

Pimp my Router

Linux für Embedded-Systeme: Router-Hack am Beispiel von OpenWrt

Joachim Schröder

Messen, Steuern, Regeln – mit dem PC kein Problem und mithilfe eines USB-Controllers wie beispielsweise dem IOWarrior-Baustein kann auch die notwendige Peripherie angeschlossen werden [1, 19]. Doch muss es immer ein PC sein? Für viele Anwendungen wird die Rechenleistung eines PCs nicht benötigt, zudem lohnt sich eine Anschaffung für dedizierte Aufgaben kaum und der Energieverbrauch spielt auch eine immer größere Rolle. Alternativ könnte ein Mikrocontroller verwendet werden, aber damit verliert man in der Regel auch die Vorteile eines PCs wie die einfache Einbindung in ein Netzwerk, Nutzung von WLAN, USB-Komponenten und vielem mehr.

Embedded-Linux-Systeme füllen diese Lücke, indem sie auf proprietärer Hardware aufsetzen, aber dennoch Funktionalität wie Netzwerkkommunikation, USB-Unterstützung, Dateisystem und Dienste wie Webserver oder SSH bieten. Entwicklungs-Boards für Embedded-Linux sind bei größeren Elektronikanbietern zu beziehen, aber für die Realisierung eines eigenen Projektes mit 200...300 Euro recht teuer. Zwar sind auch günstigere Embedded-Boards auf Linux-Basis erhältlich, allerdings ist die verwendete Linux-Variante dann oft so speziell, dass dafür kaum Treiber für Peripheriegeräte existieren. Wer ein Embedded-Board auswählt, der sollte sich auch die zugehörige Linux-Distribution genau anschauen.

Mit OpenWrt [2] steht ein minimalistisches Linux-Betriebssystem zur Verfügung, welches ursprünglich als Software für Netzwerk-Router entwickelt wurde, mittlerweile aber auch in einer Vielzahl von Projekten für gänzlich andere Zwecke verwendet wird. Der ursprüngliche Verwendungszweck bringt jedoch den größten Vorteil mit sich: OpenWrt unterstützt momentan Geräte von über 50 Herstellern und als Konsumware liegen die meisten im Preisbereich zwischen 60 und 100 Euro. OpenWrt hat zudem eine große Anwendergemeinde, wird gut gepflegt und ständig weiterentwickelt – zum Hineinschnuppern in die Embedded-Linux-Welt nahezu ideal. Im vorliegenden Artikel wird OpenWrt in Kombination mit einem WL-500gP-Router der Firma Asus verwendet, alternativ können aber auch andere Geräte aus der OpenWrt-Liste mit USB-Anschluss verwendet werden (vgl. hierzu auch die Rubrik Supported Devices unter [2]).

Für die Windows-Anwender folgt nun ein kleiner Wermutstropfen: Zwar ist die Installation eines fertigen OpenWrt-Images unter Windows grundsätzlich möglich, das OpenWrt-Build-System lässt sich aber nur unter Linux bauen. Dieses System wird benötigt, um die Toolchain samt Cross-Compiler zu erstellen und damit Anwendungen für einen OpenWrt-Router zu programmieren. Im folgenden Text wird entsprechend vorausgesetzt, dass ein debianbasiertes Linux-Betriebssystem wie z. B. Ubuntu zur Verfügung steht (vgl. [3], aktuell in der Version 8.04). Dieses kann mittlerweile problemlos auch parallel zu einem bereits existierenden Windows installiert werden – Zweifler unter den Lesern können sich also recht unverkrampft an ein Open-Source-Betriebssystem heranwagen.

Die meisten der OpenWrt-tauglichen Router besitzen einige IO-Pins, LEDs oder Taster, die sich zum Anschluss externer Komponenten nutzen bzw. zweckentfremden

lassen. Da dies aber für jedes Gerät unterschiedlich ist, kann kein allgemeines Beispiel zur Benutzung angeführt werden. Fast alle dieser Geräte verfügen jedoch über einen USB-Anschluss, so dass die Erweiterung mittels IO-Warrior-USB-Controller eine einheitliche Möglichkeit darstellt, mit der dann auch wesentlich mehr IOs, und zudem LCD-Anschluss und I²C-Bus realisiert werden können [1, 19]. Diesem Thema widmet sich der letzte Teil des Artikels, exemplarisch werden ein Temperatursensor [5] und ein Schrittmotortreiber [6] über den I²C-Bus angesteuert. Alle im Rahmen des Artikels verwendeten Quellcode-Dateien stehen unter [21] zum Herunterladen bereit.

Hardware

Bild 1 zeigt das Innenleben des als Referenz-Hardware verwendeten ASUS WL-500gP, der im Einzelhandel für etwa 80 Euro angeboten wird.

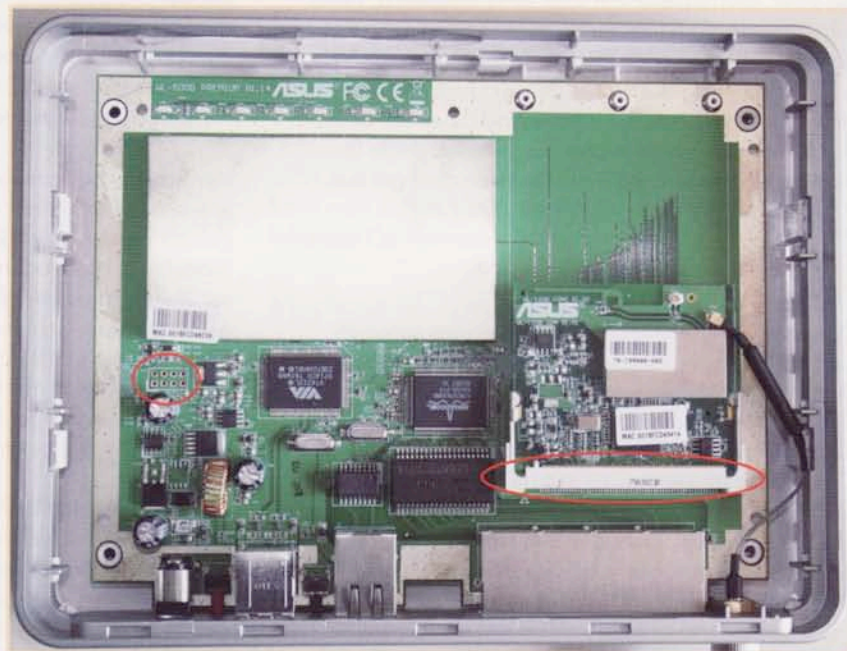


Bild 1. Hauptplatine des ASUS WL-500gP-Routers.

Neben den äußerlich sichtbaren Schnittstellen (2xUSB, 5xEthernet, WLAN) fallen bei Betrachtung der Platine zwei Besonderheiten auf: Am linken Rand auf halber Höhe befinden sich Lötunkte für einen Pfostenstecker. Hier sind zwei serielle Schnittstellen mit der in der Tabelle gezeigten Belegung verfügbar. Vorsicht: Da es sich um einen Signalpegel mit nur 3,3 V Maximalspannung handelt, muss anstelle des bekannten Pegelwandlers MAX232 unbedingt ein MAX233-Baustein verwendet werden, um RS232-konforme Pegel zu erhalten. Eine Schaltung hierfür kann entsprechend der Beschreibung in [17] realisiert werden.

Bei dem breiten, weißen Steckverbinder handelt es sich um einen Mini-PCI-Steckplatz, in welchem eine WLAN-Karte steckt. Die standardmäßig verbaute Platine mit einem Controller des Typs BCM4318 funktioniert bislang nur unter Linux Kernel 2.4. Wird eine Kernelversion

2.6 gewünscht, so bleibt nur der Tausch gegen eine Atheros MiniPCI-WLAN-Karte. Der Mini-PCI-Steckplatz macht den WL-500 auch für andere Zwecke interessant, so könnte beispielsweise eine andere Steckkarte verwendet werden, um einen CAN-Bus zu realisieren. Wer jetzt aufhorcht, der sollte sich allerdings vorher im OpenWrt-Forum nach der Realisierbarkeit und der Verfügbarkeit von Treibern erkundigen [16].

Reset	
GND	3,3V
TX 1	TX 0
RX 1	RX 0

Tabelle 1. Belegung des 8poligen Steckverbinders, der eckige Lötspitze ist Pin1/RX0.

Die weiteren technischen Daten des Routers sind wie folgt: Als Prozessor wird ein Broadcom BCM94704 mit einer Taktfrequenz von 266 MHz eingesetzt. Nicht sehr beeindruckend im Vergleich zu heutigen PCs, aber für viele Anwendungen im Bereich Messen, Steuern, Regeln völlig ausreichend. Als Hauptspeicher sind 32 MB verbaut, der interne Flash-Speicher ist mit 8 MB vergleichsweise mager ausgestattet, reicht für ein OpenWrt-Image mit Zusatzpaketen aber gut aus.

Einrichtung eines OpenWrt Build-Systems

Das OpenWrt-Build-System realisiert einen automatisierten Ablauf, um Quellen für Kernel und Pakete aus dem Internet zu laden und darauf, falls für die jeweilige Zielplattform erforderlich, Patches anzuwenden und die Quellen dann zu übersetzen. Die dazu notwendigen Cross-Compiler werden zu Beginn eines Übersetzungsdurchlaufs erstellt. Als Ergebnis entstehen komplette Firmware-Dateien, welche in den Flash-Speicher des Zielsystems übertragen werden sowie .ipk-Pakete, die auf dem Zielsystem installiert werden können.

Es ist empfehlenswert, sich vor dem Aufsetzen des OpenWrt Build-Systems zuerst mit einigen Ubuntu-Anwendungen vertraut zu machen, um später nicht an Kleinigkeiten hängen zu bleiben. Dazu zählt die Verwendung der Konsole oder Shell (in Ubuntu als Terminal bezeichnet, [7]), die Benutzung von Shell-Editoren wie vi [8] oder nano [9], und der Umgang mit Synaptic [10], einer grafischen Benutzeroberfläche für das Paket-Management-System apt [11].

Weiterhin sollte eine gewisse Erfahrung im Umgang mit SSH und Secure Copy (SCP) vorhanden sein. Diese Programme werden verwendet, um über die Konsole vom PC aus auf dem WL-500 zu arbeiten (SSH) und Dateien zu übertragen (SCP). Eine gute Einführung dazu bietet [12].

Die Quelltexte können mit der Software Subversion aus dem OpenWRT Repository heruntergeladen werden. Subversion oder auch SVN ist eine Versionsverwaltung für Dateien und Verzeichnisse und kann, falls auf dem Debian-Host-System nicht vorhanden, mit dem folgenden Befehl nachinstalliert werden:

```
$ sudo apt-get install subversion
```

Die momentan aktuelle Version 7.09 Kamikaze wird nun mit

```
$ svn co https://svn.openwrt.org/openwrt/
tags/kamikaze_7.09 <openwrt_dir>
```

in das Verzeichnis <openwrt_dir> heruntergeladen. Wer mit Subversion bislang noch nicht in Berührung gekommen ist, der findet unter [15] eine ausführliche, freie Dokumentation. Nach Ausführung der genannten Befehle ist nun das komplette Build-System auf den Rechner geladen. Mit den nachfolgenden Eingaben wird die Toolchain erstellt, um mit einer X86-Architektur Binärdateien für den Broadcom-Chip BCM947xx/953xx erstellen zu können. Neben der Toolchain selbst wird außerdem der Linux-Kernel für das Zielsystem erzeugt, es werden notwendige Pakete gebaut und es wird ein Image erstellt, das anschließend direkt in den Flash-Speicher des Routers geladen werden kann. Im Verzeichnis <openwrt_dir> wird mittels

```
$ make menuconfig
```

das Konfigurationsmenü aufgerufen, um festzulegen, wie die Zielarchitektur aussehen soll und welche Pakete gebaut werden sollen (in den Kernel integriert oder als Modul zum Nachladen). Unter Umständen müssen an dieser Stelle noch die in der Datei <openwrt_dir>/README gelisteten Pakete nachinstalliert werden. Auf einem neu eingerichteten Ubuntu-System sind diese im Terminal über

```
$ sudo apt-get install ncurses-dev gawk
bison flex autoconf automake
```

zu installieren. Im Konfigurator ist als Zielsystem Broadcom BCM947xx/953xx [2.6] auszuwählen, um später mit dem IOWarrior-Baustein arbeiten zu können. Durch die Verwendung der Linux-Kernel-Version 2.6 ergeben sich insbesondere bei der Verwendung von USB-Komponenten und bei der Treiberprogrammierung Vorteile gegenüber Kernel 2.4, es existiert jedoch das angesprochene Problem der eingeschränkten WLAN-Unterstützung.

Nun kann eine Auswahl zusätzlicher Pakete (z. B. stty für eine einfache Konfiguration der seriellen Schnittstelle) erfolgen.

Diese können mit [*] direkt in den Kernel eingebunden und damit nach dem Booten direkt als Befehl verwendet werden. Alternativ können sie auch mit der Option [M] als Modul gebaut werden, müssen dann allerdings vor der Verwendung auch manuell geladen werden. Im zweiten Fall wird der Kernel etwas kleiner gehalten.

Nach erfolgter Konfiguration kann der Editor beendet und nun mit dem Befehl

```
$ make
```



```
#### LAN configuration
config interface lan
    option type bridge
    option ifname „eth0.0“
    option proto static
    option ipaddr <wl-500-ip>
    option netmask 255.255.255.0
    option gateway 192.168.1.1
    option dns 192.168.1.1
```

Ein einfacher Editor, welcher auf jedem Linux-System installiert ist, ist vi. Hinweise zu der etwas ungewöhnlichen Bedienung finden sich unter [8].

Standardmäßig läuft auf dem WL-500 ein DHCP-Server. Für den Betrieb als Client im Hausnetz kann dieser mit

```
$ /etc/init.d/dnsmasq disable
```

abgeschaltet werden. Der WL-500 kann nun an das Netzwerk angeschlossen werden und sollte nach einem Neustart unter der eingestellten IP-Adresse zu finden sein. Jetzt kann auch der Host-PC wieder auf die ursprünglichen Einstellungen zurückgesetzt werden. Nach erneutem Einloggen auf dem WL-500 per ssh sollte eine Verbindung zum Internet vorhanden sein, was z. B. mittels ping google.de getestet werden kann.

Im nächsten Schritt kann nun die Weboberfläche installiert werden, welche bei OpenWrt nicht enthalten ist und in das separate Projekt X-Wrt [13] ausgelagert wurde. Dazu sollten zunächst die Paketinformationen auf den neuesten Stand gebracht und anschließend das Paket für die Weboberfläche installiert werden:

```
$ ipkg update
$ ipkg install http://downloads.x-wrt.org/xwrt/
kamikaze/7.09/brcm47xx-2.6/webif_latest.ipk
```

Die WL-500-Website ist nun unter voreingestellter IP über einen Browser zugänglich; Benutzername und Passwort sind die selben wie für den SSH-Login. Über die Weboberfläche können Statusinformationen abgefragt, reguläre Routereinstellungen vorgenommen und ipkg-Pakete verwaltet werden.

IPKG steht für „Itsy Package Management System“ [4] und wurde als ressourcenschonendes Paketsystem für Mikrorechner, Handhelds und Embedded-Systeme entwickelt. Die Kontrollprogramme sind auf das Notwendigste beschränkt, und auch die .ipk-Pakete sind sehr feingranular strukturiert und entsprechend schlank gehalten.

Schnelleres Einloggen mit SSH-Keys

Bevor nun mit der Software-Entwicklung begonnen wird, sollten zunächst noch einige Vorbereitungen getroffen werden, um die spätere Arbeit zu erleichtern. Die Entwicklung auf dem Host-System und das Testen auf dem WL-500 bringen häufige SSH-Verbindungen und SCP-

Kopiervorgänge (Secure Copy) mit sich. Der Anwender kann sich die häufige Eingabe des Passwortes ersparen, indem er SSH-Schlüssel verwendet. Um sich vom Hostsystem auf dem WL-500 direkt ohne Passwordeingabe einloggen zu können, muss dort der öffentliche Schlüssel des Hostrechners liegen. Falls auf dem Host noch kein Schlüsselpaar vom Typ dsa vorhanden ist, kann dieses mit folgenden Befehlen erzeugt werden:

```
$ cd ~/.ssh
$ ssh-keygen -t dsa -f id_dsa
```

Nur der öffentliche Schlüssel wird vom Host aus mittels

```
$ scp ~/.ssh/id_dsa.pub root@<wl500-ip>:/tmp
```

in das Verzeichnis /tmp (flüchtiger Speicher) des WL-500 kopiert. Dort eingeloggt, muss der Schlüssel nun der Datei authorized_keys hinzugefügt werden. Dort sind die öffentlichen Schlüssel von allen Rechnern hinterlegt, die sich direkt am WL-500 anmelden dürfen.

```
$ cd /etc/dropbear
$ cat /tmp/id_dsa.pub >> authorized_keys
$ chmod 0600 authorized_keys
```

Im Unterschied zu Mehrbenutzersystemen wie Debian liegen das Schlüsselpaar und die Autorisierungsinformationen nicht im Home-Verzeichnis, sondern global in /etc/dropbear, dem Konfigurationsverzeichnis des SSH-Server- und Client-Programms. Wer Wert auf zusätzliche Sicherheit legt, der kann in der Datei etc/config/dropbear die Möglichkeit des Passwort-Logins deaktivieren oder gegebenenfalls den SSH-Port wechseln:

```
# option PasswordAuth ,on'
option PasswordAuto ,off'
# option Port '22'
option Port '9222'
```

Wichtig: Vorher sollte man den automatischen Log-In getestet haben, da man sich sonst mit dieser Umstellung u.U. aus dem System aussperrt.

Auch wenn diese Kommunikationsrichtung der Regelfall ist, so kann es notwendig sein, den anderen Weg zu gehen und vom WL-500 beispielsweise automatisch beim Neustart eine Datei vom Server zu holen. In etc/dropbear liegen dazu bereits zwei private Schlüssel dropbear_rsa_host_key und dropbear_dss_host_key, jedoch noch keine öffentlichen Schlüssel. Diese können mit folgendem Befehl erzeugt und anschließend auf das Host-System übertragen werden, auf welchem automatisch eingeloggt werden soll:

```
$ dropbearkey -y -f /etc/dropbear/
dropbear_rsa_host_key |
grep ssh-rsa > /etc/dropbear/
dropbear_rsa_host_key.pub
$ scp /etc/dropbear/dropbear_rsa_host_key.
pub <username>@<host_address>:
```

Nach dem Anmelden am Hostsystem kann der öffentliche WL-500-Schlüssel dort zur Menge der bereits autorisierten Schlüssel im Home-Verzeichnis des Anwenders hinzugefügt werden:

```
$ cat ~/.dropbear_rsa_host_key.  
pub >> ~/.ssh/authorized_keys
```

Vom WL-500 aus ist der Hostrechner nun mit folgendem Befehl, allerdings nur unter der Angabe des öffentlichen Schlüssels mit Option `-i` per `scp` oder `ssh` erreichbar:

```
$ ssh -i /etc/dropbear/dropbear_rsa_  
host_key root@<host_address>
```

OpenWrt – „Hello World“ auf dem WL-500

Nachdem nun das OpenWrt-Build-System die gesamte Toolchain mit Cross-Compilern erstellt hat, ist die Benutzung ein Kinderspiel. Die Cross-Compiler liegen im Verzeichnis `openwrt/staging_dir_mipsel/bin` und heißen `mipsel-linux-uclibc-<...>`.

Durch Hinzufügen der Zeile

```
export PATH=<openwrt_dir>/  
staging_dir_mipsel/bin/:$PATH
```

beispielsweise zu `~/.bashrc` sind die Cross-Compiler nach einem Neustart der Shell verfügbar. Das Hallo-Welt-Beispiel in `src/helloworld.c` [21] kann durch

```
$ mipsel-linux-gcc helloworld.c -o hello
```

übersetzt werden. Ein Aufruf von

```
$ file hello
```

zeigt, dass es sich bei der Binärdatei tatsächlich um eine ausführbare Datei für die MIPS-Architektur handelt (genauer: MIPSEL, da Little Endian verwendet wird). Das Testprogramm sollte sich nach dem Kopieren auf den WL-500 auf diesem ausführen lassen. Falls das Beispiel mit dem C++ Compiler `mipsel-linux-g++` übersetzt wird, so erscheint allerdings beim Ausführen die Fehlermeldung

```
$ ./hello: can't load library ,libstdc++.so.6'
```

Die Standard C++-Library ist auf dem WL-500 nicht von Haus aus installiert, dies kann mit

```
$ ipkg install libstdc++
```

nachgeholt werden. Wenn auch in diesem Fall nicht zwingend, so ist dies doch spätestens für die folgenden Beispiele notwendig. Die Cross-Compiler können natürlich auch in einer integrierten Entwicklungsumgebung wie beispielsweise Eclipse [14] verwendet werden. Dazu muss lediglich in den Projekteinstellungen unter C/C++ Build eine weitere Konfiguration angelegt, und es müssen in den dortigen Einstellungen Settings

die Befehle für Compiler, Linker und Assembler in Aufrufe des Cross-Compilers geändert werden.

IOWarrior-Erweiterung

In [19] wurden die IOWarrior-USB-Controller bereits ausführlich vorgestellt, auf die reguläre Installation an einem Windows- oder Linux-PC soll deshalb an dieser Stelle nicht mehr eingegangen werden. Zum Betrieb eines IO-Warriors wird ein Kernelmodul benötigt, welches Routinen zur Kommunikation mit dem IO-Warrior-Baustein bereitstellt, und seinerseits weitere Kernelmodule zum USB-Zugriff verwendet.

Die IOWarrior-Platine sollte zu Beginn nicht eingesteckt sein.

Eine Installation des Kernelmoduls (siehe Dokumentation LinuxSDK in Downloads [1]) auf einem Debian-System funktioniert nach Schema F. Um ein Kernelmodul für OpenWrt zu bauen, bedarf es aber kleinerer Kniffe. Zunächst wird für das neue Paket ein Verzeichnis im Paketordner angelegt:

```
$ mkdir <openwrt_dir>/package/iowarrior
```

Jedes Paket muss bei OpenWrt in das Build-Environment integriert und diesem beschrieben werden, dies geschieht in Form einer Makefile-Datei (verfügbar unter `src/kmod-warrior/Makefile` [21]) im Paketverzeichnis. Für das Paket `kmod-iowarrior` sieht diese folgendermaßen aus:

Wenn die Paketquellen (`src/kmod-iowarrior/src`) nach `<openwrt_dir>/package/iowarrior/` kopiert wurden, dann kann nun der Konfigurationseditor über

```
$ make menuconfig
```

gestartet werden. Das Modul `kmod-iowarrior` sollte in der Rubrik `Kernel_modules->Other_modules` auftauchen. Um das Flash-Image nicht neu aufspielen zu müssen, wird das Modul nicht per `<*>` in den Kernel integriert, sondern mit `<M>` als Modul gebaut.

Nach dem Beenden und Übersetzen mit `make` liegt nach einigen Minuten im Verzeichnis `<openwrt_dir>/bin/package/` eine Datei `kmod-iowarrior_2.6.xxx.ipk`. Diese ist auf den WL-500 zu kopieren und dort mittels

```
$ ipkg install kmod-iowarrior_2.6.xxx.ipk
```

zu installieren. Durch die Installationsanweisungen sollte nun auch in `/etc/udev/rules.d/` ein Eintrag `10-iowarrior.rules` angelegt sein. In dieser Datei ist die Verbindung zwischen Kernelmodul und Gerätenamen hinterlegt. Erkennt das `iowarrior`-Modul eine angeschlossene Platine, so werden neue Geräte unter `/dev/iowarrior` erzeugt. Da das Kernelmodul auf grundlegende USB-Unterstützung angewiesen ist, werden auch die Pakete `usbcore` und `uhci_hcd` benötigt. Über den Befehl `lsusb` (enthalten im Paket `usbutils`) können angeschlossene USB-Geräte aufgelistet werden – auch dies wird später eine Hilfe sein:

```

# *****
# Dateiname:          Makefile
# Autor:              Joachim Schroeder (C)
# Datum:             12.07.2008
#
# Beschreibung:       Beschreibung für das OpenWrt-Paket
#                    kmod-iowarrior, Kerneltreiber für IOwarrior
#                    24,40 und 56.
#
# *****

include $(TOPDIR)/rules.mk
include $(INCLUDE_DIR)/kernel.mk

# Platzhalter für Paketnamen, Versions- und Releasenummer
PKG_NAME:=kmod-iowarrior
PKG_RELEASE:=1

# Build-Verzeichnis, z.B.:
# <openwrt>build_mipsel/linux-2.6-brcm47xx/
PKG_BUILD_DIR:=$(KERNEL_BUILD_DIR)/$(PKG_NAME)

include $(INCLUDE_DIR)/package.mk

# Paketbeschreibung mit Name, Kategorie und Abhängigkeiten
define KernelPackage/iowarrior
    SUBMENU:=Other modules
    DEPENDS:=@PACKAGE_kmod-usb-core
    TITLE:=IOwarrior driver
    DESCRIPTION:=\
        This package contains driver for IOwarriors 24, 40 and 56.
    VERSION:=$(LINUX_VERSION)-$(BOARD)-$(PKG_RELEASE)
    FILES:= \
        $(PKG_BUILD_DIR)/iowarrior.$(LINUX_KMOD_SUFFIX)
    AUTOLOAD:=$(call AutoLoad,80,usb-core)
endef

# Beschreibt Aktionen die vor dem Übersetzen
# ausgeführt werden sollen, hier:
# Quelldateien in das Build-Verzeichnis kopieren
define Build/Prepare
    mkdir -p $(PKG_BUILD_DIR)
    $(CP) ./src/* $(PKG_BUILD_DIR)/
endef

# Anweisungen für den Übersetzungsvorgang
define Build/Compile
    $(MAKE) -C „$(LINUX_DIR)” \
        CROSS_COMPILE=$(TARGET_CROSS) \
        ARCH=$(LINUX_KARCH) \
        SUBDIRS=$(PKG_BUILD_DIR) \
        EXTRA_CFLAGS=$(BUILDFLAGS) \
        modules
endef

# Installationsanweisungen, werden bei der Installation des
# Paketes auf dem Zielsystem ausgeführt
# $(INSTALL_DIR) prüft ob das folgende Verzeichnis existiert
# $(1) ist das Wurzelverzeichnis
# $(INSTALL_BIN) kopiert Daten, in diesem Fall die udev-Regeln
define KernelPackage/iowarrior/install
    $(INSTALL_DIR) $(1)/etc/udev/rules.d/
    $(INSTALL_BIN) ./src/10-iowarrior.rules $(1)/etc/udev/rules.d/
endef

$(eval $(call KernelPackage,iowarrior))

```

Listing 1.

```
$ ipkg install kmod-usb-core
$ ipkg install kmod-usb-uhci
$ ipkg install usbutils
```

Nach einem Neustart kann das Modul iowarrior nun mit

```
$ insmod iowarrior
```

geladen werden. Die Ausgabe von lsmod sollte dieses unter den geladenen Modulen auflisten und dessen Benutzung von usbcore zeigen. Der Befehl dmesg („diagnostic message“) zeigt den Nachrichtenpuffer des Kernels. Dieser sollte eine Nachricht von usbcore enthalten, was mitteilt, dass gerade ein neuer Treiber registriert wurde. Sollten Schwierigkeiten beim Laden der Module auftreten (falsche Reihenfolge oder verschiedene Kernelversionen), so ist dies eine Möglichkeit, der Ursache auf die Schliche zu kommen.

Eine gute Anleitung zur Erstellung eigener OpenWrt-Pakete findet sich in [20].

Praktisch wäre es, wenn das iowarrior-Modul direkt beim Start geladen würde (oder womöglich auch andere Befehle ausgeführt werden). Die sauberste Lösung ist das Anlegen eines Skripts im Verzeichnis /etc/init.d/.

Dazu ist eine Datei

```
$ vi /etc/init.d/iowarrior
```

zu erstellen und folgender Inhalt einzugeben:

```
#!/bin/sh /etc/rc.common
START=99

start () {
    insmod iowarrior
}
```

Die Startnummer gibt dabei die Reihenfolge des Skripts an, um die Aufrufe zu ordnen. In diesem Fall wird das Skript ganz am Ende der Reihe aufgerufen. Um das Skript ausführbar zu machen wird

```
$ chmod a+x iowarrior
```

angewendet. Eine Aktivierung durch

```
$ /etc/init.d/iowarrior enable
```

legt das Skript in /etc/rc.d/ ab und beim nächsten Systemstart wird das Modul direkt geladen. Eine wichtige Anwendung fehlt noch bevor getestet werden kann: Für die Erzeugung von Geräten wurden bereits Regeln hinterlegt, das eigentliche Programm zur Geräteverwaltung muss jedoch noch über

```
$ ipkg install udev
```

installiert werden. Werden auch später noch Regeln hinzugefügt, so können diese über

```
$ udevcontrol reload_rules
```

nachgeladen werden.

Jetzt ist der große Moment gekommen: Die IO-Warrior-Platine kann eingesteckt werden und sollte nach einem Aufruf von lsusb zumindest als USB-Gerät erkannt worden sein. Mit etwas Glück wurden jetzt sogar zwei Geräte als /dev/iowarrior0 bzw -1 erzeugt, die zur Kommunikation in den beiden Modi Standard und Special benutzt werden können. Im letzten Teil soll der IOWarrior nun für eine I²C-Kommunikation zum Einsatz kommen.

Benutzung der IOWarrior-I²C-Schnittstelle

Ein Test von Anwendungen auf dem PC ist natürlich wesentlich komfortabler, als eine Übertragung mit Test auf dem Zielsystem, und dank IOWarrior auch einfach möglich. Das mitgelieferte Makefile in src/ zeigt eine Möglichkeit, Compiler und Flags für zwei Zielsysteme zu setzen. Über den Eintrag

```
CXX_STD = g++
CXX_CROSS = mipsel-linux-uclibc-g++
```

werden die beiden Compiler angegeben (Das Verzeichnis des Cross-Compilers muss in der Variablen PATH eingetragen sein!), und über den Aufruf make bzw. make CROSS=1 der Standard-oder der Cross-Compiler verwendet. Alle mitgelieferten Beispiele sind in das Makefile integriert, so dass das vorige Hello World-Beispiel nun einfach für beide Systeme übersetzt und getestet werden kann.

Die Kernelfunktionen des IOWarrior-Treibers wie auch die API der Bibliothek libiowkit stellen keine Funktionen zur Verfügung, um vollständige I²C-Nachrichten unter Angabe von Adresse und Datenbytes zu schreiben. Eine Erweiterung in iowarrior_i2c.c und iowarrior_i2c.h schafft hier Abhilfe. Die Implementierung ist bewusst in C erfolgt, um die Einbindung in C-Anwendungen zu ermöglichen, wird in den folgenden Beispielen aber auch in C++ Klassen verwendet. Folgende Funktionen sind enthalten:

- void iow_print_warriors(const char* basename);
Listet alle IOWarrior-Geräte auf, die im System unter basename auftauchen (beispielsweise /dev/iowarrior). Damit kann ggf. die Seriennummer ausgelesen werden.
- int iow_find_descriptor(const char *serial, const char* basename);
Gibt den File-Descriptor für Zugriff auf den Spezialmodus zurück, über welchen auch I²C angesprochen wird. Der IOWarrior-Baustein ist über Seriennummer und Basisnamen anzugeben, so können mehrere Platinen unterschieden werden.
- int iow_i2c_write(char address, char *buf, int size, int report_size, int fd);
Schreibt an eine I²C-Adresse eine Anzahl von size Datenbytes, beginnend an Stelle *buf. Die Länge eines Reports muss zuvor mit iow_init() ermittelt werden, der Filedescriptor mit iow_find_descriptor().

```

// *****
//
// Dateiname:      Ex1_IOWarriorSimple.cpp
// Autor:         Joachim Schroeder (c)
// Datum:        2008-07-12
//
// Beschreibung:
//
// I2C-Bus-Ansteuerung mit IOWarrior am Beispiel
// einer Temperaturmessung mit Maxim DS1631. Hilfs-
// routinen in iowarrior_i2c.h erleichtern die Verwendung.
//
// *****

#include <stdio.h>
#include „iic/iowarrior_i2c.h“

int main (int argc, char **argv)
{
    char serial[9] = "00001FEC";
    char basename[20] = "/dev/iowarrior";
    char buf[20];
    int i;
    char address = 0x4f;

    iow_print_warriors(basename);

    int fd = iow_find_descriptor(serial, basename);

    if ( fd < 0 )
    {
        printf(„Device with serial 0x%s could not be found\n“, serial);
        exit(0);
    }

    int report_size = iow_i2c_init(fd);

    // read data from iic, read config
    buf[0] = 0xac;      // command: access config reg

    if (iow_i2c_write(address, buf, 1, report_size, fd) != 1)
        printf(„iic write error\n“);

    if (iow_i2c_read(address, buf, 1, report_size, fd) != 1)
        printf(„iic write error\n“);
    else
        printf(„read command byte 0x%0x\n“, buf[0]);

    // start conversion
    buf[0] = 0x51;      // command: access config reg

    if (iow_i2c_write(address, buf, 1, report_size, fd) != 1)
        printf(„iic write error\n“);

    while(1)
    {
        // read data from iic, read temperature
        buf[0] = 0xaa;      // command: access config reg
        if (iow_i2c_write(address, buf, 1, report_size, fd) != 1)
            printf(„iic write error\n“);
        if (iow_i2c_read(address, buf, 2, report_size, fd) != 2)
            printf(„iic write error\n“);
        else
            printf(„read command byte 0x%0x and 0x%0x\n“, buf[0], buf[1]);
        sleep(1);
    }

    close(fd);
}

```

Listing 2.

- `int iow_i2c_read(char address, char *buf, int size, int report_size, int fd);`
Dito, Daten werden an der Stelle `*buf` abgelegt.
- `int iow_init(int fd);`
Setzt einen IOWarrior-Baustein in den I²C-Modus und liefert die Länge eines Reports zurück. Diese kann für verschiedene Typen unterschiedlich sein und muss für Lese- und Schreibzugriffe angegeben werden.

Listing 2 `Ex1_IOWarriorSimple.cpp` [21] zeigt die Verwendung der `iowarrior_i2c`-Routinen und versucht, nach Auflisten aller vorhandenen IOWarrior-Geräte, eine Platine mit Seriennummer 00001FEC zu finden, um einen daran angeschlossenen Temperatursensor vom Typ DS1631 mit Adresse 0x4f auszulesen.

Zwei weitere Beispiele `Ex2_TempTest` und `Ex3_StepperTest` zeigen ein Konzept zur Strukturierung mehrerer I²C-Komponenten in separate C++-Klassen. Die IOWarrior-Schnittstelle wird dabei zentral von der Klasse `IICBus` geöffnet, welche alle angeschlossenen Geräte verwaltet. Die I²C-Komponenten werden bei der Erstellung direkt einem existierenden Bus-Objekt zugewiesen. Schreib- und Lesezugriffe erfolgen nur zentral über das Bus-Objekt.

Falls Multi-Threading verwendet werden soll, so sind die Zugriffsroutinen in der Klasse `IICBus` noch threadsafe zu gestalten.

Die Klasse `IICTempSensor` repräsentiert I²C-Temperatursensoren auf Basis eines Maxim DS1631 [5] und bietet Zugriffsmethoden für die einzelnen Funktionen. Damit sind Temperaturmessungen im Bereich 0 bis 70°C mit einer Auflösung von 0,5°C möglich.

Für einen I²C-Schrittmotortreiber vom Typ TMC222, wie er in [6] verwendet wird, bietet die Klasse `IICStepper` Unterstützung. Alle Zugriffsfunktionen für den Chip wurden als Klassenmethoden implementiert, aus Gründen der Nachvollziehbarkeit wurde die Namensgebung übernommen. Weitere `get()`-Methoden erlauben die gezielte Abfrage einzelner Systemwerte.

Da leider im Chip keine Funktion existiert um eine Referenzfahrt auf den Endschalter durchzuführen (welcher jedoch durch Polling abgefragt werden kann), wurde als Ersatz eine Methode `referencelnit()` nachimplementiert.

Mithilfe dieser Beispiele sollte es nun leicht möglich sein, die I²C-Schnittstelle des IOWarrior-Bausteines in eigenen Anwendungen zu nutzen oder weitere I²C-Komponenten hinzuzufügen.

Im OpenWrt-Forum [16] finden sich zahlreiche Anleitungen zur Erweiterung von Routern um USB-Audio-Sticks, SD-Karten oder Displays. Auch Fragen zu Software und Treiberproblemen werden hier beantwortet. Für weiterführende Informationen zum Thema Embedded Linux sei an dieser Stelle auf [18] verwiesen.

Quellen und Bezugsquellen

- [1] Fa. Code Mercenaries in 12529 Schönefeld / Großziethen, Hersteller der USB-Warrior-Bausteine. Downloads, Datenblätter und Forum. Anm.: Direktvertrieb an Privatkunden nur über den Webstore und nur mit den dort gelisteten Produkten.
<http://www.codemerccs.com>
<http://www.codemerccs.com/DirectsalesD.html>
<http://www.codemerccs.com/IOWdownloadsD.html>
- [2] OpenWrt Projekt-Website – Linux-Distribution für Embedded-Geräte
<http://openwrt.org>
- [3] Ubuntu-Projekt-Website – linuxbasiertes Betriebssystem
<http://www.ubuntu.com>
- [4] Itsy Package Management System (iPKG)
<http://handhelds.org/moin/moin.cgi/lpkg>
- [5] I²C-Thermometermodul, Bezug über Conrad Electronic, Artikel-Nr. 198298-62
<http://www.conrad.de>
- [6] I²C-Schrittmotormodul, eine Achse, 0,8 A, Bezug über Conrad Electronic, Artikel-Nr. 198266-62
<http://www.conrad.de>
- [7] Linux-Kompendium: Ubuntu / Arbeiten mit dem Terminal
http://de.wikibooks.org/wiki/Linux-Kompendium:_Ubuntu/_Arbeiten_mit_dem_Terminal
- [8] Kurzanleitung zum vi-Editor
<http://www.fehcom.de/pub/viref.pdf>
- [9] Grundlagen zum Nano-Editor
<http://www.gentoo.de/doc/de/nano-basics-guide.xml>
- [10] Synaptic, grafische Oberfläche zur Paketverwaltung apt
<https://help.ubuntu.com/community/SynapticHowto>
- [11] How-To für die Paketverwaltung apt
<http://www.debian.org/doc/user-manuals#apt-howto>
- [12] SSH und SCP unter Unix – eine Einführung
<http://www.uni-koeln.de/rrzk/netze/ssh/sshscp.html>
- [13] Projekt-Website X-Wrt, Web-Oberfläche für OpenWrt
<http://x-wrt.org>
- [14] Freie Entwicklungsplattform Eclipse
<http://www.eclipse.org>
- [15] Versionskontrolle mit Subversion – Freie Dokumentation
<http://svnbook.red-bean.com>
- [16] OpenWrt-Forum
<http://forum.openwrt.org>
- [17] MAX233-Adapterschaltung zur Anpassung an den RS232-Pegel
http://www.comsys1.com/workbench/On_top_of_the_Bench/Max233_Adapter/max233_adapter.html
- [18] Joachim Schröder, Tilo Gockel, Rüdiger Dillmann: Embedded Linux – Das Praxisbuch. Springer-Verlag, Heidelberg, 2008. ISBN: 9783540786191.
<http://www.praxisbuch.net>
- [19] Tilo Gockel: „USB-Anschluss leichtgemacht“ – Die USB-Mikrocontroller der Firma Code Mercenaries. Beitrag für das Elektor-Mikrocontroller-Sonderheft Nr. 3.
- [20] OpenWrt Forum – An Introduction to OpenWrt Build Environment
<http://forum.openwrt.org/viewtopic.php?pid=31794-p31794>
- [21] Quellcode-Dateien der Beispiele in diesem Artikel
<http://www.iaim.ira.uka.de/embedded-robotics/elektor>