IIO overview

Stable: 21.02.2021 - 10:24 / Revision: 21.02.2021 - 10:22

This article gives information about the Linux®IIO framework.
It explains how to activate the IIO interface and, based on examples, how to use it.

Contents

1 Framework purpose 3
2 System overview 4
2.1 Components description 5
2.2 API description 5
2.2.1 libiio 5
2.2.2 User space interface 6
2.2.3 Kernel space interface 6
3 Configuration 7
3.1 Kernel configuration 7
3.2 Device tree configuration 7
4 How to use the framework 9
4.1 How to use the IIO user space interface 9
4.2 How to use IIO kernel API 9
5 How to trace and debug the framework 10
5.1 How to trace with dynamic debug 10
5.2 How to debug with debugfs 10
6 To go further 11
6.1 How to write a kernel IIO device driver 11
6.2 Trainings documents 11
7 References 12

1 Framework purpose

IIO (Industrial I/O) is a subsystem for Analog to Digital Converters (ADCs), Digital to Analog Converters (DACs) and various types of sensors. It can be used on high speed, high data rates industrial devices. Until recently, it was mostly focused on user-space abstraction. It also includes in-kernel API for other drivers.

The Industrial I/O Linux® subsystem offers a unified framework to communicate (read and write) with drivers covering many different types of embedded sensors and a few actuators. It also offers a standard interface to user space applications manipulating sensors through sysfs and devfs.

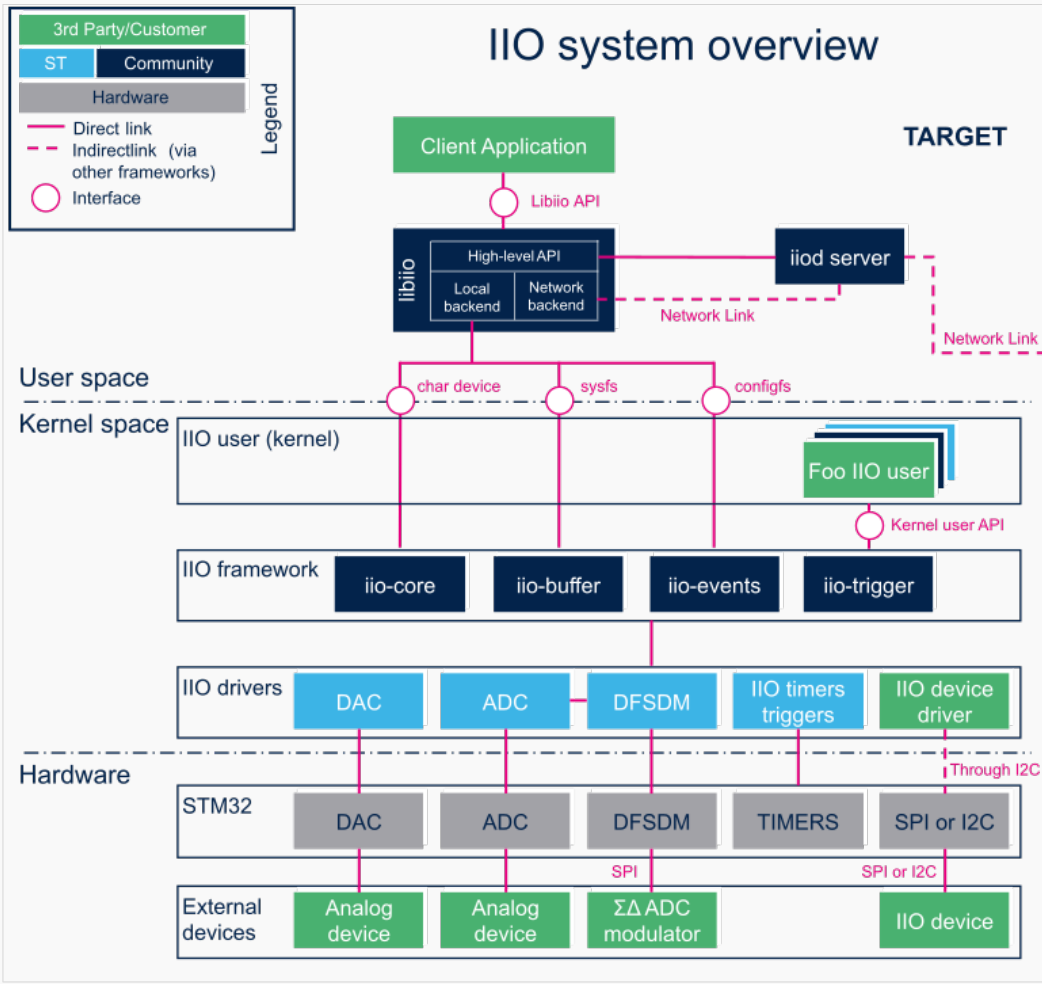
Here are some examples of supported sensor types in IIO:

- ADC / DAC
- accelerometers
- magnetometers
- gyroscopes
- pressure
- humidity
- temperature
- light and proximity

IIO can be used in many different use cases, as mentioned in How to use the framework section:

- Low speed acquisition for slow varying input signal (example: log temperature to a file)
- High speed acquisition using ADC, DFSDM or external devices (example: audio, power meter)
- Read the position of a rotary element using TIM or LPTIM quadrature encoder interface
- Driving an analog source through a DAC
- External devices connected via SPI or I2C.

2 System overview



IIO Implementation architecture

2.1 Components description

From client application to hardware

- **Client Application** (User Space): An application that configures, read or write data samples to/from *IIO device(s)* via libiio.
- **iiod server** (User Space): It is optional. Applications based on libiio can benefit from a remote access via *IIO Daemon* server, to IIO "local" backend through a network link.
- **libiio** (User Space): libiio is a complete library offering an API for developping an application. It's composed of a high-level API, and two backends:

1. The “local” backend, interfacing with the Linux kernel through the IIO API
2. The “network” backend, interfacing with the iiod server through a network link.

- **User Space interface**: It is composed of a standard **char device**, **sysfs**, **configfs** and debugfs (see API description).
- **Kernel Space user**: It can be any kernel space IIO consumer, like STM32 DFSDM audio driver or IIO hwmon driver (See How to use IIO kernel API).
- **Kernel Space interface**: It is composed of a standard API
- **IIO framework** (Kernel Space): It's composed of a core. It manages data buffers, userspace events, triggers. It also handles clients (either in kernel or in User Space).
- **IIO drivers** (Kernel Space): Linux kernel drivers to handle internal peripherals or external devices. They includes an interface that provides controls and data to the user (examples: ADC Linux driver, DAC Linux driver, DFSDM Linux driver, TIM Linux driver, LPTIM Linux driver, IIO device driver connected on SPI or I2C).
- **STM32 peripherals** (Hardware): connected to the external devices through a specific interface (examples: ADC, DAC, DFSDM , TIM, LPTIM, SPI, or I2C)
- **External devices** (Hardware): connected to the STM32 front-end through a specific interface. These can be analog devices (such as accelerometers, Inertial Measurement Units...), a Sigma Delta ADC Modulator (for audio record, energy measurements...), IIO devices on SPI or I2C...

2.2 API description

Depending on needs and location (Kernel Space or User Space), several APIs are available to control an IIO device.

2.2.1 libiio

Libiio provides a user space high-level API for client applications[1]. The library abstracts the low-level details of the hardware, and provides a simple yet complete programming interface that can be used for advanced projects.

It is a wrapper on the *user space interface* (sysfs and char device) provided by the kernel.

2.2.2 User space interface

The IIO framework provides several interfaces:

- **iio device sysfs interface**: It is used to **configure which events and data** should come out of the character device, e.g. */sys/bus /iio/devices/iio:deviceX*.

It can be used to **read** (poll) or **write data** directly **at low rates**.

The IIO sysfs ABI is documented in: *Documentation/ABI/testing/sysfs-bus-iio*[2].

See How to use the IIO user space interface and How to access information in sysfs for further details.

- **character device**[3]: It is optional in IIO. It is used to output **events and sensor data**, e.g. */dev/iio:deviceX*.

It is basically a file from application point of view. Standard file API allows to access it: open(), read(), write(), close()...

See How to use triggered buffer mode and The Linux driver implementer's API guide - Industrial I/O Buffers for further details.

- **configfs**: It allows to configure additional IIO features like *software and hrtimer triggers*.

The IIO configfs interface is documented in: *Documentation/ABI/testing/configfs-iio*[4] and *Documentation/iio/iio_configfs.txt*[5].

Note: STM32 already provides *hardware triggers* (See How to use the IIO timers triggers).

- Debugfs: May provide some debug conveniences (like *direct_reg_access* entry to read/write registers) depending on the IIO device driver in use.

2.2.3 Kernel space interface

Useful kernel API for users:

- **devm_iio_channel_get_all()** or iio_channel_get_all() / iio_channel_release_all(): Used to lookup, get, then release IIO channels.
- **iio_get_channel_type()**: get the type of a channel, such as IIO_VOLTAGE, IIO_TEMP...
- **iio_read_channel_processed()**: read channel processed value, e.g. like in micro-volts for voltage, milli-degree for temperature...
- ...

Available routines can be found in kernel header file: include/linux/iio/consumer.h[6].

3 Configuration

3.1 Kernel configuration

IIO is activated by default in ST deliveries. Nevertheless, if a specific configuration is needed, this section indicates how IIO can be activated/deactivated in the kernel.

Activate IIO in kernel configuration with Linux Menuconfig tool: Menuconfig or how to configure kernel

```
Device Drivers  --->
  <*> Industrial I/O support  --->
    [*]   Enable buffer support within IIO
     < >     IIO callback buffer used for push in-kernel interfaces
     <*>     Industrial I/O HW buffering
     <*>     Industrial I/O buffering based on kfifo
     < >   Enable IIO configuration via configfs
     [*]   Enable triggered sampling support
     (2)     Maximum number of consumers per trigger
     < >   Enable software triggers support
           Accelerometers  --->
           Analog to digital converters  --->
           Amplifiers  --->
           Chemical Sensors  --->
           Hid Sensor IIO Common  ----
           SSP Sensor Common  --->
           Digital to analog converters  --->
           IIO dummy driver  --->
           Frequency Synthesizers DDS/PLL  --->
           Digital gyroscope sensors  --->
           Health Sensors  --->
           Humidity sensors  --->
           Inertial measurement units  --->
           Light sensors  --->
           Magnetometer sensors  --->
           Inclinometer sensors  ----
           Triggers - standalone  --->
           Digital potentiometers  --->
           Pressure sensors  --->
           Lightning sensors  --->
           Proximity sensors  --->
           Temperature sensors  --->
```

IIO supports several types of sensors and devices. User can select from there any driver among the supported devices.

Please refer to ADC Linux driver, DAC Linux driver, DFSDM Linux driver, TIM Linux driver, LPTIM Linux driver articles for each peripheral.

3.2 Device tree configuration

IIO bindings[7] documentation deals with all required or optional IIO generic DT properties.

It also introduces **IIO providers** and **IIO consumers**.
Example with STM32 ADC:

```
&adc {
        adc2: adc@100 {                         /* IIO provider example */
                ...
                #io-channel-cells = <1>;
                st,adc-channels = <12>;         /* channel 12 in use */
        };
};
/ {
        consumer_device {                       /* IIO consumer example */
                io-channels = <&adc2 12>;
                io-channel-names = "example";   /* IIO consumer driver side: devm_iio_chann
el_get(&dev, "example"); */
        };

        iio-hwmon {                             /* iio_hwmon[8] is another consumer example
(See SENSORS_IIO_HWMON in kernel configuration) */
                compatible = "iio-hwmon";       /* See Documentation/devicetree/bindings
/iio/iio-bindings.txt[7]
                io-channels = <&adc2 12>;
        };
};
```

Detailed DT configuration for STM32 internal peripherals:

- ADC device tree configuration
- DAC device tree configuration
- DFSDM device tree configuration
- TIM device tree configuration
- LPTIM device tree configuration

Linux kernel provides many other supported devices[9] in **Documentation/devicetree/bindings/iio** directory.

4 How to use the framework

This section describes how to use the IIO framework from:

- User space interface: Please refer to libiio and IIO Linux kernel tools that run on top of **sysfs** and **character device** (How to use the IIO user space interface)
- Kernel space interface: How to use IIO kernel API

4.1 How to use the IIO user space interface

Please see examples based on the following use cases:

- How to read a data: How to do a simple ADC conversion using the sysfs interface
- How to write a data: How to do a simple DAC conversion using the sysfs interface
- How to setup a trigger source: How to set up a TIM or LPTIM trigger using the sysfs interface
- How to use a trigger source: How to perform multiple ADC conversions in triggered buffer mode
- How to register on an event: How to get ADC analog watchdog events
- How to use quadrature encoder: How to use the quadrature encoder with the sysfs interface

4.2 How to use IIO kernel API

Several in-kernel drivers use kernel IIO API. See HWMON client example for IIO devices, and STM32 DFSDM audio ALSA IIO client:

- iio_hwmon: *drivers/hwmon/iio_hwmon.c*[8]. See also device tree configuration example to read voltage from ADC.

```
$ cat /sys/class/hwmon/hwmon0/in1_input
1809                                    # iio_hwmon calls iio_read_channel_processed():
ADC result is in mV.
```

- stm32-adfsdm: See DFSDM Linux driver and ALSA overview for further details

5 How to trace and debug the framework

5.1 How to trace with dynamic debug

By default there is no kernel log that shows activity on IIO. However the user could enable dynamic debug for the IIO core and the IIO drivers.

```
Board $> dmesg -n8
Board $> echo "file drivers/iio/* +p" > /sys/kernel/debug/dynamic_debug/control
Board $> echo "file drivers/iio/adc/* +p" > /sys/kernel/debug/dynamic_debug/control
Board $> echo "file drivers/iio/dac/* +p" > /sys/kernel/debug/dynamic_debug/control
```

See dynamic debug for more details.

5.2 How to debug with debugfs

IIO proposes an optional debugfs entry to access registers. It is up to the IIO device driver to implement it (e.g. *debugfs_reg_access()*). When it is available:

```
$ cd /sys/kernel/debug/iio/iio:deviceX
```

To read a register from the device:

```
$ echo [register offset] > direct_reg_access
$ cat direct_reg_access
0xhhhh                                          # Register content
```

To write a register:

```
$ echo [register offset] [register value] > direct_reg_access
```

6 To go further

6.1 How to write a kernel IIO device driver

The Linux Kernel community provides all the documents needed to develop an IIO device driver :

- The Linux driver implementer's API guide - Industrial I/O[10]: This guide provides the API provided by kernel IIO core components.
- IIO staging documentation[11], included in the Kernel sources (**drivers/staging/iio/Documentation**).
- Linux Kernel IIO dummy driver example source code[12]: Dummy driver source code, included in the kernel sources (**drivers/iio/dummy/iio_simple_dummy.c**).

6.2 Trainings documents

- IIO a new subsystem[13] : Presentation of Kernel IIO subsystem
- Industrial I/O Subsystem: The Home of Linux Sensors[14]: Why IIO? What is it? Sensor types...
- Software Defined Radio using the Linux Industrial IO framework[15] : User Guide describing how to implement an application by using Linux Industrial IO framework
- Linux Device Drivers, Third Edition[16] : Reference book for linux device drivers development, for IIO see Chapter 3, Char Drivers.

7 References

- libiio High-Level API, libiio API Documentation (Library for interfacing with IIO devices)
- sysfs-bus-iio ABI, Linux standard sysfs IIO interface
- character device interface, *Linux Kernel and Driver Development* training document, see **Character drivers** and **Kernel frameworks for device drivers** chapter
- configfs-iio ABI, Linux standard configfs IIO interface
- iio_configfs interface, Linux standard configfs interface
- include/linux/iio/consumer.h , IIO 'inkern' API
- 7.07.1 Documentation/devicetree/bindings/iio/iio-bindings.txt , Linux Foundation, IIO Generic DT bindings
- 8.08.1 drivers/hwmon/iio_hwmon.c IIO HWMON, consumer driver example (kernel space)
- Kernel DT documentation IIO bindings , Linux Foundation, IIO DT bindings documents included in the Kernel sources
- Industrial I/O, The Linux driver implementer's API guide
- IIO staging documentation , Linux Foundation, IIO documents included in the Kernel sources
- drivers/iio/dummy/iio_simple_dummy.c , Linux Foundation, IIO dummy driver example source code
- IIO a new subsystem, Free Electrons, Presentation of Kernel IIO subsystem
- Industrial I/O Subsystem: The Home of Linux Sensors, Linux Foundation, IIO training
- Software Defined Radio using the Linux Industrial IO framework, Linux Foundation, User Guide describing how to implement an application by using Linux Industrial IO framework
- Linux Device Drivers, Third Edition, Pdf book, Authors Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman

Industrial I/O Linux® subsystem

Application programming interface

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

Device File System (See https://en.wikipedia.org/wiki/Device_file#DEVFS for more details)

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.)

Serial Peripheral Interface

Inter-Integrated Circuit (Bi-directional 2-wire bus standard for efficient inter-IC control.)

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

Digital Filter for Sigma-Delta Modulator

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.)

Secure Secret Provisioning

Secure secrets provisioning

Device Tree

Advanced Linux sound architecture

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

input/output