Oversampling configuration and utilization

PN154 | Posted on March 29, 2021 | Updated on May 7, 2025

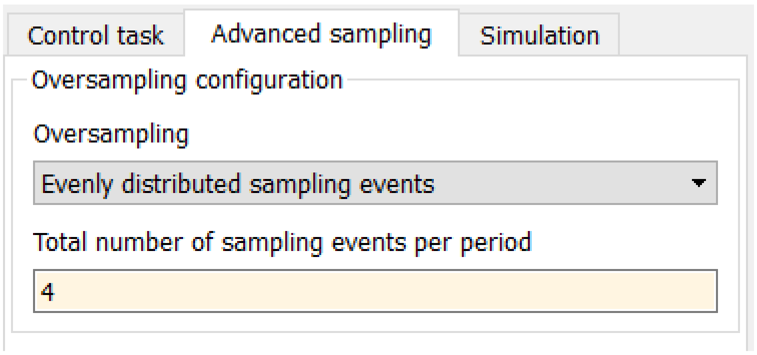
Benoît STEINMANN
Software Team Leader
imperix • in

In a standard configuration, the control algorithm is executed just after each sampling event. The oversampling feature enables the possibility to set up multiple sampling events between each control algorithm execution.

This note explains how to configure the sampling events and how to retrieve the oversampled values.

Configuring the oversampling

In the imperix ACG SDK, the sampling events are configured from the CONFIG block by setting an oversampling ratio. This spreads equidistant sampling events among the control period, starting from the main interrupt phase.

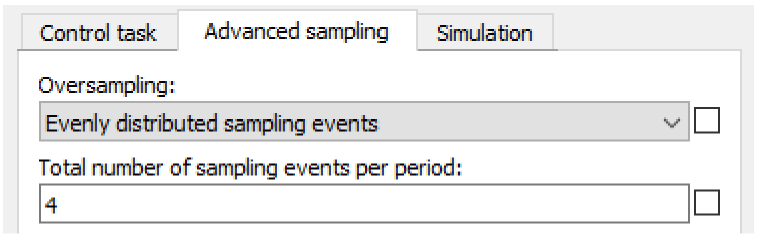


Oversampling configuration

Oversampling: Evenly distributed sampling events

Total number of sampling events per period: 4

Configuration of oversampling in Simulink

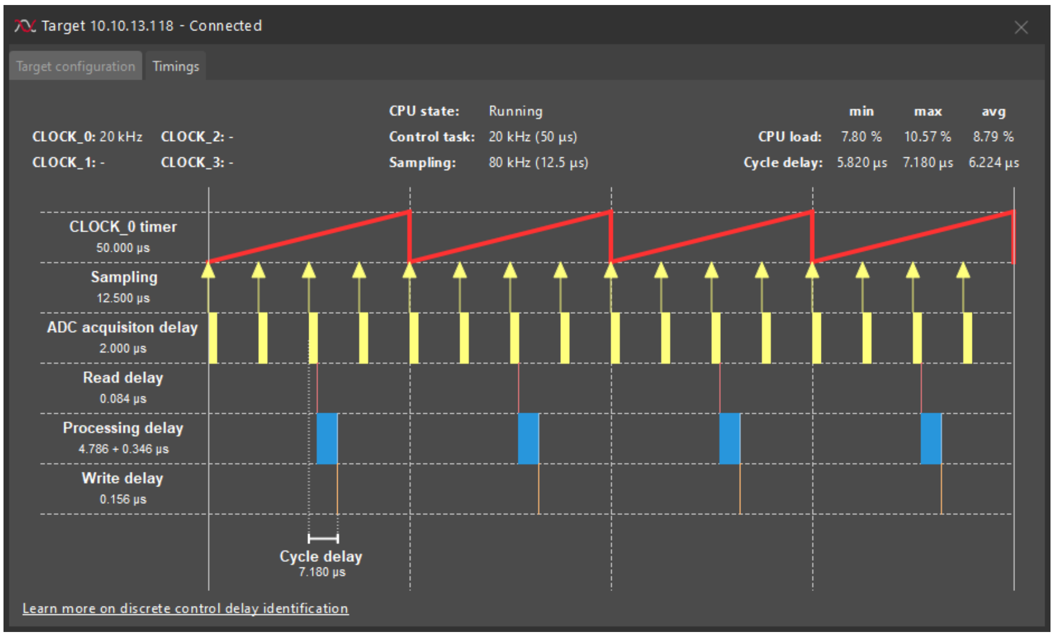


Oversampling: Evenly distributed sampling events

Total number of sampling events per period: 4

Configuration of oversampling in PLECS

The *Timings* tab in Cockpit displays where the ADC sampling events are occurring. To learn more about the Target timings tab, please refer to the Cockpit user guide.


Target 10.10.13.118 - Connected

Target configuration | Timings

						min	max	avg
		CPU state:	Running					
CLOCK_0: 20 kHz	CLOCK_2: -	Control task:	20 kHz (50 µs)		CPU load:	7.80 %	10.57 %	8.79 %
CLOCK_1: -	CLOCK_3: -	Sampling:	80 kHz (12.5 µs)		Cycle delay:	5.820 µs	7.180 µs	6.224 µs

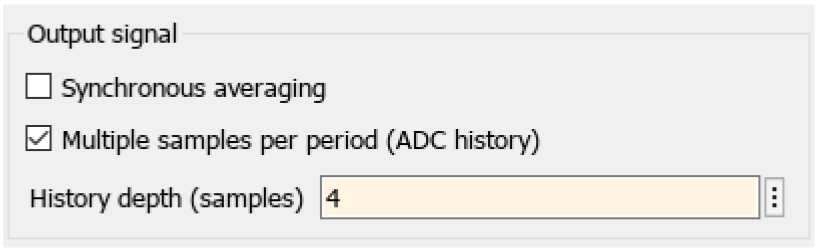
When using the C++ SDK, `void Adc_SetUserOversampling(int oversampling)` must be used in the `UserInit(void)` function.

```cpp
tUserSafe UserInit(void) {
  // Sets CLOCK_0 at 50 kHz
  Clock_SetFrequency(CLOCK_0, 50e3);

  // Set the oversampling ratio to 4 (sampling = 200 kHz)
  Adc_SetUserOversampling(4);

  ConfigureMainInterrupt(UserInterrupt, CLOCK_0, 0.5);

  // some other code...

  return SAFE;
}Code language: C++ (cpp)
```

Retrieving the oversampled analog values

In its standard configuration, the ADC block or driver will only provide the last sampled value. To retrieve older values, the **ADC history** feature must be used.

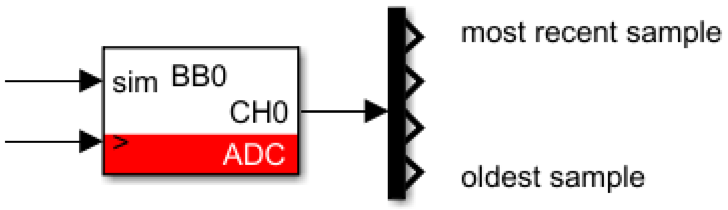
From the imperix library for Simulink or PLECS, the `ADC history` can be enabled and then the quantity of value to retrieve is configured using the `history depth` parameter. The ADC block will return a vector containing the values as shown below. The sample with index 0 is the most recent sample.


Output signal

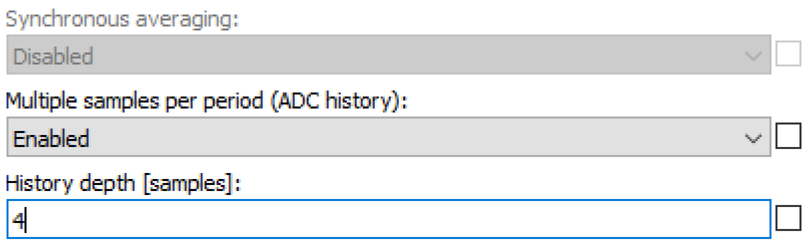
☐ Synchronous averaging

☑ Multiple samples per period (ADC history)

History depth (samples): 4

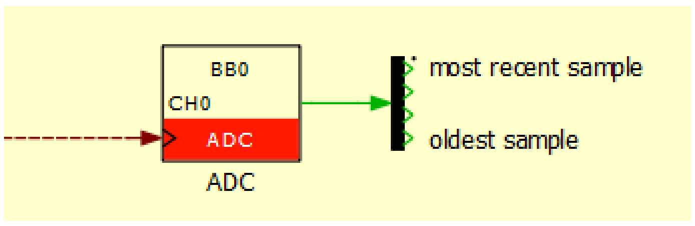


Retrieving the oversamples in Simulink


Synchronous averaging: Disabled

Multiple samples per period (ADC history): Enabled

History depth [samples]: 4



Retrieving the oversamples in PLECS

The *ADC history* and *synchronous averaging* options are mutually exclusive.

When using the C++ SDK, `Adc_ConfigureHistory` must be used in the `UserInit(void)` to enable the ADC history and configure its depth. Then the `Adc_GetHistory` can be used in the interrupt routine to get the older ADC values.

```
tUserSafe UserInit(void) {
  // Sets CLOCK_0 at 50 kHz
  Clock_SetFrequency(CLOCK_0, 50e3);

  // Set the oversampling ratio to 4 (sampling = 200 kHz)
  Adc_SetUserOversampling(4);

  ConfigureMainInterrupt(UserInterrupt, CLOCK_0, 0.5);

  // Setup a history of 4 samples for ADC0
  Adc_ConfigureHistory(ADC0, 4);

  // some other code...
  return SAFE
}

tUserSafe UserInit(void) {

  float s0, s1, s2, s3;

  s0 = Adc_GetHistory(ADC0, 0); // most recent sample
  s1 = Adc_GetHistory(ADC0, 1);
  s2 = Adc_GetHistory(ADC0, 2);
  s3 = Adc_GetHistory(ADC0, 3);

  return SAFE;
}
```
Code language: C++ (cpp)