

---

---

Hochschule für Angewandte Wissenschaften Hamburg  
Fakultät Technik und Informatik  
Department Informatik

# BeagleBone Black

Inbetriebnahme der

## General-Purpose Input/Output

Hausarbeit

Wahlpflichtmodul: Deeply Embedded  
Dozent: Prof. Dr. Stephan Pareigis

Schwensen, Lars Christian  
Matrikel-Nr.: 2124933  
Fachsemester 5

---

12. Januar 2015

---

---

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>3</b>
<b>2</b>	<b>Hardware</b>	<b>3</b>
2.1	Ressourcen . . . . .	3
2.2	Anschlüsse . . . . .	4
2.3	Pinbänke . . . . .	4
<b>3</b>	<b>TI-SDK</b>	<b>4</b>
3.1	Installation . . . . .	4
3.2	Linux Image . . . . .	5
3.2.1	Build erzeugen . . . . .	5
3.2.2	SD-Karte einrichten . . . . .	6
3.2.3	Systemstart . . . . .	7
3.3	Serielle Kommunikation . . . . .	8
3.4	Toolchain . . . . .	8
<b>4</b>	<b>GPIO</b>	<b>9</b>
4.1	Verfügbarkeit eines Pins ermitteln . . . . .	10
4.2	Pin verwenden . . . . .	12
4.3	Pulsweitenmodulation . . . . .	14
<b>5</b>	<b>Implementation</b>	<b>15</b>
<b>6</b>	<b>Zusammenfassung</b>	<b>17</b>

# Abbildungsverzeichnis

1	BeagleBone Black . . . . .	3
2	Boot-Button . . . . .	7
3	Hello World Programm in C++ . . . . .	9
4	Pinbänke des BeagleBone Black . . . . .	9
5	Ausschnitt aus dem System Reference Manual Pin P9.17 . . .	10
6	Auflistung Pin Modis für Pin P9.17 . . . . .	11
7	Aktiver Modus für Pin P9.17 . . . . .	11
8	Aktiver Modus für Pin P9.13 . . . . .	12
9	Auflistung aktiver PWM-Pins . . . . .	14
10	writeTo Funktion in C++ . . . . .	15
11	GPIO konfigurations-Funktionen in C++ . . . . .	16
12	GPIO value-Funktionen in C++ . . . . .	17

# 1 Einleitung

Raspberry Pi, pcDuino, BeagleBone Black... Der Markt bietet eine Vielzahl an erschwinglichen Einplatinen-Computern. Ist so ein System gekauft um ein konkretes Projekt zu verwirklichen, oder einfach nur zu experimentieren, stellt sich die Frage der Vorgehensweise um das System nach eigenen Wünschen zu konfigurieren und arbeiten zu lassen. Die folgenden Kapitel vermitteln eine Grundbasis an Informationen um den BeagleBone Black mithilfe des TI-SDK in Betrieb zu nehmen.

## 2 Hardware

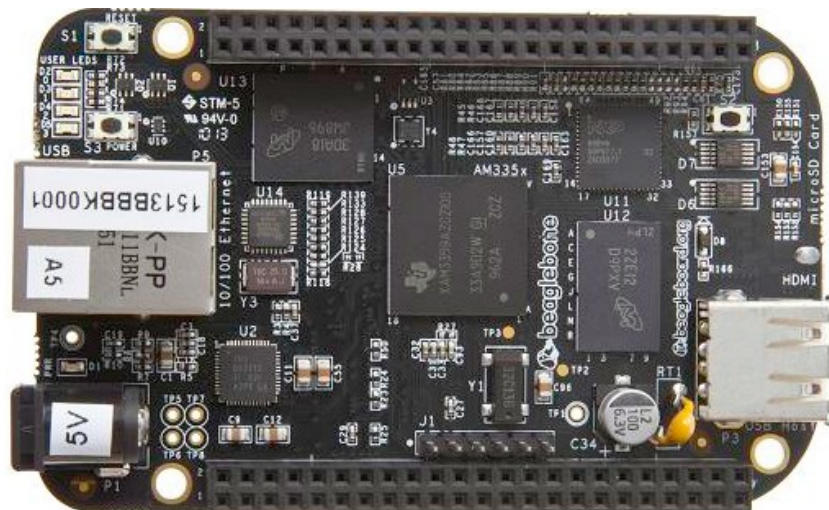


Abbildung 1: BeagleBone Black

([http://www.ti.com/ww/en/beagleboard/product\\_detail\\_black\\_lg.jpg](http://www.ti.com/ww/en/beagleboard/product_detail_black_lg.jpg))

Der BeagleBone Black (Abbildung 1) ist ein von Texas Instruments entwickelter Einplatinen-Computer. Laut dem System Reference Manual ist dieser seit dem 04.01.2013 verfügbar und aktuell in der Version (Rev C.1) erhältlich[1].

### 2.1 Ressourcen

Die aktuelle Version beinhaltet einen Sitara™AM3358 ARM Cortex-A8 Prozessor. Dieser 32Bit RISC Prozessor arbeitet mit einer Taktrate von bis zu 1GHz[6]. Außerdem sind 512MB RAM DDR3 Arbeitsspeicher und ein 4GB embedded Flash-Baustein verbaut[3].

## 2.2 Anschlüsse

Die Spannungsversorgung geschieht über eine 5V-DC-Buchse. Sofern gewährleistet ist, dass der Host-Computer die benötigte Leistung zur Verfügung stellen kann, kann die Spannungsversorgung alternativ über die MiniUSB Anschlussbuchse geschehen[2]. Die Verbindung mit einem Netzwerk geschieht über eine 10/100 Ethernet Buchse. Außerdem bietet ein USB2.0 Anschluss die Anschlussmöglichkeit von USB-Peripherie und eine microHDMI Buchse die Anschlussmöglichkeit eines Monitors. Der Interne Speicher lässt sich über einen MicroSD-Kartenslot erweitern[3].

## 2.3 Pinbänke

Der BeagleBone Black besitzt zwei Pinbänke mit jeweils 46 Anschlüssen. Diese dienen der GPIO-Kommunikation, wodurch sich das System sowohl durch gekaufte Capes, aber auch durch eigene Hardware und Schaltkreise erweitern lässt. Jeder Pin besitzt bis zu acht individuelle Modis. Diese und die Belegung der Pinbänke sind vollständig im System Reference Manual dokumentiert[5].

# 3 TI-SDK

Das TI-SDK ist ein von Texas Instruments bereitgestelltes Linux development Kit welches die benötigten Komponenten zum Entwickeln von Software für den BeagleBone Black beinhaltet in der aktuellen Version 07.00.00.00 kostenfrei zum Download bereiteht[7]. Dem Installations-Guide von Texas Instruments zu folge sei das TI-SDK derzeit ausschließlich für das Betriebssystem Ubuntu in den Versionen 10.04 und 12.04 getestet[8]. Es empfiehlt sich daher eine der genannten Versionen zu verwenden. Die Installation des Systems kann nativ auf einem Computer oder als virtuelles Betriebssystem<sup>1</sup> erfolgen.

## 3.1 Installation

Die heruntergeladene Binär-Datei lässt sich im Terminal durch den Aufruf

```
./ti-sdk-am335x-evm-07-00-00-00-Linux-x86-Install.bin
```

---

<sup>1</sup>Das virtuelle System benötigt zwingend Zugriff auf das SD-Kartenlesegerät

ausführen wodurch ein grafisches Installationsmenü gestartet wird. Innerhalb dieses Menüs erfolgt die Abfrage eines Installationspfads. Im Defaultzustand wird dieser mit `/opt/ti-sdk-am335x-evm-07.00.00.00` angegeben, welcher nach Abschluss der Installation alle relevanten Programme und Dateien beinhaltet. Anschließend wird durch Ausführen des Scripts

```
/opt/ti-sdk-am335x-evm-07.00.00.00/setup.sh
```

das System konfiguriert. Erforderlich ist zum Erstellen einer seriellen Verbindung die Eingabe der seriellen Schnittstelle<sup>2</sup> an die der BeagleBone Black angeschlossen ist. Außerdem erfordert das Script die Angabe, dass das Betriebssystem auf die SD-Karte gespeichert wird. Weitere Eingabeaufforderungen können mittels der Enter-Taste im Defaultzustand gelassen werden.

## 3.2 Linux Image

Das TI-SDK beinhaltet die Linux-Arago Distribution, welche zum Einen als sogenanntes Prebuild zur Verfügung steht, aber auch als Sourcecode, welcher zur Erzeugung eines eigenen Build genutzt werden kann. Ein vollständiges Build besteht aus

X-Loader	setzt die Pin-Multiplexer und lädt den Bootloader
Bootloader	setzt die Boot-Parameter und startet den Kernel
Kernel Image	enthält den Kernel des Betriebssystems
Root Dateisystem	beinhaltet die gesamte Struktur des Dateisystems sowie alle Dateien, Einstellungen und Geräteinformationen vom Betriebssystem
Devicetree	beinhaltet die Informationen und Parameter über die Pin-Multiplexer der Pinbuchsen-Bänke

### 3.2.1 Build erzeugen

Zur Erzeugung eines eigenen Build steht das Makefile

```
/opt/ti-sdk-am335x-evm-07.00.00.00/Makefile
```

vorkonfiguriert zur Verfügung. Wird dieses durch Aufruf von

```
make
```

ausgeführt, dann wird ein komplettes Build in den Grundeinstellungen kompiliert.

---

<sup>2</sup>der Befehl `dmesg` gibt Auskunft über angeschlossene serielle Schnittstellen

Um eigene Einstellungen am Build vornehmen zu können, muss das Einstellungsmenü "menuconfig" aufgerufen werden. Hierfür wird das Makefile im Abschnitt "Building the Linux Kernel" mit

```
$(MAKE) -j $(MAKE_JOBS) -C $(LINUXKERNEL_INSTALL_DIR)
ARCH=arm CROSS_COMPILE=$(CROSS_COMPILE)
```

ergänzt. Dies hat zur Folge, dass durch Ausführen des Makefiles das Konfigurationsmenü gestartet wird. Alle im Konfigurationsmenü gesetzten Einstellungen werden in der Datei

`/opt/ti-sdk-am335x-evm-07.00.00.00/board-support/linux-3.12.10-ti2013.12.01/.config`

abgespeichert und beim Kompilieren des Builds berücksichtigt. Nach Abschluss der Kompilation finden sich im Installationspfad folgende Dateien

X-Loader	board-support/u-boot-2013.10-ti2013.12.01/MLO
Bootloader	board-support/u-boot-2013.10-ti2013.12.01/u-boot
Kernel Image	board-support/linux-3.12.10-ti2013.12.01/arch/arm/boot/zImage
Root Dateisystem	targetNFS/
Devicetree	board-support/linux-3.12.10-ti2013.12.01/arch/arm/boot/dts/am335x-boneblack.dtb

### 3.2.2 SD-Karte einrichten

Das fertige Linux-Betriebssystem besteht mindestens aus zwei Partitionen. Die Boot-Partition enthält den Bootloader, den X-Loader, das Kernel Image und den Devicetree. Die Root-Partition enthält das Root Dateisystem. Die Partitionierung und Einrichtung der SD-Karte geschieht durch den Aufruf des Scripts

`/opt/ti-sdk-am335x-evm-07.00.00.00/bin/create-sdcard.sh`

Sofern ein eigenes Build für die Einrichtung verwendet werden soll, müssen die Pfade der erforderlichen Dateien angegeben werden. Nach Abschluss ist das Target-System komplett eingerichtet und kann im BeagleBone Black verwendet werden.

### 3.2.3 Systemstart

Im Auslieferungs-Zustand beinhaltet der BeagleBone Black als Betriebssystem eine Linux-Angstrom Distribution, welche auf dem internen Flash-Baustein gespeichert ist. Wird der BeagleBone Black durch Anschluss einer Spannungsquelle gestartet, dann sucht das System entsprechend der Reihenfolge

1. Flash-Baustein
2. microSD Kartenslot
3. UART0 Schnittstelle
4. USB0 Schnittstelle

die Quellen nach einer gültigen Boot-Partition ab und startet von dieser[4].



Abbildung 2: Boot-Button

([http://www.ti.com/ww/en/beagleboard/product\\_detail\\_black\\_lg.jpg](http://www.ti.com/ww/en/beagleboard/product_detail_black_lg.jpg))

Durch drücken des Boot-Tasters (Abbildung 2) ändert sich die Reihenfolge entsprechend[4]

1. SPI0
2. microSD Kartenslot
3. UART0 Schnittstelle
4. USB0 Schnittstelle

Wird die MLO-Datei von der Boot-Partition auf dem Flash-Baustein gelöscht oder umbenannt, so wird die Linux-Angstrom Distribution bei zukünftigen Systemstarts nicht mehr gestartet. Dies hat zur Folge, dass die SD-Karte die erste auffindbare Boot-Partition beinhaltet und somit bei Systemstart gestartet wird.

### 3.3 Serielle Kommunikation

Für den Aufbau einer seriellen Verbindung beinhaltet das TI-SDK das Programm "minicom", welches durch den Befehl

```
minicom
```

aufgerufen wird. Die im Abschnitt 3.1 erstellte Verbindung wird so automatisch gestartet und bietet vollen Zugriff auf das Dateisystem vom BeagleBone Black.

### 3.4 Toolchain

Das TI-SDK beinhaltet die Linaro-Toolchain zum Entwickeln von Software für Cortex A8 Systeme[9] in den Programmiersprachen C und C++. Der Cross-Compiler der Linaro-Toolchain kann durch einbinden der Unterordner des Installationspfads

Cross-Compiler	linux-devkit/sysroots/i686-arago-linux/usr/bin/
Bibliotheken	linux-devkit/sysroots/cortexa8hf-vfp-neon-3.8-oe- linux-gnueabi/usr/include/

den Umgebungsvariablen des Systems hinzugefügt und anschließend verwendet werden. Das TI-SDK bietet alternativ die Möglichkeit im Terminal den Befehl

```
. /opt/ti-sdk-am335x-evm-07.00.00.00/linux-devkit/environment-setup
```

auszuführen, wodurch temporär innerhalb des Terminals die Systemeigenen Umgebungsvariablen ausgeschaltet und vorkonfigurierte Systemvariablen zum Verwenden des Cross-Compilers eingeschaltet werden.



```
//helloworld.cpp

#include <iostream>

int main()
{
    std::cout << "Hello_World!" << std::endl;
    return 1;
}
```

Abbildung 3: Hello World Programm in C++

Die Kompilierung von C++ Sourcecode, wie in Abbildung 3 zu sehen, geschieht durch den Befehl

```
arm-linux-gnueabi-g++ helloworld.cpp -o helloworld
```

Der Compiler erzeugt in diesem Beispiel eine Datei namens helloworld, welche auf den BeagleBone Black transferiert und ausgeführt werden kann.

## 4 GPIO

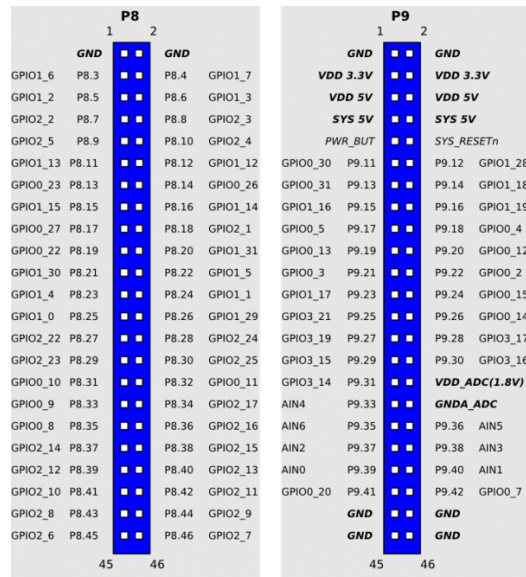


Abbildung 4: Pinbänke des BeagleBone Black

Die Pinbänke des BeagleBone Black (Abbildung 4) besitzen insgesamt 92 Pins, wovon 16 Pins der Spannungsversorgung und Masse dienen. Alle weiteren Pins stehen, je nach Konfiguration, der General-Purpose Input/Output Kommunikation zur Verfügung. Einen Zugriff auf einen Großteil der Hardware-Komponenten bietet Linux durch den Pfad

```
/sys/class/
```

Hier finden sich unter Anderem Konfigurationsmöglichkeiten für I2C- und SPI-Busse, Displays, usw.

## 4.1 Verfügbarkeit eines Pins ermitteln

Bevor ein Pin verwendet wird, sollte zunächst geprüft werden, in welchem Modus dieser sich befindet. Als erstes Beispiel dient hier der Pin P9.17, welcher auf der Bank P9 zu finden ist und mit GPIO0\_5 benannt ist. Ein Blick in das System Reference Manual gibt Auskunft über die verschiedenen Modi dieses Pins[5].

**Table 13. Expansion Header P9 Pinout**

PIN	PROC	NAME	MODE0	MODE1	MODE2	MODE3	MODE4	MODE5	MODE6	MODE7
10	P14	ehrpwm0	gpio0_03	mmc2_sdwp	gpio0_04	mimo2_uartc	gpio0_05		ehrpwm0_mux1	gpio0_113
17	A16	I2C1_SCL	spi0_cs0	mimo2_sdwp	I2C1_SCL	ehrpwm0_synci	pr1_uart0_txd			gpio0_5
18	P15	gpio0_06	spi0_cs0	mimo2_sdwp	gpio0_07	ehrpwm0_synci	pr1_uart0_txd			gpio0_114

Abbildung 5: Ausschnitt aus dem System Reference Manual Pin P9.17

Wie in Abbildung 5 zu erkennen besitzt der gewählte Pin neben dem GPIO-Modus noch die Modi: spi0\_cs0, MMC2\_sdwp, I2C1\_SCL, ehrpwm0\_synci und pr1\_uart0\_txd. Die Information des aktiven Modus erhalten wir durch den Befehl

```
ls -l /sys/kernel/debug/omap_mux
```

welcher, wie in Abbildung 6 zu erkennen, unter Anderem einen Modus des gewählten Pins auflistet. Dies ist nicht zwingend der derzeit aktive Modus des Pins.

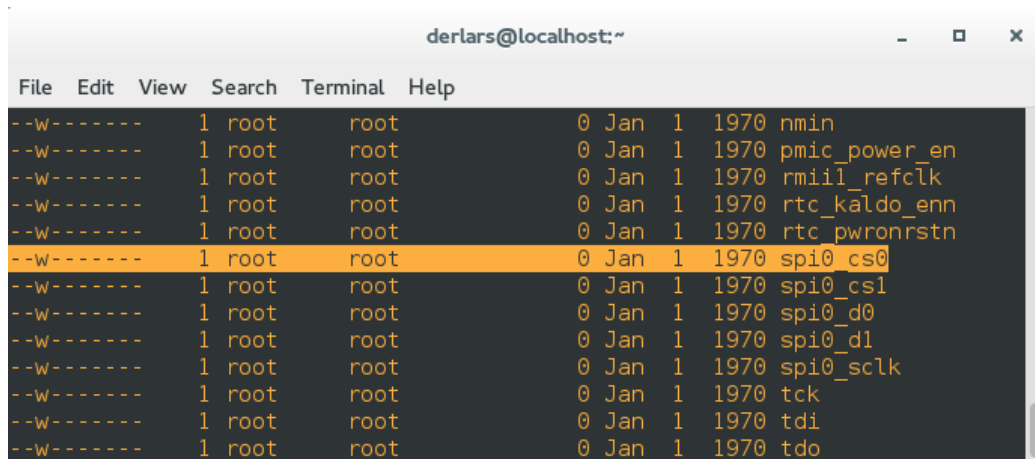


Abbildung 6: Auflistung Pin Modis für Pin P9.17

Erst durch den Befehl

```
ls -l /sys/kernel/debug/omap_mux/spi0_cs0
```

wird der aktive Modus aufgelistet.

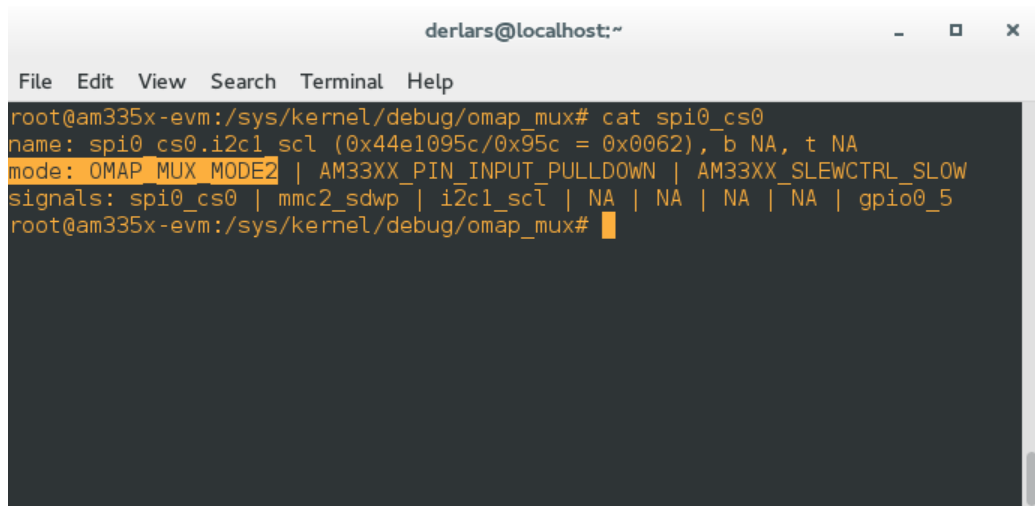
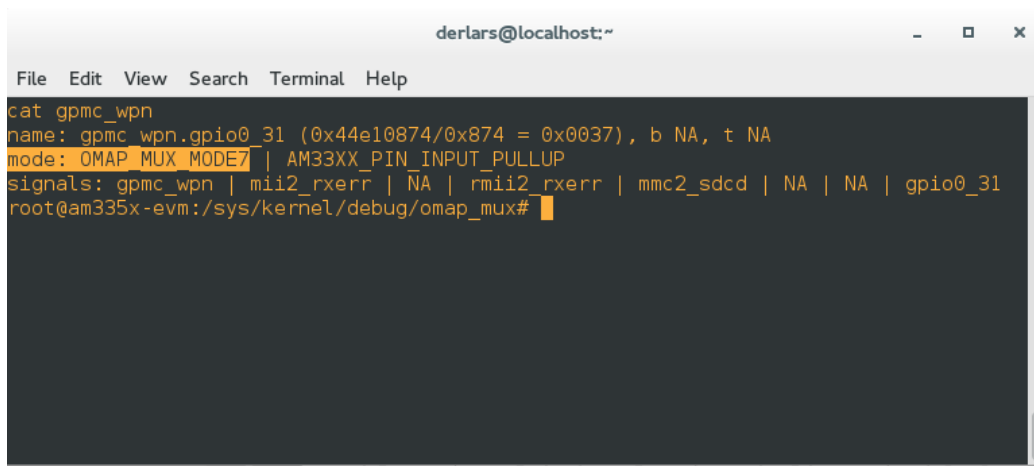


Abbildung 7: Aktiver Modus für Pin P9.17

In Abbildung 7 ist zu erkennen, dass sich der gewählte Pin derzeit im zweiten Modus (I2C1\_SCL) befindet. Sofern es keinen Einfluss auf das konfigurierte System hat, kann der GPIO-Modus durch den Befehl

```
echo 7 >/sys/kernel/debug/omap_mux/spi0_cs0
```

erreicht werden. Diese Änderung ist bis zu einem erneuten Wechsel des Modus oder einem Neustart des Systems gültig. Kann eine negative Beeinflussung durch den Moduswechsel auf das System nicht ausgeschlossen werden, weil zum Beispiel weitere Hardware an den I2C-Bus angeschlossen ist, muss ein anderer freier Pin gesucht werden. Als zweites Beispiel dient der Pin P9.13, welcher ebenfalls auf der Bank P9 platziert ist und den Namen GPIO0\_31 hat. Eine Auflistung des aktiven Modus nach dem gleichen Schema gibt preis, dass dieser Pin für die Verwendung des GPIO-Modus bedenkenlos geeignet ist, da sich dieser schon im entsprechenden Modus befindet. (Abbildung 8)



```

derlars@localhost:~
File Edit View Search Terminal Help
cat gpmc_wpn
name: gpmc_wpn.gpio0_31 (0x44e10874/0x874 = 0x0037), b NA, t NA
mode: OMAP_MUX_MODE7 | AM33XX_PIN_INPUT_PULLUP
signals: gpmc_wpn | mi2_rxerr | NA | rmii2_rxerr | mmc2_sdcd | NA | NA | gpio0_31
root@am335x-evm:/sys/kernel/debug/omap_mux#

```

Abbildung 8: Aktiver Modus für Pin P9.13

## 4.2 Pin verwenden

Steht ein Pin, wie in Abschnitt 4.1 ermittelt, zur Verfügung dann muss dieser zunächst durch den Befehl

```
echo 31 >/sys/class/gpio/export
```

in das Dateisystem eingebunden werden. Hierdurch wird der Pfad

```
/sys/class/gpio/gpio31
```

erzeugt, welcher zur Modifikation des Pins P9.13 dient. Eine Auflistung dieses Pfads beinhaltet unter Anderem die Dateien `direction`, `value`, `edge` und `active_low`[10]. Die Inhalte der jeweiligen Dateien lassen sich durch den Befehl

```
cat [Dateiname]
```

aufflisten. Der Inhalt aus `direction` definiert ob es sich um einen input-Pin oder einen output-Pin handelt. Im Defaultzustand ist jeder Pin als input-Pin definiert. Durch schreiben eines `out` in die Datei mittels des Befehls

```
echo out >direction
```

wird der Pin zu einem output-Pin. Der aktive Pegel des Pins wird durch `value` definiert. Ist ein Pin als input-Pin definiert, dann enthält `value` die Information darüber, ob an dem Pin eine Spannung, also ein high-Pegel, anliegt. Dies wird durch eine 1 signalisiert und kann, wie beschrieben, mittels des Befehls

```
cat value
```

abgefragt werden. Sofern keine Spannung am Pin anliegt, beinhaltet `value` eine 0. Das System Reference Manual weist darauf hin, dass die Pins des BeagleBone Black nur 3,3V tolerant seien und nicht 5V tolerant[5]. Liegt dennoch eine Spannung von mehr als 3,3V an einem Pin an, kann der BeagleBone Black dadurch Schaden nehmen. Ist ein Pin als output-Pin definiert, kann der Pegel des Pins durch schreiben einer 1 in `value` auf high gesetzt und schreiben einer 0 in `value` auf low gesetzt werden. Durch die Datei `active_low` lässt sich dieses Verhalten invertieren. Beinhaltet `active_low` eine 0, bedeutet eine 0 in `value` ein low-Pegel und eine 1 in `value` einen high-Pegel. Beinhaltet `active_low` allerdings eine 1, dann bedeutet eine 0 in `value` ein high-Pegel und eine 1 in `value` ein low-Pegel. Die Datei `edge` kann die Werte `none`, `rising`, `falling` oder `both` annehmen und dient der Interrupt-Erkennung[10]. Auf das Thema Interrupt wird an dieser Stelle nicht weiter eingegangen.

### 4.3 Pulsweitenmodulation

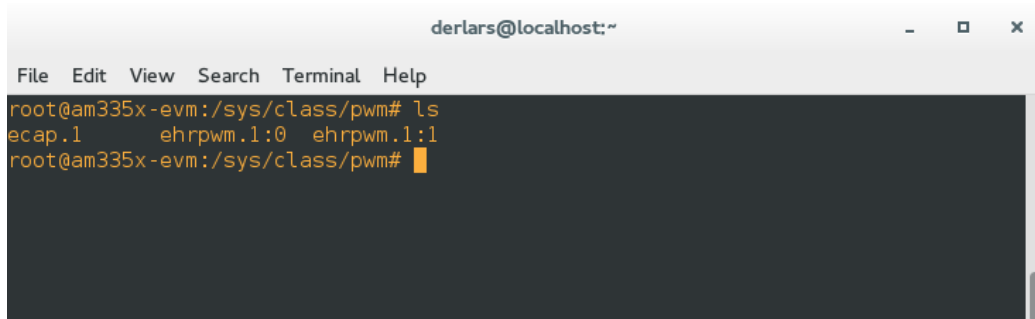


Abbildung 9: Auflistung aktiver PWM-Pins

Bei der Pulsweitenmodulation wird die Ein- und Ausschaltzeit eines Rechtecksignals bei fester Grundfrequenz variiert[12]. Das Tastverhältnis (engl. duty cycle) errechnet sich aus

$$t_{ein}/(t_{ein} + t_{aus})$$

Laut beagleboard.org besitzt der BeagleBone Black acht Pins für die Pulsweitenmodulation[13]. Aktive PWM-Pins sind im Pfad

`/sys/class/pwm/`

als Ordner aufgelistet (Abbildung 9). Zur Modifizierung des jeweiligen Pins stehen in dem entsprechenden Ordner unter Anderem die Dateien `run`, `tick_hz`, `period_ns`, `period_freq`, `duty_ns`, `duty_percent`, `polarity` und `request` zur Verfügung[11]. `run` definiert den aktuellen Status des PWM-Pins. Beinhaltet `run` eine 1, dann ist der Pin eingeschaltet. Beinhaltet `run` eine 0, dann ist der Pin ausgeschaltet. `tick_hz` enthält die aktuelle Systemfrequenz. `period_ns` enthält die Zeit einer gesamten Periode in Nanosekunden. `period_freq` errechnet sich nach folgender Formel

$$period_{freq} = \frac{1}{\frac{period_{ns}}{1000000000ns/s}} = \frac{1000000000ns}{period_{ns} * s}$$

Durch `duty_ns` wird die Anzahl der Nanosekunden definiert, die der Pegel innerhalb einer Periode auf high gesetzt ist. Somit errechnet sich `duty_percent` nach der Formel

$$duty_{percent} = \frac{duty_{ns}}{period_{ns}} * 100$$

`polarity` beeinflusst die Polarität des PWM-Pins. Eine 0 definiert den Pin als output-Pin und eine 1 definiert diesen als input-Pin. Die Datei `request` kann

die Information beinhalten, wofür dieser PWM-Pin verwendet wird. Dies kann zum Beispiel die Hintergrundbeleuchtung eines Displays oder die Ansteuerung eines Schrittmotors sein. Ist keine Verwendung eingetragen, dann beinhaltet request die Information, dass der Pin frei ist. Da eine Eintragung einer Verwendung nicht zwingend erforderlich ist, ist es möglich, dass der Pin als frei definiert ist, obwohl dieser schon verwendet wird. Hier ist also Vorsicht geboten!

## 5 Implementation

Um die in Abschnitt 4 erläuterten Zugriffe auf GPIO-Pins zu implementieren bedarf es nur wenig Aufwand.

```
#include <fstream>

int writeTo(char target[], char value[])
{
    std::ofstream targetFile;
    targetFile.open(target);
    if(targetFile.is_open())
    {
        targetFile << value;
        targetFile.close();
        return 1;
    }
    return -1;
}
```

Abbildung 10: writeTo Funktion in C++

Die Hauptaufgabe übernimmt eine Funktion die in Dateien des Dateisystems schreibt. (Abbildung 10). Diese Funktion wird anschließend mit den entsprechenden Parametern von den weiteren Funktionen aufgerufen, sodass ein Pin mittels der Funktionen aus Abbildung 11 erzeugt und konfiguriert werden kann.

```
#define MAXSIZE 64

int exportPin(int pin)
{
    char target[] = "/sys/class/gpio/export";
    char value[MAXSIZE];
    sprintf(value, "%d\n", pin);

    return writeTo(target, value);
}

int unexportPin(int pin)
{
    char target[] = "/sys/class/gpio/unexport";
    char value[MAXSIZE];
    sprintf(value, "%d\n", pin);

    return writeTo(target, value);
}

int setOutputPin(int pin)
{
    char target[MAXSIZE];
    char value[] = "out";
    sprintf(target, ".../gpio%d/direction", pin);

    return writeTo(target, value);
}

int setInputPin(int pin)
{
    char target[MAXSIZE];
    char value[] = "in";
    sprintf(target, ".../gpio%d/direction", pin);

    return writeTo(target, value);
}
```

Abbildung 11: GPIO konfigurations-Funktionen in C++



```
int setValue(int pin)
{
    char target[MAXSIZE];
    char value[] = "1";
    sprintf(target, ".../gpio%d/value", pin);

    return writeTo(target, value);
}

int revokeValue(int pin)
{
    char target[MAXSIZE];
    char value[] = "0";
    sprintf(target, ".../gpio%d/value", pin);

    return writeTo(target, value);
}
```

Abbildung 12: GPIO value-Funktionen in C++

Das Verändern des Pegels geschieht ebenfalls nach diesem Muster. Entsprechende Funktionen sind in Abbildung 12 definiert.

## 6 Zusammenfassung

Das Beispiel des Verwenden eines einfach GPIO-Pins zeigt, dass die Hauptaufgabe des Entwicklers darin besteht, dass Dateisystem von Linux zu überblicken. Ist dieser Schritt geschehen, lässt sich das erlangte Wissen schnell auf weitere Fragestellungen anwenden. Steht einem ein SDK zur Verfügung ist auch der erste Schritt zu einer Lauffähigen Entwicklungsumgebung schnell gemacht.

## Literatur

- [1] System Reference Manual Revision C.1, Seite 14 [Stand 22.05.2014]
- [2] System Reference Manual Revision C.1, Seite 18 f [Stand 22.05.2014]
- [3] System Reference Manual Revision C.1, Seite 30 [Stand 22.05.2014]
- [4] System Reference Manual Revision C.1, Seite 69 [Stand 22.05.2014]
- [5] System Reference Manual Revision C.1, Seite 82 ff [Stand 22.05.2014]
- [6] Texas Instruments - <http://www.ti.com/product/am3358>  
[Stand 08.01.2014]
- [7] Texas Instruments -  
[http://software-dl.ti.com/sitara\\_linux/esd/AM335xSDK/latest/index\\_FDS.html](http://software-dl.ti.com/sitara_linux/esd/AM335xSDK/latest/index_FDS.html)  
[Stand 08.01.2014]
- [8] Texas Instruments -  
[http://processors.wiki.ti.com/index.php/Sitara\\_SDK\\_Installer](http://processors.wiki.ti.com/index.php/Sitara_SDK_Installer)  
[Stand 02.04.2014]
- [9] Texas Instruments -  
<http://www.ti.com/lsds/ti/tools-software/mainlinelinux.page>  
[Stand 24.06.2013]
- [10] Linux Kernel Organization -  
<https://www.kernel.org/doc/Documentation/gpio/sysfs.txt>  
[Stand 15.12.2014]
- [11] Linux Kernel Organization -  
<https://www.kernel.org/doc/Documentation/pwm.txt>  
[Stand 01.12.2014]
- [12] mikrocontroller.net - <http://www.mikrocontroller.net/articles/Pulsweitenmodulation>  
[Stand 10.01.2014]
- [13] BeagleBoard.org -  
<http://beagleboard.org/support/BoneScript/analogWrite/>  
[Stand 19.08.2013]