How to use the IIO user space interface

Stable: 30.03.2021 - 10:20 | Revision: 29.03.2021 - 00.01

How to use the IIO user space interface with a user terminal.

Contents

1 Purpose

This article describes how to use the IIO with a user terminal.

The use cases of the following examples are:

- **analog to digital**
- **digital to analog**
- **quadrature encoder**[1] **monitoring**

Conversions between an STM32 board and an external device:

- Basic reads from ADC (for example by polling) or writes to a DAC are performed using **sysfs**
- More advanced use cases (with timer triggers and buffers) are performed using **sysfs configuration** and **character devices** either directly or with tools
- Simulation of a quadrature encoder device using GPIOs

Some **IIO tools** are used in this article (e.g. lsiio). A list of IIO tools is defined in dedicated articles: IIO Linux kernel tools and libiio tools

2 How to do a simple conversion using the sysfs interface

The IIO sysfs interface can be used to configure devices and do **simple conversions at low rates**.

This is usually referred to as **IIO direct mode** in IIO device drivers.

Documentation/ABI/testing/sysfs-bus-iio[2] is the Linux® kernel documentation that fully describes the IIO standard ABI.

Note: To convert a raw value to standard units, the IIO defines this formula: **Scaled value = (raw + offset) * scale**

2.1 How to do a simple ADC conversion using the sysfs interface

This example shows **how to read** a single data from the ADC, using sysfs.

The ADC is enabled by thedevice tree: ADC DT configuration example

First, look for the IIO device matching the ADC peripheral:

```
$ grep -H "" /sys/bus/iio/devices/*/name | grep adc                    # or use
'lsiio | grep adc'
/sys/bus/iio/devices/iio:device0/name:48003000.adc:adc@0               # Going to
use iio:device0 sysfs, that matches ADC1
/sys/bus/iio/devices/iio:device1/name:48003000.adc:adc@100
```

Then, perform a single conversion on an ADC, and also read the ADC scale and offset:

```
$ cd /sys/bus/iio/devices/iio:device0/
$ cat in_voltage6_raw                                                  # Convert
ADC1 channel 0 (analog-to-digital): get raw value
40603
$ cat in_voltage_scale                                                 # Read
scale
0.044250488
$ cat in_voltage_offset                                                # Read
offset
0
$ awk "BEGIN{printf (\"%d\n\", (40603 + 0) * 0.044250488)}"            # Scaled
value = (raw + offset) * scale
1796                                                                   # Result:
1796 mV
```

2.2 How to do a simple DAC conversion using the sysfs interface

This example shows **how to write** single data to the DAC, using sysfs.

The DAC is enabled by the device tree: DAC DT configuration example

First, look for the IIO device matching the DAC peripheral:

```
$ lsiio | grep dac
Device 003: 40017000.dac:dac@1                                          # Going to
use iio:device3 sysfs, that matches DAC1
Device 004: 40017000.dac:dac@2
```

Then, check the DAC *scale* to compute the *raw* value:

```
$ cd /sys/bus/iio/devices/iio:device3/
$ cat out_voltage1_scale                                                # Read
scale
0.708007812
$ awk "BEGIN{printf (\"%d\n\", 2000 / 0.708007812)}"                    # Example
to convert convert 2000 mV (millivolts)
2824
$ echo 2824 > out_voltage1_raw                                          # Write
raw value to DAC1
$ echo 0 > out_voltage1_powerdown                                       # Enable
DAC1 (out of power-down mode): DAC now converts from digital to analog
                                                                        # User can
now convert new value with 'echo xxxx > out_voltage1_raw'
```

3 Convert one or more channels using triggered buffer mode

Building upon on what is described in the article User space interface, the user should:

- configure and enable the **IIO trigger via sysfs** (*/sys/bus/iio/devices/triggerX*)
- configure and enable the **IIO device via sysfs** (*/sys/bus/iio/devices/iio:deviceX*)
- access configured events and **data from character device** (*/dev/iio:deviceX*)

This is typically the case when using one of the **IIO buffer modes**.

See The Linux driver implementer's API guide - Industrial I/O Buffers for further details.

The STM32 provides several hardware triggers, among which TIM and LPTIM can be used in IIO.

3.1 How to set up a TIM or LPTIM trigger using the sysfs interface

This example shows **how to set up** a TIM or an LPTIM **trigger**, using sysfs.

TIM and/or LPTIM are enabled by device tree: See TIM configured in PWM mode and trigger source example and/or LPTIM DT configuration as PWM and trigger source example

Runtime configuration is performed using the sysfs interface:

```
$ lsiio | grep tim                                                    # Look for
IIO device that matches TIM and/or LPTIM peripheral
Device 010: 44000000.timer:trigger@0
Trigger 000: tim6_trgo
Trigger 001: tim1_trgo
Trigger 002: tim1_trgo2
Trigger 003: tim1_ch1
Trigger 004: tim1_ch2
Trigger 005: tim1_ch3
Trigger 006: tim1_ch4
```

Either the TRGO or the PWM output can be configured, and used as the trigger source for analog conversions.

- To configure the timX_trgo trigger, the **"sampling_frequency"** (Hz) can be set directly:

```
$ cd /sys/bus/iio/devices/trigger0/
$ cat name
tim6_trgo
$ echo 10 > sampling_frequency                                        # Set up
10Hz sampling frequency on tim6_trgo
```

- When using the timX_chY or the **lptimX_outY** trigger, the frequency must be set using the PWM framework. See How to use PWM with sysfs interface.

```
$ cd /sys/bus/platform/devices/44000000.timer:pwm/pwm/pwmchip0
$ echo 0 > export                                                     # Export ti
m1_ch1 PWM
$ echo 100000000 > pwm0/period
$ echo 50000000 > pwm0/duty_cycle
$ echo 1 > pwm0/enable                                                # Enable ti
m1_ch1 with 10Hz frequency and 50% duty cycle
```

3.2 How to perform multiple ADC conversions in triggered buffer mode

This example shows **how to read** multiple data from an ADC, to **scan one or more channels**.

> The ADC is enabled by the device tree: ADC DT configuration example
>
> Conversions are triggered by the **TIM or LPTIM hardware trigger**, See How to set up a TIM or LPTIM trigger using the sysfs interface.

As an example, ADC *in0* and *in1* can be converted in sequence.

- **sysfs** interface overview:

```
$ cd /sys/bus/iio/devices/iio\:device0
$ cat name
48003000.adc:adc@0
$ ls scan_elements
in_voltage0_en  in_voltage0_index  in_voltage0_type  in_voltage1_en  in_voltage1_index
in_voltage1_type
$ ls trigger
current_trigger
$ ls buffer
enable  length  watermark
```

- Example to enable ADC channel 0 and channel 1, and use the tim6_trgo trigger source :

```
$ echo 1 > scan_elements/in_voltage0_en                    # Enable channel 0
$ echo 1 > scan_elements/in_voltage1_en                    # Enable channel 1
$ echo "tim6_trgo" > trigger/current_trigger               # Assign tim6_trgo
trigger to ADC
$ cat trigger/current_trigger
tim6_trgo
$ echo 1 > buffer/enable                                   # Start ADC in buffer mode
```

- **character device** data out:

```
$ hexdump -e '"iio0 :" 8/2 "%04x " "\n"' /dev/iio:device0 &     # Read data from /dev/iio:
device0, display by group of 8, 2 bytes.
iio0 :9f15 0000 9e9f 0000 9f18 0000 9ee4 0000                   # Result: raw data out in
the form of: in0 data | in1 data | in0 data...
...
```

3.3 How to perform multiple ADC conversions in triggered buffer mode using libiio

Prerequisite: please see the similar example: How to perform multiple ADC conversions in triggered buffer mode.

That example uses **iio_readdev**[3] provided by libiio tools.

The example below requests 8 data samples on the ADC configured with:

- channel 0 and channel 1, also referred to as **voltage0** and **voltage1**, enabled
- tim6_trgo, also referred to as **trigger0** to trigger conversions, see How to set up a TIM or LPTIM trigger using the sysfs interface

```
$ iio_readdev -t trigger0 -s 8 -b 8 iio:device0 voltage0 voltage1 | hexdump
0000000 9efe 0000 9ed9 0034 9eff 0000 9ee5 0000
0000010 9edb 0011 9ecc 000b 9eb0 0000 9ed4 0001
```

4 How to use the quadrature encoder with the sysfs interface

Take care this section is no more dedicated to IIO but is related to the new Linux counte r framework coming with STM32 MPU ecosystem-v3

This example shows **how to monitor the position** (count) of a **linear** (or rotary) **encoder**.

It uses quadrature the encoder[1] interface available on the TIM and LPTIM internal peripherals.

4.1 How to set up the TIM quadrature encoder with the sysfs interface

The TIM quadrature encoder is enabled by the device tree: TIM configured as quadrature encoder interface

```
Board $> grep -H "" /sys/bus/counter/devices/*/name   # Look for TIM counter devices
/sys/bus/counter/devices/counter0/name:44000000.timer:counter
Board $> cd /sys/bus/counter/devices/counter0
Board $> cat count0/function_available                # List available modes:
quadrature x2 a
quadrature x2 b
quadrature x4
Board $> echo "quadrature x4" > count0/function       # set quadrature mode
Board $> echo 65535 > count0/ceiling                  # set ceiling value (upper limit
for the counter)
Board $> echo 0 > count0/count                        # reset the counter
```

Runtime configuration is performed using the sysfs interface[4]:

```
Board $> echo 1 > count0/enable                       # enable the counter
```

Once started, the encoder value and direction are available using:

```
Board $> cat count0/count
0
Board $> cat count0/direction
forward
```

4.2 How to set up the LPTIM quadrature encoder with the sysfs interface

The LPTIM quadrature encoder is enabled by the device tree: LPTIM configured as quadrature encoder interface

Runtime configuration is performed using the sysfs interface[4]:

```
Board $> grep -H "" /sys/bus/counter/devices/*/name   # Look for TIM counter devices
/sys/bus/counter/devices/counter0/name:40009000.timer:counter
Board $> cd /sys/bus/counter/devices/counter0
Board $> cat count0/function_available                # List available modes:
increase
quadrature x4
Board $> echo "quadrature x4" > count0/function       # set quadrature mode
Board $> echo 65535 > count0/ceiling                  # set ceiling value (upper limit
for the counter)
Board $> echo 1 > count0/enable                       # enable the counter
```

Once started, the encoder value is available using:

```
Board $> cat count0/count
0
```

4.3 How to use the TIM or LPTIM quadrature encoder with the sysfs interface

This example shows how to monitor the TIM quadrature encoder interface via sysfs (the LPTIM case is very similar):

- In this example, two GPIO lines (PD1, PG3) are externally connected to the TIM (or LPTIM)
- Then libgpiod[5] is used to set and clear the encoder input pins, to '**emulate**' an external quadrature encoder device.

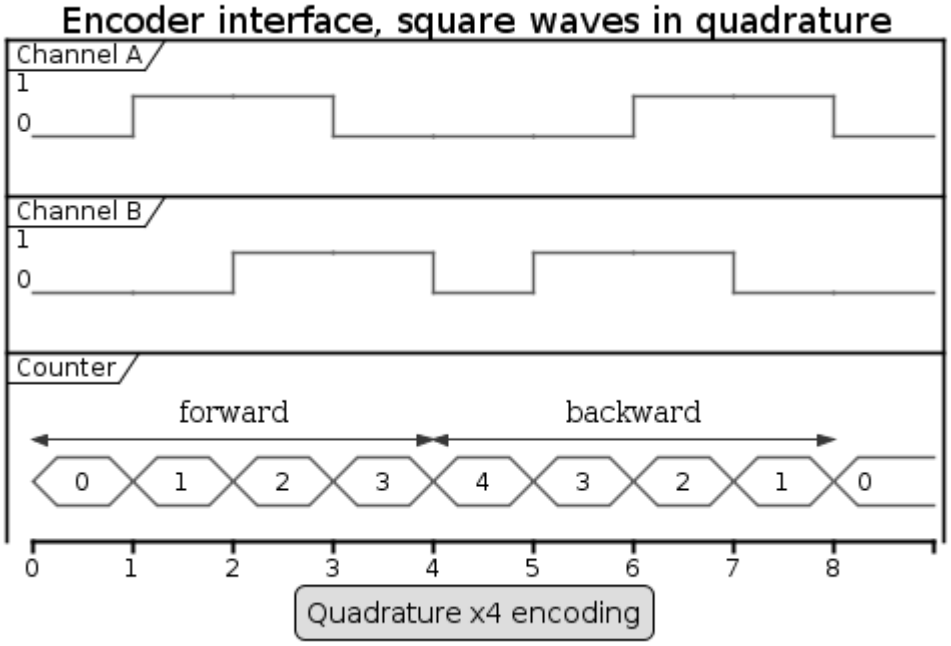
On the STM32MP157X-DKX discovery board, PD1, PG3, TIM1_CH1 and TIM1_CH2 signals are accessible via respectively the D7, D8, D6 and D10 pins of the Arduino Uno connector.

Step-by-step example:

- **Externally connect** and initialise **GPIO pins to TIM** or LPTIM encoder input pins, to '**emulate**' an external quadrature encoder

```
Board $> gpiodetect
...
gpiochip6 [GPIOG] (16 lines)
...
gpiochip3 [GPIOD] (16 lines)
...
Board $> gpioset gpiochip3 1=0     # initialize PD1 to 0 as GPIO, connect it to TIM or LPT
IM channel A input
Board $> gpioset gpiochip6 3=0     # initialize PG3 to 0 as GPIO, connect it to TIM or LPT
IM channel B input
```

- Set up the TIM or LPTIM quadrature encoder with the sysfs interface, see How to set up the TIM quadrature encoder with the sysfs interface or How to set up the LPTIM quadrature encoder with the sysfs interface
- GPIO pins are then set or cleared as follows:




```
Board $> cd /sys/bus/counter/devices/counter0/
Board $> cat count0/count                          # [channel A, channel B] = [0, 0]
0
Board $> gpioset gpiochip3 1=1                     # [channel A, channel B] = [1, 0]
Board $> cat count0/count
1
Board $> gpioset gpiochip6 3=1                     # [channel A, channel B] = [1, 1]
Board $> cat count0/count
2
Board $> gpioset gpiochip3 1=0                     # [channel A, channel B] = [0, 1]
Board $> cat count0/count
3
Board $> gpioset gpiochip6 3=0                     # [channel A, channel B] = [0, 0]
Board $> cat count0/count
4
Board $> cat count0/direction
forward
Board $> gpioset gpiochip6 3=1                     # [channel A, channel B] = [0, 1]
Board $> cat count0/count
3
Board $> cat count0/direction                      # Direction has changed, down-
counting now
backward
...
```

5 References

- 1.01.1 Quadrature encoder, Incremental encoder overview
- Documentation/ABI/testing/sysfs-bus-iio , Linux standard sysfs IIO interface
- https://wiki.analog.com/resources/tools-software/linux-software/libiio/iio_readdev, iio_readdev
- 4.04.1 Documentation/ABI/testing/sysfs-bus-counter , Counter sysfs ABI
- Control GPIO through libgpiod

Industrial I/O Linux® subsystem

Analog-to-digital converter. The process of converting a sampled analog signal to a digital code that represents the amplitude of the original signal sample.

Digital-to-analog converter (Electronic circuit that converts a binary number into a continuously varying value.)

System File System (See https://en.wikipedia.org/wiki/Sysfs for more details)

Application binary interface. (In computer software, an application binary interface (ABI) describes the low-level interface between a computer program and the operating system or another program.)

Linux® is a registered trademark of Linus Torvalds.

low-power timer (STM32 specific)

Pulse Width Modulation

Microprocessor Unit

General-Purpose Input/Output (A realization of open ended transmission between devices on an embedded level. These pins available on a processor can be programmed to be used to either accept input or provide output to external devices depending on user desires and applications requirements.)