EVAL-AD4052-ARDZ

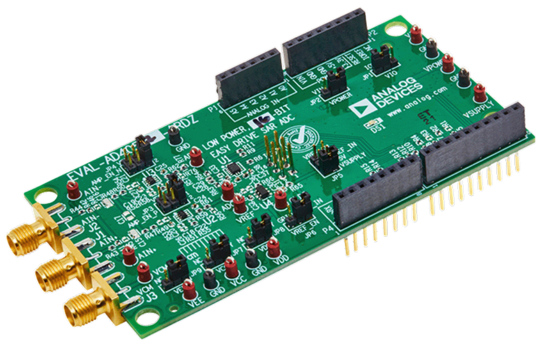
Evaluating the AD4050/AD4052 Compact, Low Power, 12-Bit/16-Bit, 2 MSPS Easy Drive SAR ADCs

Table of contents

EVAL-AD4050/AD4052-ARDZ

The EVAL-AD4050-ARDZ and EVAL-AD4052-ARDZ evaluation boards enable quick and easy evaluation of the performance and features of the AD4050 or the AD4052, respectively. The AD4050 and AD4052 are compact, low power, 12-bit or 16-bit (respectively) Easy Drive successive approximation register (SAR) analog-to-digital converters (ADCs).

The evaluations board are designed to conform to the Arduino Uno Shield mechanical and electrical standard.



Overview

This section provides a general overview on the evaluation board, all supported carriers, firmware and software.

The following carriers are supported, followed by the target firmware:

Carrier	no-OS	Linux
NUCLEO-H503RB	✓	
NUCLEO-H563ZI	✓	
Cora Z7S		✓
DE10-Nano		✓
SDP-K1 [1]	✓	

[1] The SDP-K1 uses the Precision-converters-firmware or the closed-source project instead of a no-OS project.

Features

- Full featured evaluation boards for the AD4050 and AD4052 with a USB power solution.
- Single differential channel and common-mode input available through SMA connectors.
- PC software (ACE plugin/IIO Oscilloscope) for control and data analysis of the time and frequency domains.
- Compatible with other Arduino form factor controller boards.

Evaluation board kit contents

- EVAL-AD4050-ARDZ/EVAL-AD4052-ARDZ evaluation board.

Equipment needed

- Host PC.
- One of the supported carriers.
- Precision signal source with SMA cable.

Hardware

The evaluation board is connected to the Arduino Uno compatible headers of the carrier.

When powering from the carrier, ensure the JP2 jumper is to the +5V position and power on the carrier board.

When powering from an external power supply, ensure the JP2 jumper is set to the VIN position, power on the carrier board, and provide power to VIN (according to the carrier board user guide).

For more information about hardware specifications, see the EVAL-AD4050-ARDZ/ EVAL-AD4052-ARDZ evaluation board pages, in particular, the user guide and design support files.

User Guides

This chapter is aimed to everyone using the evaluation board. It provides instructions on bringing-up the evaluation board with pre-built binaries for supported carries, and how to interact with it.

Evaluating the device

Note

This section describes how to evaluate the device using the open-source drivers, for the SDP-K1's closed-source platform, see UG-2222.

The first step is on evaluating the device is choosing one of the following carriers and firmware, and downloading the pre-built files:

Carrier	no-OS	Linux
NUCLEO-H503RB	download	
NUCLEO-H563ZI	download	
Cora Z7S		download
DE10-Nano		download
SDP-K1	download	

Tip

For Linux, the latest Kuiper relase may contain the provided files already, in this case, you shall use those instead.

Hardware setup

Note

These steps occurs prior to flashing the firmware for no-OS and after flashing it for Linux.

To set up the hardware, complete the following steps:

- Disconnect both the evaluation board and the carrier from all power sources.
- Connect the evaluation board to the carrier using the Arduino Uno compatible headers (there is only one position where all pins are connected).

- Check jumpers and powering on instructions specific for the carrier.
- When powering the evaluation board from the carrier, follow:
  - Set JP2 jumper on the evaluation board to the +5V position. This position connects the evaluation board power management circuitry to the +5 V pin on the Arduino Uno power header
  - Connect the carrier to a PC with a USB cable, the evaluation board DS1 LED should turn on, as others LEDs on the carrier.
- When powering the evaluation board from an external power supply, follow:
  - Set JP2 jumper on the evaluation board to the VIN position. This position connects the evaluation board power management circuitry to the VIN pin on the Arduino Uno power header.
  - Power on the carrier via the external power supply option (in general, via a DC jack), the evaluation board DS1 LED should turn on, as others LEDs on the carrier.
  - Connect the carrier to a PC with a USB cable.

Flashing the firmware

no-OS

These steps are done after the hardware setup, with the board powered on and connected to a PC.

Unpack the downloaded file to a folder and flash one of the example projects (*.elf* files) to the STM32 board using the STM32 Cube IDE or copy over into the USB driver hosted by the STM32 board.

See AD405x no-OS Example Project STM32 for the description of each example.

Precision-converters-firmware

These steps are done after the hardware setup, with the board powered on and connected to a PC.

Unpack the downloaded file to a folder and flash the *.elf* file to the STM32 board using the STM32 Cube IDE or copy over into the USB driver hosted by the STM32 board.

Linux

These steps are done before the hardware setup, with the board powered off.

For both CoraZ7S and DE10-Nano, prepare a SD Card with Kuiper.

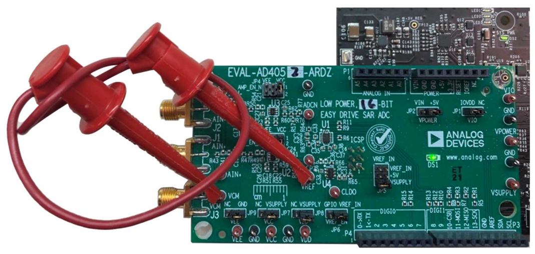
Then, patch the SD Card with the downloaded files:

- CoraZ7S (Zynq)
- DE10-Nano (Cyclone5)

Insert the flashed SD Card on the carrier powered off and follow the hardware setup steps.

Quick start

Connect a precision signal source or signal generator to the analog input Subminiature Version A (SMA) connectors to drive the AD4050/AD4052 inputs into their specified operating ranges.


Biasing the EVAL-AD4052-ARDZ Inputs Without Signal Generator Hardware for Software Validation

IIf no signal generator is available, a jumper cable between the VREF and VCM test points can be used to bias the AD4050/ AD4052 analog inputs to VREF. This is preferred over connecting the amplifier inputs to GND, because the amplifier VEE rails are connected to GND by default.

Evaluation board hardware

Follow UG-2222, Evaluation Board Hardware section.

Evaluation board software

For no-OS basic examples, the evaluation board is interfaced through any serial software such as minicom, picocom and putty (see AD405x no-OS Example Project STM32, Basic Example section).

Using no-OS tinyIIO example, precision-converters-firmare or Linux, the interface is done through Libiio

For no-OS tinyIIO and precision-converters-firmware, execute on the host PC:

```
~$ iio_info -u serial:/dev/ttyACM0,115200,8n1
```

And for Linux, on the carrier Linux shell:

```
~$ iio_info
```

Or from the host, with a Ethernet cable connected to the carrier:

```
~$ iio_info -u ip:192.168.2.1
```

(the IP address depends on your local network and carrier settings).

You can also use IIO Oscilloscope on to obtain waveforms using a GUI.

Developers

This chapter summarizes all source code and related documentation of the evaluation board.

To work with the source code, you should have prior knowledge on software development and HDL design (for the FPGA). We provide pointers to introductory guides, although their scope is limited to topics that particularly relate to our codebase and do not replace the full documentation of the tools used.

Drivers

The drivers source code are available at:

Firmware	Source code	Documentation
no-OS	drivers/adc/ad405x	doc
Linux	drivers/iio/adc/ad4052.c	doc

The no-OS driver is divided into a core driver and a tinyIIO layer to be used with Libiio. The Linux driver is always exposed via the Linux Industrial I/O Subsystem.

To get started with no-OS drivers, checkout no-OS drivers guide, and for Linux drivers the Kernel and devicetrees page.

Projects

The source code for baremetal projects can be found at:

Carrier	Firmware	Project	Documentation
NUCLEO-H503RB	no-OS	projects/ad405x	docs
NUCLEO-H563ZI	no-OS	projects/ad405x	docs
SDP-K1	precision-converters-firmware	projects/ad405x_iio	docs
CoraZ7S	Linux	-	-
DE10-Nano	Linux	-	-

For the no-OS project, the basic examples use only the core driver, while the iio example uses the tinyIIO layer to expose the device to Libiio.

The precision-converters-firmware project also expose the device to Libiio, differentiating only on the target carrier.

Since the Linux driver exposes the device via the Linux Industrial I/O Subsystem, no project is required to leverage the device on Linux targets (CoraZ7S and DE10-Nano).

Follow no-OS projects and precision-converters-firmware projects to comprehend the project structure for each.

Linux devicetrees

For the carriers targeting Linux, the devicetrees are available at:

Carrier	Devicetree
Cora Z7S	zynq-coraz7s-ad4052.dts
DE10-Nano	-

HDL reference design

The DE10-Nano and Cora Z7s use the FPGA to instantiate the controllers to interface the evaluation board.

The source code is available at projects/ad4052_ardz and documented at AD4052-ARDZ HDL project.

Get start with the HDL reference design reading the User Guide.

Software & Bindings

Using any IIO or TinyIIO driver layer, the device can be interacted through Libiio, language bindings on top of libiio and the IIO Oscilloscope GUI.

For the Python language a class abstraction of the device is available at adi/ad405x.py (class doc), with an example at examples/ad4052_example.py

Help and Support

For questions and more information, please visit the EngineerZone Support Community.

ATTENTION
OBSERVE PRECAUTIONS FOR HANDLING
ELECTROSTATIC SENSITIVE DEVICES

All the products described on this page include ESD (electrostatic discharge) sensitive devices. Electrostatic charges as high as 4000V readily accumulate on the human body or test equipment and can discharge without detection. Although the boards feature ESD protection circuitry, permanent damage may occur on devices subjected to high-energy electrostatic discharges. Therefore, proper ESD precautions are recommended to avoid performance degradation or loss of functionality. This includes removing static charge on external equipment, cables, or antennas before connecting to the device.

HDL Design

AD4052-ARDZ HDL project

Overview

The HDL reference design for the ▶ AD4050, ▶ AD4052, ▶ AD4056, and ▶ AD4058 . They are versatile, 16-bit/12-bit, successive approximation register (SAR) analog-to-digital converters (ADCs) that enable low-power, high-density data acquisition solutions without sacrificing precision. These ADCs offer a unique balance of performance and power efficiency, plus innovative features for seamlessly switching between high-resolution and low-power modes tailored to the immediate needs of the system.

The ▶ AD4050/ ▶ AD4052/ ▶ AD4056/ ▶ AD4058 are ideal for battery-powered, compact data acquisition and edge sensing applications.

The ▶ EVAL-AD4050-ARDZ/ ▶ EVAL-AD4052-ARDZ evaluation boards enable quick and easy evaluation of the performance and features of the ▶ AD4050 or the ▶ AD4052, respectively. The AD4050 and AD4052 are compact, low power, 12-bit or 16-bit (respectively) Easy Drive successive approximation register (SAR) analog-to-digital converters (ADCs).

This project has a SPI Engine instance to control and acquire data from the precision ADC. This instance provides support for capturing continuous samples at the maximum sample rate.

Supported boards

- ▶ EVAL-AD4050-ARDZ
- ▶ EVAL-AD4052-ARDZ

Supported devices

- ▶ AD4050
- ▶ AD4052
- ▶ AD4056
- ▶ AD4058

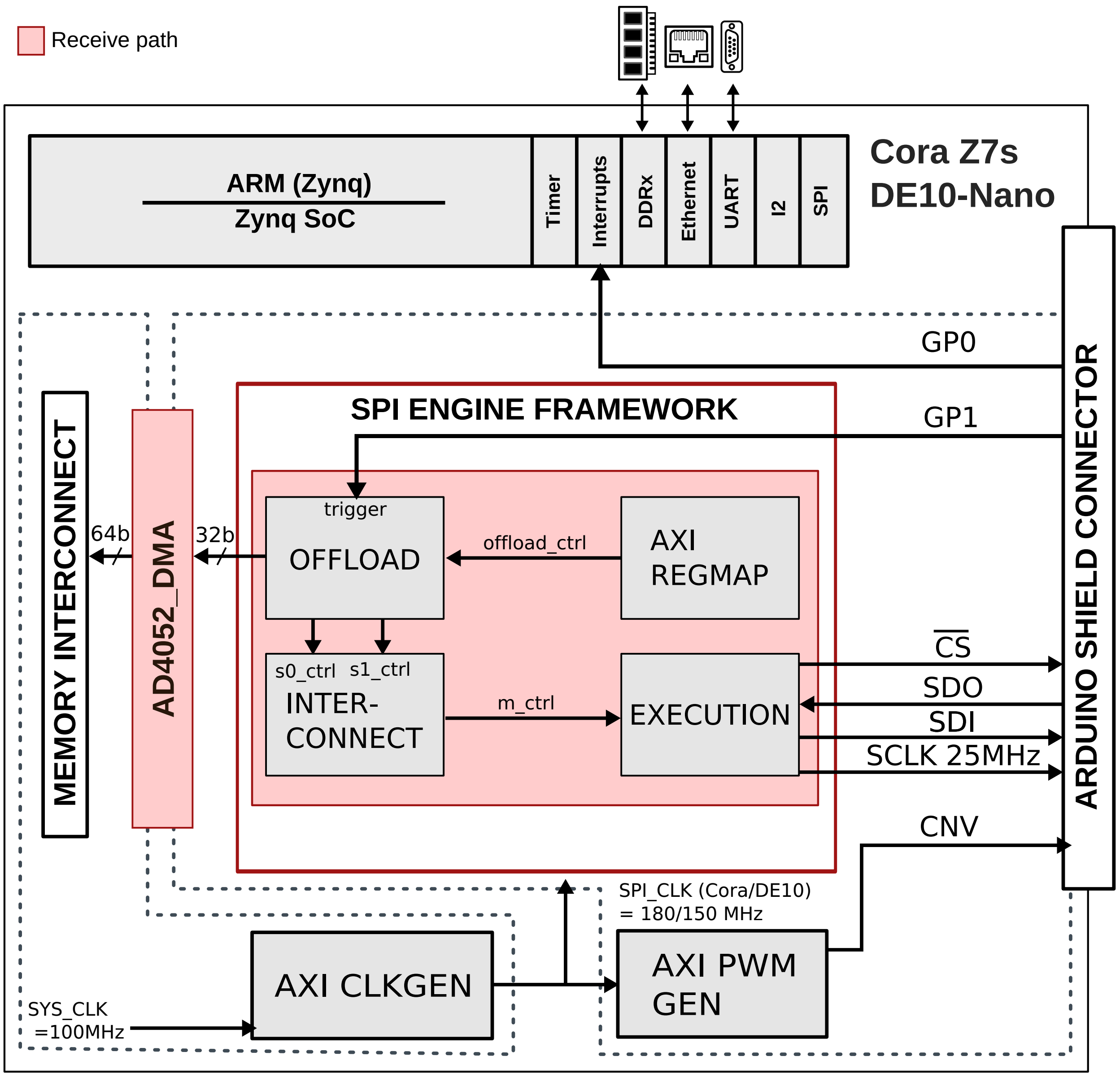
Supported carriers

- Cora Z7-07S Arduino shield connector
- DE10-Nano Arduino shield connector

Block design

The data path and clock domains are depicted in the below diagram:

🟥 Receive path



CPU/Memory interconnects addresses

The addresses are dependent on the architecture of the FPGA, having an offset added to the base address from HDL (see more at HDL Architecture).

Cora Z7S

Instance	Address
spi_adc_axi_regmap	0x44A0_0000
spi_adc_dmac	0x44A3_0000
axi_iic_eeprom	0x44A4_0000
spi_clkgen	0x44A7_0000
adc_trigger_gen	0x44B0_0000

DE10-Nano

Instance	Address
axi_dmac_0	0x0002_0000
axi_spi_engine_0	0x0003_0000
pwm_trigger	0x0004_0000
spi_clk_pll_reconfig	0x0005_0000

I2C connections

Cora Z7s

I2C type	I2C manager instance	Alias	Address	Device Address	I2C subordinate
PS	axi_iic_eeprom	axi_iic_eeprom_io	0x44A4_0000	0x52	EEPROM

DE10-Nano

I2C type	I2C manager instance	Alias	Address	Device Address	I2C subordinate
PS	i2c1	sys_hps_i2c1	—	0x52	—

Device address considering the EEPROM address pins `A0=0`, `A1=1`, `A2=0`.

SPI connections

SPI type	SPI manager instance	SPI subordinate	CS
PL	axi_spi_engine	ad4052	0

GPIOs

The Software GPIO number is calculated as follows:

- Cora Z7S: the offset is 54

GPIO signal	Direction	HDL GPIO EMIO	Software GPIO
	(from FPGA view)		**Zynq-7000**
adc_cnv	OUTPUT	34	88
adc_gp1	INOUT	33	87
adc_gp0	INOUT	32	86

- DE10-Nano: the offset is 32

GPIO signal	Direction	HDL GPIO EMIO	Software GPIO
	(from FPGA view)		**DE10-Nano**
adc_cnv	OUTPUT	34	2

GPIO signal	Direction	HDL GPIO EMIO	Software GPIO
	(from FPGA view)		DE10-Nano
adc_gp1	INPUT	33	1
adc_gp0	INPUT	32	0

Interrupts

Below are the Programmable Logic interrupts used in this project.

Instance name	HDL	Linux Zynq	Actual Zynq
axi_adc_dma	13	57	89
spi_adc_axi_regmap	12	56	88
axi_iic_eeprom	11	55	87

Instance name	HDL	Linux DE10-Nano	Actual DE10-Nano
axi_dmac_0	4	44	76
axi_spi_engine_0	3	43	75

Building the HDL project

The design is built upon ADI's generic HDL reference design framework. ADI distributes the bit/elf files of these projects as part of the ADI Kuiper Linux. If you want to build the sources, ADI makes them available on the HDL repository. To get the source you must clone the HDL repository, and then build the project as follows:

Linux/Cygwin/WSL

```
~$ cd hdl/projects/ad4052_ardz/coraz7s
~/hdl/projects/ad4052_ardz/coraz7s$ make
```

```
~$ cd hdl/projects/ad4052_ardz/de10nano
~/hdl/projects/ad4052_ardz/de10nano$ make
```

A more comprehensive build guide can be found in the Build an HDL project user guide.

Resources

Hardware related

- Product datasheets:
  - AD4050
  - AD4052

HDL related

- AD4052-ARDZ HDL project source code

IP name	Source code link	Documentation link
AXI_PWM_GEN	library/axi_pwm_gen	here
AXI_CLKGEN	library/axi_dmac *	here
AXI_DMAC	library/axi_dmac	here
AXI_HDMI_TX	library/axi_hdmi_tx **	here
AXI_SYSID	library/axi_sysid	here
AXI_SPI_ENGINE	library/spi_engine/axi_spi_engine	here
SPI_ENGINE_EXECUTION	library/spi_engine/ spi_engine_execution	here
SPI_ENGINE_INTERCONNECT	library/spi_engine/ spi_engine_interconnect	here
SPI_ENGINE_OFFLOAD	library/spi_engine/ spi_engine_offload	here
SYSID_ROM	library/sysid_rom	here

Legend

- `*` instantiated only for Cora Z7S
- `**` instantiated only for DE10-Nano

- SPI Engine Framework documentation

Software related

- AD4052 Linux driver ad4052.c

More information

- ADI HDL User guide

Support

Analog Devices, Inc. will provide **limited** online support for anyone using the reference design with ADI components via the EngineerZone FPGA reference designs forum.

For questions regarding the ADI Linux device drivers, device trees, etc. from our Linux GitHub repository, the team will offer support on the EngineerZone Linux software drivers forum.

For questions concerning the ADI No-OS drivers, from our No-OS GitHub repository, the team will offer support on the EngineerZone microcontroller No-OS drivers forum.

It should be noted, that the older the tools' versions and release branches are, the lower the chances to receive support from ADI engineers.

Linux IIO Driver

AD4052

The AD4050, AD4052, AD4056, and AD4058 . are versatile, 16-bit/12-bit, successive approximation register (SAR) analog-to-digital converters (ADCs) that enable low-power, high-density data acquisition solutions without sacrificing precision. These ADCs offer a unique balance of performance and power efficiency, plus innovative features for seamlessly switching between high-resolution and low-power modes tailored to the immediate needs of the system.

The AD4050/ AD4052/ AD4056/ AD4058 are ideal for battery-powered, compact data acquisition and edge sensing applications.

The EVAL-AD4050-ARDZ/ EVAL-AD4052-ARDZ evaluation boards enable quick and easy evaluation of the performance and features of the AD4050 or the AD4052, respectively. The AD4050 and AD4052 are compact, low power, 12-bit or 16-bit (respectively) Easy Drive successive approximation register (SAR) analog-to-digital converters (ADCs).

Supported Devices

- AD4050
- AD4052
- AD4056
- AD4058

Evaluation Boards

- EVAL-AD4050-ARDZ
- EVAL-AD4052-ARDZ

Source Code

Status

Source	Mainlined?
git	[No]

Files

Function	File
driver	ad4052.c
devicetree	ad4052.dts
devicetree bindings doc	ad4052.yaml

Devicetree

External clock

Reference clock used for sampling timing. May be the same clock as the SPI Controller driver.

```
clocks {
    ref_clk: ext-clk {
        #clock-cells = <0x0>;
        compatible = "fixed-clock";
        clock-frequency = <150000000>;
        clock-output-names = "ref_clk";
    };
};
```

Sampling trigger

This PWM generator is used to start the sampling procedure.

```
adc_trigger: axi-pwm-gen@ {
    compatible = "adi,axi-pwmgen";
    reg = <0x44b00000 0x1000>;
    label = "ad4052_cnv";
    #pwm-cells = <2>;
    clocks = <&cnv_ext_clk>;
};
```

ADC node

The AD4052 is a SPI-compatible ADC so it should be under a SPI controller.

Important

The `spi-max-frequency` must be a multiple of a 25MHz clock and the `dmas` handle should use a 64-bit wide data bus.

```
&spi{
    ad4052: ad4052@0 {
        compatible = "adi,ad4052";
        reg = <0>;
        spi-max-frequency = <25000000>;
        clocks = <&spi_clk>;
        dmas = <&rx_dma 0>;
        dma-names = "rx";
        pwm-names = "cnv";
        pwms = <&adc_trigger 0 0>,
        cnv-gpios = <&gpio0 88 GPIO_ACTIVE_HIGH>;
        gp1-gpios = <&gpio0 87 GPIO_ACTIVE_HIGH>;
        gp0-gpios = <&gpio0 86 GPIO_ACTIVE_HIGH>;
    };
};
```

Usage

Kernel configuration

This device depends on a PWM based trigger used to start the sampling procedure. The first step is to enable the support for AXI_PWMGEN.

```
Symbol: PWM_AXI_PWMGEN [=y]
Type  : tristate
Prompt: Analog Devices AXI PWM generator
   Location:
     -> Device Drivers
       -> Pulse-Width Modulation (PWM) Support (PWM [=y])
   Defined at drivers/pwm/Kconfig:78
   Depends on: PWM [=y] && HAS_IOMEM [=y]
   Selected by [y]:
   - KERNEL_ALL_ADI_DRIVERS [=y]
```

Another required component is the SPI controller. The AD4052 has a specific set of SPI timing requirements that are supported by the SPI Engine IP.

```
Symbol: SPI_AXI_SPI_ENGINE [=y]
Type  : tristate
Prompt: Analog Devices AXI SPI Engine controller
  Location:
    -> Device Drivers
      -> SPI support (SPI [=y])
  Defined at drivers/spi/Kconfig:112
  Depends on: SPI [=y] && SPI_MASTER [=y] && HAS_IOMEM [=y]
  Selected by [y]:
  - KERNEL_ALL_ADI_DRIVERS [=y]
```

And finally, enable support for the AD4630 device family.

```
Symbol: AD4052 [=y]
Type  : tristate
Prompt: Analog Device AD4052 ADC Driver
  Location:
    -> Device Drivers
      -> Industrial I/O support (IIO [=y])
        -> Analog to digital converters
  Defined at drivers/iio/adc/Kconfig:47
  Depends on: IIO [=y] && SPI [=y] && PWM [=y] && GPIOLIB [=y]
  Selects: IIO_BUFFER [=y] && IIO_BUFFER_DMA [=y] && IIO_BUFFER_DMAENGINE [=y]
```

Driver testing

This device can be found under */sys/bus/iio/devices/*

One way to check if the device and driver are present is using **iio_info**:

```
$ iio_info
    iio:device0: ad4052 (buffer capable)
            1 channels found:
                    voltage0:  (input, index: 0, format: le:s16/32>>0)
                    2 channel-specific attributes found:
                            attr  0: raw value: 12167
                            attr  1: sampling_frequency value: 1000000
            1 device-specific attributes found:
                            attr  0: waiting_for_supplier value: 0
            3 buffer-specific attributes found:
                            attr  0: data_available value: 0
                            attr  1: direction value: in
                            attr  2: length_align_bytes value: 8
            1 debug attributes found:
                            debug attr  0: direct_reg_access value: 0x10
            No trigger on this device
```

You can go to the device folder using:

```
$ cd $(grep -rw /sys/bus/iio/devices/*/name -e "ad4052" -l | xargs dirname)
```

Note

As specified in the devicetree section, the device supports multiple functional modes. This example describes the steps for the 24-bit burst averaging mode.

In the folder there are several files that can set specific device attributes:

```
/sys/bus/iio/devices/iio:device0$ ls -l
```

```
total 0
drwxr-xr-x 2 root root    0 Nov 16 21:27 buffer
drwxr-xr-x 2 root root    0 Nov 16 21:27 buffer0
-r--r--r-- 1 root root 4096 Nov 16 21:27 dev
drwxr-xr-x 2 root root    0 Nov 16 21:27 events
-rw-r--r-- 1 root root 4096 Nov 16 21:27 in_voltage_oversampling_ratio
-rw-r--r-- 1 root root 4096 Nov 16 21:27 in_voltage_raw
-rw-r--r-- 1 root root 4096 Nov 16 21:27 in_voltage_sampling_frequency
-r--r--r-- 1 root root 4096 Nov 16 21:27 in_voltage_sampling_frequency_available
-r--r--r-- 1 root root 4096 Nov 16 21:27 name
lrwxrwxrwx 1 root root    0 Nov 16 21:27 of_node -> ../../../../../../../../firmware/devicetree/base/fpga-axi@0/spi
drwxr-xr-x 2 root root    0 Nov 16 21:27 power
drwxr-xr-x 2 root root    0 Nov 16 21:27 scan_elements
lrwxrwxrwx 1 root root    0 Nov 16 21:27 subsystem -> ../../../../../../../../bus/iio
-rw-r--r-- 1 root root 4096 Nov 16 21:27 uevent
-r--r--r-- 1 root root 4096 Nov 16 21:27 waiting_for_supplier
```

Oversampling

The AD4052 device family has support for a burst averaging mode that's exposed as the oversampling attribute. Writing value 0 or 1 returns the device to sample mode.

Display current oversampling value:

```
$ cat in_voltage_oversampling_ratio
64
```

Change the sample averaging count:

```
$ echo 2048 > in_voltage_oversampling_ratio
$ cat in_voltage_oversampling_ratio
2048
```

Important

Please note that averaging on low sampling rates will timeout if the default buffer wait time is not modified. For single shot readings, it will also timeout in 1 second.

Sample rate for burst and monitor mode

Leveraging the device internal clock, the sample rate for burst and monitor mode can be configured.

Display all available sample rates:

```
$ cd /sys/bus/iio/devices/iio\:device0 ; pwd
/sys/bus/iio/devices/iio:device0
$ cat in_voltage_sampling_frequency_available
2000000 1000000 300000 100000 33300 10000 3000 500 333 250 200 166 140 125 111
```

Set the desired sample rate:

```
$ echo 1000000 > in_voltage_sampling_frequency
$ cat in_voltage_sampling_frequency
1000000
```

Sample rate for buffer reading with PWM trigger

Set the desired period of the PWM trigger to set the sampling frequency of bufferred reading.

```
$ cd /sys/bus/iio/devices/iio\:device0/buffer ; pwd
/sys/bus/iio/devices/iio\:device0/buffer
$ echo 1000000 > sampling_frequency
$ cat sampling_frequency
1000000
```

Auto suspend

The device enters sleep mode (low power) when no acquisition is being made. Display the current runtime status:

```
$ cat /sys/bus/spi/devices/spi0.0/power/runtime_status
suspend
```

Caution

The power management methods are coupled to the spi device and not the iio device.

There is a timeout of 1 second before the power management puts the device in sleep mode. This is to avoid putting the device to sleep when sampling single shot readings without a buffer.

Monitor mode

The driver yield a IIO Event for the device threshold interrupt in device monitor mode. The event is triggered for either direction (rising or falling the max/min threshold values). Configure the device threshold and hysteresis values:

```
$ cd /sys/bus/iio/devices/iio\:device0 ; pwd
/sys/bus/iio/devices/iio:device0
$ echo 1000 > events/thresh_rising_value
$ echo -1000 > events/thresh_falling_value
$ echo 125 > events/thresh_rising_hysteresis
$ echo 125 > events/thresh_falling_hysteresis
```

Enable monitor mode:

```
$ echo 1 > events/thresh_either_en
```

At monitor mode, since the device is contiguously sampling, the device is active:

```
$ cat /sys/bus/spi/devices/spi0.0/power/runtime_status
active
```

Threshold events will increment the interrupt count:

```
$ cat /proc/interrupts
           CPU0
 ...
 46:          3 GIC-0  90 Edge      ad4052
```

The driver puts the device in monitor mode after every device access, until disabling the threshold event with:

```
$ echo 0 > events/thresh_either_en
```

Caution

The device is locked from any other access until the monitor mode is disabled.

The user is responsible to catching IIO Event and clearing the device status register.

```
$ echo 0 > events/thresh_either_en
$ echo 0x41 > direct_reg_access
$ cat direct_reg_access
0x88
$ echo 0x41 0x02 > direct_reg_access
$ cat direct_reg_access
0x80
```

Data acquisition

The data acquisition is performed using a buffer system:

```
$ cd /sys/bus/iio/devices/iio\:device0 ; pwd
/sys/bus/iio/devices/iio:device0
$ ls buffer
data_available  enable  length  length_align_bytes  watermark
$ ls scan_elements
in_voltage0_en  in_voltage0_index  in_voltage0_type
```

Every buffer implementation features a set of files:

- `buffer`: length Get/set the number of sample sets that may be held by the buffer.
- `buffer`: enable Enables/disables the buffer. This file should be written last, after length and selection of scan elements
- `scan_elements`: in_voltage0_en enable/disables the channel output so the data won't be buffered for that specific channel.

Enable and read 400 samples:

```
$ echo 1 > scan_elements/in_voltage0_en
$ echo 400 > buffer/length
$ echo 1 > buffer/enable
$ hexdump -n 400 /dev/iio\:device0
0000000 0eaf 0000 0ead 0000 0eb0 0000 0ead 0000
0000010 0ead 0000 0ea7 0000 0e9d 0000 0e9c 0000
0000020 0e97 0000 0e93 0000 0e84 0000 0e80 0000
0000030 0e7f 0000 0e7e 0000 0e7c 0000 0e7d 0000
0000040 0e7c 0000 0e79 0000 0e75 0000 0e6c 0000
0000050 0e64 0000 0e63 0000 0e5f 0000 0e51 0000
0000060 0e4c 0000 0e4f 0000 0e4d 0000 0e4b 0000
0000070 0e4d 0000 0e4b 0000 0e4c 0000 0e42 0000
0000080 0e39 0000 0e36 0000 0e33 0000 0e31 0000
0000090 0e24 0000 0e1c 0000 0e19 0000 0e1a 0000
00000a0 0e1d 0000 0e1a 0000 0e16 0000 0e19 0000
00000b0 0e15 0000 0e0c 0000 0e07 0000 0e03 0000
00000c0 0e03 0000 0e06 0000 0e05 0000 0e04 0000
00000d0 0e03 0000 0dff 0000 0df0 0000 0dea 0000
00000e0 0ded 0000 0de5 0000 0ddd 0000 0dd9 0000
00000f0 0dd7 0000 0dd3 0000 0dd2 0000 0dd1 0000
0000100 0dcd 0000 0dd1 0000 0dc8 0000 0dc1 0000
0000110 0dbe 0000 0dbb 0000 0dbc 0000 0daa 0000
0000120 0da5 0000 0da3 0000 0da2 0000 0d9f 0000
0000130 0da0 0000 0da1 0000 0da2 0000 0d9d 0000
0000140 0d91 0000 0d8e 0000 0d89 0000 0d8a 0000
0000150 0d7a 0000 0d74 0000 0d6f 0000 0d70 0000
0000160 0d6b 0000 0d6f 0000 0d70 0000 0d73 0000
0000170 0d6c 0000 0d63 0000 0d5f 0000 0d59 0000
0000180 0d58 0000 0d4e 0000 0d46 0000 0d42 0000
0000190
$ echo 0 > buffer/enable
```

Debug mode

You can write and read the ADC registers using debugfs. Here we will read and write the scratchpad register:

First go to the device debug folder:

```
~$ cd /sys/kernel/debug/iio/iio\:device0 ; pwd
/sys/kernel/debug/iio/iio:device0
```

Read the 0xA register:

```
$ echo 0xA > direct_reg_access
$ cat direct_reg_access
0x0
```

Write and verify the value:

```
$ echo 0xA 0x5F > direct_reg_access
$ cat direct_reg_access
0x5F
```

no-OS Driver&Project

AD405x no-OS Example Project STM32

Supported Evaluation Boards

EVAL-AD4050-ARDZ EVAL-AD4052-ARDZ

Overview

Applications

No-OS Build Setup

Please see: https://wiki.analog.com/resources/no-os/build

For the `.ioc`, copy the desired carrier target file from the *carrier* folder to the project directory.

No-OS Supported Examples

Basic example

This example prints sample data out to the uart.

Here is an example on how see the sample data of the basic example:

```
minicom -b 115200 -D /dev/ttyACM0 -C ./serial.dat
make run
cat ./serial.dat | grep  ADC | cut -d ' ' -f 2 > ./plot.dat
echo "set terminal svg; set output './o.svg';plot './plot.dat' with lines" | gnuplot
```

IIO example

This project is a IIOD demo for AD405X device. The project launches a IIOD server on the board so that the user may connect to it via an IIO client.

At a host, use IIO utilities, for example:

```
iio_info -u serial:/dev/ttyACM0,115200,8n1
```

Using IIO-Oscilloscope, the user can configure the ADC and view the measured data on a plot.

If you are not familiar with ADI IIO Application, please take a look at: IIO No-OS

If you are not familiar with ADI IIO-Oscilloscope Client, please take a look at: IIO Oscilloscope

The no-OS IIO Application together with the No-OS IIO AD405X driver take care of all the back-end logic needed to setup the IIO server.

The read buffer is used for storing the burst data which shall be retrieved by any LibIIO client.

No-OS Supported Platforms

STM32 Platform

Used hardware:

- STM32 NUCELO-H503RB
- STM32 NUCELO-H563ZI

- ST debugger

Prerequisites

- export STM32CUBEMX=<path/to/stm32cubemx>
- export STM32CUBEIDE=<path/to/stm32cubeide>
- firmware for the target platform (download on stm32cubemx beforehand)
- nucleo-*.ioc file in the project folder

Build and flash

- Copy the target carrier configuration from the carrier folder e.g. `cp carrier/nucleo-h503rb.ioc .`
- make run

Project Options

- Use basic interactive example that prints samples to uart:

  ./Makefile

  BASIC_EXAMPLE = y

  IIO_EXAMPLE = n
- Specify the AD405X part and instance ID in use:

  ./Makefile:

  DEV_TYPE = AD4052
- Specify the carrier in use:

  ./Makefile:

  CARRIER = NUCLEO_H503RB

AD405x no-OS Driver

Supported Devices

AD4052 AD4058

Overview

The AD4052/AD4058 are low power, compact 16-bit successive approximation register (SAR) analog-to-digital converters (ADC) designed for battery powered precision measurement and monitoring applications. The AD4052/AD4058 feature set supports event-driven programming for dynamic tradeoff between system power and precision. Using a patented, power efficient window comparator, the AD4052 autonomously monitors signals while the host sleeps. The programmable averaging filter enables on-demand high resolution measurements for optimizing precision for the power consumed.

The AD4052 includes AFE control signals to minimize the complexity of host timers. The control signals automate the power cycling of the AFE relative to ADC sampling to reduce system power while minimizing settling error artifacts. The Easy Drive analog inputs enables compact and low power signal conditioning circuitry by reducing the dependence on high-speed ADC driver amplifiers. The small 3.4 pF sampling capacitors result in low dynamic and average input current, broadening compatibility with low power amplifiers or direct sensor interfacing. The AD4052 wide common mode input range supports both differential and single-ended input signals.

The AD4052 family features a 4-wire SPI with a dedicated CNV input. Cyclic redundancy check (CRC) is available on all interface read and write operations and internal memory to ensure reliable device configuration and operation.

AD405X Device Configuration

Driver Initialization

SPI

In order to be able to use the device, you will have to provide the support for the communication protocol (SPI) as well as 3 external GPIOs for the CNV pin and two general-purpose input/output pins (GP0 and GP1).

The first API to be called is **ad405x_init**. Make sure that it returns 0, which means that the driver was initialized correctly.

GPIO Configuration

The device has two general purpose output pins, GP0 and GP1. These pins can be configured as threshold events, data ready, among other status signals. In the driver files the **ad405x_set_gp_mode** can be found and used to choose the specific signal for the GPIOs.

If GP0 is set as DRDY, the device will assert the pin on the CONV assertion, and the ADC driver will wait the pin to desert before issuing the ADC data acquisition. During initialization, GP1 is used to track the DEV_RDY state, and no further behaviour is defined at the driver level.

Channel Configuration

Channel data can be fetched with **ad405x_get_adc**.

The channel data format can be set using **ad405x_set_data_format**

Channel operation mode can also be configured using **ad405x_set_operating_mode**.

Soft Reset

The device can be soft reset by using **ad405x_soft_reset**.

AD405X Driver Initialization Example

SPI

```
struct ad405x_dev *ad405x;
const struct no_os_spi_init_param ad405x_spi_ip = {
     .device_id = SPI_DEVICE_ID,
     .max_speed_hz = 100000,
     .mode = NO_OS_SPI_MODE_0,
     .chip_select = GPIO_CS_PIN,
     .bit_order = NO_OS_SPI_BIT_ORDER_MSB_FIRST,
     .platform_ops = SPI_OPS,
     .extra = &ad405x_spi_extra_ip
};
const struct no_os_gpio_init_param gpio_cnv_param = {
     .port = GPIO_CNV_PORT,
     .number = GPIO_CNV_PIN,
     .platform_ops = GPIO_OPS,
     .extra = &gpio_init
};
const struct no_os_gpio_init_param gpio_gpio0_param = {
     .port = GPIO_GPIO0_PORT,
     .number = GPIO_GPIO0_PIN,
     .platform_ops = GPIO_OPS,
     .extra = &gpio_init
};
const struct no_os_gpio_init_param gpio_gpio1_param = {
     .port = GPIO_GPIO1_PORT,
     .number = GPIO_GPIO1_PIN,
     .platform_ops = GPIO_OPS,
     .extra = &gpio_init
};
struct ad405x_init_param ad405x_ip = {
     .comm_type = AD405X_COMM,
     .comm_init.spi_init = ad405x_spi_ip,
     .dev_type = AD405X_DEV_TYPE,
     .gpio_cnv = &gpio_cnv_param,
     .gpio_gpio0 = &gpio_gpio0_param,
     .gpio_gpio1 = &gpio_gpio1_param
};
ret = ad405x_init(&ad405x, &ad405x_ip);
if (ret)
     goto error;
```