/USB-Adapter, USB-Soundkarten und viele weitere USB-Geräte verwenden.

Im nächsten Teil werden wir weitere Geräte und Schnittstellen anschauen und diese mit kleinen Versuchen testen!

(120181)