Hochschule Augsburg | Fakultät für Informatik | Sommer 2021
Prof. Dr. Högl | Embedded Linux | 2021-05-05

Memo-#7: **GPIOs ansteuern mit libgpiod**

Inhalt

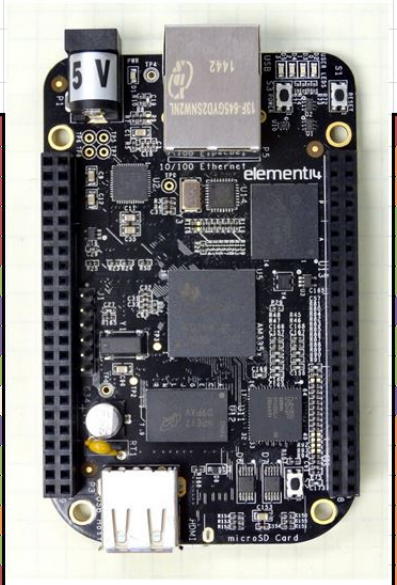
1 Ein/Ausgabe am BBB

Die folgende Abbildung zeigt die Ein-/Ausgabe Pins des Beagle Bone Black ([BBBPINS]).

Beaglebone Black Pinout Diagram

P9

Function	Physical Pins		Function
DGND	1	2	DGND
VDD 3.3 V	3	4	VDD 3.3 V
VDD 5V	5	6	VDD 5V
SYS 5V	7	8	SYS 5V
PWR_BUT	9	10	SYS_RESET
UART4_RXD	11	12	GPIO_60
UART4_TXD	13	14	EHRPWM1A
GPIO_48	15	16	EHRPWM1B
SPIO_CSO	17	18	SPIO_D1
I2C2_SCL	19	20	I2C_SDA
SPIO_DO	21	22	SPIO_SLCK
GPIO_49	23	24	UART1_TXD
GPIO_117	25	26	UART1_RXD
GPIO_115	27	28	SP11_CSO
SP11_DO	29	30	GPIO_112
SP11_SCLK	31	32	VDD_ADC
AIN4	33	34	GND_ADC
AIN6	35	36	AIN5
AIN2	37	38	AIN3
AIN0	39	40	AIN1
GPIO_20	41	42	ECAPWMO
DGND	43	44	DGND
DGND	45	46	DGND



LEGEND
Power, Ground, Reset
Digital Pins
PWM Output
1.8 Volt Analog Inputs
Shared I2C Bus
Reconfigurable Digital

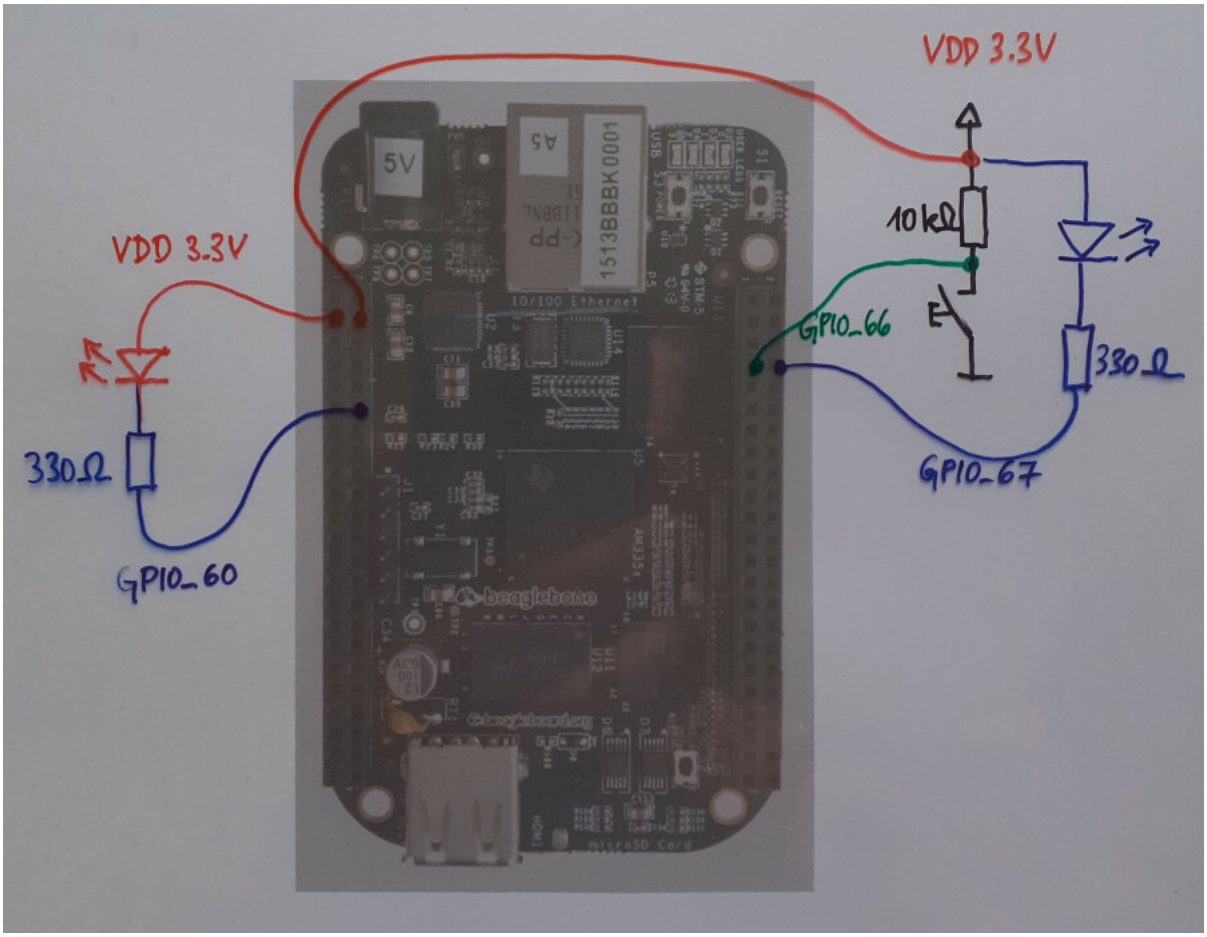
P8

Function	Physical Pins		Function
DGND	1	2	DGND
MMC1_DAT6	3	4	MMC1_DAT7
MMC1_DAT2	5	6	MMC1_DAT3
GPIO_66	7	8	GPIO_67
GPIO_69	9	10	GPIO_68
GPIO_45	11	12	GPIO_44
EHRPWM2B	13	14	GPIO_26
GPIO_47	15	16	GPIO_46
GPIO_27	17	18	GPIO_65
EHRPWM2A	19	20	MMC1_CMD
MMC1_CLK	21	22	MMC1_DAT5
MMC1_DAT4	23	24	MMC1_DAT1
MMC1_DATO	25	26	GPIO_61
LCD_VSYNC	27	28	LCD_PCLK
LCD_HSYNC	29	30	LCD_AC_BIAS
LCD_DATA14	31	32	LCD_DATA15
LCD_DATA13	33	34	LCD_DATA11
LCD_DATA12	35	36	LCD_DATA10
LCD_DATA8	37	38	LCD_DATA9
LCD_DATA6	39	40	LCD_DATA7
LCD_DATA4	41	42	LCD_DATA5
LCD_DATA2	43	44	LCD_DATA3
LCD_DATA0	45	46	LCD_DATA1

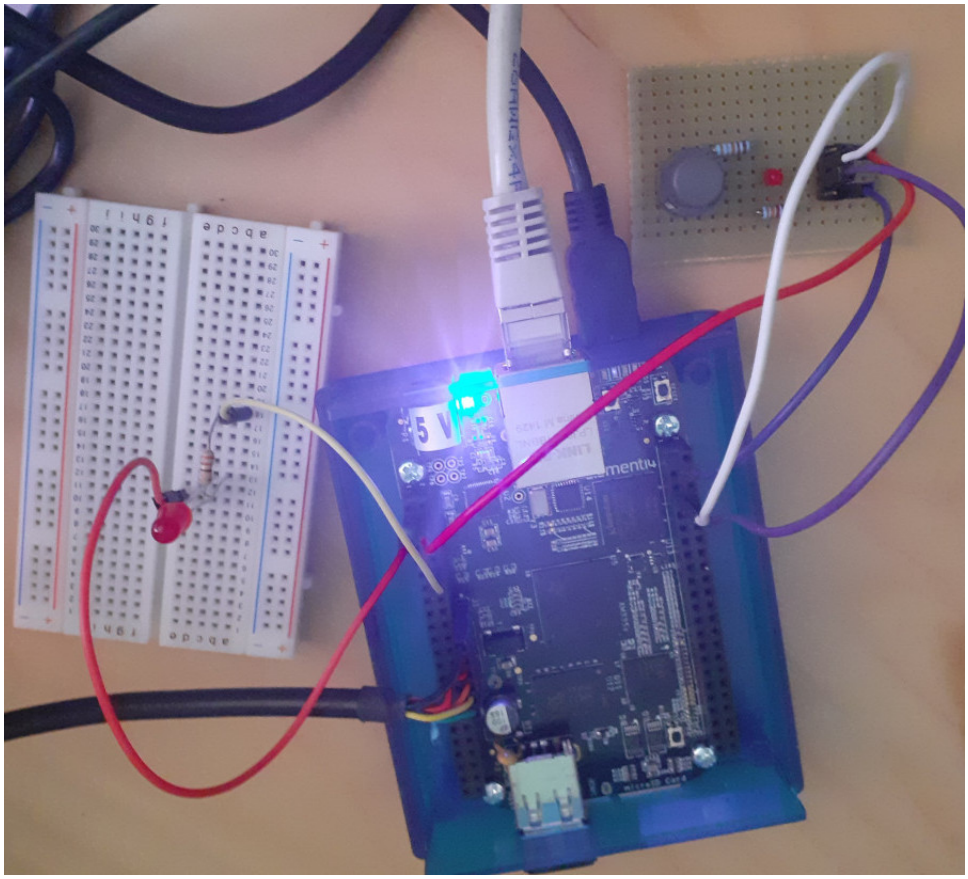
2 Schaltplan

Im Schaltplan sieht man links eine LED und einen Vorwiderstand. Das Dreieck der LED geht in Richtung von Anode (+) zu Kathode (-). Die LED muss unbedingt mit einem Vorwiderstand betrieben werden. Der Wert des Widerstandes ist nicht so wichtig, irgend ein Wert zwischen 200 und 400 Ohm passt. Der Widerstand ist an GPIO_60 angeschlossen. Siehe GPIO_60 auch dem vorherigen Bild.

Auf der rechten Seite sieht man eine zweite LED, die über einen Vorwiderstand an GPIO_67 angeschlossen ist. Ausserdem gibt es nun einen Taster, der über einen 10 kOhm Vorwiderstand an 3.3V angeschlossen ist. Am Taster wird eine Verbindung zu GPIO_66 gemacht.



Die Schaltung sieht in Wirklichkeit nun wie in der folgenden Abbildung aus: Der linke Teil ist auf einem Steckbrett gesteckt, beim rechten Teil sind die Bauteile auf einer Lochrasterplatine verlötet.



3 Software

Wir möchten nun mit einem Programm die LEDs ein- und ausschalten (Ausgang) sowie die Taste einlesen (Eingang). Üblicherweise werden Ein-/Ausgabeoperationen auf GPIO Pins mit Hilfe des `sysfs` gemacht, siehe z.B. [MOLLOY]. Tatsächlich ist das `sysfs` schon ein Weilchen als veraltet markiert, die aktuelle Lösung nennt sich `libgpiod` [QUADE]. Diesen Weg möchte ich gehen.

Zunächst muss man die Pakete `gpiod`, `libgpiod-dev` und `python3-libgpiod` auf dem Beaglebone installieren. Danach gibt es die Programme

- gpiodetect
- gpiofind

- gpioget
- gpioinfo
- gpiomon
- gpioset

Jedes Kommando kann mit der `-h` Option gestartet werden und gibt dann einen kurzen Hilfetext aus.

Mit

```
gpioset --mode=wait gpiochip1 28=0
```

kann man z.B. die LED an GPIO_60 einschalten. Die GPIO Pinnamen, die auch im Bild ganz oben zu finden sind, müssen zuerst übertragen werden in ein Paar gpiochip/bit. Erläuterungen dazu findet man wieder im Molloy oder an vielen Stellen im Internet. Das Python Skript `gpiofind2.py` in diesem Verzeichnis kann diese Umrechnung erledigen (wie funktioniert es?):

```
debian@beaglebone:~$ python3 gpiofind2.py GPIO_60
gpiochip1 28
```

Die Schreibweise `28=0` bedeutet, dass das Bit 28 auf log. "0" gesetzt wird, d.h. ca. 0 Volt Ausgangsspannung. Dadurch kann Strom durch die LED fliessen und diese zum Leuchten bringen. Wenn das Programm mit der Eingabetaste abgeschlossen wird, geht das Bit 28 wieder auf log. "1", d.h. ca. 3,4 Volt Spannung. Dadurch kann kein Strom mehr fliessen, die LED geht aus.

Selber ausprobieren:

```
gpioset --mode=time --sec=1 gpiochip1 28=0
```

Insgesamt gibt es bei `gpioset` vier Modi: `--mode=[exit|wait|time|signal]`. Bitte alle ausprobieren.

Den Tastenzustand kann man einlesen mit `gpioget`. Vom Pin GPIO_66 liest man mit:

```
gpioget gpiochip2 2
```

Mit `gpiomon` kann man Eingänge auf das Eintreffen von bestimmten Bedingungen überwachen:

```
gpiomon --num-events=5 --rising-edge gpiochip2 2
gpiomon --format="%e %o %s %n" --falling-edge gpiochip2 2
gpiomon --num-events=1 --silent gpiochip2 2
# Mehrere Eingaenge ueberwachen, exit beim ersten Event
gpiomon --silent --num-events=1 gpiochip0 <bit1> <bit2> ...
```

Wenn man die Anzahl Events pro Tastendruck untersucht, dann bekomme ich eine unterschiedliche Zahl von 1 bis etwa 10. Ich lasse mir dazu einfach die Events wie folgt ausgeben und drücke mehrmals auf die Taste:

```
gpiomon --num-events=100 --rising-edge gpiochip2 2
```

Woran könnte das liegen?

Man kann die libgpiod auch in Python verwenden. Demoprogramme findet man auf dem BBB. Die folgenden Dateien zunächst in ein Unterverzeichnis in Home kopieren, z.B. `python3-libgpiod`:

```
/usr/share/doc/python3-libgpiod/examples/gpiodetect.py
/usr/share/doc/python3-libgpiod/examples/gpiofind.py
/usr/share/doc/python3-libgpiod/examples/gpioget.py
/usr/share/doc/python3-libgpiod/examples/gpioinfo.py
/usr/share/doc/python3-libgpiod/examples/gpiomon.py
/usr/share/doc/python3-libgpiod/examples/gpioset.py
```

4 Ideen für Aufgaben

- Taste und LED an das BBB anschliessen, entweder auf einem Steckbrett (Anfänger) oder auf einer Lochrasterplatine löten (für Bastler und Technische Informatiker).
- Andere GPIO Pins als die Beschriebenen ausprobieren.

- Die vier Modi von `gpioset` ausprobieren.
- Wie schnell kann ich maximal einen Pin abwechselnd ein- und ausschalten?

  Da man das nicht mehr mit dem Auge wahrnehmen kann, benötigt man entweder ein Oszilloskop oder einen Logikanaysator. Ich kann den Logikanalysator von Saleae (https://www.saleae.com) empfehlen. Wer Freude an Embedded Linux hat, sollte sich sowas anschaffen, eines der ersten Modelle gibt es auch als Klon aus China zu kaufen für etwa 10 Euro. Modernere Modelle sind zwar toll, haben aber auch einen relativ hohen Preis. Die Bediensoftware ist gratis und für alle Betriebssysteme zu haben.
- Auch mal das `sysfs` ausprobieren, mit dem immer noch die meisten GPIO Zugriffe programmiert werden. Seit Kernel 4.8 ist das aber als *deprecated* (veraltet) markiert. Welches ist besser, alt oder neu?
- Einen *Reaktionszeitmesser* in Python programmieren, der eine LED nach einer zufälligen Wartezeit aktiviert und danach die Zeit misst, bis die Taste gedrückt wurde. Die Zeit soll auf die Millisekunde genau sein. Es soll dabei das Python Binding für `libgpiod` verwendet werden.

5 Literatur

[BBBPINS]

https://beagleboard.org/Support/bone101

[QUADE]

Eva-Katharina Kunst und Jürgen Quade, Kernel- und Treiberprogrammierung mit dem Linux-Kernel – Folge 101

http://hhoegl.informatik.hs-augsburg.de/elinux/kt/kt-101/kt101.pdf

"Viele Applikationen, die Sensoren und Aktoren über GPIOs ansteuern, benutzen ein längst abgekündigtes Interface. Das ist ein bisschen verrückt, weil das moderne Kernel-Subsystem Gpiolib und die passende Userland-Bibliothek Libgpiod viel schneller und sicherer arbeiten."

[MOLLOY]

Derek Molloy, Exploring the Beaglebone Black.
http://exploringbeaglebone.com