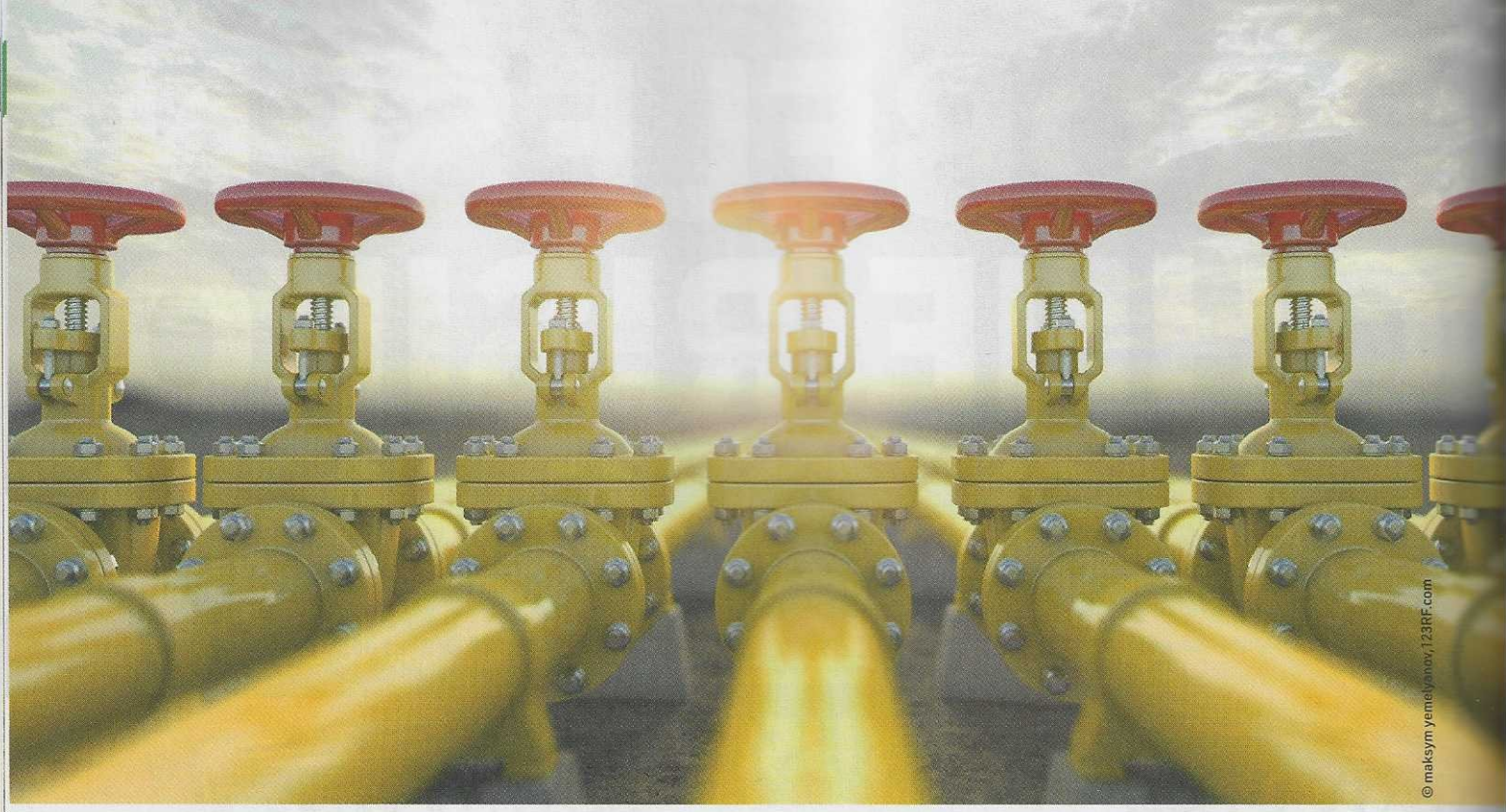


Kernel- und Treiberprogrammierung mit dem Linux-Kernel – Folge 115

Industrielle Kanäle

Das Industrial-IO-Subsystem des Kernels standardisiert die softwaretechnische Ankopplung insbesondere von Sensoren. Ungeachtet der Bezeichnung profitieren davon nicht nur die Profis, sondern auch Maker und Hobbyisten. Eva-Katharina Kunst, Jürgen Quade

Die Autoren

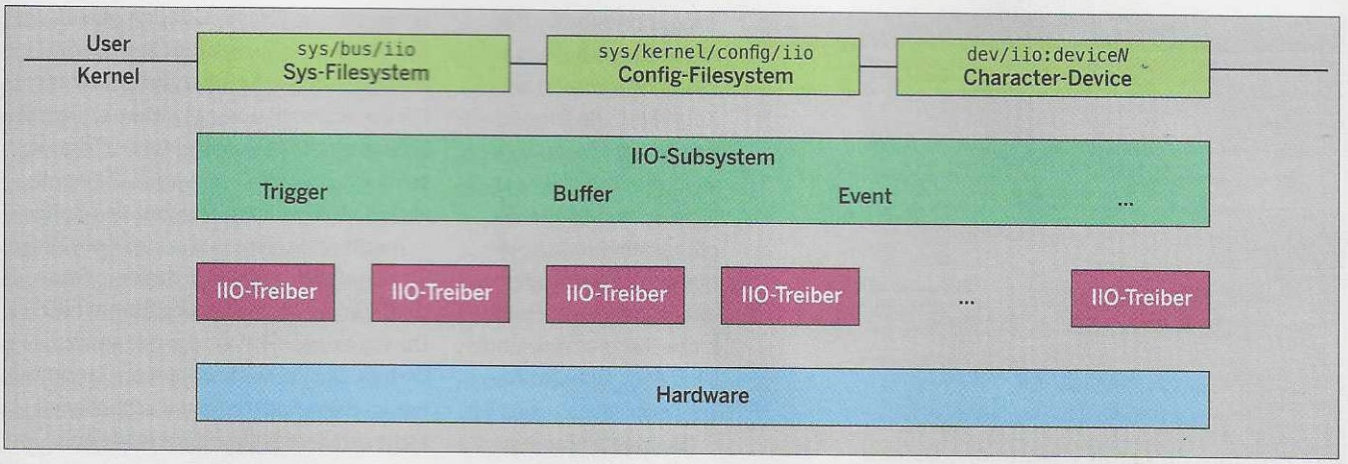
Eva-Katharina Kunst ist schon seit den Anfängen von Linux Fan von Open Source. **Jürgen Quade**, Professor an der Hochschule Niederrhein, bietet auch für Unternehmen Schulungen zu den Themen Treiberprogrammierung und Embedded Linux an.

Linux steuert Mähroboter, Satelliten und Autos. Dazu liest es über Gerätetreiber Sensorwerte ein, arbeitet Regelalgorithmen ab und gibt Parameter an Aktoren aus. Da jeder Sensor- und Aktortyp einen Treiber benötigt, entsteht ein verstreuter Mix an Treiber-Software.

Um mehr Ordnung in die Angelegenheit zu bringen, hob Jonathan Cameron 2009 das Industrial-IO-Subsystem (IIO) aus der Taufe. Damit gab er nicht nur den unzähligen Sensoren und – partiell – auch Aktoren ein wohliges Zuhause im großen Linux-Quellcodebaum, sondern stellte für Anwender und Entwickler ein einheitliches Interface für den professionellen Umgang mit den Daten bereit.

Dazu gehören etwa die Attribuierung mit Zeitstempeln und die Standardisierung von Sensorwerten. Was nutzt den Anwendern ein Temperaturwert, wenn nicht klar ist, ob es sich um Grad Celsius oder Fahrenheit handelt oder ob ein Zeitwert in Sekunden oder Millisekunden vorliegt? Der Name Industrial IO führt allerdings etwas in die Irre, assoziiert er doch eine industrielle Anwendung. Dabei handelt es sich eher um ein professionelles, standardisiertes Sensor-Interface, das besonders für Maker oder Hobbyisten interessant und relevant ist.

Geschätzt haben sich mittlerweile rund 300 Sensoren und Aktoren im IIO-Bereich des Quellcodes eingefunden, wo sie sich in Kategorien wie Drucksensoren (*pressure*), Temperatursensoren (*temperature*), Beschleunigungsmesser (*accel*), Entfernungsmesser (*proximity*), Feuchtigkeitsmesser (*humidity*), Gesundheit (*health*) und chemische Sensoren (*chemical*) tummeln. Zusätzlich gibt es jeweils eine generalisierte Kategorie für Analog-Digital-



1 **Das über Verzeichnisse hierarchisch strukturierte IIO-Subsystem im Linux-Kernel bedient mittlerweile rund 300 Sensoren und Aktoren.**

und Digital-Analog-Wandler. Damit befindet sich der Code für eine große Komponentensammlung bereits im Linux-Archiv und wartet nur darauf, als Modul generiert, geladen und genutzt zu werden.

Prinzipienreiterei

Für den Zugriff auf das Subsystem und auf Daten sowie für deren Management setzt IIO auf das alte Unix-Prinzip: Alles ist eine Datei. Eine hierarchische Strukturierung über Verzeichnisse garniert das Ganze. Im Sys-Filesystem findet sich das IIO primär unter `/sys/bus/iio/` **1**.

Im Unterverzeichnis `devices/` tauchen nebst anderem alle zur Verfügung stehenden Geräte in Form von weiteren Unterverzeichnissen auf. Das IIO-Framework hält sich an dieser Stelle nicht mit einer ausgefeilten Namensgebung auf, sondern zählt die Geräte einfach durch. Dazu ergänzt es den Namensstamm `iio:device` um eine nachfolgende Nummer. Das erste Gerät wird über das Verzeichnis `iio:device0` repräsentiert, das zweite über `iio:device1` und so weiter.

Für den Zugriff auf das Gerät sammeln sich im zugehörigen Unterverzeichnis – im Folgenden nennen wir es Geräteverzeichnis – eine Fülle von Dateien, deren Namen Auskunft über die dahinter stehende Funktion geben **2**. Die Datei `name` enthält die aussagekräftige Bezeichnung des Geräts, etwa in Form der Typenbezeichnung. In der Datei `dev` finden sich die Major- und Minor-Nummer einer zugehörigen Gerätedatei, die wie üblich im Ordner `/dev/` liegt.

Kanalarbeiter

Da moderne Peripherie oft mehr als einen einzelnen Messwert liefert, führt das IIO-Subsystem das Konzept der Kanäle ein. Ein Beschleunigungsmesser beispielsweise stellt drei Werte bereit, nämlich die Beschleunigung in X-, Y- und Z-Richtung; AD-Wandler verfügen meist nicht nur über einen Analogeingang, sondern häufig über vier oder acht. Jeder Messwert wird durch einen Kanal repräsentiert. Der Beschleunigungsmesser hat also drei Kanäle, der AD-Wandler vier oder acht.

Jeder Kanal wird im Geräteverzeichnis des IIO-Subsystems über mehrere Dateien repräsentiert. Der Kanalname, etwa `accel_x`, bildet den Namensstamm einer Datei, die das Framework um ein Präfix und um ein Suffix erweitert. Das Präfix `in_` steht für einen Eingang, das Präfix `out_` für einen Ausgang. Das Suffix `_raw` steht für den unskalierten Rohwert, wie ihn beispielsweise ein AD-Wandler liefert. Weitere Suffixe sind `_offset` und `_scale`. Mithilfe der in den Dateien gespeicherten Werte lässt sich ein Rohwert in einen gebrauchsfähigen Messwert umrechnen, beispielsweise eine Geschwindigkeit in Meter pro Sekunde. Die simple Formel dazu lautet *value=(raw+offset)*scale*.

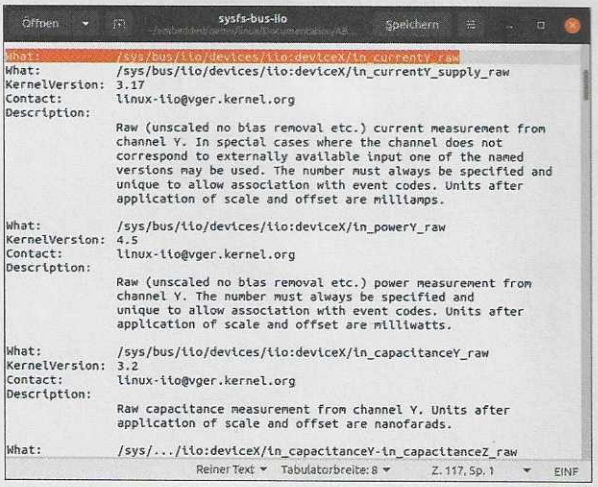
Um einen ersten Eindruck vom IIO-Subsystem zu erhalten, hat Entwickler Cameron ein Dummy-Geräte-Template implementiert, das Ubuntu als fertiges Kernel-Modul mitliefert. Es lässt sich per `modprobe iio_dummy` laden. Damit es ein Dummy-Gerät instanziiert, müssen Sie unterhalb des Ordners `/sys/kernel/config/iio/devices/dummy/` ein Verzeichnis mit dem Namen des betreffenden Geräts anlegen. Rufen Sie also dort `mkdir foo` auf, erscheint unter `/sys/bus/iio/devices/` das neue Verzeichnis `iio:device0`.

Doppelter Punkt

Um auf der Kommandozeile in das neue Verzeichnis zu wechseln, müssen Sie dort den Doppelpunkt durch einen vorangestellten Rückstrich maskieren (`cd iio\:device0`); schließlich dient der Doppelpunkt in der Shell als Metazeichen.

```
root@raspberrypi:/sys/bus/iio/devices/iio:device0# ls
buffer                      in_sampling_frequency     in_voltage-voltage_scale
current_timestamp_clock     in_steps_calibheight      name
dev                         in_steps_en               out_voltage0_raw
events                      in_steps_input            power
in_accel_x_calibbias        in_voltage0_offset        scan_elements
in_accel_x_calibscale       in_voltage0_raw           subsystem
in_accel_x_raw              in_voltage0_scale         trigger
in_activity_running_input   in_voltage1-voltage2_raw  uevent
in_activity_walking_input   in_voltage3-voltage4_raw
```

2 **IIO-Geräte lassen sich über Dateizugriffe konfigurieren und auslesen.**



3 Sensor- und Aktorwerte sind gut dokumentiert.

Das neue Dummy-Foo-Gerät bietet eine Fülle von Kanälen an: Es stellt zum Beispiel eine Beschleunigung in X-Richtung zur Verfügung, eine Spannung, zwei Spannungsdifferenzen, einen Zeitstempel, einen Schrittzähler sowie einen Kanal, der die Aktivitäten des Laufens und Wanderns in Form einer prozentualen Wahrscheinlichkeit signalisiert. Eine Beschreibung dieser und anderer im IIO-Subsystem verwendeten Attributdateien findet sich in der Linux-Kernel-Dokumentation. Diese zentrale Dokumentation beschreibt unter anderem die Einheiten, in denen die Messwerte vorliegen 3.

Auf Shell-Ebene lesen Sie einen Wert per Cat-Kommando ein. So offenbart ein `cat` auf die Datei `name`, dass unser Gerät `foo` heißt. `in_voltage0_raw` liefert die Zahl 73, `in_voltage0_offset` 7 und `in_voltage0_scale` 0.00133. Nach der bereits vorgestellten Formel lässt sich daraus eine Spannung von (73+7) x 0.00133 = 0.10640 berechnen, also 106,4 Millivolt.

Kernel als Macher

Besonders spannend wird das IIO-Framework bei der automatischen Erfassung von Messwerten. Dazu bietet es in separaten Kernel-Modulen implementierte Trigger und Buffer. Per `modprobe iio_trig_sysfs` bekommen Sie Zugriff auf die Trigger, `modprobe iio_buffer` stellt die Buffer zur Verfügung. Von Triggern gibt es diverse Ausprägungen; insbesondere der auf den High-Resolution-Timern basierende `iio_trig_hrtimer` ist sehr nützlich.

Die Idee hinter den Triggern und Buffern besteht darin, dass das IIO-Subsystem Messwerte automatisiert erfasst. Der Anwender definiert die Trigger-Ereignisse, etwa durch Angabe einer Abtastfrequenz, startet die Messung und liest dann zeitlich entkoppelt die Daten über eine Gerätedatei ein. Einziger Wermutstropfen: Nicht alle IIO-Geräte unterstützen Trigger und Buffer.

Sowohl die Kanalauswahl als auch das automatisierte Erfassen und das Einlesen durch eine Applikation lassen sich über das Sys-Filesystem parametrieren. Sind die entsprechenden Kernel-Module geladen und unterstützt das IIO-Gerät das automatische Erfassen, finden sich im Sys-Filesystem-Verzeichnis zum Gerät die Unterordner `trigger/` und `buffer/`. Für die Auswahl und das automatisierte Erfassen von Messwerten ist der Trigger zuständig, das Einlesen durch die Applikation konfigurieren Sie über den Buffer.

An dieser Stelle schon einmal ein Hinweis: Auf einem Ubuntu mit dem Standard-Kernel-Modul für das Dummy-IIO-Device existiert keiner der genannten Ordner. Die Kernel-Macher von Canonical haben diese Funktionalität schlichtweg nicht aktiviert, sodass Sie zum Testen selbst Hand anlegen müssen (siehe Kasten Dummy-IIO-Devices auf dem RasPi).

Abbildung 4 zeigt die Schritte zum Etablieren der automatisierten Erfassung. Als Erstes instanziieren sie einen Trigger. Dazu legen Sie im Config-Filesystem an dedizierter Stelle (`/sys/kernel/config/iio/triggers/hrtimer/`) ein Unterverzeichnis mit dem Namen des neuen Triggers an (Listing 1).

Konfiguration mit Bordmitteln

Das unscheinbare `mkdir` bewirkt, dass im Ordner `/sys/bus/iio/devices/` das neue Unterverzeichnis mit dem Namen `trigger0/` aufploppt. Wie die Geräte

Listing 1: Neuer Trigger

```
$ cd /sys/kernel/config/iio/triggers/hrtimer/
$ sudo mkdir foo_periodic
```

Dummy-IIO-Devices auf dem RasPi

Bedauerlicherweise haben die Ubuntu-Entwickler im Dummy-Treiber die Trigger- und Buffer-Unterstützung ausgelassen, sodass ein Test dieser Funktionen nicht ohne Weiteres gelingt. Vorher müssen Sie die Kernel-Konfiguration anpassen sowie die IIO-Dummy-Module neu generieren und einrichten.

Unter Raspberry Pi OS sieht es nicht viel besser aus. Da sich aber der RasPi besser für Experimente mit dem IIO-Subsystem eignet – schließlich lassen sich hier kinderleicht eigene Sensoren anschließen – experimentieren wir mit der Himbeere weiter. Dazu holen wir uns die Kernel-Quellen, konfigurieren sie und generieren und installieren den Kernel. Die Schritte zur Kernel-Generierung auf dem Raspberry Pi beschreibt eine eigene Dokumentation unter „Local building" bestens.

Bei der Wahl der RasPi-Version gilt es, Vorsicht walten zu lassen – üblicherweise kommt derzeit noch ein 32-Bit-Kernel zum Einsatz. Nach der Grundkonfiguration, also bei einem RasPi 4 durch Aufruf des Befehls `make bcm2711_defconfig`, müssen Sie noch über `make menuconfig` die spezifische IIO-Konfiguration vornehmen. Der Kasten Kernel-Konfigurationsoptionen des IIO-Subsystems führt dazu die einzustellenden Optionen auf, die auch Abbildung 5 zeigt.

Mit der neuen Konfiguration lässt sich anschließend der Kernel generieren. Dazu benötigen Sie reichlich Geduld. Auf einem RasPi 4 dauert der Compilerlauf bei Verwendung sämtlicher Prozessorkerne knapp 100 Minuten. Wer weniger Geduld mitbringt oder – insbesondere mit einer etwas betagteren Version des Raspberry Pi – nicht die ganze Nacht Zeit hat, weicht besser auf den Weg des Cross-Generierens aus.

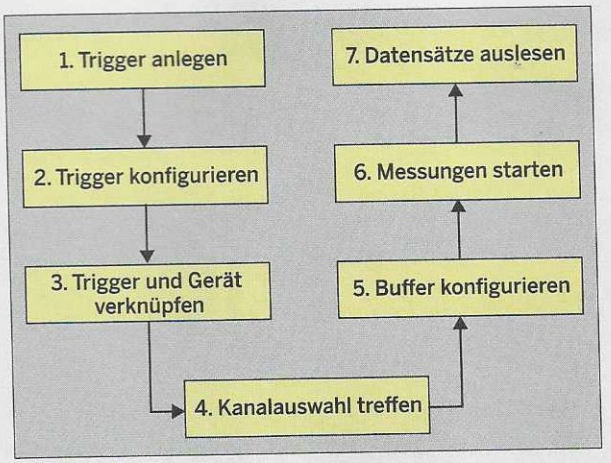
Sämtliche im Artikel verwendeten Befehle fasst Listing 2 noch einmal zusammen. Nach der Installation und einem Neustart steht dem Test der beschriebenen Funktionen mit dem Dummy-Device nichts mehr im Weg.

hlt das IIO-Framework auch die Trigger
rch. Hätten wir gleich zwei Trigger an-
legt, entstünde ein zweites Verzeichnis
it dem Namen `trigger1/`.
Im Subdirectory `trigger0/` finden sich
nter anderem die Dateien `name` und
`ampling_frequency`, wobei `name` wieder
en aussagekräftigen Namen des Trig-
ers enthält. Über `sampling_frequency`
sst sich die Abtastrate einstellen. In der
orgabe beträgt sie 100 Hz, zum Ändern
er Frequenz schreiben Sie einfach per
cho die neue Abtastrate in die Datei. Der
rigger ist damit konfiguriert.

Im dritten Schritt verknüpfen Sie Trigger
nd Gerät. Dazu wechseln Sie in den
Unterordner `trigger/` des Gerätever-
eichnisses. Den Namen des Triggers, im
Beispiel `foo_periodic`, tragen Sie per
cho in die Datei `current_trigger` ein.

Im nächsten, vierten Schritt wählen Sie
die zu überwachenden Kanäle aus. Im
IO-Verzeichnis des Geräts hat sich nach
der Zuordnung des Triggers zum Gerät
schon wieder ein neuer Ordner namens
`scan_elements/` aufgetan. Er enthält di-
verse Dateien, die die Suffixe `_en`, `_index` und `_type` tragen. `_en` steht dabei für *enable*. Schreiben Sie eine 1 in die Datei, wird dieser Wert bei einem Triggerereignis erfasst.

Die Dateien mit dem Suffix `_type` sind etwas komplizierter zu interpretieren. Sie definieren das Format des gespeicherten Messwerts 6. Der später über die Gerätedatei auszulesende Wert `voltage0` etwa verwendet gemäß `in_voltage0_type` (`le:u13/16>>>0`) das Byte-Ablageformat Little Endian (`le`).



4 **Das IIO-Subsystem erfasst automatisiert Messwerte.**

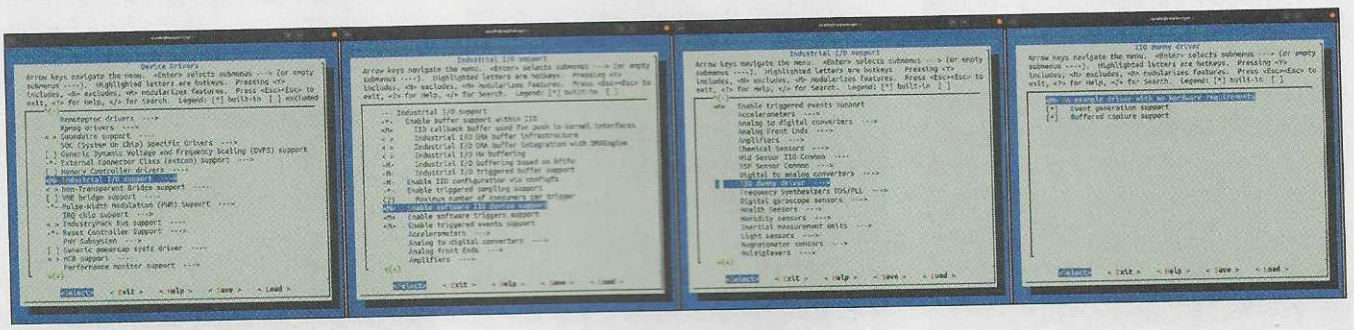
Das Byte-Ablageformat gibt vor, in welcher Reihenfolge die Bytes eines 16-, 32- oder 64-Bit-Worts im Speicher liegen und folglich zu interpretieren sind. Im Beispiel handelt es sich um einen vorzeichenlosen Wert (`u`), wobei die unteren 13 Bits des 16-Bit-Worts relevant sind. Sie gilt es im konkreten Fall noch um `0` Bits nach rechts (`>>>`) zu schieben 7, sprich: Im Beispiel kann das Schieben entfallen.

Nach Auswahl der Kanäle fehlt im Wesentlichen noch die Konfiguration des Zwischenspeichers (Buffer). Tatsächlich lässt sich das IIO-Subsystem so konfigurieren, dass es eine Applikation erst dann aufweckt, wenn eine definierte Anzahl Messwerte erfasst wurden. Dazu spezifizieren Sie zum einen die Größe des Buffers und zum anderen den Füllstand, ab dem die über den Systemcall `read()` schlafende Applikation einen Wakeup-Stupser bekommt. Standardmäßig nimmt der Buffer zwei Datensätze auf. Sobald ein Datensatz vorliegt, wird die Applikation aufgeweckt. Um das System zu entlasten, spezifizieren Sie beispielsweise eine Puffergröße von 20 und lassen die Applikation jeweils nach 15 Datensätzen aufwecken. Sie hat dann 5 Datensätze Zeit, die abgelegten 15 Datensätze auszulesen.

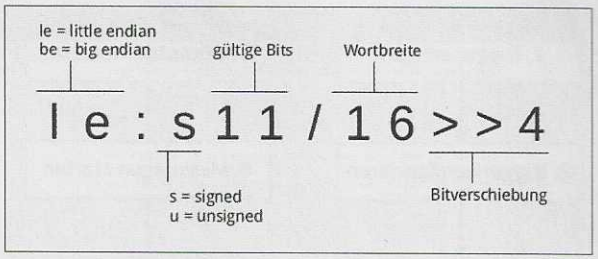
Der Begriff Datensatz verdeutlicht, dass alle ausgewählten Kanäle als ein Datensatz über eine einzelne Gerätedatei ausgelesen werden. Die Reihenfolge der Ab-

Kernel-Konfigurationsoptionen des IIO-Subsystems

Device Drivers | Industrial I/O support | Enable IIO configuration via configfs
Device Drivers | Industrial I/O support | Enable software IIO device support
Device Drivers | Industrial I/O support | Enable software triggers device support
Device Drivers | Industrial I/O support | Enable triggered events device support
Device Drivers | Industrial I/O support | IIO dummy driver | An example driver with no hardware requirements
Device Drivers | Industrial I/O support | IIO dummy driver | Event generation support
Device Drivers | Industrial I/O support | IIO dummy driver | Buffered capture support
Device Drivers | Industrial I/O support | Triggers - standalone | High resolution timer trigger
Device Drivers | Industrial I/O support | Triggers - standalone | Generic interrupt trigger
Device Drivers | Industrial I/O support | Triggers - standalone | SYSFS trigger



5 **Die Kernel-Konfiguration über `make menuconfig` hält für das Industrial-IO-Subsystem eine ganze Reihe von Opionen bereit.**



6 Die Kodierung von Daten bei automatisierter Erfassung.

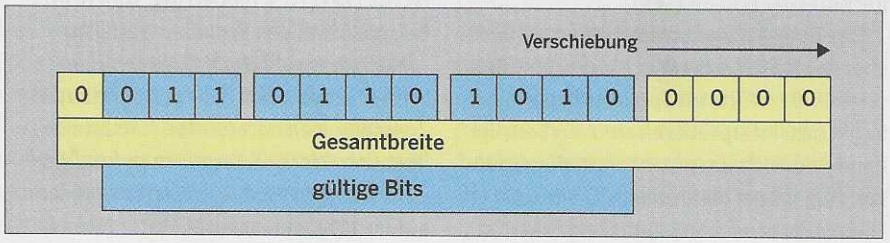
lage erschließt sich über den Inhalt der Dateien mit dem Suffix _index. Die bereits erwähnte Gerätedatei befindet sich erwartungsgemäß im Verzeichnis /dev/ und trägt denselben Namen wie das Verzeichnis im Sys-File-system, also beispielsweise /dev/iio:device0. Die hexadezimale Ausgabe der Datensätze durch Lesen der Gerätedatei zeigt Abbildung 8.

Tools

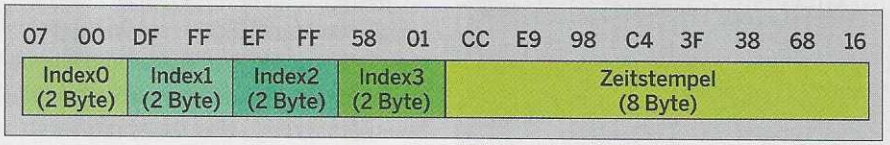
Im professionellen Einsatz greift man wohl häufiger über selbst gestrickte Software auf die IIO-Geräte zu als direkt auf Shell-Ebene. Auch dabei helfen die Kernel-Entwickler, indem sie eine passende Bibliothek samt Dokumentation zur Verfügung stellen (Listing 3, erste Zeile). Des Weiteren gibt es einen Reigen von Tools (letzte Zeile), mit denen Sie bequem und ohne lange Pfadeingabe auf die IIO-Geräte zugreifen. Als Gratisbeigabe steckt auch noch ein kleiner Netzwerk-Daemon für den Remote-Zugriff im Paket.

Fazit und Ausblick

Intuitiv und gut unterstützt, so präsentiert sich das IIO-Subsystem dem Anwender. Ob das auch für den Treiberentwickler gilt, der sein Gerät per Industrial IO zugreifbar machen möchte, klären wir in der nächsten Folge der Kern-Technik. (*jlu*)



7 Beim Zugriff über die Gerätedatei muss die Anwendung die Daten zurechtschieben.



8 Alle ausgewählten Messwerte des Sensors werden in einem Datensatz abgespeichert.

Weitere Infos und interessante Links
www.lm-online.de/qr/44255

Dateien zum Artikel herunterladen unter
www.lm-online.de/dl/44255

Listing 3: IIO-Bibliothek und Tools

```
$ sudo apt install libiio0
libiio0-dev libiio0-doc
$ sudo apt install libiio-utils
```

Listing 2: Test mit dem Dummy-Device

```
# Kernel-Module laden
modprobe iio_dummy
modprobe iio_trig_sysfs
modprobe iio_trig_hrtimer
#
# Dummy Device mit Namen foo
anlegen
cd /sys/kernel/config/iio/
devices/dummy/
mkdir foo
#
# Trigger anlegen und
konfigurieren
cd /sys/kernel/config/iio/
triggers/hrtimer/
mkdir foo_periodic
cd /sys/bus/iio/devices/trigger0
cat name
echo 2 >sampling_frequency
#
# Trigger mit Gerät verknüpfen
# (nur bei selbstgeneriertem
Modul möglich)
cd /sys/bus/iio/devices/
iio\:device0/trigger/
echo "foo_periodic" >current_
trigger
#
# Einzulesende Kanäle auswählen
cd ../scan_elements/
echo 1 >in_timestamp_en
echo 1 >in_voltage0_en
echo 1 >in_accel_x_en
#
# Auslesen konfigurieren
cd ../buffer
echo 10 >length
echo 5 >watermark
#
# Messungen starten
echo 1 >enable
#
# Daten hexadezimal auslesen
hd /dev/iio\:device0
```