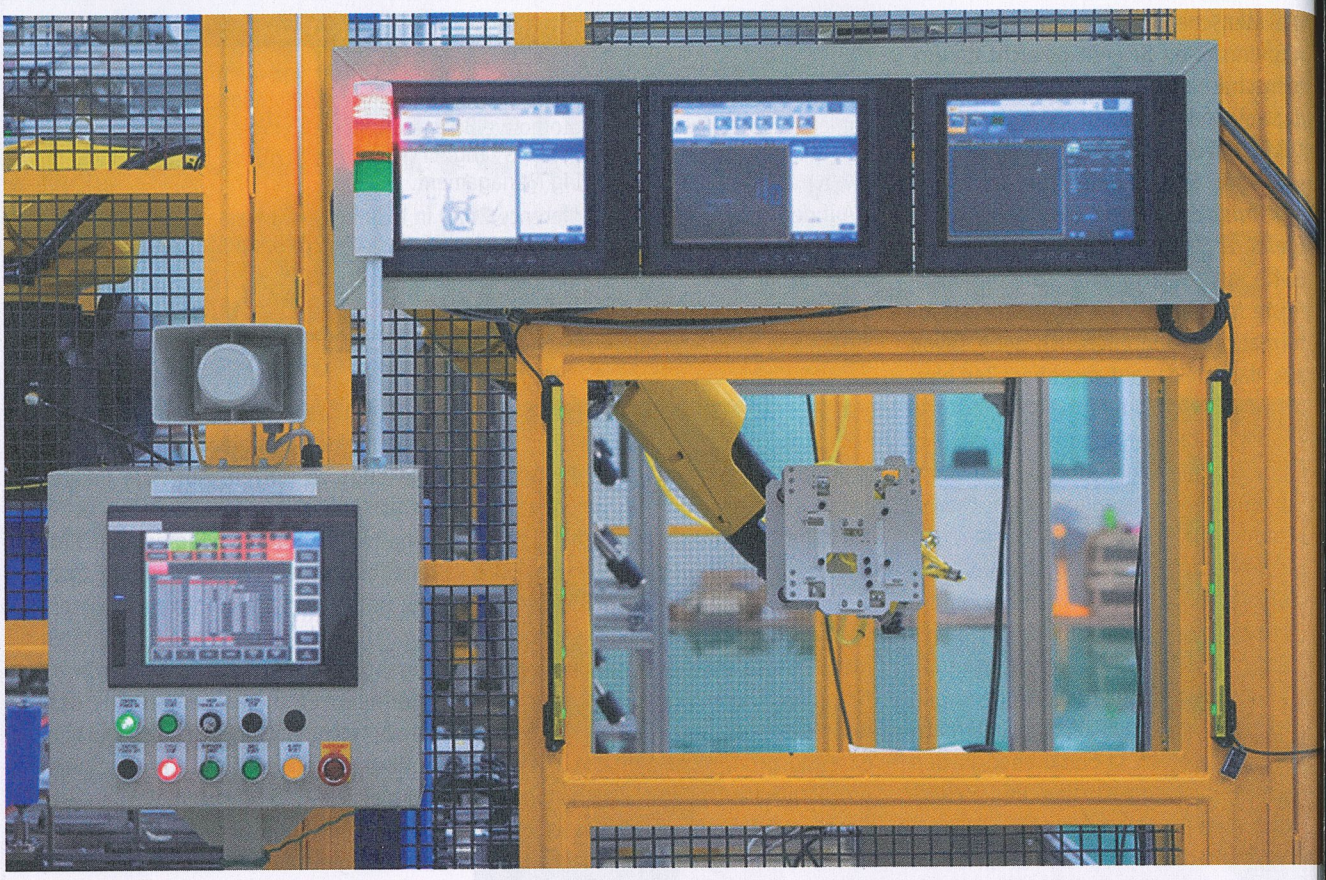

IIO-Framework bindet Sensoren an Linux an

Bei der Entwicklung von Linux-Treibern wurde alles, was den Horizont PC-üblicher Peripheriegeräte übersteigt, stark vernachlässigt. Aber es gibt Licht am Horizont: Das IIO-Framework bindet Sensoren und Aktoren an Linux-Systeme an. Für die Erstellung von IIO-Treibern gibt es im Internet viele Anleitungen. Hier zeigen wir ihre Benutzung im User-Space.

Martin Hofherr
Entwickler für Software und FPGAs, Mixed Mode

Embedded-Linux-Geräte übernehmen immer mehr Aufgaben wie Messen, Steuern, Regeln, die bisher entweder von analoger Elektronik, einem separaten Mikrocontroller oder von einer Speicherprogrammierbaren Steuerung (SPS) erledigt wurden. Sei es um Kosten und Platz zu sparen, oder einfach weil modernere Prozessoren und SoCs so leistungsfähig geworden sind, dass sie neben ihren traditionellen Aufgabenfeldern wie Netzwerkkommunikation und Bereitstellung einer Benutzeroberfläche diese Aufgaben noch zusätzlich übernehmen können. So hat auch der bekannte Einplatinencomputer »Raspberry Pi« seinen Weg auf eine Hutschiene in manchen Schaltschrank gefunden. Auch die stetig steigende Zahl von mobilen Geräten, Hausautomatisierungslösungen und Smart Devices, die auf Linux setzen, sind mit den unterschiedlichsten Sensoren ausgestattet, mit denen sie ihre Umgebung erfassen. Um all diese neuen Aufgaben erledigen zu können, ist eine effiziente Schnittstelle zur Kommunikation mit Sensoren und Aktoren notwendig. Lange Zeit führte diese Peripherie ein Schattendasein und wurde oft mit Treibern angesteuert, die entweder im *Hwmon* oder *Input subsystem* angesiedelt waren, oder gar gleich ein komplett eigenes Interface hatten. Diese Lösungen wurden weder den immer größer werdenden Anforderungen hinsichtlich Datenrate und Steuerung des Abtastzeitpunkts gerecht noch waren sie skalierbar. Deswegen wurde 2009 mit der Entwicklung des Industrial IO-Treiberframeworks (IIO) im Linux-Kerne begonnen.

Funktionsumfang

Wann immer es um die Nutzung von Messdaten geht, werden auch zeitliche Aspekte wichtig. Zum einen ist es wichtig zu wissen, wann ein Datensatz aufgenommen wurde, zum anderen ist es oft notwendig, den Zeitpunkt festlegen zu können, an dem ein Datensatz ermittelt wird. Mit anderen Worten: Ein Framework zur Erfassung von Messdaten sollte erstens die Möglichkeit bieten, zu jedem Sample einen Timestamp zu erhalten und zweitens einen Mechanismus zur Einrichtung von Triggern zur Steuerung des Samplezeitpunkts. Beide Möglichkeiten bietet das IIO-Framework durch *IIO-Trigger*. IIO bietet verschiedene Triggermechanismen: Das Lesen eines oder mehrerer Kanäle kann entweder durch den User-Space ausgelöst, periodisch von einem Timer oder durch einen externen oder internen Interrupt gesteuert werden.

Neben der Steuerung des Samplings verwaltet das IIO-Framework auch den Speicher, in dem die Samples abgelegt werden. IIO kann dabei neben reinen Softwarepuffern auch mit Peripherie umgehen, die eigene Hardware-Ringpufffer enthält.

Die Erfassung von Messdaten kann durch IIO-Framework auch mit Ereignissen verknüpft werden, die an den User-Space weitergegeben werden. Ein Beispiel für ein solches Event ist die abrupte Beschleunigung oder ein Spannungswert, der einen bestimmten Schwellwert überschreitet. IIO bietet hierfür spezielle *Character Devices*, aus denen User-Space-Prozesse lesen können. Der Leseprozess wird hier so lange blockiert, bis ein Event auftritt.

Die angeführten Beispiele belegen, dass das IIO-Framework im Linux-Kernel inzwischen recht mächtig geworden ist. Allerdings zeigt es sich, dass es zwar reichlich Information zur Erstellung von neuen IIO-Treibern im Linux-Kernel gibt, aber Anleitungen, die die Nutzung des Frameworks aus dem User-Space heraus beschreiben, eher spärlich zu finden sind. Deswegen soll im Folgenden eine kurze Einführung gegeben werden.

IIO-Dummy-Treiber

Da die besten Lernerfolge erzielt werden, wenn man das neue Wissen gleich praktisch anwenden kann, wäre es wünschenswert, dass jeder Leser dieser Anleitung ein IIO-fähiges Gerät hat, mit dem er die Funktionen des IIO-Frameworks testen kann. Zum Glück existiert im Linux-Kernel ein IIO-Dummy-Treiber, der die Funktion einer echten Hardware simuliert und so jedem die Möglichkeit bietet, das IIO-Framework zu testen. Für diese Anleitung wurde mit dem Linux-Kernel Version 4.13 gearbeitet (es ist davon auszugehen, dass die Funktion des Treibers in allen 4.x-Versionen des Kernels gleich ist, was der Autor aus nachvollziehbaren Gründen nicht für alle überprüfen konnte). Der folgende Abschnitt führt Schritt für Schritt durch die Benutzung des Dummy-IIO-Treibers und gibt damit einen ersten Einblick in die Funktionen des Linux-IIO-Frameworks.

IIO-Dummy-Treiber im Kernel aktivieren

Um den Treiber einsetzen zu können, muss er im Linux-Kernel aktiviert werden. Neben dem IIO-Dummy-Treiber müssen noch weitere Kernel-Module geladen werden, die der Bereitstellung von Sample Triggern dienen. Die Treiber werden als Module kompiliert, um den Austausch des Kernels zu ersparen. Zuerst sollte allerdings geprüft werden, ob die Module bereits auf dem eigenen System vorhanden sind. Wenn das der Fall ist, kann man das Erstellen der Kernel-Module überspringen. Diese Prüfung geschieht durch fehlerlose Ausführung folgender Kommandos:

```
modprobe iio_dummy
modprobe iio_trig_sysfs
modprobe iio_trig_hrtimer
```

Lassen sie sich ohne Fehler ausführen, so sind die nötigen Kernel-Module bereits auf dem System vorhanden. Andernfalls müssen die Einträge aus *Listing 1* im Konfigurationsmenü des Kernels aktiviert werden, um die Module zu erstellen.

Wie man die Kernel-Module kompiliert und installiert, ist bei den verschiedenen Distributionen jeweils unterschiedlich und würde den Rahmen dieses Artikels sprengen. Jedoch existieren zu verbreiteten Distributionen wie Ubuntu Linux zahlreiche Anleitungen, wie unter der jeweiligen Distribution am besten Kernelmodule erstellt und installiert werden. Diese Anleitungen sind leicht im Internet zu finden.

Erstellung eines eigenen IIO-Dummy-Devices

Wenn alle Kernelmodule geladen sind, kann man mit der Untersuchung des Dummy-IIO-Moduls beginnen. Im ersten Schritt muss ein eigenes IIO-Dummy-Device für die Experimente erstellt werden.

LISTING 1

```
Device Drivers --->
  {M} Industrial I/O Support --->
    <M> Enable software IIO device support
    <M> Enable software triggers support
    IIO dummy driver --->
      <M> An example driver with no hardware requirements
      [*]   Event generation support
      [*]   Buffered capture support
    Triggers - standalone --->
      <M> High Resolution Timer Trigger
      <M> SYSFS Trigger
```

Listing 1: Diese Einträge im Konfigurationsmenü des Kernels müssen aktiviert werden, damit die IIO-Dummy-Treiber als Kernel-Module erstellt werden.

Da es sich bei IIO-Dummy-Devices über reine Softwaredevices ohne tatsächliche Hardware im Hintergrund handelt, kann im Betrieb ein neues Gerät hinzugefügt werden. Dies geschieht über configfs. Um configfs auf dem Gerät zu mounten, falls noch nicht vorhanden, führt man folgendes aus:

```
sudo mkdir -p /config
sudo mount -t configfs none /config
```

configfs ist ein virtuelles Dateisystem, das es erlaubt, aus dem User-Space heraus Kernel-Objekte zu erstellen, zu verwalten und zu löschen. Um ein eigenes IIO-Dummy-Device zu erstellen muss ein neuer Ordner mit dem Namen des Geräts in /config/iio/devices/dummy angelegt werden. Soll das neue Gerät my_dummy heißen, so kann es wie folgt erstellt werden:

```
mkdir /config/iio/devices/dummy/my_dummy
```

Im User-Space wird das neue Device wie bei Linux üblich durch mehrere Dateien und Ordner dargestellt. Die Einträge unter /sys/bus/iio/devices/iio:device0 repräsentieren das IIO-Device und seine Kanäle. Die Datei /dev/iio:device0 erlaubt den Zugriff auf den Buffer und die Events des Devices.

Dateien als Ein-/Ausgänge des IIO-Dummy

In *Sysfs* wird das neue IIO-Device durch den Ordner */sys/bus/iio/devices/iio:device0* dargestellt. Die Dateien in diesem Ordner stellen die verschiedenen Kanäle dar, die vom IIO-Dummy-Device simuliert werden und bieten zusätzlich die Möglichkeit, die Datenerfassung zu konfigurieren. Dateien mit dem Präfix *in_* und dem Suffix *_raw* stellen die einzelnen Eingangskanäle des Devices dar. Mit diesen Dateien kann man direkt den Wert des Eingangskanals lesen. So stellt die Datei *in_accel_x_raw* den aktuellen Wert der Beschleunigung in x-Richtung dar:

```
cat /sys/bus/iio/devices/iio:device0/in_accel_x_raw
34
```

Dieser Wert ist bei dem simulierten Beschleunigungskanal fest auf den Wert 34 codiert. Zusätzlich können den Dateien weitere Informationen zu den einzelnen Kanälen entnommen werden. So stellen die Dateien mit dem Suffix *_scale* den passenden Konvertierungsfaktor zur Verfügung, um die Rohwerte in physikalische Einheiten umzurechnen.

Mit IIO ist es nicht nur möglich Werte einzulesen, sondern auch Ausgänge zu steuern. So stellt zum Beispiel die Datei *out_voltage0_raw* einen D/A-Ausgang dar. Für erste Tests ist es natürlich interessant, die Werte direkt über die Dateien des Interfaces *Sysfs* auszulesen. Für eine reale Anwendung ist es jedoch oft notwendig, den Abtastzeitpunkt genauer zu steuern. Außerdem wird diese Methode bei mehreren Kanälen, die immer wieder gelesen werden müssen, schnell aufwendig. Hier kommen die Trigger ins Spiel. Im nächsten Abschnitt wird deren Funktionsweise an einem Beispiel dargestellt.

Trigger für Eingangssignale

Mit dem bisher Dargestellten ist es möglich, ein durch einen Trigger gesteuertes Sampling durch das IIO-Dummy-Device aufzusetzen. Dazu wird ein *Sysfs*-Trigger verwendet. Falls noch nicht geschehen, muss dazu als erstes das IIO-Sysfs-Triggermodul geladen werden:

```
modprobe iio_trig_sysfs
```

Danach existiert ein neues Device u
/sys/bus/iio/devices/iio_sysfs_trigger.
ser Ordner enthält mehrere Dateien,
zur Konfiguration des IIO-Sysfs-Trig
dienen, jedoch noch kein eigentli
Trigger-Device. Um dieses Trigger-De
zu erstellen, wird in die Datei *add_trig*
geschrieben:

```
echo 0 > /sys/bus/iio/devices/iio_sysfs_trigger/add_trigger
```

Dadurch wurde ein neues Triggerde
trigger0 erzeugt, dessen Konfiguratio
dateien unter */sys/bus/iio/devices/trigg*
liegen. Die Datei *name* enthält die Beze
nung des Triggers, unter der er im IIO-
registriert ist.

```
cat /sys/bus/iio/devices/i
sysfs_trigger/trigger0/name
sysfstrig0
```

Mit diesem Bezeichner ist es möglich,
Trigger an IIO-Devices zu binden, in
man ihn in die Datei *current_trigger*
entsprechenden Devices schreibt. Der
erstellte Trigger wird als Trigger für
Dummy-Device registriert:

```
cat /sys/bus/iio/devices/
trigger0/name > trigger/cu
rent_trigger
```

Im nächsten Schritt muss ausgewählt
den, welche Kanäle durch das Trigg
vent ausgelesen werden sollen. Zur
figuration dienen die Dateien im Or
scan_elements. Im Folgenden soll der
IIO-Dummy-Treiber simulierte Besch
nigungskanal in x-Richtung ausgele
werden.

```
echo 1 > /sys/bus/iio/devices/iio:device0/scan_elements/in_accel_x_raw
```

Im letzten Schritt muss der Puffer
Devices aktiviert werden. Dazu wird
Wert 1 in die Datei */sys/bus/iio/dev*
iio:device0/buffer/enable geschrieben

```
echo 1 > /sys/bus/iio/devices/iio:device0/buffer/enable
```

Jedes Auftreten eines Triggersignales
jetzt einen Wert der x-Achsenbeschle
gung im Puffer des IIO device *my_du*
ab. Die Daten werden über das Chara
Device */dev/iio:device0* bereitgestellt
Lesezugriff auf dieses Device wird block
bis genügend Daten vorhanden sind.
Größe des Puffers wird über die Datei
bus/iio/devices/iio:device0/buffer/leng
eingestellt. Bis zu welchem Füllgrad
lesender Prozess blockiert werden
wird über die Datei */sys/bus/iio/dev*

iio:device0/buffer/watermark festgelegt. Folgende Anweisungen konfigurieren, dass der Puffer zum Beispiel 100 Werte groß sein und lesende Prozesse blockieren soll, bis 64 Werte im Puffer abgelegt worden sind:

```
echo 100 > /sys/bus/iio/devices/iio:device0/buffer/length
echo 64 > /sys/bus/iio/devices/iio:device0/buffer/watermark
```

Um sofort ein Ergebnis zu erhalten, werden die Pufferlänge auf 2 und die Triggerschwelle auf 1 gesetzt. In einem separaten Terminal wird das Char Device ausgelesen:

```
cat /dev/iio:device0 | xxd
```

Wenn */sys/bus/iio/devices/iio_sysfs_trigger/trigger0/trigger_now* auf den Wert 1 gesetzt wird, wird in dem Terminal mit dem lesenden Prozess ein Wert angezeigt. Natürlich ist das Lesen eines einzelnen Kanals mit einem Sysfs-Trigger nicht besonders sinnvoll, da man den Wert auch direkt aus der entsprechenden Datei aus dem Sysfs-Interface lesen könnte. Zum simultanen Auslesen mehrerer Kanäle ist es hingegen sehr nützlich, einen Sysfs-Trigger zu verwenden. Die jeweiligen Kanäle müssen im Ordner *scan_elements* aktiviert werden.

Trigger mit hochauflösendem Timer

Der *High-Resolution-Timer*-Trigger ermöglicht es, periodische Trigger aufzusetzen. Der Trigger existiert als das Kernelmodul *iio_trig_hrtimer*, für das ebenfalls ein eigenes Trigger-Device über Configfs erstellt werden muss. Nach dem Laden des Moduls taucht in der IIO-Hierarchie in Configfs ein neuer Ordner mit dem Namen *triggers/hrtimer* auf. Ein High-Resolution-Timer-Trigger-Device kann durch Anlegen eines Unterordners in diesem Ordner angelegt werden:

```
mkdir /config/iio/triggers/hrtimer/my_hrt_trigger
```

Dadurch wird unter */sys/bus/iio/devices/* ein neuer Trigger angelegt. Da durch das vorherige Experiment mit dem Sysfs-Trigger bereits ein Trigger-Device *trigger0* existiert, erhält der neue Trigger die Bezeichnung *trigger1*. Der Inhalt des neues Trigger-Ordners enthält neben der bereits bekannten Datei *name* noch eine weitere *sampling_frequency*, in die die Abtastfrequenz in Hertz geschrieben werden kann. Steht der Wert 10 in der Datei, entspricht dem ein Triggersignal mit einer Frequenz von 10 Hz. Natürlich ist zu beachten, dass bei realen Geräten die tatsächlich erreichbare Samplingfrequenz noch immer von der Hardware abhängig ist. Der High-Resolution-Timer-Trigger wird genauso genutzt wie der Sysfs-Trigger.

Dieses Framework bietet noch wesentlich mehr Möglichkeiten, als in dieser Übersicht dargestellt werden konnten und die Zahl der unterstützten Geräte wächst stetig. Zusätzlich steht mit *libiio* auch eine Bibliothek zur vereinfachten Nutzung von IIO-Geräten zur Verfügung, die noch zusätzliche Funktionen wie Steuerung über das Netzwerk bieten. (jk)