Linux Iio Documentation

Release 6.8.0

The kernel development community

Jan 16, 2026

CONTENTS

CHAPTER

ONE

INDUSTRIAL IIO CONFIGFS SUPPORT

1.1 1. Overview

Configfs is a filesystem-based manager of kernel objects. IIO uses some objects that could be easily configured using configfs (e.g.: devices, triggers).

See Documentation/filesystems/configfs.rst for more information about how configfs works.

1.2 2. Usage

In order to use configfs support in IIO we need to select it at compile time via CONFIG_IIO_CONFIGFS config option.

Then, mount the configfs filesystem (usually under /config directory):

```
$ mkdir /config
$ mount -t configfs none /config
```

At this point, all default IIO groups will be created and can be accessed under /config/iio. Next chapters will describe available IIO configuration objects.

1.3 3. Software triggers

One of the IIO default configfs groups is the ”triggers” group. It is automagically accessible when the configfs is mounted and can be found under /config/iio/triggers.

IIO software triggers implementation offers support for creating multiple trigger types. A new trigger type is usually implemented as a separate kernel module following the interface in include/linux/iio/sw_trigger.h:

```
/*
 * drivers/iio/trigger/iio-trig-sample.c
 * sample kernel module implementing a new trigger type
 */
#include <linux/iio/sw_trigger.h>


static struct iio_sw_trigger *iio_trig_sample_probe(const char *name)
{
```

```
	/*
	 * This allocates and registers an IIO trigger plus other
	 * trigger type specific initialization.
	 */
}

static int iio_trig_sample_remove(struct iio_sw_trigger *swt)
{
	/*
	 * This undoes the actions in iio_trig_sample_probe
	 */
}

static const struct iio_sw_trigger_ops iio_trig_sample_ops = {
	.probe		= iio_trig_sample_probe,
	.remove		= iio_trig_sample_remove,
};

static struct iio_sw_trigger_type iio_trig_sample = {
	.name = "trig-sample",
	.owner = THIS_MODULE,
	.ops = &iio_trig_sample_ops,
};

module_iio_sw_trigger_driver(iio_trig_sample);
```

Each trigger type has its own directory under /config/iio/triggers. Loading iio-trig-sample module will create 'trig-sample' trigger type directory /config/iio/triggers/trig-sample.

We support the following interrupt sources (trigger types):

- hrtimer, uses high resolution timers as interrupt source

1.3.1 3.1 Hrtimer triggers creation and destruction

Loading iio-trig-hrtimer module will register hrtimer trigger types allowing users to create hrtimer triggers under /config/iio/triggers/hrtimer.

e.g:

```
$ mkdir /config/iio/triggers/hrtimer/instance1
$ rmdir /config/iio/triggers/hrtimer/instance1
```

Each trigger can have one or more attributes specific to the trigger type.

1.3.2 3.2 ”hrtimer” trigger types attributes

”hrtimer” trigger type doesn't have any configurable attribute from /config dir. It does introduce the sampling_frequency attribute to trigger directory. That attribute sets the polling frequency in Hz, with mHz precision.

CHAPTER

TWO

CIRRUS LOGIC EP93XX ADC DRIVER

2.1 1. Overview

The driver is intended to work on both low-end (EP9301, EP9302) devices with 5-channel ADC and high-end (EP9307, EP9312, EP9315) devices with 10-channel touchscreen/ADC module.

2.2 2. Channel numbering

Numbering scheme for channels 0..4 is defined in EP9301 and EP9302 datasheets. EP9307, EP9312 and EP9315 have 3 channels more (total 8), but the numbering is not defined. So the last three are numbered randomly, let's say.

Assuming ep93xx_adc is IIO device0, you'd find the following entries under /sys/bus/iio/devices/iio:device0/:

sysfs entry	ball/pin name
in_voltage0_raw	YM
in_voltage1_raw	SXP
in_voltage2_raw	SXM
in_voltage3_raw	SYP
in_voltage4_raw	SYM
in_voltage5_raw	XP
in_voltage6_raw	XM
in_voltage7_raw	YP

CHAPTER

THREE

BNO055 DRIVER

3.1 1. Overview

This driver supports Bosch BNO055 IMUs (on both serial and I2C busses).

Accelerometer, magnetometer and gyroscope measures are always provided. When ”fusion_enable” sysfs attribute is set to 1, orientation (both Euler angles and quaternion), linear velocity and gravity vector are also provided, but some sensor settings (e.g. low pass filtering and range) became locked (the IMU firmware controls them).

This driver supports also IIO buffers.

3.2 2. Calibration

The IMU continuously performs an autocalibration procedure if (and only if) operating in fusion mode. The magnetometer autocalibration can however be disabled writing 0 in the sysfs in_magn_calibration_fast_enable attribute.

The driver provides access to autocalibration flags (i.e. you can known if the IMU has successfully autocalibrated) and to the calibration data blob.

The user can save this blob in a firmware file (i.e. in /lib/firmware) that the driver looks for at probe time. If found, then the IMU is initialized with this calibration data. This saves the user from performing the calibration procedure every time (which consist of moving the IMU in various way).

The driver looks for calibration data file using two different names: first a file whose name is suffixed with the IMU unique ID (exposed in sysfs as serial_number) is searched for; this is useful when there is more than one IMU instance. If this file is not found, then a ”generic” calibration file is searched for (which can be used when only one IMU is present, without struggling with fancy names, that change on each device).

Valid calibration file names would be e.g.

bno055-caldata-0e7c26a33541515120204a35342b04ff.dat bno055-caldata.dat

In non-fusion mode the IIO 'offset' attributes provide access to the offsets from calibration data (if any), so that the user can apply them to the accel, angvel and magn IIO attributes. In fusion mode they are not needed (the IMU firmware internally applies those corrections) and they read as zero.