Kernel- und Treiberprogrammierung mit dem Linux-Kernel – Folge 108

Kern-Technik

Über Pulsweitenmodulation lassen sich mit Single-Board-Computern Blinklichter, gedimmte LEDs und variable Motordrehzahlen unkompliziert realisieren – vorausgesetzt, der Kernel ist richtig konfiguriert. Die vorliegende Folge der Kern-Technik-Serie demonstriert das am Beispiel eines Raspberry Pi. Eva-Katharina Kunst, Jürgen Quade

Der Raspberry Pi gehört neben dem Arduino und dem ESP32 zu den Lieblingsobjekten jedes Elektronik-Makers. Kein Wunder, lässt sich doch bei dem preiswerten Linux-Rechner über die Steckerleiste mit wenig Aufwand Hardware ankoppeln und vor allem mit ein paar Zeilen Python ansprechen. Mal eben eine LED anschalten oder einen Sensor ankoppeln? Kein Problem.

An den Pins der Steckerleiste lässt sich per Software durch das Schreiben einer »1« beziehungsweise »0« eine Spannung von 3,3 respektive 0 Volt anlegen. Zwar liefern die Ausgänge nur begrenzt Strom, aber mit ein wenig zwischengeschalteter Hardware (H-Brücke) lassen sich auch Motoren betreiben.

Schreibt die Software eine »1« auf den zum Ausgang gehörenden GPIO-Pin, liegen 3,3 Volt an, und der Motor dreht sich. Seine Drehzahl richtet sich nach der angelegten Spannung und der Bauform. Sobald man eine »0« auf den Pin schreibt, sinkt die Spannung auf 0 Volt; der Motor steht. Motor an, Motor aus – da muss doch mehr gehen?

Gepulster Betrieb

Betreibt man den Motor gepulst, indem man ihn an- und, bevor er auf volle Drehzahl kommt, wieder abschaltet und dieses Spiel in schneller Folge wiederholt, dann dreht sich der Motor aufgrund der Trägheit insgesamt langsamer. Die Drehzahl hängt dabei von der Häufigkeit der Schaltvorgänge ab sowie vom Verhältnis der Zeitspannen, in denen man den Motor mit Strom versorgt und abschaltet. Diese Art, einen Ausgang gepulst zu betreiben, nennt sich konsequenterweise Pulsweitenmodulation (englisch Pulse Width Modulation) oder kurz PWM (**Abbildung 1**). Verschaltet man anstelle des Motors eine LED am Ausgang und betreibt ihn pulsweitenmoduliert mit einem Tastgrad von 50 Prozent, leuchtet die Diode (etwa) nur noch halb so hell. Eine sehr niedrige Frequenz mit einer langen Periodendauer erzeugt dagegen ein klassisches Blinklicht.

Per Software erzeugt man ein pulsweitenmoduliertes Signal einfach mithilfe einer Schleife, in der der Ausgang den Zustand ständig wechselt und zwischendurch ein wenig pausiert (siehe **Kasten „Softwaregesteuerte PWM“**). Die damit erzielbare Genauigkeit lässt allerdings zu wünschen übrig, der Code muss parallel zur eigentlichen Applikation arbeiten und verbraucht zudem noch Rechenzeit.

Kollision mit Audio

Eine so simple Aufgabe wie das Erzeugen eines gepulsten Signals lässt sich jedoch

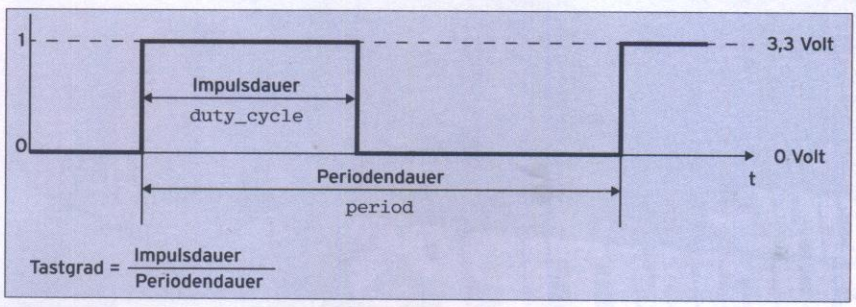

Abbildung 1: Das Prinzip und die Parameter der Pulsweitenmodulation.

Die Autoren

Eva-Katharina Kunst ist seit den Anfängen von Linux Fan von Open Source. Jürgen Quade, Professor an der Hochschule Niederrhein, bietet auch für Unternehmen Schulungen zu den Themen Treiberprogrammierung und Embedded Linux an.

wesentlich besser durch Hardware realisieren, weswegen moderne Controller entsprechende Funktionsblöcke gleich mitbringen. Der Raspberry Pi beispielsweise verfügt über zwei PWM-Kanäle, die sich dank Linux und eines PWM-Subsystems im Kernel einfach über das Sys-Filesystem ansteuern lassen.

Allerdings gelingt standardmäßig der Zugriff auf diese per Hardware realisierte Pulsweitenmodulation nicht: Der auf dem Raspberry Pi verbaute Broadcom-Controller bietet zwar eine schier unglaubliche Funktionsvielfalt, doch kann der Anwender diese Möglichkeiten nicht unbedingt alle zeitgleich nutzen. Die angesprochene Hardware-PWM beispielsweise kollidiert mit der internen Audioverarbeitung, die in der Standardkonfiguration auf dem RasPi den Vorzug genießt.

Hilfe vom Bootloader

Um die Hardware-PWM zu nutzen, steht folglich am Anfang das Aktivieren und Konfigurieren durch den Bootloader. Dazu erweitert man auf einem aktuellen Raspbian die in der Boot-Partition befindliche Datei »config.txt« um die Zeile »dtoverlay = pwm«. Kommt ein Kernel mit einer Version kleiner als 4.9 zum Einsatz, lautet die Zeile stattdessen »dtoverlay = pwm-with-clk«. Weitere Einzelheiten nennt der Kasten „Boot-Konfiguration für PWM".

Bootet man mit dieser Konfiguration, dann lädt der Raspberry Pi automatisch den Treiber »pwm_bcm2835« und füttert ihn mit den notwendigen Hardwareadressen. Im Verzeichnis »/sys/class/pwm/« gibt es jetzt ein neues Unterverzeichnis »pwmchip0/« und darin unter anderem die virtuellen Dateien »export«, »npwm« und »unexport«.

Die Datei »npwm« gibt die Anzahl der Kanäle für die Pulsweitenmodulation zurück. Im Fall des Raspberry Pi sind das zwei, von denen sich jedoch, abhängig von der beim Booten aktivierten Hardware-Konfiguration, eventuell nur einer nutzen lässt.

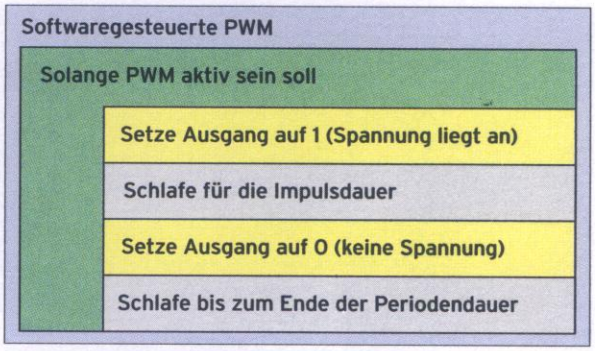

Abbildung 2: Funktionsweise der Pulsweitenmodulation per Software.

Das Reservieren eines PWM-Kanals erfolgt über die Datei »export«. Schreibt der Anwender, mit Root-Rechten versehen, in diese Datei eine »0«, erscheint das neue Verzeichnis »pwm0/«, das unter anderem die Dateien »capture«, »duty_cycle«, »enable«, »period« und »polarity« beherbergt. Über diese Dateien lässt sich die Pulsweitenmodulation am konfigurierten Ausgang (GPIO 12 oder GPIO 18) in Betrieb nehmen.

Dazu schreibt der Anwender die Periodendauer (Kehrwert der Frequenz) in die Datei »period« und die Impulsdauer respektive Pulsweite – beides angegeben in Nanosekunden – in die Datei »duty_cycle«. Um etwa für ein Blinklicht einen Impuls mit einer halben Sekunde Länge zu erzeugen, der jede Sekunde einmal auftritt, muss der Anwender in »period« den Wert 1 000 000 000 und in »duty_cycle« 500 000 000 ablegen. Das Schreiben

Softwaregesteuerte PWM

Der Raspberry Pi bringt zwei in Hardware realisierte PWM-Kanäle mit, die sich auf zwei der vier GPIO-Pins 12, 13, 18 und 19 konfigurieren lassen. Setzt man die PWM per Software um, kann man auch andere Pins nutzen. Das Struktogramm in Abbildung 2 zeigt das prinzipielle Vorgehen: In einer Schleife erzeugt der Code den Impuls durch Ausgabe einer »1« auf den gewünschten GPIO-Pin, danach schläft er für die Impulsdauer (Pulsweite). Anschließend schaltet er durch Ausgabe einer »0« die Spannung am Ausgang auf 0 Volt und pausiert dann noch einmal bis zum Ende der Periodendauer.

Sieht man die PWM als Teil eines Ganzen, beispielsweise als Komponente einer Regelung, wird klar, dass dieser Part parallel zum Berechnen der Regelparameter laufen muss. Das Anpassen von Pulsweite oder Frequenz erfordert zudem eine Interprozesskommunikation, im einfachsten Fall über globale Variablen. Das macht die zunächst theoretisch einfache Realisierung in der Praxis erheblich komplizierter als eine Lösung via Hardware.

Hinzu kommt ein weiterer wesentlicher Nachteil: Während bei einer per Hardware gesteuerten PWM die Genauigkeit im unteren Mikrosekundenbereich liegt, muss man auf Softwareebene mit Abweichungen einzelner Perioden im Millisekundenbereich rechnen. Selbst das klappt nur mithilfe tiefer Griffe in die Trickkiste der Echtzeitprogrammierung. Obendrein frisst die softwareseitige PWM der CPU Rechenzeit weg. Bei kurzen Periodendauern (sprich: hohen Frequenzen) macht sich das signifikant bemerkbar und darf keinesfalls vernachlässigt werden.

Boot-Konfiguration für PWM

Raspberry Pi A und B, RasPi 2 und 3, Pi Zero, Raspberry Pi 4, Compute- und Industrie-Module – die Hardware der einzelnen Raspberry-Pi-Modelle unterscheidet sich teils recht grundlegend. Um die Hardware trotz aller Unterschiede weitestgehend mit nur einem Kernel zu bedienen, lagerten die Linux-Entwickler schon vor Jahren die eigentliche Hardwarebeschreibung (Adressen, Interrupts, DMA-Kanäle) in sogenannte Devicetrees aus und trennten sie so vom Kernel-Code. Beim Booten legt der Bootloader den für den jeweiligen Kernel relevanten Devicetree in den Hauptspeicher und übergibt dem Kernel beim Starten dessen Adresse. Devicetrees liegen mittlerweile modular in Form sogenannter Overlays vor, der Bootloader zeichnet für das Laden der benötigten Overlays verantwortlich.

Auf dem Raspberry Pi finden sich in der Datei »/boot/overlays/README« die Beschreibung der entsprechenden Overlays sowie deren Konfigurationsparameter. Demnach konfiguriert das Overlay »pwm« einen einzelnen PWM-Kanal, während »pwm-2chan« die Konfiguration von zwei Kanälen abhandelt. Dabei kann der Anwender die einzelnen Kanäle auf unterschiedliche GPIOs legen (Abbildung 3): *PWM0* lässt sich im Wesentlichen auf GPIO 12 (auf der Steckerleiste Pin 32) oder GPIO 18 (Pin 12) ausgeben, *PWM1* auf GPIO 13 (Pin 33) oder GPIO 19 (Pin 35). Wer Audio und PWM zusammen einsetzen will, der stößt dabei auf Probleme: Pin 18 wird vom I²S-Audio-Interface verwendet, der analoge Audioausgang benötigt beide PWM-Kanäle.

Um die Pulsweitenmodulation zu konfigurieren, stehen die Overlay-Parameter »pin«, »func«, »pin2« und »func2« zur Verfügung. Tabelle 1 auf Seite 74 zeigt die Konfigurationszeile, die man der Datei »/boot/config.txt« anhängen muss, um die jeweiligen PWMs freizugeben und mit den gewünschten GPIO-Pins zu verknüpfen.

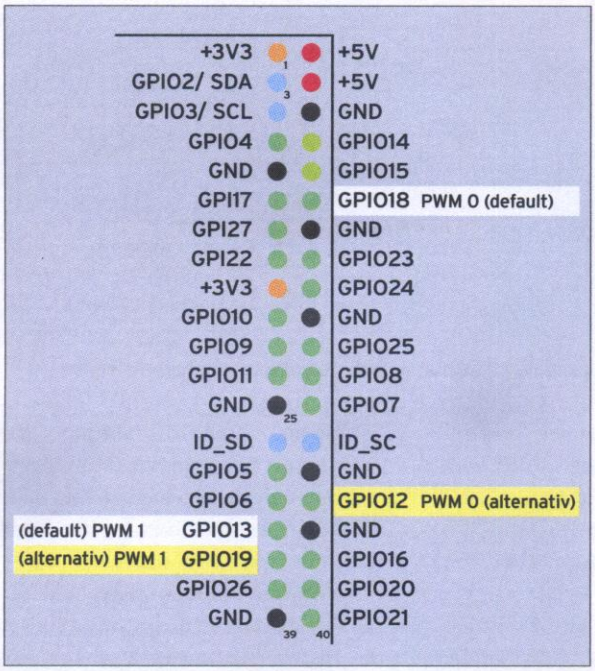

Abbildung 3: Via Hardware generierte PWM-Signale lassen sich beim Raspberry Pi an zwei von vier Pins ableiten.

einer »1« in die Datei »enable« aktiviert anschließend die PWM (Listing 1). Um sie wieder abzuschalten, schreibt man eine »0« in die Datei »enable«.

Namensirritation

Bei der Konfiguration der Perioden- und der Impulsdauer überprüft der Linux-Kernel die Sinnhaftigkeit der Daten. Er lehnt eine Impulsdauer, die länger als die Periodendauer ausfällt, ebenso ab wie eine Periodendauer, die kürzer ist als die Impulsdauer. Konsequenz: Erst schreibt man die Periodendauer (»period«), anschließend die Impulsdauer (»duty_cycle«). Den Namen der Datei »duty_cycle« haben die Linux-Macher allerdings etwas unglücklich gewählt, bedeutet duty cycle in der Fachsprache doch Tastgrad, was gemäß Norm eine dimensionslose Verhältniszahl von Pulsweite zu Periodendauer darstellt. Um also bei gegebenem Tastgrad die Pulsweite zu berechnen, wird der zwischen Null und Eins, also zwischen 0 und 100 Prozent liegende Tastgrad mit der Periodendauer multipliziert. In unserem Blinklicht-Beispiel ergibt sich damit ein Tastgrad respektive duty cycle von 50 Prozent.

Darüber hinaus kann man anstelle der Impulsdauer (Pulsweite) die Zeitdauer innerhalb der Periodendauer angeben, in der kein Impuls ausgegeben wird. Dazu trägt man in die Datei »polarity« den Wert »inversed« ein. Jetzt interpretiert der Kernel den Inhalt von »duty_cycle« als die Zeit ohne Impuls. Ein »normal« in derselben Datei schaltet wieder alles zurück auf die ursprüngliche Interpretation. Abbildung 4 zeigt die Ausgänge der beiden PWM-Kanäle des Raspberry Pi, wenn der Anwender einen Kanal normal und den anderen mit denselben Parametern invertiert betreibt. Die Periodendauer ist ebenso wie die Impulsdauer sehr deutlich zu erkennen.

Nur in den seltensten Fällen aktiviert ein Anwender die Pulsweitenmodulation nach dem Muster aus Listing 1 über die Konsole. Da die Konfiguration in Unix-Manier über Dateizugriffe erfolgt, fällt eine Umsetzung beispielsweise in C-Code leicht, wie Listing 2 zeigt. Der Funktion »write_pwm_config()« übergibt man als Parameter den Namen der zu ändernden Konfigurationsdatei sowie den zu schreibenden Wert.

Insbesondere die Datei »export« wird in einer realen Applikation nur einmal geschrieben, während man die Pulsweite oder auch die Periodendauer häufiger verändert, um beispielsweise einen Motor langsam anzufahren.

Sicherheit geht vor

Um aus dem Quellcode »pwm.c« das ausführbare Programm »pwm« zu generieren, genügt der Aufruf »make pwm« im Verzeichnis mit dem Quellcode. Da die Konfiguration der Pulsweitenmodulation normalerweise Superuser-Rechte erfordert, passt der IT-sicherheitstechnisch versierte

Listing 1: Konfiguration der PWM in der Konsole

```
01 pi@raspberrypi:~ $ sudo su
02 root@raspberrypi:/home/pi# cd /sys/class/pwm/pwmchip0/
03 root@raspberrypi:/sys/class/pwm/pwmchip0/# echo 0 > export
04 root@raspberrypi:/sys/class/pwm/pwmchip0/# cd pwm0
05 root@raspberrypi:/sys/class/pwm/pwmchip0/pwm0# echo 500000000 > period
06 root@raspberrypi:/sys/class/pwm/pwmchip0/pwm0# echo 250000000 > duty_cycle
07 root@raspberrypi:/sys/class/pwm/pwmchip0/pwm0# echo 1 > enable
```

Tabelle 1: Konfigurationszeile zur Aktivierung von PWM in »/boot/config.txt«

PWM 0 GPIO 12	PWM 0 GPIO 18	PWM 1 GPIO 13	PWM 1 GPIO 19	Konfiguration
X				»dtoverlay=pwm,pin=12,func=0«
	X			»dtoverlay=pwm,pin=18,func=5«
		X		»dtoverlay=pwm,pin=13,func=4«
			X	»dtoverlay=pwm,pin=19,func=2«
X		X		»dtoverlay=pwm,pin=12,func=0,pin2=13,func2=4«
X			X	»dtoverlay=pwm,pin=12,func=0,pin2=19,func2=2«
	X	X		»dtoverlay=pwm,pin=18,func=5,pin2=13,func2=4«
	X		X	»dtoverlay=pwm,pin=18,func=5,pin2=19,func2=2«

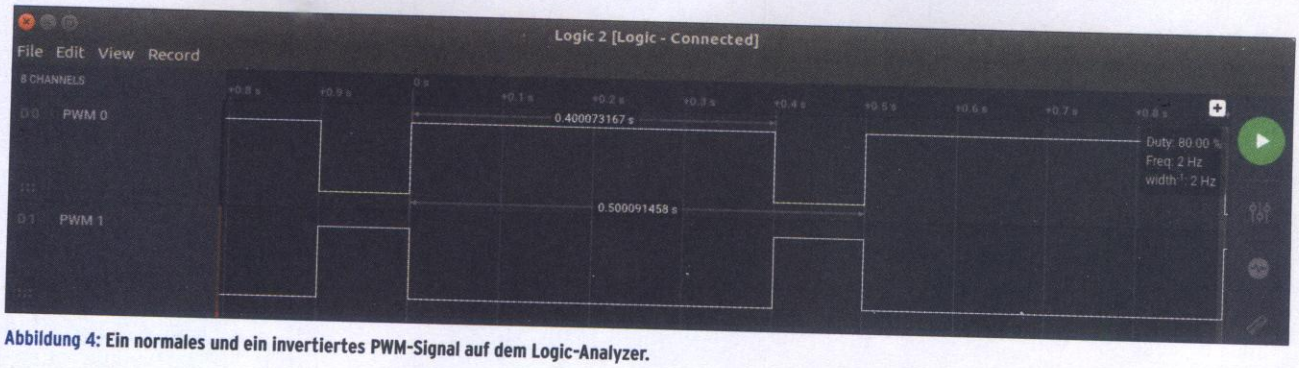

Abbildung 4: Ein normales und ein invertiertes PWM-Signal auf dem Logic-Analyzer.

Admin die Zugriffsrechte der Konfigurationsdateien im Sys-Filesystem an.

Er ändert die Besitzverhältnisse der zugehörigen Verzeichnisse und Konfigurationsdateien so, dass diese der Gruppe *gpio* gehören. Um die Änderungen über Boot-Vorgänge hinweg persistent zu machen, ergänzt er als Superuser auf dem Raspberry Pi die Datei »/etc/udev/rules.d/99-com.rules« um den Eintrag aus Listing 3.

Nach einem Reboot greifen diese Änderungen und ermöglichen den Zugriff für alle User, die der Gruppe *gpio* angehören.

Der Default-Benutzer *pi* gehört bereits zum Club; die Login-Namen anderer User hängt der Admin gegebenenfalls in der Datei »/etc/group« an die Zeile an, die mit »gpio« beginnt. Jetzt startet das Programm »pwm« ohne Root-Rechte.

Ausblick

Die hardwaregesteuerte Pulsweitenmodulation lässt sich erwartungsgemäß auch innerhalb des Kernels konfigurieren und nutzen, also in Gerätetreibern. Pfiffige Treiberprogrammierer realisieren zusätzliche hardwaregesteuerte PWM-Kanäle, indem sie noch DMA (Direct Memory Access) mit den zwei vorhandenen PWM-Kanälen kombinieren [1]. Damit lassen sich etwa alle vier Rotoren eines Quadrocopters unabhängig voneinander ansteuern. Doch dieses Thema bleibt einer späteren Kern-Technik-Folge vorbehalten. (*jlu*) ■

Infos

[1] DMA: Eva-Katharina Kunst, Jürgen Quade, „Kern-Technik", LM 06/2016, S. 74, [https://www.linux-magazin.de/36766]

Listing 2: Konfiguration der Hardware-PWM per C-Programm

```
01 #include <stdio.h>
02 #include <fcntl.h>
03 #include <unistd.h>
04 #include <string.h>
05
06 static int write_pwm_config( char *fname,
07   char *value )
08 {
09   int fd_pwm, ret, length;
10   char fpath[128];
11
12   if (fname==NULL || value==NULL) {
13     return -1;
14   }
15   snprintf( fpath, sizeof(fpath),
16     "/sys/class/pwm/pwmchip0/%s",
17     fname);
18   length = strlen( fpath );
19   if (length >= sizeof(fpath)) {
20     fprintf( stderr,
21       "path %s too long\n", fname);
22     return -1;
23   }
24
25   fd_pwm = open( fpath, O_WRONLY );
26   if (fd_pwm<0) {
27     perror( fpath );
28     return -1;
29   }
30   ret = write(fd_pwm,value,
31     strlen(value));
32   if (ret<0) {
33     perror( value );
34     close( fd_pwm );
35     return -2;
36   }
37   close( fd_pwm );
38   return 0;
39 }
40
41 int main( int argc, char **argv )
42 {
43   int ret;
44
45   ret = write_pwm_config("export","0");
46   if (ret) return ret;
47   ret = write_pwm_config(
48     "pwm0/period", "500000000" );
49   if (ret) return ret;
50   ret = write_pwm_config(
51     "pwm0/duty_cycle", "400000000");
52   if (ret) return ret;
53   ret = write_pwm_config(
54     "pwm0/enable", "1" );
55   if (ret) return ret;
56
57   return 0;
58 }
```

Listing 3: Berechtigungen der Konfigurationsdateien ändern

```
01 SUBSYSTEM=="pwm*", PROGRAM="/bin/sh -c '\
02   chown -R root:gpio /sys/class/pwm \
03   && chmod -R 770 /sys/class/pwm; \
04   chown -R root:gpio \
05   /sys/devices/platform/soc/*.pwm/pwm/
     pwmchip* \
06   && chmod -R 770 \
07   /sys/devices/platform/soc/*.pwm/pwm/
     pwmchip*'"
```