Embedded Linux leicht gemacht (5)

I/O, ADC, PWM, LAN & Webserver

Das Elektor-Linux-Board bringt eine Vielzahl von Möglichkeiten für die Anwendungsentwicklung mit. In dieser Folge werden wir digitale und analoge Signale einlesen. Dazu kommen eine Netzwerkverbindung und ein kleiner Webserver, der HTML-Seiten dynamisch generiert. Damit kann man sich den Status von LEDs und vielem mehr in einem Browser anzeigen lassen, und auch das Schalten und Walten aus der Ferne ist möglich.

Von Benedikt Sauter [1]

Inzwischen laufen Bootloader und Kernel, und viele Leser haben auch schon erste Erfahrungen mit dem Dateisystem und der SD-Karte gemacht (siehe hierzu auch den Textkasten „SD-Karten-Image“). In diesem Artikel geht es nun stetig auf die erste „richtige“ Anwendung zu. Da Embedded-Linux-Applikationen meist dort zum Einsatz kommen, wo Maschinen gesteuert und Daten erfasst werden, zeigen wir in diesem Kursteil zuerst, wie man digitale und analoge Signale einliest und ausgibt. Danach richten wir eine Netzwerkverbindung ein. Hiermit bekommen wir einen Fernzugang zum Board, das sich nun zum Beispiel über eine Webseite steuern lässt.

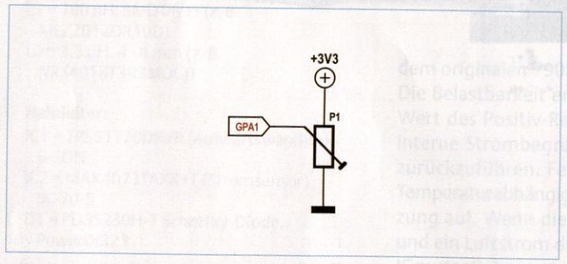

Bild 1. Mit einem Poti kann man den A/D-Wandler testen.

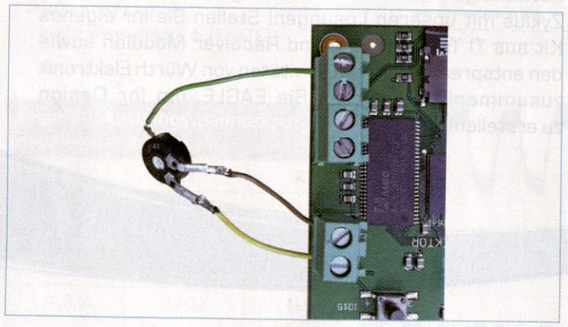

Bild 2. Der analoge Eingang ist auf eine Schraubklemme herausgeführt, so fällt das Anschließen leicht.

Digitale I/O-Pins

Im zweiten Teil dieser Serie haben wir bereits die LED1 auf dem Board ein- und ausgeschaltet. Die LED hängt am GPIO3-Pin des Prozessors. Diese GPIO-Pins können als Ein- oder Ausgang, aber auch als Interrupteingang verwendet werden. Die Vorgehensweise bei der Initialisierung solch eines I/O-Pins dürfte für die meisten Elektor-Leser nichts Neues sein:

- Aktivierung des Pins als GPIO.
- Initialisierung der Datenrichtung.
- Ausgabe eines Wertes bzw. Einlesen des aktuell anliegenden Signalpegels.

Unter Embedded Linux können wir die GPIO-Pins über die Gerätetreiber von der Konsole aus ansprechen.
Zuerst wechseln wir in den Ordner für die Kommunikation mit dem GPIO-Treiber:

```
cd /sys/class/gpio
```

Dann folgt die Aktivierung des mit der LED verbundenen Pins als GPIO (siehe Schaltplan in [2]):

```
echo 3 > export
```

Nun wollen wir gleich auch den Pin, der mit dem Taster verbunden ist, als GPIO aktivieren:

```
echo 15 > export
```

Dann konfigurieren wir die Pins als Aus- bzw. Eingang:

```
cd gpio3
echo „out“ > direction
```



```
cd ../gpio15
echo „in“ > direction
```

Nun können wir die LED (wie schon einmal gezeigt) mit den folgenden Befehlen ein und ausschalten:

```
cd ../gpio3
echo 1 > value

echo 0 > value
```

Der Zustand des Tasters steht immer in einer (virtuellen) Datei namens „value“. Deren Inhalt kann man einfach mit dem Tool `cat` ausgeben:

```
cd ../gpio15
cat value
```

jetzt sollte man in der Lage sein, auch das Relais auf dem Board zu schalten. Das Relais hängt an GPIO18, der Pin ist natürlich als Ausgang zu konfigurieren.

Analog/Digital-Wandler

Der LPC3131 bietet vier analoge Eingänge mit je maximal 10 bit Auflösung. Der Wertebereich des Ergebnisses ist dann 0..1023 (bzw. 0..0x3FF in Hex-Darstellung). Als Spannungsreferenz wird 3,3 V verwendet, die Versorgungsspannung der I/O-Bank, auf der die Pins liegen.
Der Zugriff auf die gewandelten Werte läuft ähnlich wie beim Taster. Für die A/D-Wandler gibt es einen eigenen Treiber, der aber die Werte nur immer einzeln pro Kanal ausgeben kann. Zuvor muss man einstellen, welchen Kanal man abfragen möchte.

Am einfachsten kann man das Verhalten des A/D-Wandlers mit einem einfachen (Trimm-)Poti testen. Die Schaltung ist in **Bild 1** dargestellt. Das Potentiometer wird direkt an die 3,3-V-Versorgungsspannung angeschlossen, den Schleifer verbinden wir mit dem Pin GPA1, der auf eine Schraubklemme herausgeführt ist.
Das Ganze sollte dann grob wie in **Bild 2** aussehen. Jetzt können wir den A/D-Wandler initialisieren und nacheinander Werte ausgeben lassen, siehe hierzu **Bild 3**.
Möchte man zum Testen nicht immer wieder den gleichen Befehl eingeben, kann man dies mit dem Programm „watch“ automatisieren. Nach Eingabe von:

```
watch -n 1 cat /dev/lpc313x_adc
```

ruft das Tool ein Mal pro Sekunde den gewünschten Befehl auf. Mit Strg-C bricht man dies wieder ab.
Neben GPA1 sind auf dem Elektor-Linux-Board auch die ADC-Kanäle GPA0 und GPA3 verfügbar, und zwar an der Stiftleiste J5 (GPA2 ist nicht herausgeführt).

Möchte man den Eingang des A/D-Wandlers noch (in gewissen

```
root@gnublin:~# echo 1 > /dev/lpc313x_adc
set res to 10
root@gnublin:~# cat /dev/lpc313x_adc
0x21f
root@gnublin:~# cat /dev/lpc313x_adc
0x11d
```

Bild 3. Initialisierung des ADCs und Ausgabe von zwei Werten.

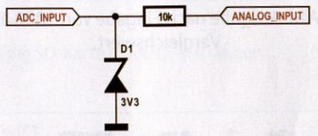

Bild 4. Schutz des ADC-Eingangs.

Grenzen) gegen Überstrom und Überspannung schützen, so kann man einen 10-k-Widerstand in Serie schalten und zusätzlich eine 3,3-V-Zener-Diode gegen Masse anschließen (**Bild 4**).

PWM-Signal erzeugen

Die so genannte Puls-Weiten-Modulation hat viele Anwendungen: Spannungen erzeugen, Servo-Ansteuerung, Audio und vieles mehr [3]. Auf dem Elektor-Linux-Board ist ein PWM-Ausgang des Controllers auf die Stiftleiste J5 herausgeführt. Für einen Test schließen wir

Listing 1: PWM.

```
#include <stdio.h>
#include <stdlib.h>

#ifndef abs
#define abs(x) ((x) < 0 ? -(x) : (x))
#endif

int pwm(int value) {
   FILE* f = fopen(„/dev/lpc313x_pwm“, „wb“);
   fputc(value & 0xff, f);
   fputc((value >> 8) & 0xff, f);
   fclose(f);
}

int main() {
   int value = 0;
   int b;

   while(1) {
      b = abs(63 - 2*value);
      pwm(b * b);
      value = (value + 1) % 64;
      usleep(1000);
   }
}
```

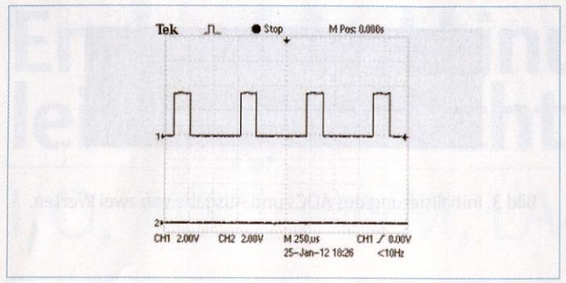

Bild 5. PWM-Ausgabe nach Eingabe von 1000 als Zähler-Vergleichswert.

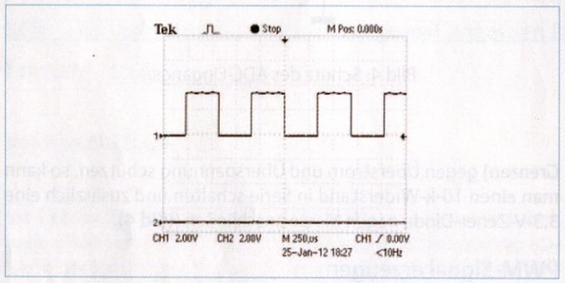

Bild 6. PWM mit 50 % Tastverhältnis.

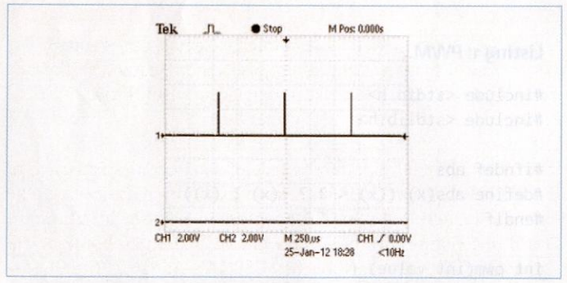

Bild 7. PWM mit 1 % Tastverhältnis.

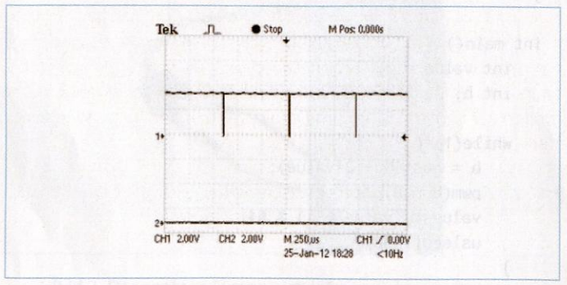

Bild 8. PWM mit 99 % Tastverhältnis.

dort am besten ein Oszilloskop an.
Im PWM-Modus inkrementiert der Controller einen 12 bit-Zähler pro Zählertakt. Sobald ein bestimmter Vergleichswert erreicht ist, wechselt die Flanke am PWM-Pin von High nach Low (wenn der Zähler überläuft, wird er zurückgesetzt und gleichzeitig der Pin wieder auf High gezogen). Als Vergleichswert kann man einen beliebigen 12-bit-Wert heranziehen, das entspricht einem Bereich von 0..4095. Wird als Wert eine 0 eingegeben, so wechselt das PWM-Signal sofort auf Low. Bei einer Eingabe von 2000 bekommen wir ein Tastverhältnis von etwa 50 %.

Da der PWM-Treiber im Gegensatz zu den I/O- und ADC-Treibern einen binären Eingabewert benötigt, können wir nicht einfach mit `echo` und `cat` arbeiten, da diese eine Eingabe automatisch als Zeichen (ASCII) interpretieren. Wir benötigen daher ein kleines Hilfsprogramm.
In C haben wir uns dies recht schnell auf dem Board selbst geschrieben. Doch man kann es noch etwas bequemer haben, denn solch ein Programm gibt es bereits (**Listing 1**). Im Home-Ordner, in dem man sich automatisch nach dem Anmelden befindet, liegt die Datei „pwm.c“. Im Code muss nur noch der Name der Gerätedatei angepasst werden.
Zuerst öffnen wir die Datei mit einem Editor auf dem Board:

```
nano pwm.c
```

Mit den Pfeiltasten geht man nun zur folgenden Zeile ...

```
FILE* f = fopen(“/dev/pwm”, “wb”);
```

... und passt sie entsprechend an:

```
FILE* f = fopen(“/dev/lpc313x_pwm”, “wb”);
```

Jetzt kann man die Datei mit Strg-o speichern und den Editor mit Strg-x beenden. Übersetzen kann man den Code auf dem PC, aber auch direkt auf dem Linux-Board, da wir dort ebenfalls über einen Compiler verfügen:

```
gcc -o pwm pwm.c
```

Ist das Programm übersetzt (dies dauert einen Moment) kann man es im Anschluss direkt ausführen:

```
./pwm
```

Im Oszi sieht man nun, wie sich das Tastverhältnis zwischen den Impulsen und Pausen stetig ändert.

Möchte man ein Signal mit einem festen Puls-/Pausenverhältnis ausgeben, kann man dies zum Beispiel mit einem kleinen Skript erledigen, das in der Programmiersprache Python geschrieben ist. Im Home-Ordner liegt schon eine Datei „pwm.py“.
Zuerst starten wir den Python-Interpreter:

SD-Karten-Image

Beim Experimentieren – zu dem wir herzlich einladen! – kann es durchaus einmal sein, dass man in eine Sackgasse gerät. Da ist es mitunter hilfreich, wenn man den ursprünglichen Zustand der SD-Karte wieder herstellen kann. Deshalb bieten wir den Inhalt der SD-Karte als Extra-Download an. Zuerst lädt man das Image von der Elektor-Website herunter [8] (Download „SD Card Image“, 120180-12.zip). Anschließend entpackt man das Archiv:

```
unzip 120180-12.zip
```

Es dauert dann einen kurzen Moment bis die folgende Ausgabe erscheint:

```
Archive: ../120180-12.zip inflating: Elektor_Linux_Board -
Build_New_SD_Card.txt inflating: gnublin.img
```

Jetzt kann man die zu beschreibende SD-Karte in den PC oder Kartenleser stecken. Das System hängt die SD-Karte nun automatisch ein, was wir hier aber nicht brauchen, denn wir möchten das Image 1:1 auf die SD-Karte schreiben. Daher muss man die Karte zuerst wieder manuell aushängen.

Am besten geht man auf die Konsole und gibt dort nach dem Anstecken des Kartenlesers den Befehl `dmesg` ein. Nun erhält man eine Ausgabe wie:

```
[ 1069.427374] sdf: sdf1 sdf2 [ 1069.430857] sd
 5:0:0:0: [sdf] No Caching mode page present [
 1069.430863] sd 5:0:0:0: [sdf] Assuming drive cache:
 write through [ 1069.430868] sd 5:0:0:0: [sdf]
 Attached SCSI removable disk [ 1070.002620] EXT2-
 fs (sdf1): warning: mounting unchecked fs, running
 e2fsck is recommended
```

In den letzten Zeilen steht, was für Gerätenamen der Kernel für die SD-Karte vergeben hat.

Jetzt muss man die SD-Karte manuell aushängen ...

```
umount /dev/sdf1
```

... wobei `sdf1` durch den Gerätenamen der SD-Karte zu ersetzen ist (genau genommen den Gerätenamen der ersten Partition).

Dann kann man das heruntergeladene Image auf die SD-Karte schreiben:

```
sudo dd if=gnublin.img of=/dev/sdf
```

`sdf` ist hier die Bezeichnung der ganzen Karte als Blockdevice.

Hinweis: Der Schreibvorgang kann bis zu 10 Minuten dauern.

```
python
```

In den Interpreter (wir befinden uns nun bereits in einem interaktiven Modus) laden wir anschließend das PWM-Modul (eine Bibliothek mit Python-Funktionen):

```
import pwm
```

Jetzt kann man die Funktionen des Moduls aufrufen. Eine dieser Funktionen nimmt direkt den PWM-Zählervergleichswert entgegen:

```
pwm.pwm_raw(1000)
```

Das Signal im Oszilloskop müsste nun wie in **Bild 5** aussehen.

Eine Eingabe des Tastverhältnisses in Prozent, über

```
pwm.pwm(50)
```

führt zu einem Signal wie in **Bild 6**.

Mit den Befehlen

```
pwm.pwm(1)
pwm.pwm(99)
```

lassen sich entsprechend Puls-/Pausenverhältnisse von 1 und 99 % generieren (**Bild 7** / **Bild 8**).

Mit Strg-d kann man den Python-Interpreter wieder beenden. Leider dauert es immer ein wenig, bis Python gestartet ist. Wenn der Interpreter aber einmal läuft, reagiert er recht flüssig.

Netzwerkschnittstelle

Im letzten Teil haben wir bereits gezeigt, wie man das System dazu bringt, mit einem USB/UART-Adapter zusammenzuarbeiten. Nun wollen wir einen weiteren USB-Adapter installieren, um das Linux-Board mit einem Ethernet-Netzwerk verbinden zu können. Wir verwenden einen handelsüblichen USB/LAN-Adapter (**Bild 9**). Hier gibt es viele verschiedene Modelle, aber die meisten Geräte greifen auf ähnliche, klassische Chips zurück. Wer ganz auf Nummer sicher gehen will: Für diese Ausgabe haben wir einen „D-Link DUB-E100“ verwendet [4].

In der letzten Folge hatten wir für den USB/UART-Adapter einen eigenen Treiber in den Kernel integriert. Wie schon im Laufe der Serie erwähnt, bietet der Kernel aber auch die Möglichkeit, Gerätetreiber als Modul zur Laufzeit nachzuladen. Beim Netzwerkadapter wollen wir genau das tun. Im Dateisystem liegen bereits verschiedene Treiber bereit.

Im Falle des D-Link-Adapters müssen wir folgenden Befehl eingeben:

```
modprobe asix
```

Wir müssten nun eine Ausgabe wie in **Bild 10** erhalten.
Im Dateisystem befinden sich insgesamt drei Treiber:

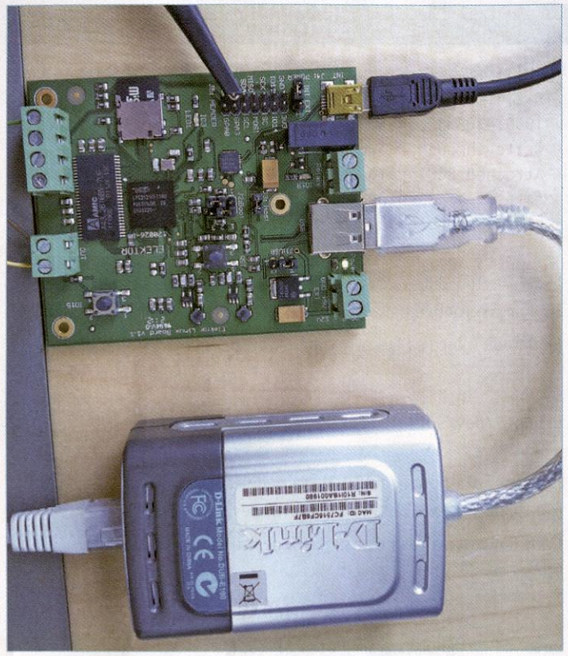

Bild 9. Ein USB/LAN-Adapter erweitert unser Board um eine Netzwerk-Schnittstelle.

- asix
- pegasus
- net1080

Nutzt man einen anderen Adapter, kann man einfach die Treiber testweise nacheinander laden. Ob man den richtigen gefunden hat, sieht man nach der Eingabe von

```
ifconfig -a
```

Erscheint das „eth0"-Interface in der Ausgabe, dann ist das Netzwerk bereit zum Übertragen von Daten.

Falls keiner der Treiber passt, kann man im Kernel unter „Device Drivers" -> „Network device support" -> „USB Network Adapters" weitere Treiber manuell hinzufügen. Entweder man nimmt den Treiber fest in den Kernel auf (Empfehlung für Einsteiger, siehe [2]). Oder man muss die Module nach dem Übersetzen in das Dateisystem kopieren und später vom Kernel nachladen lassen.
Alle Module übersetzen kann man mit:

```
make modules
```

Auf die Karte installiert man die neuen Module mit

```
make modules_install INSTALL_MOD_PATH=/mnt
```

Statt `/mnt` muss man hier den Pfad zur eingehängten SD-Karte angeben.

Wird der Netzwerkadapter jetzt erkannt, dann kann man temporär eine IP-Adresse vergeben. Sicherheitshalber sollte man zuvor mit einem PC im gleichen Netzwerk prüfen, ob die IP-Adresse frei ist. Das geht unter Linux (und Windows) von der Konsole aus mit dem Befehl `ping`:

```
ping 192.168.0.7
```

Wenn das Programm meldet, dass kein Gerät antwortet ...

```
2 packets transmitted, 0 received, 100% packet loss, time 1006ms
```

... dann kann man diese IP-Adresse vergeben. Hierzu geben wir auf dem Linux-Board ein:

```
ifconfig eth0 192.168.0.7
```

Ein erneutes Ping vom PC aus sollte jetzt zu einer positiven Meldung führen (**Bild 11**).
Optional kann man sich auch von einem DHCP-Server automatisch eine Netzwerkadresse zuweisen lassen (**Bild 12**).

Soll der Treiber bei jedem Start automatisch geladen werden, dann kann man dessen Namen in die Datei „/etc/modules" eintragen. Hier stehen alle Module untereinander, die automatisch beim Boot von Linux geladen werden sollen.
Die IP-Adresse hinterlegt man in der Datei „/etc/network/interfaces". Diese Datei existiert bereits in unserem Dateisystem. Mit einem Editor öffnet man diese und trägt bei „eth0" die eigene IP-Adresse ein.
Nach dem Start ist das Elektor-Linux-Board nun immer im hauseigenen Netzwerk verfügbar.

```
root@gnublin:/lib/modules/2.6.33/kernel/drivers/net/usb# usb 1-1: new high speed USB device using lpc-ehci and address 2
usb 1-1: New USB device found, idVendor=2001, idProduct=3c05
usb 1-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
usb 1-1: Product: DUB-E100
usb 1-1: Manufacturer: D-Link Corporation
usb 1-1: SerialNumber: 000001

root@gnublin:/lib/modules/2.6.33/kernel/drivers/net/usb# modprobe asix
eth0: register 'asix' at usb-lpc-ehci.0-1, ASIX AX88772 USB 2.0 Ethernet, fc:75:16:cf:6b:7f
usbcore: registered new interface driver asix
root@gnublin:/lib/modules/2.6.33/kernel/drivers/net/usb#
```

Bild 10. Automatische Erkennung des USB/LAN-Adapters.

Webserver

Nachdem jetzt steht die Netzwerkverbindung steht, können wir einen kleinen Webserver starten, um eine erste Demoseite per Browser anzeigen zu können. Im Homeverzeichnis des Benutzers „root" liegt ein kleines Skript, das den bekannten Webserver „lighttpd" anwirft:

```
root@gnublin:~# ./lighttpd-init.sh

Syntax OK

root@gnublin:~#
```

Besucht man mit einem Browser die vorher vergebene IP-Adresse, dann erscheint die Webseite in **Bild 13**.

Ein Webserver stellt typischerweise statische HTML-Seiten zur Verfügung. Wenn wir uns nun zum Beispiel den Status einer LED im Browser anzeigen lassen wollen, dann muss unser Webserver eine HTML-Seite ausliefern, die dynamisch (abhängig vom LED-Status) zusammengebaut wird. Wir benötigen hierfür eine Schnittstelle zwischen dem Webserver und einem externen Programm, das erkennen kann, ob die LED nun gerade an oder aus ist, und eine entsprechende Webseite generiert.

Die einfachste Verbindung heißt hier CGI. Das „Common Gateway Interface" ist eine Schnittstelle, mit welcher der Webserver nahezu beliebige Programme aufrufen kann. Eine Bedingung ist nur, dass das Programm kommandozeilenorientiert arbeitet, also von der Konsole aus (mit eventuellen Parametern) gestartet werden könnte. Außerdem muss das Hilfsprogramm direkt eine HTML-Seite ausgeben. Als CGI-Programm kann man ein einfaches Linux-Shell-Skript, ein PHP- oder Python-Programm und sogar ein C-Programm verwenden.

LED schalten vom Browser aus

Für eine kleine Demo-Applikation wollen wir gleich die erste Möglichkeit nutzen. Wir erstellen uns eine einfache Skript-Datei, welche die Shell (Konsole) direkt ausführen kann. Zuerst aber müssen wir uns das CGI-Interface im Webserver einrichten.
In der Datei „/etc/lighttpd/modules.conf" muss man den Eintrag ...

```
#include "conf.d/cgi.conf"
```

... mit einem Editor (z.B. nano oder vi) ändern in:

```
include „conf.d/cgi.conf"
```

```
sauter@pc:~$ ping 192.168.0.7
PING 192.168.0.7 (192.168.0.7) 56(84) bytes of data.
64 bytes from 192.168.0.7: icmp_req=1 ttl=64 time=1.24 ms
64 bytes from 192.168.0.7: icmp_req=2 ttl=64 time=1.50 ms
--- 192.168.0.7 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 1.248/1.375/1.502/0.127 ms
```

Bild 11. Meldung nach einem Ping.

```
root@gnublin:~# udhcpc
udhcpc (v1.17.3) started
Setting IP address 0.0.0.0 on eth0
Sending discover...
Sending select for 192.168.0.182...
Lease of 192.168.0.182 obtained, lease time 864000
Setting IP address 192.168.0.182 on eth0
Deleting routers
Adding router 192.168.0.1
Recreating /etc/resolv.conf
 Adding DNS server 192.168.0.1
root@gnublin:~#
```

Bild 12. Hier lassen wir uns vom DHCP-Server eine Adresse zuweisen.

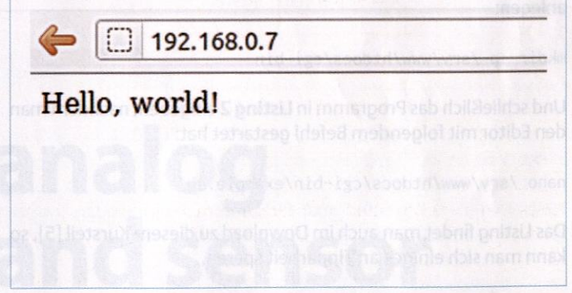

Bild 13. Testseite des Webservers.

Anschließend muss in der Datei „/etc/lighttpd/conf.d/cgi.conf" der Eintrag

```
#alias.url += ( "/cgi-bin" => server_root + "/cgi-bin" )
```

in

```
alias.url += ( "/cgi-bin" => var.server_root + "/cgi-bin" )
```

geändert werden. Jetzt weiß der Webserver, dass er die Dateien im Ordner „/cgi-bin" als Programm ausführen soll (und nicht als HTML-Seite an den Browser senden).

Damit wir schließlich ein einfaches Shell-Skript als CGI-Programm verwenden können muss in der Datei der Bereich ...

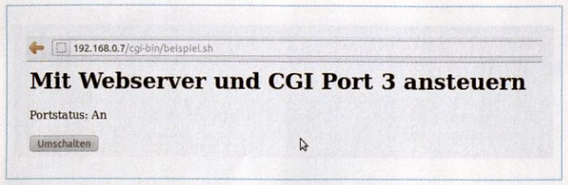

Bild 14. LED schalten im Browser.

```
cgi.assign              = ( ".pl"  => "/usr/bin/perl",
                            ".cgi" => "/usr/bin/perl",
                            ".rb"  => "/usr/bin/ruby",
                            ".erb" => "/usr/bin/eruby",
                            ".py"  => "/usr/bin/python" )
```

... mit der Zeile `".sh"  => "/bin/sh"` erweitert werden:

```
cgi.assign              = ( ".pl"  => "/usr/bin/perl",
                            ".cgi" => "/usr/bin/perl",
                            ".rb"  => "/usr/bin/ruby",
                            ".sh"  => "/bin/sh",
                            ".erb" => "/usr/bin/eruby",
                            ".py"  => "/usr/bin/python" )
```

Im nächsten Schritt muss man den Ordner für die CGI-Programme anlegen:

```
mkdir -p /srv/www/htdocs/cgi-bin
```

Und schließlich das Programm in **Listing 2** eingeben, nachdem man den Editor mit folgendem Befehl gestartet hat:

```
nano /srv/www/htdocs/cgi-bin/example.sh
```

Das Listing findet man auch im Download zu diesem Kursteil [5], so kann man sich einiges an Tipparbeit sparen.

Damit der Webserver die LED ansteuern kann, muss man zuvor noch den Pin konfigurieren und die Datenrichtung auf der Konsole einstellen:

```
echo 3 > /sys/class/gpio/export
echo out > /sys/class/gpio/gpio3/direction
```

Weil der Webserver aus Sicherheitsgründen nie als User „root" läuft, müssen wir noch temporär Rechte vergeben, mit denen der Webserver auf die LED zugreifen darf:

```
chown  lighttpd:lighttpd /sys/class/gpio/gpio3/value
```

Jetzt darf jeder im System auf die LED zugreifen. Das ist eigentlich nicht optimal, doch ein detailliertes Vergeben der Rechte würde an dieser Stelle zu weit führen. Infos findet man hierzu unter [6]. Außerdem braucht der Webserver noch die passenden Rechte, um Logdateien in einem vorher generierten Verzeichnis ablegen zu dürfen:

```
mkdir /var/log/lighttpd
chown -R lighttpd:lighttpd /var/log/lighttpd
```

Startet man den Webserver jetzt ...

```
root@gnublin:~# /etc/init.d/lighttpd restart
```

... dann müsste die Meldung erscheinen:

```
Syntax OK
```

Wenn man per Browser auf die zuvor eingestellte IP-Adresse zugreift, dann öffnet sich die Seite in **Bild 14**, die den Status des LED-Pins anzeigt.
Auch eine kleine Steuerung haben wir eingebaut, mit Hilfe eines Mini-HTML-Formulars, das in diesem Falle nur einen Submit-Button enthält. Drückt man auf einen solchen Button, dann übermittelt der Browser normalerweise den Inhalt der Formular-Steuerelemente an unseren Webbrowser. In diesem Fall benutzen wir den Mechanismus nur, um dem Webserver mitzuteilen, dass er erneut unser CGI-Skript „/cgi-bin/example.sh" aufrufen soll. Dieses toggelt die LED1 auf dem Board und baut die neue Webseite mit der veränderten Statusmeldung zusammen.

Listing 2: CGI-Skript zur Generierung der Webseite.

```
#!/bin/sh

if [ "$REQUEST_METHOD" == "POST" ]
then
   if [ 'cat /sys/class/gpio/gpio3/value' == 1 ]
   then
      echo 0 > /sys/class/gpio/gpio3/value
   else
      echo 1 > /sys/class/gpio/gpio3/value
   fi
fi

echo "Content-Type: text/html; charset=utf-8"
echo ""
echo "<html>"
echo " <head>"
echo "   <title>Webserver CGI Port 3 (LED)</title>"
echo " </head>"
echo " <body>"
echo "  <h1>Control-Panel CGI Port 3</h1>"
if [ 'cat /sys/class/gpio/gpio3/value' == 1 ]
then
   echo "  Port: On"
else
   echo "  Port: Off"
fi
echo "  <br><br>"
echo "  <form action\"/cgi-bin/example.sh \" method =
\"POST\">"
echo "  <input type=\"submit\" value=\"Click\">"
echo " </form>"
echo " </body>"
echo "</html>"
```


Ausblick

In der nächsten Ausgabe werden wir eine etwas komplexere HTML-Oberfläche erstellen, mit der wir weitere Funktionen auf dem Board steuern können. Doch natürlich ist es mit einem reinen User-Interface nicht getan, unsere Steuerung benötigt auch etwas Intelligenz. Dies können wir mit einem kleinen Programm, das im Hintergrund läuft, verwirklichen.

Für den übernächsten und letzten Teil dieser Serie (in der ersten Ausgabe des Jahres 2013) haben wir uns noch etwas Besonderes ausgedacht: Unsere Leser können selbst auswählen, welche Themen noch behandelt werden sollen. Auf einer speziellen Webseite [7] können Sie hierüber abstimmen!

(120182)

Weblinks

[1] sauter@embedded-projects.net
[2] www.elektor.de/120181
[3] http://de.wikipedia.org/wiki/Pulsweitenmodulation
[4] http://shop.embedded-projects.net/gnublin
[5] www.elektor.de/120182
[6] http://en.gnublin.org/index.php/Permissions_GPIO
[7] www.elektor.de/linux-feedback
[8] www.elektor.de/120180