ADC - Analog data acquisition

SD003 | Posted on April 2, 2021 | Updated on July 24, 2025

Benoît STEINMANN
Software Team Leader
imperix

Table of Contents

The ADC block is used to retrieve the measurements from the analog inputs of an imperix controller.

This help documentation deals with the **software part** of the analog data acquisition. To set-up the hardware part, please refer to the following documents:

- Analog front-end configuration on B-Box RCP (PN105)
- B-Box RCP datasheet (PDF)
- B-Board PRO datasheet (PDF)

Note on sampling: The sampling instant is the same for all the analog inputs, across all devices (when in a multi-devices configuration). The sampling is, by design, linked to the same clock generator as the interrupt and is configured using the CONFIG block. See CONFIG – Interrupt configuration to learn more.

Simulink block

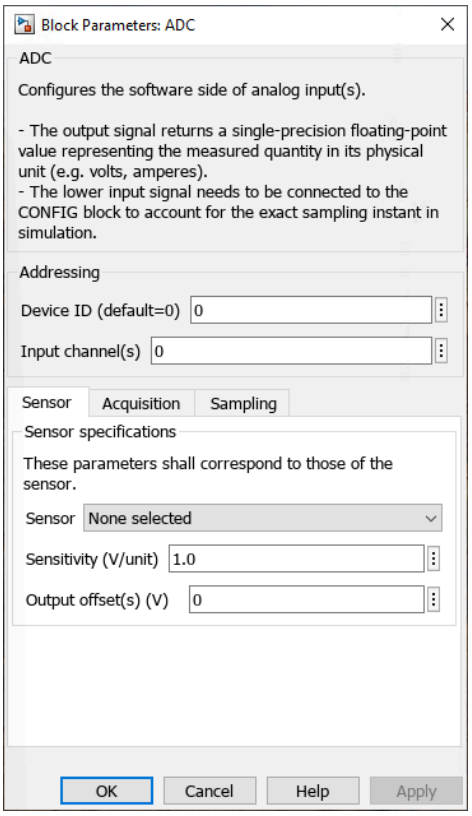
Signal specification

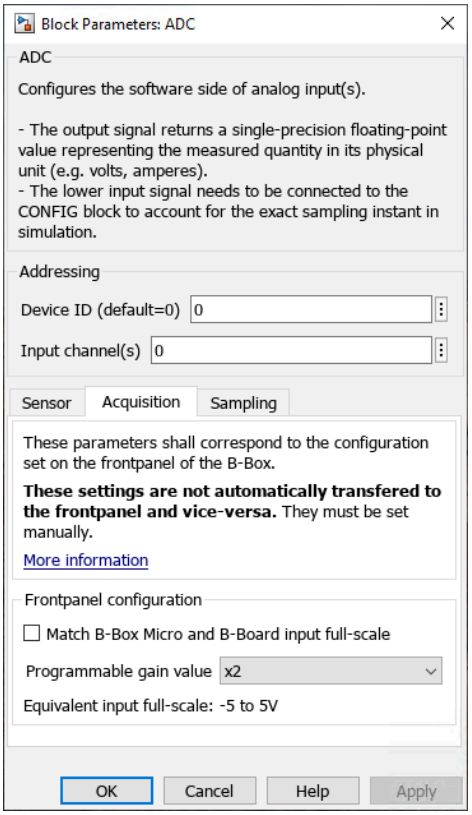
- The output signal returns a single-precision floating-point value representing the measured quantity in its physical unit (e.g. Volts, Amperes).
- The `sim` input signal is used in simulation and documented in Simulation essentials with Simulink (PN135).
- The > input signal needs to be connected to the CONFIG block in order to account for the exact sampling instant in simulation.

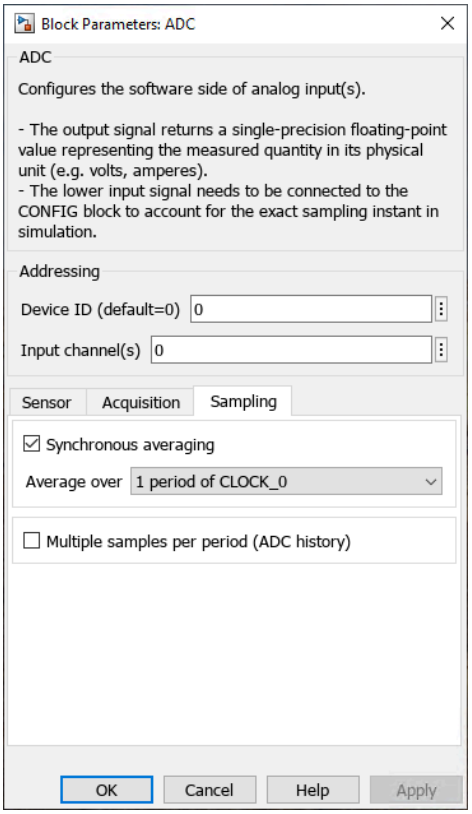
Parameters

- **Device ID** selects which B-Box/B-Board to address when used in a multi-device configuration.
- **Input channel(s)** (vectorizable) selects which physical input channel to read from (e.g. 0 to 15 on B-Box RCP).
- Sensor
  - **Sensitivity** (vectorizable) is the sensor sensitivity in Volts per measured unit (e.g. V/V for a voltage sensor and V/A for a current sensor).
  - **Output offset(s)** (vectorizable) compensates for the sensor offset. It is expressed in Volts at the output of the sensor.
- Acquisition

    - **Programmable gain value** must match the configuration set on the frontpanel of the B-Box.
    - **Match B-Box Micro and B-Board input full-scale** must be checked if the device is a B-Box Micro B-Board PRO. It forces the programmable analog gain to x2, to account for the +/- 5V input full-scale (as opposed to +/- 10V on the B-Box RCP).
- Sampling
    - **Synchronous averaging** configures the block to output the average ADC value over 1 or 2 periods of *CLOCK_0*.
      See Synchronous averaging (PN124) for more details.
    - **Multiple samples per period (ADC history)** configures the block to output a vector of the N last values, as documented in Oversampling (PN154). This option is mutually exclusive with the synchronous averaging.



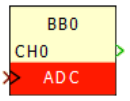




PLECS block

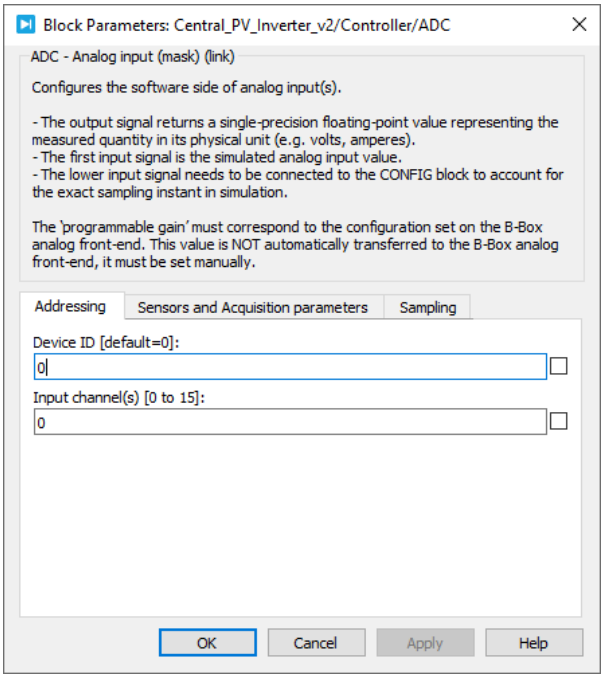
Signal specification

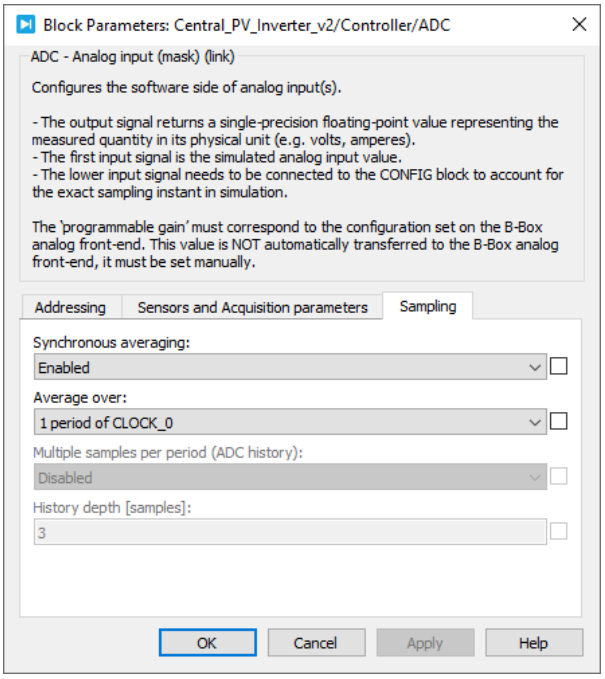
- The output signal returns a single-precision floating-point value representing the measured quantity in its physical unit (e.g. Volts, Amperes).
- The target inport (only visible at the atomic subsystem level) is used in simulation and documented in Simulation essentials with PLECS (PN137).
- The > input signal needs to be connected to the CONFIG block to account for the exact sampling instant in simulation.

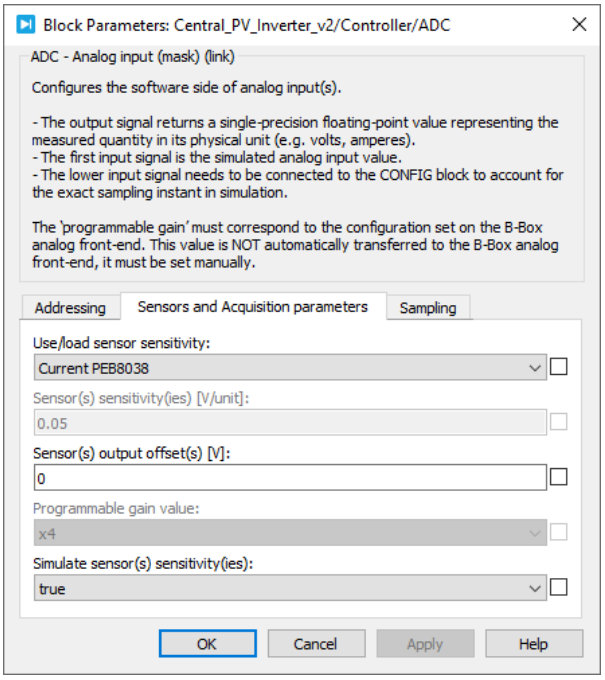


Parameters

- Addressing
  - **Device ID** selects which B-Box/B-Board to address when used in a multi-device configuration.
  - **Input channel** (vectorizable) selects which physical input channel to read from (e.g. 0 to 15 on B-Box RCP).
- Sensor and Acquisition parameters
  - **Use/load sensor parameters** loads the parameters of an imperix sensor.
  - **Sensor sensitivity** (vectorizable) is the sensor sensitivity in Volts per measured unit (e.g. V/V for a voltage sensor and V/A for a current sensor).
  - **Sensor output offset** (vectorizable) compensates for the sensor offset. It is expressed in Volts at the output of the sensor.
  - **Programmable gain value** must match the configuration set on the frontpanel of the B-Box RCP. If the device is a B-Box Micro or a B-Board PRO, it must be set to x2, to account for the +/- 5V input full-scale (as opposed to +/- 10V on the B-Box RCP).
  - **Simulate sensor(s) sensitivity(ies)**: when *true*, the ADC block expects its input to be the value of the sensor's output (typ. ±10V). When *false*, it expects the physical measure value.
- Sampling
  - **Synchronous averaging** configures the block to output the average ADC value over 1 or 2 periods of *CLOCK_0*.
    See Synchronous averaging (PN124) for more details.
  - **Multiple samples per period (ADC history)** configures the block to output a vector of the N last values, as documented in Oversampling (PN154). This option is mutually exclusive with the synchronous averaging.


Block Parameters: Central_PV_Inverter_v2/Controller/ADC
ADC - Analog input (mask) (link)
Configures the software side of analog input(s).
- The output signal returns a single-precision floating-point value representing the measured quantity in its physical unit (e.g. volts, amperes).
- The first input signal is the simulated analog input value.
- The lower input signal needs to be connected to the CONFIG block to account for the exact sampling instant in simulation.
The 'programmable gain' must correspond to the configuration set on the B-Box analog front-end. This value is NOT automatically transferred to the B-Box analog front-end, it must be set manually.
Addressing
Sensors and Acquisition parameters
Sampling
Device ID [default=0]:
0
Input channel(s) [0 to 15]:
0
OK
Cancel
Apply
Help


Block Parameters: Central_PV_Inverter_v2/Controller/ADC
ADC - Analog input (mask) (link)
Configures the software side of analog input(s).
- The output signal returns a single-precision floating-point value representing the measured quantity in its physical unit (e.g. volts, amperes).
- The first input signal is the simulated analog input value.
- The lower input signal needs to be connected to the CONFIG block to account for the exact sampling instant in simulation.
The 'programmable gain' must correspond to the configuration set on the B-Box analog front-end. This value is NOT automatically transferred to the B-Box analog front-end, it must be set manually.
Addressing
Sensors and Acquisition parameters
Sampling
Synchronous averaging:
Enabled
Average over:
1 period of CLOCK_0
Multiple samples per period (ADC history):
Disabled
History depth [samples]:
3
OK
Cancel
Apply
Help


Block Parameters: Central_PV_Inverter_v2/Controller/ADC
ADC - Analog input (mask) (link)
Configures the software side of analog input(s).
- The output signal returns a single-precision floating-point value representing the measured quantity in its physical unit (e.g. volts, amperes).
- The first input signal is the simulated analog input value.
- The lower input signal needs to be connected to the CONFIG block to account for the exact sampling instant in simulation.
The 'programmable gain' must correspond to the configuration set on the B-Box analog front-end. This value is NOT automatically transferred to the B-Box analog front-end, it must be set manually.
Addressing
Sensors and Acquisition parameters
Sampling
Use/load sensor sensitivity:
Current PEB8038
Sensor(s) sensitivity(ies) [V/unit]:
0.05
Sensor(s) output offset(s) [V]:
0
Programmable gain value:
x4
Simulate sensor(s) sensitivity(ies):
true
OK
Cancel
Apply
Help

C++ functions

The sampling is by design linked to the same clock generator as the interrupt, thus the sampling frequency and phase are configured using the `ConfigureMainInterrupt` function as explained in the related note: Interrupt configuration.

Standard functions

`Adc_ConfigureInput` **— Configure an ADC channel**

```
void Adc_ConfigureInput(unsigned int input, float gain, float offset, unsigned int device=0);
```
Code language: C++ (cpp)

Configures the desired ADC channel with the desired `gain` and `offset` that is applied on the 16-bit digitally-converted value to obtain a floating-point quantity such as $\text{valuefloat} = \texttt{gain} \times \text{value16bit} + \texttt{offset}$.

It has to be called in `UserInit()` for each channel that the user wants to use.

Parameters

- `input`: the analog input channel number (e.g. 0 to 15 on B-Box RCP)
- `gain`: the gain applied on the 16-bit digitally-converted value
- `offset`: the offset applied to the returned value
- `device`: the id of the addressed device (optional, used in multi-device configuration only)

The **gain** is not equal to the **sensor sensitivity**. The example of use section below shows how to compute the `gain` and `offset` parameters.

`Adc_GetValue` **— Read the ADC value**

```
float Adc_GetValue(unsigned int input, unsigned int device=0);
```
Code language: C++ (cpp)

Retrieves the value of an ADC channel, with the gain and offset already applied.

It has to be called in the interrupt.

Parameters

- `input`: the analog input channel number (0 to 15)
- `device`: the id of the addressed device (optional, used in multi-device configuration only)

`Adc_EnableSynchronousAveraging` **— Enables synchronous averaging on an ADC channel**

```
void Adc_EnableSynchronousAveraging(unsigned int input, unsigned int device);
```
Code language: C++ (cpp)

When enabled, Adc_GetValue returns the average ADC value over a period of CLOCK_0.

It has to be called in the interrupt.

Parameters

- `input`: the analog input channel number (0 to 15)
- `device`: the id of the addressed device (optional, used in multi-device configuration only)

Advanced functions for oversampling

These functions are used to perform oversampling, read Oversampling (PN154) to learn more.

`Adc_AddSamplingEvent` **— Configure a new sampling instant**

```
void Adc_SetUserOversampling(int oversampling);
```
Code language: C++ (cpp)

Sets the number of ADC samples to acquire at each CLOCK_0 period.

The samples are evenly distributed and can be retrieved using and `Adc_GetHistory`.

It has to be called in `UserInit()`.

Parameters

- `oversampling`: number of samples to acquire at each CLOCK_0 period

`Adc_ConfigureHistory` **— Configure the ADC history**

```
void Adc_ConfigureHistory(unsigned int input, unsigned int depth, unsigned int device=0);Code language: C++ (cpp)
```

Enables the possibility to have access to the N last ADC samples using `Adc_GetHistory()`.

It has to be called in `UserInit()`.

Parameters

- `input`: the analog input channel number
- `depth`: the number of samples available
- `device`: the id of the addressed device (optional, used in multi-device configuration only)

`Adc_GetHistory` **— Read the historical samples**

```
float Adc_GetHistory(unsigned int input, unsigned int n, unsigned int device=0);Code language: C++ (cpp)
```

Gets the **N**th historical sample of a given ADC channel.

Using `Adc_GetHistory(n=0)` is equivalent to using `Adc_GetValue()`.

It has to be called during the control interrupt.

Parameters

- `input`: the analog input channel number (0 to 15)
- `n`: the historical index of the sample to read (0 is the most recent one)
- `device`: the id of the addressed device (optional, used in multi-device configuration only)

Example of use

For the sake of this example, we recompute the necessary ADC gain. However, the gains for imperix standard sensors and power modules are already defined in `sensors.h`

This example considers the current sensor of a PEB8024 module. Its sensitivity is $S = 50.0\,[\mathrm{mV/A}]$. As recommended in the datasheet, the chosen front-end gain is selected as $G = 2$. Considering that the ADC offers 16 bits over the ±10V input range, this results in a total sensitivity $\alpha = S \cdot G \cdot 32768/10 = 327.68\,[\mathrm{bit/A}]$.

In this example, `gain` must therefore be equal to `gain` $= 1/\alpha = 3.052\,[\mathrm{mA/bit}]$. The `offset` value can be adjusted empirically to cancel the measured value when no current is flowing through the sensor (static offset).

```
#define ADCONV (32768.0/10.0) // +/- 10V input range, 16-bit ADC
#define SENSITIVITY (0.05*2*ADCONV) // total sensitivity
#define I_GAIN (1.0/SENSITIVITY )

float I_meas = 0;

tUserSafe UserInit(void)
{
  Adc_ConfigureInput(0, I_GAIN, 0.0);
  return SAFE;
}

tUserSafe UserInterrupt(void)
{
  I_meas = Adc_GetValue(0);
  return SAFE;
}Code language: C++ (cpp)
```