

ADC - Analog data acquisition

SD003 | Posted on April 2, 2021 | Updated on March 2, 2026



Benoît STEINMANN
Software Team Leader
imperix · in

Table of Contents

- [Principles of operation](#)
- [Block parameters](#)
- [Simulink block](#)
 - [Signal specification](#)
 - [Mask](#)
- [PLECS block](#)
 - [Signal specification](#)
 - [Mask](#)
- [C++ functions](#)

The ADC block (or C++ routines) are used to access data from a given Analog-to-Digital Converter (ADC) channel. They also serve to configure how this data is sampled, filtered, and rescaled, prior to being used inside the application-level user code. Together, these parameters constitute the so-called **software part** of the ADC channel configuration, as opposed to the hardware part.

The **hardware part**, on the other hand, also requires some configuration, whose details depend on the controller model. The table below summarizes these differences and provides direct links to the associated user guides. Comparative information about imperix's programmable controllers is also given in [PN250](#).

Param.	B-Box 4	B-Box 3 RCP	B-Box micro	B-Board 3 PRO
Resources	- 24x 20Msps / 16bits - ±10V ADC range	- 16x 500ksps / 16bits - ±10V ADC range	- 8x 2Msps / 16bits - ±5V ADC range	- 8x 2Msps / 16bits - ±5V ADC range
HW param.	- Safety limits - Reaction speed - Calibration [Y/N]	- Input impedance - Safety limits - Pre-ADC gain - Low-pass filter	- Safety limits	N/A
SW param.	- Low-pass filter - Sampling method - Scaling (gain+offset)	- Sampling method - Scaling (gain+offset)	- Sampling method - Scaling (gain+offset)	- Sampling method - Scaling (gain+offset)
Doc. link	AI/AO config.	AI config.	AI config.	N/A (HW-dependent)

In addition to the proper configuration of ADC resources, the correct configuration of the **over-current and over-voltage protections** is also essential. Related information is provided in [PN257](#).

Principles of operation

The sampling of all analog inputs is always conducted simultaneously, driven by **SCLK**, which is a simple derivative of **CLK0**, only differing by the **SAMPLING_PHASE**.

Both the frequency and the phase of the ADC sampling are defined from the [CONFIG](#) block. They set part of the overall timing configuration, which must be properly coordinated with modulation-related actions to guarantee the desired execution of the control algorithms. Notably, the two following options are possible:

- **Single-rate sampling**, in which case the control task is executed once per PWM period. This is the default configuration used in most examples.
- **Double-rate sampling**, in which case the control task is executed twice per PWM period, requiring ADC sampling to be executed twice faster than the modulation. This case gives best performance, provided that sufficient CPU time is available.

- **Advanced sampling configurations** are also possible, typically authorizing to retrieve multiple samples at once (data history), variable-frequency switching, or running FPGA-based control tasks at a higher rate than the CPU.

Frequency and phase aside, all other ADC parameters are individual, channel-specific settings that can be configured independently within each ADC block.

Block parameters

Addressing

- **Device ID** selects which device to address when used in a multi-device configuration.
- **Input channel(s)** (vectorizable) selects which physical input channel to read from.

Sensor specifications

- The **Sensor** provides a list of imperix sensors. Selecting a sensor automatically populate the sensitivity parameter. When *None* is selected, the user must manually enter the sensitivity.
- **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.

Input full scale

- The **Input full scale** parameter can be set to
 - *Maximum (device dependant)*
 - *Programmable (B-Box RCP 3.0 only)*
- The **Programmable gain value** can be set to 1x, 2x, 4x or 8x and must match the B-Box RCP 3.0 front panel setting.

The [B-Box RCP 3.0](#) features a programmable gain amplifier, which can be set to 1x, 2x, 4x or 8x. These correspond to input scales of $\pm 10V$, $\pm 5.0V$, $\pm 2.5V$ and $\pm 1.25V$, respectively. This gain **must be** configured via the B-Box front panel, as described in [PN105](#).

The **programmable gain value** in the ADC block **must also match** the front panel setting, as gain mismatch will occur at the ADC block output otherwise.

The [B-Box 4](#) have a fixed input full scale of $\pm 10V$ while the [B-Box Micro](#) and [B-Board PRO](#) have a fixed input full scale of $\pm 5V$. For these devices, the **input full scale** parameter should remain to *Maximum*. A different setting would be ignored otherwise, and a warning message displayed in the log.

Sampling

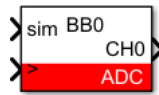
- **Synchronous averaging** computes the average of multiple samples over a specific and synchronous time interval. The interval can be configured to *1 period of CLK0* or *2 periods of CLK0*. This technique very effectively rejects high-frequency artifacts and is therefore enabled by default.
- A **Low-pass filter** is available in B-Box 3 and 4, providing a more aggressive attenuation in the high-frequency range. The chosen filter cut-off frequency may however introduce a non-negligible group delay, which should be accounted for in the control algorithm (and, ideally, the selection of the sampling phase).
For the B-Box RCP 3.0, the low-pass filter must be configured via the front panel, as explained in [PN105](#).
- **Data history** configures the block to output a vector of the N most recent ADC samples, which can be useful when the sampling frequency F_{SCLK} is larger than the CPU task frequency F_{CPU} . The maximum history depth is 64 samples.
- **Simulate synchronous averaging / low-pass filter** allows for a more accurate simulation, but it increases the simulation time.

Using synchronous averaging should be the preferred approach in most cases. The article [Sampling techniques for power electronics](#) compares the different methods and can help decide between them.

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.



Mask

Block Parameters: ADC1

ADC

Configures the software side of analog input(s).

- The output returns the measured quantity in its physical unit (e.g. volts, amperes).
- The lower input signal needs to be connected to the CONFIG block to account for the exact sampling instant in simulation.

Addressing

Device ID (default=0)

Input channel(s)

Sensor

Sensor specifications

These parameters shall correspond to those of the sensor.

Sensor

Sensitivity (V/unit)

Output offset(s) (V)

Input full scale

Input full scale

Programmable gain value

Equivalent input full-scale: -5 to 5V

For **B-Box RCP 3.0**:

- The maximum input full scale corresponds to a *programmable gain* of **x1**.
- The *programmable gain* must match the configuration set on the B-Box analog front-end. This parameters is not automatically transferred, it must be set manually via the front panel.

[More information](#)

OK Cancel Help Apply

Block Parameters: ADC1

ADC

Configures the software side of analog input(s).

- The output returns the measured quantity in its physical unit (e.g. volts, amperes).
- The lower input signal needs to be connected to the CONFIG block to account for the exact sampling instant in simulation.

Addressing

Device ID (default=0)

Input channel(s)

Sensor

Filter type

Synchronous averaging

Average over

Low-pass filter (B-Box 4 and B-Box RCP 3.0 only)

Data history

Use ADC history

Simulation

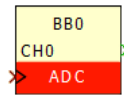
Simulate synchronous averaging / low-pass filter

OK Cancel Help Apply

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.



Mask

Block Parameters: screenshots/Imperix 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.

For **B-Box RCP 3.0**: the *programmable gain* and *low-pass filter* must correspond to the configuration set on the B-Box analog front-end. These parameters are NOT automatically transferred. They must be set manually via the front panel.

Addressing Sensor Sampling

Device ID [default=0]:

Input channel(s):

OK Cancel Apply Help

Block Parameters: screenshots/Imperix 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.

For **B-Box RCP 3.0**: the *programmable gain* and *low-pass filter* must correspond to the configuration set on the B-Box analog front-end. These parameters are NOT automatically transferred. They must be set manually via the front panel.

Addressing Sensor Sampling

Synchronous averaging:

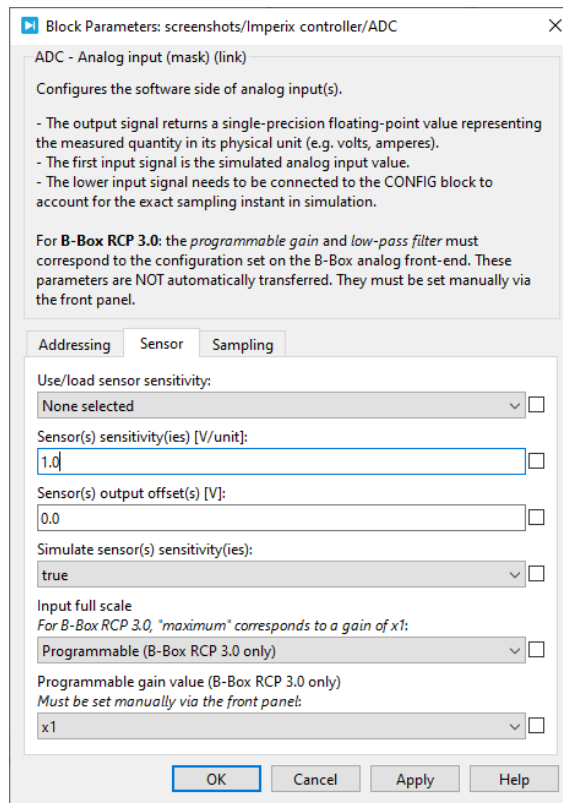
Average over:

Low-pass filter
On B-Box RCP 3.0, must be set manually via the front panel:

Multiple samples per period (ADC history):

Simulate synchronous averaging / low-pass filter:

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_ConfigureSensor — Enable and configure an ADC channel

```
void Adc_ConfigureSensor(unsigned int input, float sensitivity, float offset, unsigned int device=0);Code language: C++
```

Enable and configure the desired ADC channel with the desired sensor sensitivity and offset, such as:

`Adc_GetValue()` return value = $\frac{1}{\text{sensitivity}} \times \text{analog input value} + \text{offset}$.

Can only be called in `UserInit()`.

Parameters

- `input`: the analog input channel number (e.g. 0 to 23 on B-Box 4)
- `sensitivity`: the sensor sensitivity in (V/unit)
- `offset`: the offset (V)
- `device`: the ID of the addressed device (optional, used in multi-device configuration only)

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.

Can only be called in the interrupt routine.

Parameters

- `input`: the analog input channel number (e.g. 0 to 23 on B-Box 4)
- `device`: the ID of the addressed device (optional, used in multi-device configuration only)

Return value

- Returns the measured ADC sample.

Adc_EnableSynchronousAveraging — Enable 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 CLK0.

Can only be called in UserInit().

Parameters

- input: the analog input channel number (e.g. 0 to 23 on B-Box 4)
- device: the ID of the addressed device (optional, used in multi-device configuration only)

Advanced functions

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().

Can only be called in UserInit().

Parameters

- input: the analog input channel number (e.g. 0 to 23 on B-Box 4)
- 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 Nth historical sample of a given ADC channel.

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

Can only be called in the interrupt routine.

Parameters

- input: the analog input channel number (e.g. 0 to 23 on B-Box 4)
- 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)

Return value

- Returns the Nth measured ADC sample.

Adc_ConfigureSclkMultiplier — Configure the ADC sampling events

```
void Adc_ConfigureSclkMultiplier(int multiplier);Code language: C++ (cpp)
```

Configures the number of sampling events within a single control cycle. Refer to [SCLK Multiplier](#) for further information regarding the multiplier settings.

Can only be called in UserInit().

Parameters

- multiplier: the number of sampling events per period

Legacy 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 $value_{float} = gain \times value_{16bit} + offset$.

Can only be called in UserInit().

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)

Example of use

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

In this example, gain must therefore be equal to $\text{gain} = 1/\alpha = 3.052 \text{ [mA/bit]}$. 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)
```

[Adc_SetUserOversampling — Configure the ADC sampling events](#)

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

Configures the number of sampling events within a single control cycle. The oversampling parameter has been renamed to **SCLK multiplier**, and `Adc_ConfigureSclkMultiplier()` should be used instead. Refer to [SCLK Multiplier](#) for further information regarding the multiplier settings.

Can only be called in `UserInit()`.

Parameters

- oversampling: the number of sampling events per period