



## ADC Linux driver



# ADC Linux driver

Stable: 16.01.2020 - 15:02 / Revision: 16.01.2020 - 14:59

## Contents

1 Article purpose .....	2
2 Short description .....	2
3 Configuration .....	2
<b>3.1 Kernel configuration .....</b>	<b>2</b>
<b>3.2 Device tree .....</b>	<b>3</b>
4 How to use .....	3
5 How to trace and debug .....	3
6 Source code location .....	4
7 References .....	4

## 1 Article purpose

This article introduces the Linux<sup>®</sup> driver for the ADC<sup>[1]</sup> internal peripheral:

- Which ADC features are supported by the driver
- How to configure, use and debug the driver
- What is the driver structure, and where the source code can be found.

## 2 Short description

The ADC Linux<sup>®</sup> driver (kernel space) is based on the IIO framework. It supports two modes:

1. **IIO direct mode:** single capture on a channel (using interrupts)
2. **IIO triggered buffer mode:** capture on one or more channels (preferably using DMA).  
It uses the hardware triggers available in IIO. See [TIM Linux driver](#) and [LPTIM Linux driver](#).

## 3 Configuration

### 3.1 Kernel configuration

Activate the ADC<sup>[1]</sup> Linux<sup>®</sup> driver in the kernel configuration using the Linux Menuconfig tool: [Menuconfig](#) or [how to configure kernel](#) (enable both `CONFIG_STM32_ADC_CORE` and `CONFIG_STM32_ADC`).



```
Device Drivers --->
  <*> Industrial I/O support --->
    Analog to digital converters --->
      <*> STMicroelectronics STM32 adc core
      <*> STMicroelectronics STM32 adc
```

## 3.2 Device tree

Refer to the [ADC device tree configuration](#) article when configuring the ADC Linux kernel driver.

## 4 How to use

In "**IIO direct mode**", the conversion result can be read directly from **sysfs** (refer to [How to do a simple ADC conversion using the sysfs interface](#)).

In "**IIO triggered buffer mode**", the configuration must be performed using **sysfs** first. Then, **character device** (/dev/iio:deviceX) is used to read data (refer to [Convert one or more channels using triggered buffer mode](#)).

## 5 How to trace and debug

Refer to [How to trace with dynamic debug](#) for how to **enable the debug logs** in the driver and in the framework.

Refer to [How to debug with debugfs](#) for how to **access the ADC registers**.

The ADC has system wide dependencies towards other key resources:

- **runtime power management** can be disabled, for example it may be forced **on** via *power/control* sysfs entry:

```
Board $> cd /sys/devices/platform/soc/48003000.adc/48003000.adc:adc@0
Board $> cat power/autosuspend_delay_ms
2000
Board $> cat power/control
auto # kernel is allowed to automatically suspend
the ADC device after autosuspend_delay_ms
Board $> echo on > power/control # force the kernel to resume the ADC device
(e.g. keep clocks and regulators enabled)
```



It might be useful to disable runtime power management, in order to dump registers by any means or to check clock and regulator usage (see example below).

- **clock**<sup>[2]</sup> usage can be verified by reading *clk\_summary*.



```
Board $> cat /sys/kernel/debug/clk/clk_summary | grep adc
adc12_k          1      1      0      24000000      0 0
                1      1      0      196607910     0 0
```

- **regulator**<sup>[3]</sup> tree and usage can be verified (e.g. use count, open count or regulator reference voltage) as follows:

```
Board $> cat /sys/kernel/debug/regulator/regulator_summary
regulator          use open bypass voltage current      min      max
-----
v3v3                4   5      0  3300mV      0mA  3300mV  3300mV
vdda                1   2      0  2900mV      0mA  2900mV  2900mV
  40017000.dac      0mV      0mV
  48003000.adc      0mV      0mV
```

- **pinctrl**<sup>[4]</sup> usage can be verified by reading *pinmux-pins*:

```
Board $> cd /sys/kernel/debug/pinctrl/soc\:pin-controller@50002000/
Board $> cat pinmux-pins | grep adc
pin 92 (PF12): device 48003000.adc function analog group PF12 # check pin is
assigned to ADC and is configured as "analog"
```

- **interrupts** can be verified by reading "interrupts":

```
Board $> cat /proc/interrupts
56:      CPU0      CPU1      dummy    0 Edge      48003000.adc:adc@0
```

## 6 Source code location

The ADC source code is composed of:

- **stm32-adc-core** driver to handle common resources such as clock (selection, prescaler), **regulator** used as reference voltage, interrupt and common registers.
- **stm32-adc** driver to handle the resources available for each ADC such as channel configuration and buffer handling.

## 7 References

- 1.01.1 ADC internal peripheral
- Clock overview
- Regulator overview
- Pinctrl overview



## ADC Linux driver

---

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.

Industrial I/O Linux subsystem

Direct Memory Access

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