Kernel- und Treiberprogrammierung mit dem Linux-Kernel – Folge 71

Kern-Technik

Mit dem flexiblen Bootloader U-Boot, einem Tftp-Server, angepasstem Kernel und Userland sowie etwas Konfiguration booten eingebettete Systeme wie der Raspberry Pi vom Netz. *Jürgen Quade*, *Eva-Katharina Kunst*

Der mitgelieferte Bootloader des Raspberry Pi hat einen großen Nachteil: Er kann nur Dateien von der ersten Partition der SD-Karte laden, zudem muss

diese mit einem FAT-32-Dateisystem formatiert sein. Für die meisten Fälle scheint das ausreichend, für Entwickler aber, die mit Kernel oder Root-Filesystem experimentieren, ist es zu umständlich. Das hängt mit der Host-Target-Entwicklung zusammen: Dabei übersetzt der Entwickler etwa einen neuen Kernel auf seinem Arbeitsrechner (Host), den er dann irgendwie auf die Bootpartition des Raspberry Pi (Target) bringen muss.

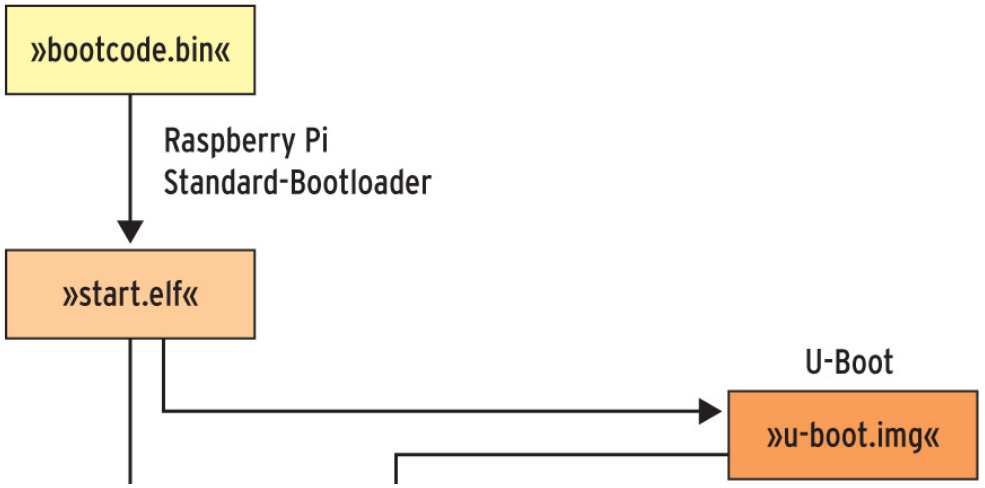
Läuft auf dem Raspberry Pi bereits ein Linux und ist die Himbeere per LAN-Kabel im Netzwerk erreichbar, kann das sehr einfach per »scp« oder »netcat« erfolgen. Ist der kopierte Kernel aber fehlerhaft, bleibt spätestens nach dem nächsten Reboot nur noch der Transfer über die SD-Karte. Konkret: SD-Karte aus dem Raspberry Pi, SD-Karte in den Entwicklungsrechner, Bootpartition einhängen (erfolgt oft automatisch), funktionstüchtigen Kernel auf die Bootpartition kopieren, Bootpartition aushängen, SD-Karte entfernen und in den Raspberry Pi einstecken. Da muss es doch etwas Besseres geben!

Moderne Bootloader, etwa der in der Embedded-Branche verbreitete Loader "Das U-Boot" [1], bieten den Systemstart vom Netzwerk an. Das mit dem U-Boot ausgestattete Gerät initialisiert direkt nach dem Einschalten sein Netzwerk, lässt sich per DHCP eine Adresse geben, sucht einen Tftp-Server, der den zu bootenden Kernel hostet, und holt sich von diesem den Betriebssystemkern. Man kann sogar das Root-Filesystem laden lassen, doch dazu später mehr.

DELUG-DVD

Die Delug-DVD zu diesem Magazin enthält ein angepasstes U-Boot, Tools und eine Bootpartition für den Raspberry Pi.

Leider lässt sich der Bootloader des Raspberry Pi nicht einfach gegen das U-Boot austauschen. Aber der findige Anwender bedient sich eines Tricks: Er konfiguriert den Original-Bootloader so, dass er nicht wie üblich den Linux-Kernel in den Hauptspeicher lädt, sondern stattdessen U-Boot (Abbildung 1). Das beschleunigt zwar nicht gerade den normalen Bootprozess, bringt aber im Gegenzug Features wie Netzwerkboot oder die Auswahl aus verschiedenen Kerneln oder Root-Filesystemen beim Booten.



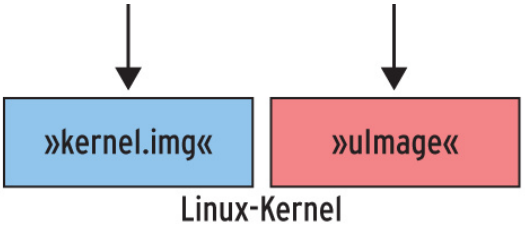


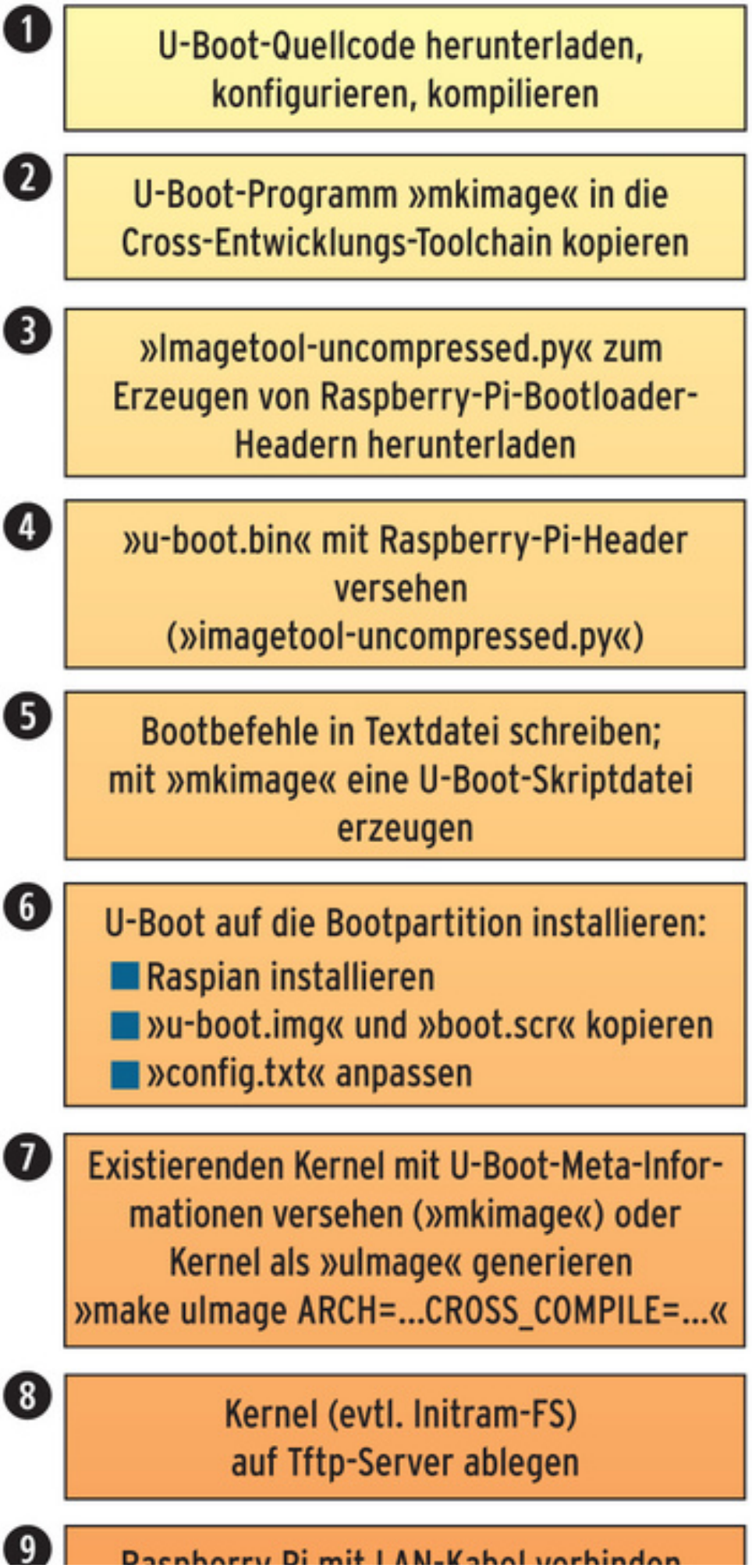
Abbildung 1: Bootsequenz des Raspberry Pi mit untergeschobenem U-Boot.

Aus der Quelle

Um dies zu realisieren, benötigt man U-Boot für den Raspberry Pi. Im Netz findet sich der Bootloader aber zunächst nur im Quellcode, der zum einen für den Raspberry Pi cross-kompiliert, zum anderen aber zusätzlich mit einem Header versehen werden muss, damit ihn der normale Bootloader des Raspberry Pi überhaupt laden kann. Deshalb haben die Autoren eine lauffähige Version generiert, die auf der Delug-DVD dieses Magazins sowie unter [2] als Tar-File bereitsteht. Dieses rund 40 MByte große Archiv enthält noch weitere Dateien, die dem Leser einige aufwändige Generierungsschritte ersparen (siehe Tabelle 1).

Tabelle 1: Dateien in [2]

Objektname	Funktion	Schritt in Abbildung 2
mkimage.32, mkimage.64	Generator der U-Boot-Meta-Information	1
mkimage/imagetool-uncompressed.py	Generator für die RPI-Meta-Information	3
u-boot.img	Einsatzbereiter Bootloader	4
boot.txt	U-Boot-Befehle zum Booten vom Flash	5
boot.scr	U-Boot-Skript zum Booten vom Flash	5
boot-net.txt	U-Boot-Befehle zum Netzwerk-Boot des Kernels	5
boot-initramfs.txt	U-Boot-Befehle zum Netzwerk-Boot (Kernel und Initram-FS)	5
uImage	Präparierter Linux-Kernel	7
initramfs.uboot	Schlankes Initram-FS	8
boot.dd	Image einer Bootpartition, Installation auf eine bereits partitionierte SD-Karte per »dd if=boot.dd of=/dev/sd*X*1«	


1
U-Boot-Quellcode herunterladen, konfigurieren, kompilieren
2
U-Boot-Programm »mkimage« in die Cross-Entwicklungs-Toolchain kopieren
3
»Imagetool-uncompressed.py« zum Erzeugen von Raspberry-Pi-Bootloader-Headern herunterladen
4
»u-boot.bin« mit Raspberry-Pi-Header versehen (»imagetool-uncompressed.py«)
5
Bootbefehle in Textdatei schreiben; mit »mkimage« eine U-Boot-Skriptdatei erzeugen
6
U-Boot auf die Bootpartition installieren:
Raspian installieren
»u-boot.img« und »boot.scr« kopieren
»config.txt« anpassen
7
Existierenden Kernel mit U-Boot-Meta-Informationen versehen (»mkimage«) oder Kernel als »uImage« generieren »make uImage ARCH=...CROSS_COMPILE=...«
8
Kernel (evtl. Initram-FS) auf Tftp-Server ablegen
9

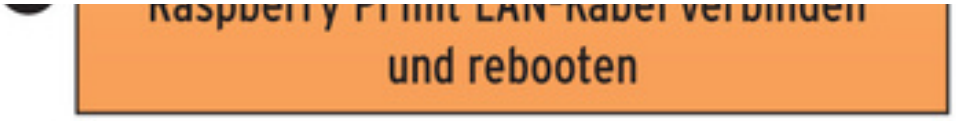


Abbildung 2: Neun Schritte sind erforderlich, um den Raspberry Pi auf Netzwerkboot umzurüsten.

Für den ganz Neugierigen liegt sogar das Image einer Raspberry-Pi-Bootpartition mit U-Boot, einem Kernel, einem Initram-FS und der benötigten Konfiguration bei. Das braucht er nur per »dd« auf die erste Partition einer SD-Karte zu schreiben, um den modifizierten Bootvorgang testen zu können. Wer dagegen den Bootloader selbst erzeugen und dabei möglicherweise anders konfigurieren oder erweitern möchte, findet eine Anleitung dazu im Kasten "U-Boot für den Raspberry Pi übersetzen".

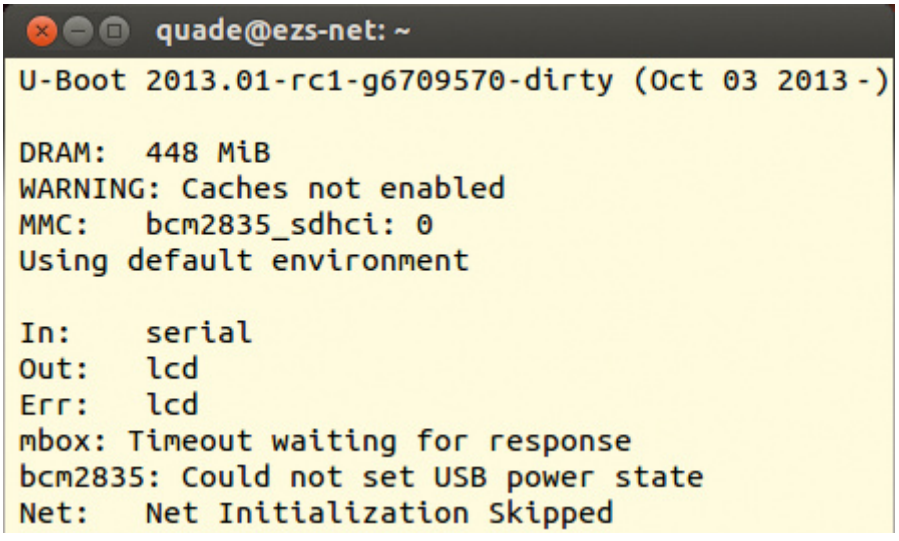
Ob selbst kompiliert oder in der Version des Linux-Magazins: Der mit dem Header versehene Bootloader »u-boot.img« wandert im nächsten Schritt auf die SD-Karte. Auch hier gibt es mehrere Wege. Der wohl einfachste besteht darin, die auf einer SD-Karte vorhandene Installation eines Raspbian zu modifizieren [4] und das U-Boot-Image anstelle der Datei »kernel.img« auf die Bootpartition der SD-Karte zu kopieren.

Etwas übersichtlicher ist es allerdings, nicht einfach plump »kernel.img« mit »u-boot.img« zu überschreiben, sondern »u-boot.img« zusätzlich auf die Karte zu packen und dann die ebenfalls auf der Bootpartition zu findende Datei »config.txt« um die Zeile »kernel=u-boot.img« zu ergänzen.

Ist die Bootpartition der SD-Karte auf dem Host unter dem Verzeichnis »/media/boot/« eingehängt, geht das mit den folgenden beiden Befehlen:

```
cp kernel.img /media/boot/u-boot.img
echo "kernel=u-boot.img">>/media/boot/config.txt
```

Nach der Installation des Bootloaders ist die Zeit für den ersten Probelauf gekommen. Steckt man die präparierte SD-Karte in den Raspberry Pi und versorgt den Mini-Computer mit Strom, bootet U-Boot in einen Shellprompt (siehe Tabelle 2 "Wichtige U-Boot Kommandos"). Die dabei erscheinenden Bootmeldungen sind in Abbildung 3 zu sehen.



```
No ethernet found.
Hit any key to stop autoboot:  0
U-Boot> usb start
```

Abbildung 3: U-Boot auf dem Raspberry Pi meldet sich auf der seriellen Konsole.

Tabelle 2: Wichtige U-Boot-Kommandos

Kommando	Bedeutung
help *Kommando*	Ohne Option werden die zur Verfügung stehenden Kommandos gelistet (Abbildung 4). Ansonsten gibt es eine Kurzhilfe zu »*Kommando*«.
fatls mmc 0	Listet den Inhalt auf der ersten Partition der ersten SD-Karte (»mmc«) auf.
fatload mmc 0 *Speicheradresse Datei*	Lädt »*Datei*« von der ersten Partition des Geräts »mmc« an die »*Speicheradresse*«.
usb start	Initialisierung des USB-Stacks (notwendig, um Zugriff auf das Netzwerk zu bekommen).
bootm *Speicheradresse Initramfs Devicetree*	Bootet das Image an der »*Speicheradresse*«. Falls »*Initramfs*« angegeben ist, wird dem Kernel dessen Adresse übergeben. Falls sich im Speicher der »*Devicetree*« befindet, wird auch dessen Adresse übergeben.
dhcp *Speicheradresse IP-Adresse*:*Bootfile*	Lässt sich per DHCP eine IP-Adresse zuweisen und lädt danach vom Tftp-Server mit »*IP-Adresse*« die Datei »*Bootfile*« an die »*Speicheradresse*«.
tftpboot *Speicheradresse IP-Adresse*:*Bootfile*	Lädt die Datei »*Bootfile*« vom Tftp-Server mit der »*IP-Adresse*« an die »*Speicheradresse*«.
reset	Neustart

```
quade@ezs-net: ~
loadb   - load binary file over serial line (kermit mode)
loads   - load S-Record file over serial line
loady   - load binary file over serial line (ymodem mode)
loop    - infinite loop on address range
ls      - list files in a directory (default /)
md      - memory display
mm      - memory modify (auto-incrementing address)
mmc     - MMC sub system
mmcinfo - display MMC info
mtest   - simple RAM read/write test
mw      - memory write (fill)
nfs     - boot image via network using NFS protocol
nm      - memory modify (constant address)
part    - disk partition related commands
ping    - send ICMP ECHO_REQUEST to network host
printenv- print environment variables
reset   - Perform RESET of the CPU
run     - run commands in an environment variable
setenv  - set environment variables
```

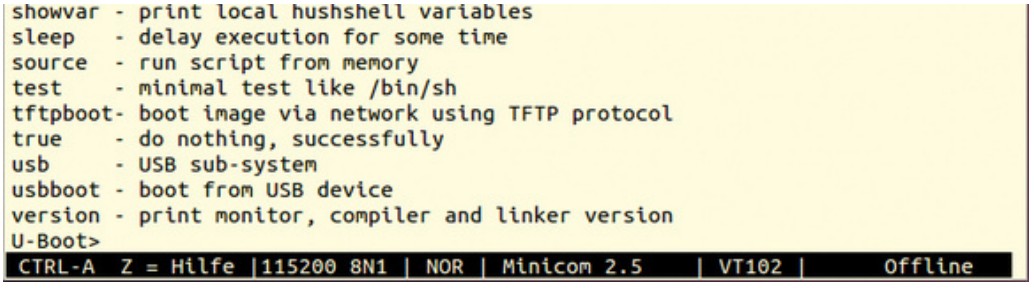


Abbildung 4: Booten zum Prompt: Per serieller Schnittstelle lässt sich das U-Boot interaktiv kommandieren.

Kein USB-Keyboard

Wer versucht mit einer angesteckten USB-Tastatur Kommandos aufzurufen, wird allerdings enttäuscht: Kommandos nimmt der Bootloader in der vorliegenden Konfiguration nur über eine serielle Schnittstelle entgegen, über die U-Boot seine Meldungen auch ausgibt. Wer über die serielle Schnittstelle das Kommando »fatls mmc 0« eintippt, bekommt alle auf der ersten Partition der SD-Karte abgelegten Dateien angezeigt. Auf das Kommando »dhcp« holt sich die über Ethernet angeschlossene Himbeere von einem im Netzwerk vorhandenen DHCP-Server eine IP-Adresse. Ansonsten kann man eine lokale Adresse mit »setenv ipaddr *IP-Adresse*« einstellen.

Weniger erfolgreich dürfte jedoch der Versuch ablaufen, einen per »fatload mmc 0 00200000 kernel.img« in den Hauptspeicher geladenen Kernel über das Kommando »bootm« zu starten. Das hängt damit zusammen, dass nicht nur der hauseigene Bootloader, sondern auch U-Boot für die zu ladenden Dateien einen speziellen Header benötigt.

Wie es für den Pi-Bootloader ein entsprechendes Programm für das Schreiben der Meta-Informationen gibt, bringt auch U-Boot in seinem Quellcode mit »mkimage« ein solches Tool mit. Wer U-Boot selbst kompiliert hat, findet »mkimage« im U-Boot-Quellcodeverzeichnis unter »u-boot-pi/tools/mkimage«. Eine fertige Version sowohl für 32 als auch für 64 Bit gibt es auch unter [2].

Das Kommando in Listing 1 generiert die notwendige Headerinformation für den von Raspbian mitgelieferten Kernel (auf einer mit Raspbian installierten SD-Karte die Datei »kernel.img«).

Listing 1: *mkimage*-Aufruf

```
01 $ mkimage -A arm -O linux -C none -T kernel \
02     -a 0x00008000 -e 0x00008000 -n "linux-kernel" -d kernel.img uI
03 Image Name:   linux-kernel
04 Created:      Thu Oct  3 16:45:13 2013
05 Image Type:   ARM Linux Kernel Image (uncompressed)
06 Data Size:    2802568 Bytes = 2736.88 kB = 2.67 MB
07 Load Address: 00008000
08 Entry Point:  00008000
```

Wer den Kernel ohnehin selbst generiert, wie beispielsweise in [3] beschrieben,

der kopiert als Root »mkimage« in das Verzeichnis der Cross-Toolchain und ergänzt den Programmnamen noch durch den Namensvorsatz der Toolchain. Bei der Toolchain von Ubuntu 12.04 beispielsweise ruft er »cp mkimage /usr/bin/arm-linux-gnueabi-mkimage« auf. Danach gibt er im Quellcodeverzeichnis des Linux-Kernels das Kommando

```
make uImage ARCH=armCROSS_COMPILE=arm-linux-gnueabi-
```

ein. Das Kernel-Buildsystem generiert jetzt nicht nur den Kernel, sondern versieht ihn auch direkt mit den Meta-Informationen für U-Boot. Das erspart das separate Aufrufen von »mkimage«.

Aufgetaucht

Im ersten Test bootet U-Boot in einen Prompt, der über die serielle Schnittstelle Kommandos entgegennimmt. Damit der Bootloader ohne Interaktion den präparierten Kernel lädt, ist noch ein weiteres U-Boot-Skript erforderlich. Es schiebt per »fatload« zunächst einmal den Kernel von der SD-Karte in den Hauptspeicher (an Adresse 0x00200000) und startet ihn danach per »bootm« (Listing 2).

Listing 2: Texteingabe für U-Boot-Skript *boot.txt*

```
01 echo "Loading uImage..."
02 fatload mmc 0:1 00200000 uImage
03 bootm 00200000
```

Um ein lauffähiges U-Boot Skript zu bekommen, schreibt man die Kommandos in eine Textdatei (»boot.txt«) und versieht sie per »mkimage« mit der notwendigen Meta-Information. Das so präparierte Skript findet unter dem Namen »boot.scr« ebenfalls seinen Platz auf der Bootpartition der SD-Karte:

```
# mkimage  -A arm -O linux -T script-C none -d boot.txt boot.scr
```

Auf der Bootpartition müssen sich damit mindestens die folgenden Dateien befinden (Abbildung 5):

- »bootcode.bin«
- »start.elf«
- »uImage«
- »boot.scr«
- »u-boot.img«
- »config.txt«

Damit bootet Raspbian wie gehabt, sieht man einmal von dem Zwischenschritt U-Boot ab. Wer mit dem Raspberry Pi per serieller Schnittstelle Verbindung hat, kann den Bootprozess anhalten. Gibt es alternative Kernel, kann er nun diese ausprobieren.

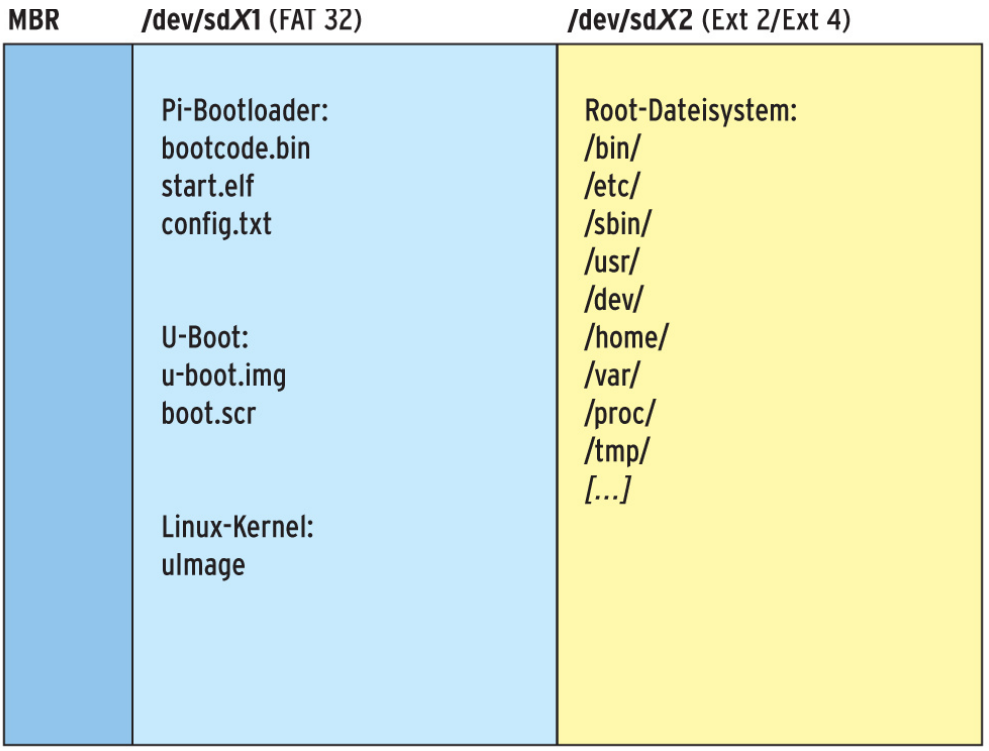


Abbildung 5: Diese Dateien auf der SD-Karte sind am Bootvorgang beteiligt.

Besonders interessant ist es jedoch, den Kernel über das Netzwerk zu laden. Das erfordert einen Tftp-Server, der die Dateien vorhält. Viele DHCP-Server, beispielsweise eine Fritzbox, bieten auch Tftp-Dienste an. Da man dort aber nicht so einfach Dateien ablegen kann, ist es sinnvoll, neben einem vorhandenen DHCP-Server einen eigenen Tftp-Server auf dem lokalen Entwicklungssystem zu installieren.

Als Tftp-Server bietet sich unter Ubuntu »tftpd-hpa« an, der in Version 12.04 mit »sudo apt-get install tftpd-hpa« rasch installiert ist. Der Server stellt die Dateien aus dem Verzeichnis »/var/lib/tftpboot/« im Netzwerk zur Verfügung. Allerdings gibt es bei der installierten Version einen Bug: Nach einem Reboot startet der Server nicht korrekt, der Ubuntu-Anwender muss ihn daher händisch per »sudo service tftpd-hpa restart« anschubsen.

Ins Netz gegangen

Damit U-Boot den richtigen Tftp-Server wählt, benötigt man dessen IP-Adresse, die beispielsweise auf dem lokalen Entwicklungssystem per »ifconfig« zu erfahren ist. Mit diesen Informationen lässt sich das U-Boot-Skript »boot-net.txt« schreiben:

```
usb start
dhcp 00200000 IP-Adresse :uImage
bootm 00200000
```

Dabei ist die IP-Adresse des Tftp-Servers einzusetzen, in vielen Fällen also die IP-Adresse des Entwicklungsrechners. Die Skriptbefehle gehören wieder in eine Textdatei geschrieben, per »mkimage« mit der notwendigen Meta-Information versehen und schließlich auf der SD-Karte abgelegt. Wenn jetzt noch der Kernel

unter dem Namen »uImage« auf dem Tftp-Server liegt und der Raspberry Pi per LAN-Kabel am Netzwerk angeschlossen ist, steht dem ersten Netzwerkboot nichts mehr entgegen. Einmal eingerichtet ist die Anwendung einfach. Um einen neuen Kernel auszuprobieren, legt man diesen einfach mit dem Namen »uImage« in das Verzeichnis »/var/lib/tftpboot/«. Beim nächsten Booten holt sich die Himbeere den Kernel, legt ihn in den Hauptspeicher und startet ihn.

Besonders interessant ist diese Technik, wenn der Benutzer nicht nur den Kernel, sondern auch das Userland per Netzwerk lädt. Linux bietet mit der Technologie Initram-FS [5] die Möglichkeit, einen Teil des Hauptspeichers als eine Art Ramdisk zu verwenden, in der sich die zum Betrieb notwendigen Verzeichnisse und Dateien befinden. Dieses Verfahren wird vor allem für eingebettete System eingesetzt und bringt einige Vorteile: Mit jedem Neustart liegt ein konsistentes Filesystem vor, auch wenn dem System mal der Strom abgeschaltet wurde, ohne es sauber herunterzufahren. Ein Filesystem-Check ist daher unnötig.

Alles im Speicher

Das System lässt sich einfach durch Austausch der zugehörigen Imagedatei aktualisieren, der Zugriff ist sehr schnell, da die Daten alle im Hauptspeicher liegen. Das typischerweise schlanke Initram-FS bootet deutlich schneller als ein ausgewachsenes Userland von Raspbian. Allerdings befinden sich Änderungen im Userland zunächst nur im RAM, gehen also beim nächsten Reboot ohne weitere Maßnahmen verloren.

Um ein Initram-FS zu nutzen, muss der Anwender den Kernel passend konfigurieren. Dazu aktiviert er in der Kernelkonfiguration mit »make menuconfig« den Punkt »General setup | Initial RAM filesystem and RAM disk (initramfs/initrd) support«. Danach übersetzt er den Kernel und legt ihn auf den Tftp-Server:

```
make uImage ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-
cp arch/arm/boot/uImage /var/lib/tftpboot/
```

Daneben ist das Initram-FS als solches erforderlich. Der Einfachheit halber gibt es auf der Delug-DVD und dem Server des Linux-Magazins ein schlankes Initram-FS, das sich für Tests eignet [2]. Der Benutzer kann sich darauf mit dem Passwort »root« als Root anmelden. Viele Befehle sind allerdings nicht eingebaut, denn es dient hauptsächlich zur Demonstration des Bootprozesses.

Jetzt fehlt nur noch das U-Boot-Skript »boot-initramfs.txt«. Unter der Voraussetzung, dass der Tftp-Server die IP-Adresse 192.168.178.25 hat, sieht es aus wie in Listing 3. Das Kommando »mkimage« macht aus der Textdatei das lauffähige Skript. Das Initram-FS wird im Speicher des Raspberry Pi an die hexadezimale Adresse 00f00000 gelegt, um dann beim Booten vom Kernel an einen geeigneten Platz verschoben zu werden.

Listing 3: U-Kernel und Initram-FS laden

```
01 usb start
02 dhcp 00200000 192.168.178.25:uImage
03 tftpboot 00f00000 192.168.178.25:initramfs.uboot
04 bootm 00200000 00f00000
```

U-Boot für den Raspberry Pi übersetzen

Die Standardversion von U-Boot bietet zwar Unterstützung für den Raspberry Pi, sie funktioniert aber nicht out of the Box. Insbesondere macht sie noch Probleme mit USB und damit auch der Netzwerkfunktionalität. Glücklicherweise findet sich aber eine gepatchte Version unter [http://github.com/gonzoua/u-boot-pi]. Deren Quellcode lässt sich per »git clone« auf den Entwicklungsrechner holen.

Um aus dem Quellcode ein auf dem Raspberry Pi lauffähiges Programm zu generieren, ist ein Cross-Compiler erforderlich. Ubuntu 12.04 hat zwar standardmäßig einen ARM-Cross-Compiler im Paketarchiv, doch der taugt für diesen Zweck nicht. Daher muss der Pi-Besitzer eine eigene Toolchain installieren.

Vergleichsweise einfach geht das mit dem Systembuilder Buildroot (siehe [6], [7]). Wer unter Ubuntu 12.04 die in Listing 4 abgedruckten Kommandos eingibt, der hat anschließend unterhalb von »~/linux-magazin/« – im Verzeichnis »buildroot-2013.08.01/output/host/usr/bin/« – die Programme der Cross-Toolchain liegen.

Listing 5 zeigt die Kommandos, die erforderlich sind, um den U-Boot-Quellcode zu laden, die Pfadvariable anzupassen (damit der Cross-Compiler zu finden ist) und den Bootloader zu konfigurieren, zu generieren sowie mit einem Header zu versehen. Dieser Header ist notwendig, da der hauseigene Bootloader des Raspberry Pi zum Laden des alternativen Bootloaders Meta-Informationen benötigt, beispielsweise die Adresse, ab der der Bootloader in den Speicher geladen werden soll.

Das Programm »imagetool-uncompressed.py« verknüpft diese Informationen mit dem selbst übersetzten U-Boot-Binary. Listing 5 kopiert das Ergebnis (»kernel.img«) in das Verzeichnis »~/linux-magazin«, um es leichter zu finden. So lässt sich U-Boot auf der SD-Karte quasi anstelle des Linux-Kernels installieren.

Listing 4: Generieren einer Cross-Toolchain

```
01 mkdir ~/linux-magazin
02 cd ~/linux-magazin
03 wget http://www.buildroot.net/downloads/buildroot-2013.08.1.tar.bz
04 tar xvf buildroot-2013.08.1.bz2
05 cd buildroot-2013.08.1
06 make rpi_defconfig
07 make toolchain
```

Listing 5: Generieren von U-Boot

```
01 cd ~/linux-magazin
02 git clone git://github.com/gonzoua/u-boot-pi
03 cd u-boot-pi
04 export CROSS_COMPILE=arm-linux-
05 PATH=$PATH:~/linux-magazin/buildroot-2013.08.1/output/host/usr/bin
06 make rpi_b_config
07 make rpi_b
08 cd ..
09 git clone http://github.com/raspberrypi/tools
10 cd tools/mkimage/
11 ./imagetool-uncompressed.py ../../u-boot-pi/u-boot.bin
12 cp kernel.img ~/linux-magazin/
```

Einmaliger Aufwand

Zugegebenermaßen ist anfänglich etwas Aufwand zu treiben, um den Raspberry Pi vom Netzwerk sowohl Kernel als auch Initram-FS holen zu lassen. Es lohnt sich aber: Einmal eingerichtet spart das Booten vom Netz viel Zeit. *(mhu)*

Infos

1. Das U-Boot: [http://www.denx.de/wiki/U-Boot]
2. Listings und Dateien zum Artikel: [http://www.linux-magazin.de/static/listings/magazin/2013/12/kern-technik/]
3. Quade, Kunst: "Kern-Technik – Folge 69": Linux-Magazin 08/13, S. 86
4. Raspbian: [http://www.raspbian.org]
5. Quade, Kunst: "Kern-Technik – Folge 39": Linux-Magazin 05/08, S. 98
6. Buildroot: [http://buildroot.uclibc.org]
7. Quade, Kunst, "Kern-Technik – Folge 59": Linux-Magazin 11/11, S. 96

Die Autoren

Eva-Katharina Kunst, Journalistin, und Jürgen Quade, Professor an der Hochschule Niederrhein, sind seit den Anfängen von Linux Fans von Open Source. In der Zwischenzeit ist die dritte Auflage ihres Buches "Linux Treiber entwickeln" erschienen.