

Grasshopper: GPIOs über ein Webinterface steuern

Christian Zietz

Die folgende Anleitung soll zeigen, wie man die GPIOs (General Purpose Inputs/Outputs, also Port-Pins) des Grasshopper- bzw. ICnova-AP7000-Base-Boards über ein Webinterface setzen und auslesen kann.

Vorausgesetzt werden Kenntnisse über die grundlegenden Linux-/Unix-Kommandos sowie optimalerweise minimale Kenntnisse des einzigen auf dem Board installierten Editors *vi*.

Die in dieser Anleitung erwähnten Dateien finden sich nach Kapiteln sortiert in *grasshopper-gpio.tgz*.

1 Inbetriebnahme

Für diese Anleitungen verbinden wir das Board sowohl über USB mit dem PC als auch über Ethernet mit dem Netzwerk. Über den USB-Anschluss wird das Board mit Strom versorgt. Außerdem bietet eine virtuelle serielle Schnittstelle darüber sowohl die Boot-Meldungen als auch eine Konsole. Details zur Nutzung der seriellen Schnittstelle nennt die Projektseite des Grasshoppers. Die Ethernetschnittstelle wird standardmäßig nicht beim Booten aktiviert. **Achtung:** So lange das Board über USB am PC angeschlossen ist, ist eine externe Stromversorgung über die vorhandene Buchse weder notwendig noch sinnvoll. Soll das Board jedoch später einmal an einem Netzteil betrieben werden, weist In-Circuit darauf hin, dass der Aufdruck 7-20V auf der Platine falsch ist. Es dürfen keinesfalls mehr als 10V angelegt werden.

Im Folgenden wird davon ausgegangen, dass das Board nun über die Konsole auf der virtuellen seriellen Schnittstelle angesprochen werden kann. Wir loggen uns mit dem Benutzernamen *default* (ohne Passwort) ein und geben dann das Kommando *su* ein, um als *root* zu arbeiten. Es gibt noch ein paar weitere Kleinigkeiten, die wir zunächst korrigieren wollen:

- Damit die Ethernetschnittstelle beim Booten aktiviert wird und sich das Board von einem DHCP-Server automatisch eine IP-Adresse besorgt, muss die Datei */etc/network/interfaces* editiert werden (mit *vi*). Wir entfernen die Raute (#) vor der Zeile

```
auto eth0
```

Einziger Nachteil dieser Änderung: Ist kein DHCP-Server auffindbar, verzögert sich der Boot-Vorgang leicht. Andernfalls müssten wir jedoch nach jedem Login auf der seriellen Konsole ein *ifup eth0* eingeben, um die Ethernetschnittstelle zu aktivieren.

- Um die Meldung *can't access tty: job control turned off* beim Login loszuwerden, ändern wir in */etc/inittab* die Zeile

```
null::respawn:/sbin/getty -L console 115200 vt100
```

```
in
```

```
null::respawn:/sbin/getty -L /dev/ttyS0 115200 vt100
```

Ein weiterer Vorteil: Nun ist an der seriellen Konsole auch der direkte Login als *root* möglich. **Achtung:** Sollte die *inittab* beschädigt werden, ist evtl. kein regulärer Login mehr möglich. Die Lösung in diesem Fall: Beim Booten bei der Meldung *Press SPACE to abort autoboot in 3 seconds* die Leertaste drücken und dann im Bootloader U-Boot die folgenden Kommandos eingeben:

```
setenv bootargs $bootargs init=/bin/sh
boot
```

Damit landet man nach dem Booten einmalig direkt in einer Shell und kann die *inittab* reparieren.

- Bei manchen Grasshoppern ist offenbar die Datei */var/www/index.html* beschädigt. Dies führt beim Booten zu der Meldung *Unknown node type: e002 len 766 offset 0x508da8*. Auch liefert der Webserver dann keine Startseite aus. Eine reparierte *index.html* ist in den Dateien zu dieser Anleitung enthalten.

2 Dateien mit dem Board austauschen

Es gibt verschiedene Wege, Dateien mit dem Board auszutauschen. Standardmäßig läuft ein Webserver, der die Dateien unterhalb von */var/www* bereitstellt. Auf dem Board ist *wget* installiert, mit dem Dateien von einem Webserver auf das Board geladen werden können. Wir wollen jedoch den ebenfalls vorhandenen TFTP-Server verwenden. (Hinweis: Außer dem Namen hat TFTP (Trivial File Transfer Protocol) nichts mit dem bekannteren FTP gemeinsam.) Standardmäßig läuft der Server nicht. Wir starten ihn mit folgendem Kommando und begrenzen den Zugriff sicherheitshalber auf Dateien in */tmp*:

```
in.tftpd -l -c -s /tmp
```

Windows und die gängigen Linux-Distributionen bringen einen TFTP-Client mit, so dass wir nun Dateien austauschen können. Um beispielsweise die Datei *test.bin* vom PC zum Board zu senden ist unter Windows folgender Aufruf auf der Kommandozeile nötig (IP-Adresse dabei durch die tatsächliche Adresse des Boards ersetzen):

```
tftp -i IP-Adresse PUT test.bin
```

Sollte im LAN bereits ein NFS-Server (Network File System) existieren, kann auch er zum Datenaustausch mit dem Grasshopper dienen. Dazu muss auf dem Board nur *portmap* gestartet werden. Danach ist ein Mounten von NFS-Freigaben möglich.

3 Einrichtung der GPIOs

Der Zugriff auf die Port-Pins erfolgt nach Unix-Art über eine Device-Datei. Diese legen wir mit dem Kommando

```
mknod /dev/gpio0 c 254 0
```

an. Außerdem muss nach jedem Neustart konfiguriert werden, welcher Port und welche Pins über diese Datei als Ein- und Ausgänge angesprochen werden sollen. In Anleitung werden wir die Pins PORTA00 (kurz PA00), PA01 und PA02 als Ausgänge und die Pins PA03, PA04 und PA05 als Eingänge konfigurieren. Diese Konfiguration erfolgt über Dateien in einem noch anzulegenden Unterverzeichnis von */config/gpio*:

```
mkdir /config/gpio/gpio0
```

Achtung: Die Zuordnung zwischen der Device-Datei und der eben angelegten Konfiguration erfolgt **nicht** über den Namen sondern über die Reihenfolge der Erstellung der Unterordner. Unsere Device-Datei */dev/gpio0* spricht immer die Konfiguration an, die als erste unter */config/gpio* angelegt wird, unabhängig von deren Namen.

Der Ordner */config/gpio/gpio0* enthält vier Dateien, in die wir unsere Konfiguration schreiben müssen:

- *gpio.id* gibt den Port an. Hierbei steht 0 für Port A, 1 für Port B usw.

- *pin_mask* gibt an, welche Pins des Ports überhaupt angesprochen werden sollen. Die entsprechenden Bits sind zu setzen, der resultierende Wert muss als Hexadezimalzahl in *pin_mask* geschrieben werden. In unserem Fall wollen wir mit *0x3f* die untersten 6 Pins ansprechen.
- *oe_mask* gibt an, welche Pins Ausgänge sein sollen. In dieser Anleitung steht *0x7* für die untersten 3 Pins.
- Schließlich muss in die Datei *enabled* eine 1 geschrieben werden, um die Konfiguration zu aktivieren.

Insgesamt ergeben sich damit folgende Kommandos:

```
mkdir /config/gpio/gpio0
cd /config/gpio/gpio0
echo 0 > gpio.id
echo 0x3f > pin_mask
echo 0x7 > oe_mask
echo 1 > enabled
```

Da diese Kommandos nach jedem Neustart ausgeführt werden müssen, enthalten die Dateien zu dieser Anleitung das Shell-Skript *gpio0*. Dieses kopieren wir nach */etc/init.d*, machen es mit *chmod a+x gpio0* ausführbar und legen dann eine Verknüpfung darauf in */etc/rc.d* an, damit das Shell-Skript automatisch beim Booten aufgerufen wird:

```
cd /etc/rc.d
ln -s ../init.d/gpio0 S20gpio0
```

Wenn das Blinken der LED1 stört, so ist dies der richtige Zeitpunkt es abzustellen. Hierfür geben wir ein bzw. erweitern */etc/init.d/gpio0* um:

```
echo none > /sys/class/leds/led1\:green/trigger
```

4 GPIOs setzen und lesen

Die gewählten Port-Pins können nun über die angelegte Device-Datei (*/dev/gpio0*) angesprochen werden. Beim Lesen dieser Datei werden nach dem Öffnen und danach nach jeder Änderung der mittels *pin_mask* gewählten Port-Pins vier Bytes (= 32 Bits) übermittelt, die den Status des Ports enthalten. Die über *oe_mask* als Ausgänge definierten Port-Pins können durch das Schreiben der Device-Datei gesetzt oder gelöscht werden. Auch hier müssen vier Bytes an binären Daten übergeben werden.

Um den Umgang mit GPIOs zu vereinfachen, liegt dieser Anleitung ein kleines C-Programm als Quellcode (*gpio.c*) sowie passend für das Board kompiliert (*gpio*) bei. Wir kopieren die kompilierte Variante auf dem Board in das Verzeichnis */usr/bin*. Hier ein paar Beispiele für mögliche Aufrufe des Programms:

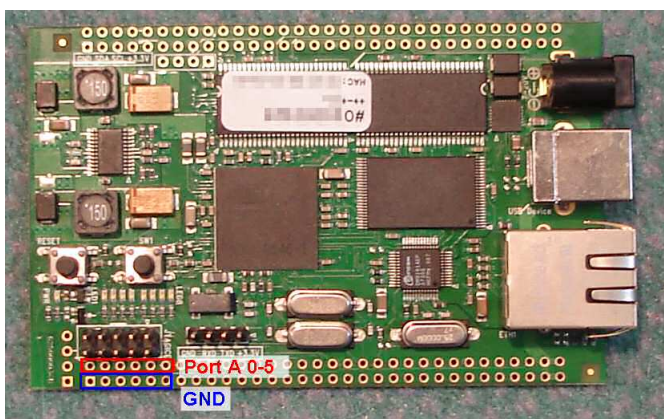
- `gpio /dev/gpio0 on 0`
setzt das niederwertigste (nullte) Bit des konfigurierten Ports (in unserem Fall Port A) auf logisch high. Dies funktioniert natürlich nur bei als Ausgang konfigurierten Pins.
- `gpio /dev/gpio0 off 1`
setzt den Pin Nummer 1 des konfigurierten Ports auf logisch low, sofern der entsprechende Pin ein Ausgang ist.
- `gpio /dev/gpio0 query 2`
fragt den Zustand des Pins 2 ab. Dies funktioniert sowohl für Ausgänge als auch für Eingänge. Eingänge haben einen (schwachen) internen Pull-Up. Der Rückgabewert von *gpio* ist in diesem Fall 0, wenn der Pin high ist und 1, falls er low ist. (Hinweis: Diese scheinbare Verdrehung der Logik des Rückgabewerts erlaubt dann wieder die logisch schlüssigere Verwendung in *if*-Abfragen in Shell-Skripts.)

5 CGI-Skript für den Webserver einrichten

Der auf dem Board vorhandene und automatisch gestartete Webserver (bereitgestellt von *busybox*) unterstützt CGIs, d.h. die Generierung interaktiver Webseiten. Es ist allerdings wenig sinnvoll, wie auf einem "großen" Webserver Skripte in *php*, *perl* oder *python* für CGI-Programme zu verwenden. Wie jedoch bereits das auf dem Board vorhandene Beispiel (*/var/www/cgi-bin/led.sh*) zeigt, kann jede beliebige ausführbare Datei, also auch ein Shell-Skript, als CGI-Programm dienen. Die Kommunikation zwischen Webserver und Skript erfolgt, wie beim CGI üblich, über Umgebungsvariablen sowie über die Standardausgabe.

Dieser Anleitung liegt mit *gpio.sh* ein passendes Skript bei, um entsprechend der vorhin vorgenommenen Konfiguration die Pins 0 bis 2 des Ports A zu setzen und den Logikpegeln an den Pins 3 bis 5 zu lesen. Nachdem wir das Skript nach */var/www/cgi-bin* kopiert und mittels *chmod a+x gpio.sh* ausführbar gemacht haben, können wir es auf dem PC im Browser als <http://IP-Adresse/cgi-bin/gpio.sh> aufrufen.

Das beigefügte Foto zeigt die Lage der entsprechenden GPIOs, die sich nun über das Netz schalten und auslesen lassen:



6 Rechtliches

Diese Anleitung und alle begleitenden Dateien mit Ausnahme von *chap1/index.html* wurden von Christian Zietz <czietz@gmx.net> erstellt und können unter den Bedingungen der *Creative Commons Attribution 2.0 Germany* Lizenz verbreitet werden. Es wird keine Gewähr für die Richtigkeit und Fehlerfreiheit übernommen.