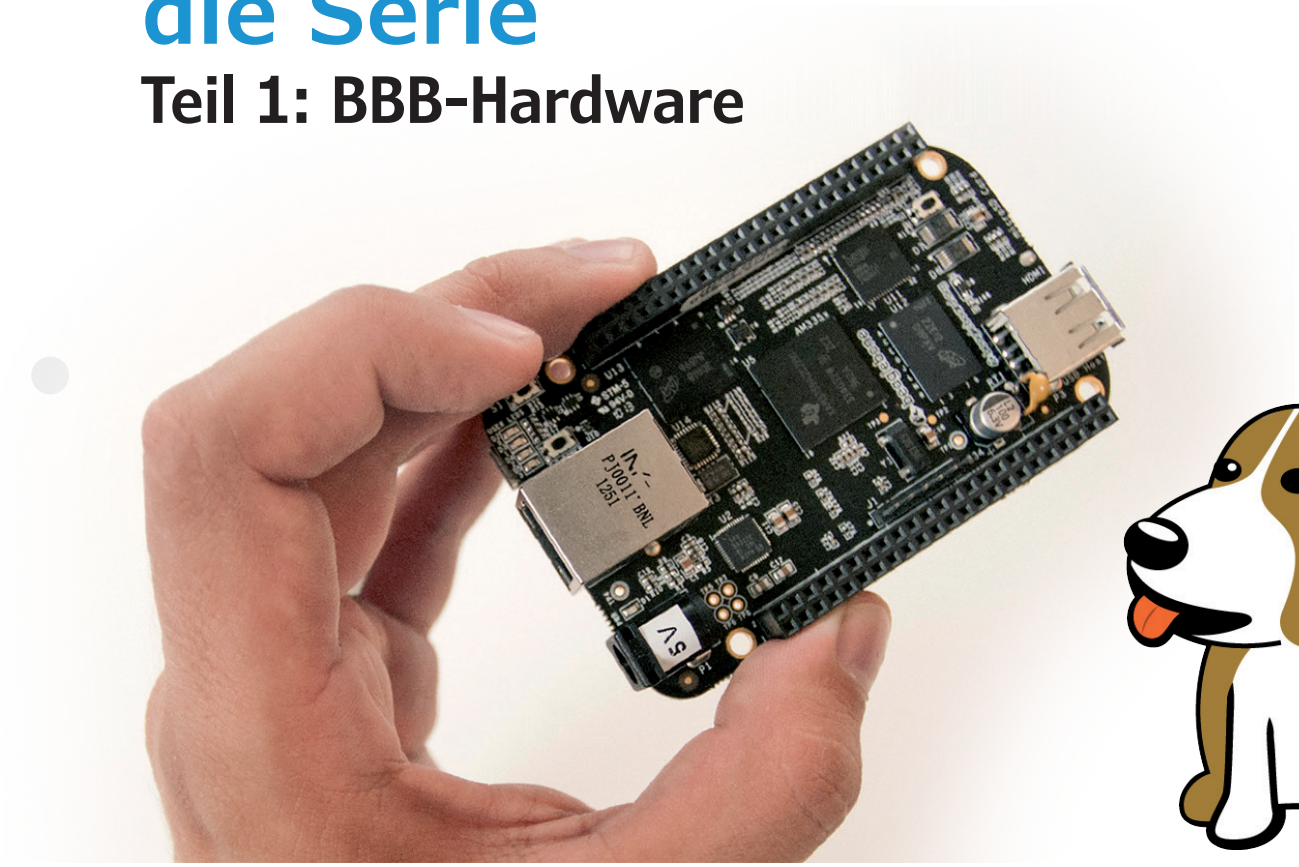
BeagleBone Black, die Serie

Teil 1: BBB-Hardware



Von **Tony Dixon** (UK)

Wenn Sie bei einem Raspberry Pi mehr I/O benötigen oder aber bei einem Arduino Due mehr Rechenleistung, dann wäre vielleicht ein BBB (**B**eagle**B**one **B**lack) das Richtige für Sie. In diesem ersten **.**Post-Projekt mit dem BBB geht es um dessen Erweiterungs-Steckplätze. Darauf folgt dann die obligatorische Demo in Form des Programms Blinky für eine blinkende LED.

Falls Sie die BBB-Einführung von Thijs Beckers in Elektor Dezember 2013 [1] verpasst haben, informiert Sie **Tabelle 1** über die Fähigkeiten des Boards. Die Linux-Distribution Ångström ist schon vorinstalliert. Weitere Infos zum BBB gibt es unter [2].

Hardware genug

Der BBB ist ziemlich gut für Hardware-Projekte präpariert, mit jeder Menge GPIOs sowie analogen, PWM- und seriellen Interfaces. Es gibt zwei Steckplätze für Erweiterungen: P8 (Expansion B) und P9 (Expansion A). Sie sind

Tabelle 1. Technische Daten des BeagleBone Black.

CPU	Sitara AM3359AZC100 ARM Cortex-A8 von TI
Takt	1 GHz
RAM	512 MB DDR3-SDRAM
Video	uHDMI
Speicher	2 GB eMMC (on-board), microSD-Karte
Ports	Ethernet 10/100 Mbit/s, USB-Host, USB-Client
I/O	65 x GPIO, 7 x analog, 8 x PWM, 4.5 x UARTs, 2 x I2C, 2 x SPI
Preis	45 $

als praxisfreundliche 46-polige Buchsen im Rastermaß 0,1" ausgeführt. **Tabelle 2** zeigt deren Pin-Belegung nach dem Einschalten. Per Software lassen sich auch andere Signale auf die Pins legen. Genaueres hierzu findet man im *BBB System Reference Manual*.

Die I/O-Signale haben 3,3-V-Logik. Man darf sie deshalb nicht direkt mit 5-V-Schaltungen verbinden, wenn man das Board nicht in den Beagle-Himmel schicken will.

Software-Ausstattung

Da es sich beim BBB um einen Linux-Computer handelt, gibt es eine große Auswahl an Programmiersprachen. Die Favoriten sind sicher C/C++ und Python, doch für den BBB gibt es mit BoneScript eine ganz eigene Sprache, die schon mit den Vorgängern BeagleBoard, BeagleBoard-XM und dem originalen BeagleBone eingeführt wurde.
Bei BoneScript handelt es sich um eine auf Node.js basierende Library, die viele arduino-artige Funktionen zur Interaktion mit der BBB-Hardware mit sich bringt. BoneScript basiert auf JavaScript, und wie bei Arduino gibt es eine IDE namens Cloud9, mit der man prima eigene Programme schreiben kann.

Blinky _._._._

Das erste BBB-Programm steht voll in der Tradition von LED-Blink-Demo-Software für Embedded-Systeme. Man könnte dazu das zuvor erwähnte BoneScript verwenden, doch der Einfachheit halber ist es in „normalem" C/C++ verfasst. Für das Programm Blinky muss eine LED über einen 680-Ω-Widerstand an den Header-Pin P8.03 (GPIO1_6) angeschlossen werden.

Beim BBB sind die GPIOs in 32er-Blöcken gruppiert und diese Blöcke werden startend mit 0 indiziert. Die GPIO-Nummer ergibt sich daher aus dem Index multipliziert mit 32 plus der Nummer im Block. GPIO1_6 hat demnach die Nummer 1 * 32 + 6 = 38.

Tabelle 2. Pinbelegung der Expansion Header P8 und P9

Signal	P8			Signal
GND	1		2	GND
GPIO1_6	3		4	GPIO1_7
GPIO1_2	5		6	GPIO1_3
TIMER4	7		8	TIMER7
TIMER5	9		10	TIMER6
GPIO1_13	11		12	GPIO1_12
EHRPWM2B	13		14	GPIO2_26
GPIO1_15	15		16	GPIO1_14
GPIO0_27	17		18	GPIO2_1
EHRPWM2A	19		20	GPIO1_31
GPIO1_30	21		22	GPIO1_5
GPIO1_4	23		24	GPIO1_1
GPIO1_0	25		26	GPIO1_29
GPIO2_22	27		28	GPIO2_24
GPIO2_23	29		30	GPIO2_25
UART5_CTS	31		32	UART5_RTS
UART4_RTS	33		34	UART3_RTS
UART4_CTS	35		36	UART3_CTS
UART5_TXD	37		38	UART5_RXD
GPIO2_12	39		40	GPIO2_13
GPIO2_10	41		42	GPIO2_11
GPIO2_08	43		44	GPIO2_09
GPIO2_6	45		46	GPIO2_07

Signal	P9			Signal
GND	1		2	GND
3,3V	3		4	3,3V
5V	5		6	5V
5V_SYS	7		8	5V_SYS
PWR_BUTTON	9		10	SYS_RESET
UART4_RXD	11		12	GPIO1_28
GPIO4_TXD	13		14	EHRPWM1A
GPIO1_16	15		16	EHRPWM1B
I2C1_SCL	17		18	I2C1_SDA
I2C2_SCL	19		20	I2C2_SDA
UART2_TXD	21		22	UART2_RXD
GPIO1_17	23		24	UART1_TXD
GPIO3_21	25		26	UART1_RXD
GPIO3_19	27		28	SPI1_CS0
SPI1_D0	29		30	SPI1_D1
SPI1_SCLK	31		32	AVCC
AIN4	33		34	AGND
AIN6	35		36	AIN5
AIN2	37		38	AIN3
AIN0	39		40	AIN1
GPIO_20	41		42	GPIO_7
GND	43		44	GND
GND	45		46	GND

Vereinfacht gesagt betrachtet Linux praktisch alles als Datei. Das gilt auch für Hardware-Ports wie UARTs oder USB. Folglich kann man GPIOs durch den Linux-Kernel ansprechen, wie wenn es sich um einen File Descriptor handeln würde. Die folgenden File Descriptors realisieren den Zugriff auf die GPIOs:

```
/sys/class/gpio/export
/sys/class/gpio/gpio38/direction
/sys/class/gpio/gpio38/value
/sys/class/gpio/unexport
```

Zuerst startet man eine Terminal-Session und dann den Editor **nano**. Im Terminal tippt man:

```
nano blinky.cpp
```

Nun schreibt man den Code von **Listing 1** am Artikelende ab oder kopiert ihn einfach. Anschließend sichert man das neue Programm durch die Shortcuts Ctrl+X und Y sowie ENTER zur Bestätigung des Sicherns.

Nach dem Sichern kompiliert man das Programm durch die Terminal-Eingaben:

```
g++ blinky.cpp -o blinky
```

Wenn es keine Compiler-Fehler gab, startet man das Programm durch Eingabe von:

```
./blinky
```

Jetzt sollte die angeschlossene LED im Sekundentakt blinken. Wie man sieht, wäre etwas im Stil von: „Warnung vor dem Beagle“ ganz sicher übertrieben.

(130472)

Weblinks

[1] BeagleBone Black, Elektor Dezember 2013, www.elektor.de/130279

[2] Beagle-Webseite: http://beagleboard.org

Listing 1. blinky.cpp

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

using namespace std;

int main() {
          FILE *export_file = NULL;
          FILE *IO_dir = NULL;
          char str_low[] = "low";
          char str_high[] = "high";
          char str_port[] = "38";

          // Open Port
          export_file = fopen ("/sys/class/gpio/export", "w");
          fwrite (str_port, 1, sizeof(str_port), export_file);
          fclose (export_file);

  while (1) {
          IO_dir = fopen ("/sys/class/gpio/gpio38/direction", "w");
          fwrite (str_high, 1, sizeof(str_high), IO_dir);   // pin = HIGH
          fclose (IO_dir);
          sleep (1);
```

```
            IO_dir = fopen ("/sys/class/gpio/gpio38/direction", "w");
            fwrite (str_low, 1, sizeof(str_low), IO_dir);   //pin = LOW
            fclose (IO_dir);
            sleep (1)
  }

}
```