



[Knowledge base](#) » [Product notes](#) » Simulation essentials with Simulink

Simulation essentials with Simulink

By Julien Orsinger March 23, 2021 Updated on February 15, 2024 PN135

This note provides in-depth content for an accurate and efficient offline simulation of an imperix controller and the corresponding plant model using **ACG SDK** on Simulink.

A general overview of software-related notes is given on [this page](#).

Table of Contents

1. Related content
2. Fundamental concepts
3. Working principle of the main blocks
 - 3.1. Clock and Configuration
 - 3.2. ADC
 - 3.3. PWM
4. Mastering the sample times
 - 4.1. Verifying the sample times
5. Altering the sample times
6. Further readings

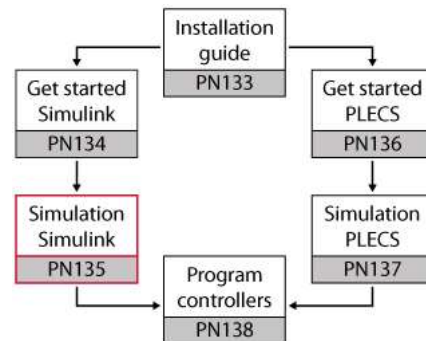
Related content

Suggested prerequisites

- [Installation guide for ACG SDK](#)
- [Getting started with Simulink](#)

Suggested further readings

- [Programming and operating imperix controllers](#)
- [Cockpit – User guide](#)
- [Speeding up Simulink simulation](#)



Getting started with imperix ACG SDK on Simulink [Part 2]



Fundamental concepts

The offline simulation is meant to reproduce as faithfully as possible the behavior of the overall system (controller and plant). To that end, the Simulink blockset was designed with the following guidelines in mind:

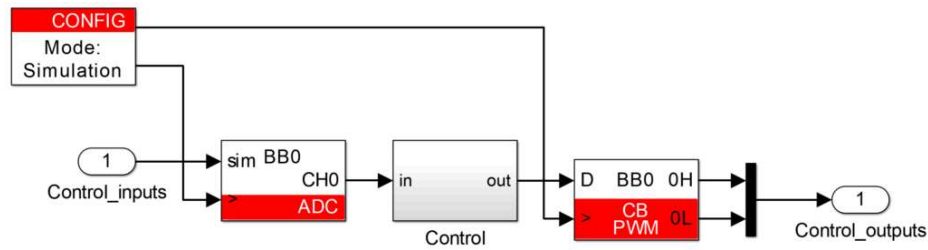
- The **plant** quantities are to be modeled with **continuous** signals
 - This requires a **variable-step solver**
- The **control** algorithm is modeled with **discrete** signals, sampled at a rate equal to the interrupt frequency, and with a phase corresponding to the sampling phase.
 - This requires a discretized algorithm (in z domain)
 - This is modeled accurately with the variable-step solver, as it is forced to take a major step at each execution of the interrupt
- The behavior of the real PWM generators is modeled, in particular:
 - The frequency and phase of the carrier
 - The instants when the duty-cycle and phase parameters are updated (at zero and/or max of the carrier)
- The duration of the algorithm execution is modeled
 - This induces a delay between the start of the interruption and the availability of the new data

With all the phases and delays modeled accurately, the simulated controller has the same dynamics as the real controller, which allows tuning the control parameters during the offline simulation.

The imperix blockset is already implemented such that these guidelines are automatically observed. As such, no particular action is required from the user.

Working principle of the main blocks

The three fundamental blocks are the Configuration, ADC, and PWM blocks. Most applications can work with only those three, as in the standard configuration shown below.



Typical content of the controller model

Clock and Configuration

The Configuration block configures the main global parameters of the model, such as

- The model execution purpose (offline simulation or code generation for a real-time target)
- The frequency of `CLOCK_0` (sampling and interrupt clock)
- The interrupt phase and postscaler

To configure and implement `CLOCK_0`, the Configuration block contains a Clock block. A clock is a time base serving as a time reference for different peripherals (e.g. ADC or PWM).

In simulation, the Clock block outputs a sawtooth signal with the clock period $T_{clk0} = 1/f_{clk0}$ and a phase of zero. Then, this clock signal is passed through a subsystem that generates a sampling clock, with a relative phase-shift of ϕ_s and a period defined by

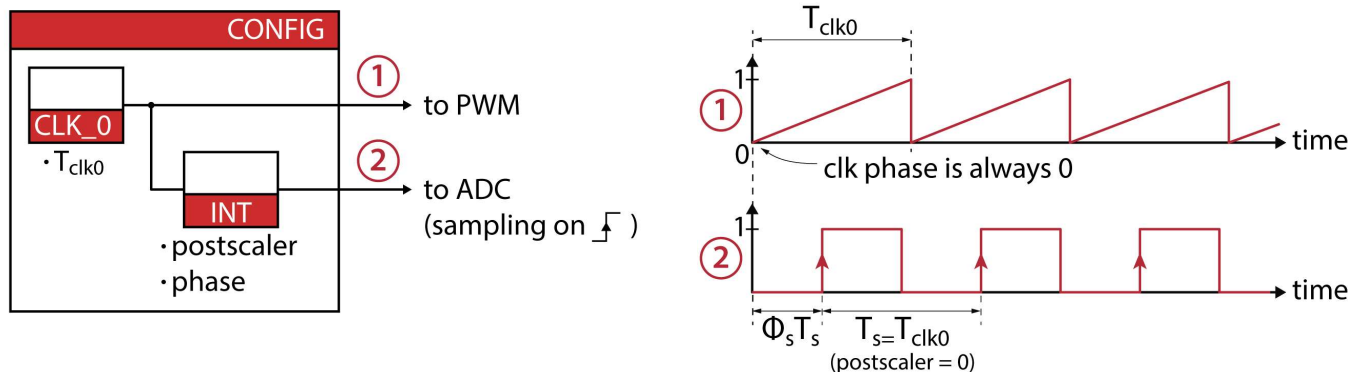
$$T_s = \begin{cases} T_{clk0} & \text{if postscaler} = 0 \\ 2 \cdot \text{postscaler} \cdot T_{clk0} & \text{otherwise} \end{cases}$$

Finally, the configuration block sets the value of the sample time variable

$$\text{CTRLPERIOD} = [T_s, \phi_s T_s]$$

This variable can be used in any block requiring a sample time (e.g. discrete transfer functions) to make sure it is executed at the interrupt frequency and with the proper phase.

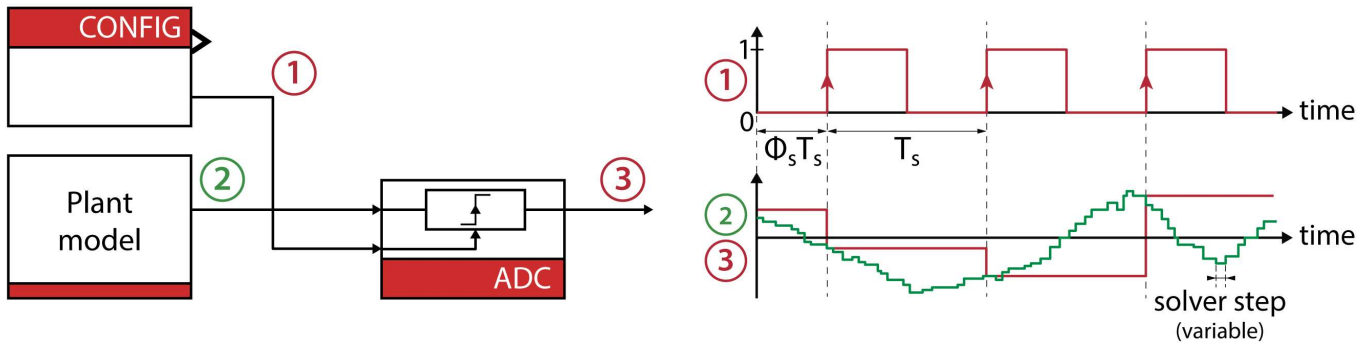
The fundamental elements of the Configuration block are mapped below, with the waveform of the generated signals.



ADC

The ADC block allows retrieving the sampled value of a given ADC channel and converts it to its value in physical unit.

The simulation model simply sample-and-holds the input signal (2) with the rising edges of the sampling clock (1). The input signal is typically a continuous signal coming from the plant model and representing a measurement value. The sample time of the output signal (3) is set to CTRLPERIOD in order to be automatically propagated to other connected blocks.

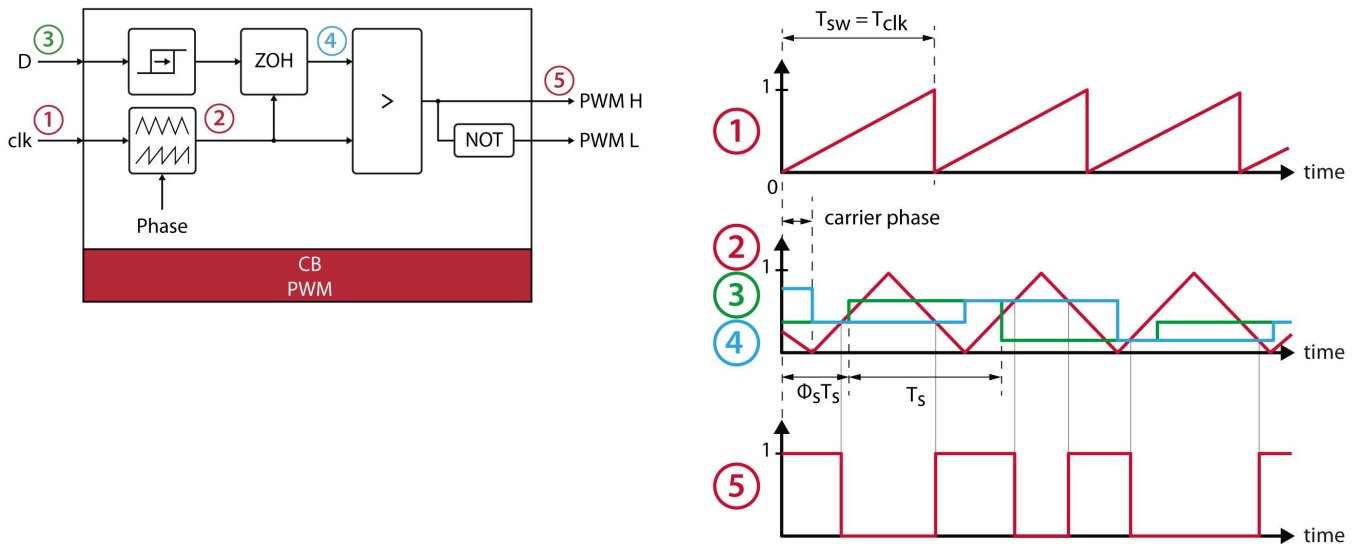


PWM

Various types of pulse-width modulators exist within the imperix blockset.

Among them, the carrier-based PWM modulator (CB PWM block) configures the corresponding FPGA peripheral and generates the PWM signals according to the duty-cycle and carrier phase parameters.

In the simulation model, the clock signal (1) connected to the clock input is used as a time reference to generate the carrier signal (2). In parallel, the duty-cycle value (3) is sampled once or twice per switching period (depending on the update-rate parameter) and compared with the carrier to produce the output PWM signal (5). (A more complex model is used to generate a carrier with a variable phase, which is beyond the scope of this document.)



Mastering the sample times

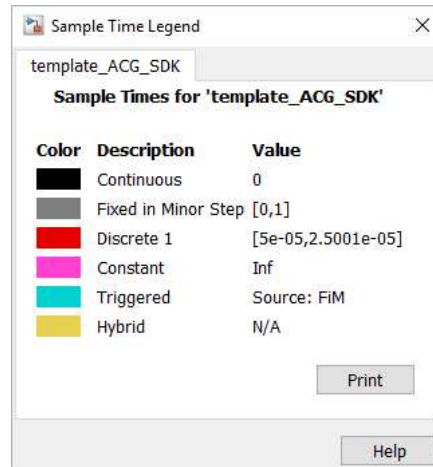
Mastering the *sample time* (essentially the *execution rate*) of each block is key for an accurate and efficient simulation, as well as the generation of proper run-time code. In particular, the execution rates must comply with the fundamental concepts listed above, namely:

- The plant (i.e. **physical**) signals are represented by **continuous** signals
- The **control** signals (i.e. those computed during the controller main interrupt) are represented by **discrete** signals with a sampling rate and phase corresponding to the configuration of the main interrupt. Their sample time is, therefore, the vector CTRLPERIOD.

The blocks of the library are built in such a way that they comply with these concepts. The user is only recommended to make sure that his control implementation is executed at the proper rate. In some situations, the `CTRLPERIOD` may not be propagated automatically to the whole control model (see the example of the step block below).

Verifying the sample times

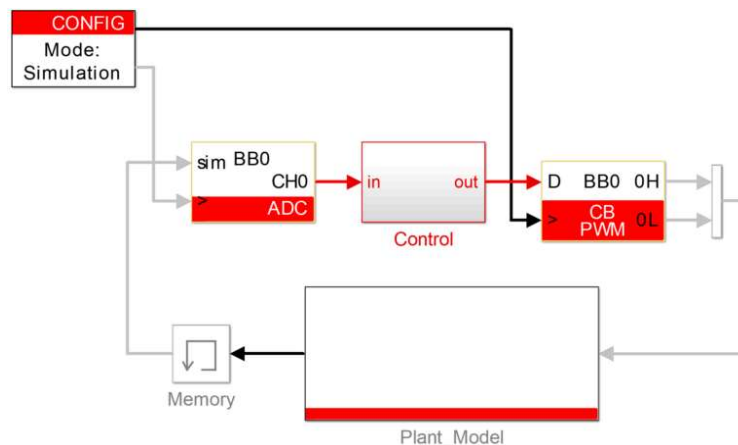
In Simulink, the sample time of each signal can be displayed by choosing Display > Sample Time > Colors. The color legend can be displayed by pressing Ctrl+J (see below)



Sample time color legend

In Simulink, the sample time of “continuous” signals (as opposed to discrete signals with a fixed period) can either be **Continuous** or **Fixed in Minor Step**. The latter is essentially an optimized version of “continuous,” applicable when the value of the signal cannot change between the major steps of the solver. For instance, the PWM signals do not vary between the switching instants. They can be considered as “semi-continuous” signals. For further reading, see <https://mathworks.com/help/simulink/ug/types-of-sample-time.html>.

With the colors defined above, a typical control implementation should look like this:

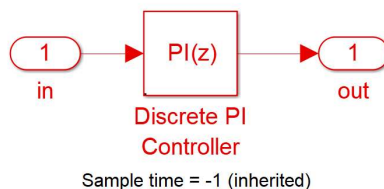


Typical sample times of a simulation model

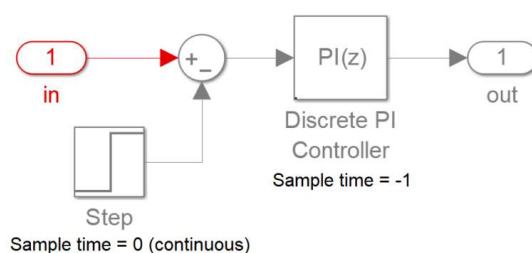
As the ADC and PWM blocks serve as an interface between the continuous plant signals and the discretized control, they contain blocks with different sample times and have thus a **Hybrid** sample time.

Altering the sample times

In most cases, inherited sample time (-1) ensures that the whole discretized control is executed at the proper rate. The illustration below shows a discrete PI controller whose sample time is -1 and the sample time is correctly propagated from outside the system:

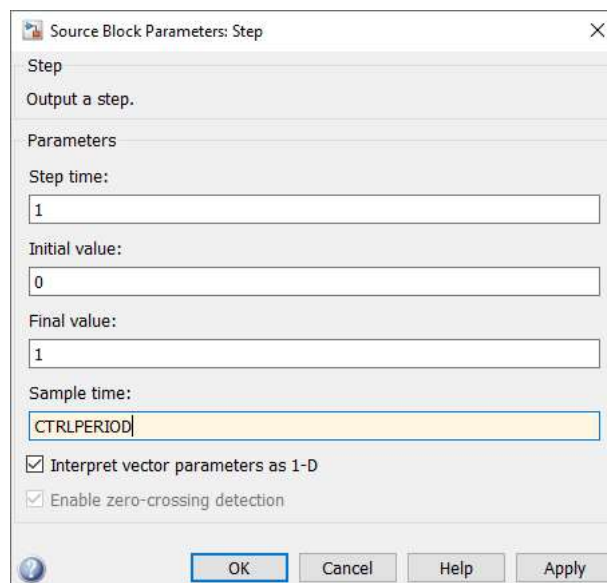
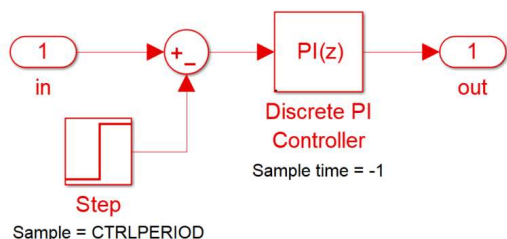


In some cases, a certain block can enforce a wrong sample time, as in the example below, where the step block is continuous (its default sample time) and propagates the wrong sample time to the PI controller (in this case, this even results in an error, since the discrete PI cannot run at a continuous sample time):



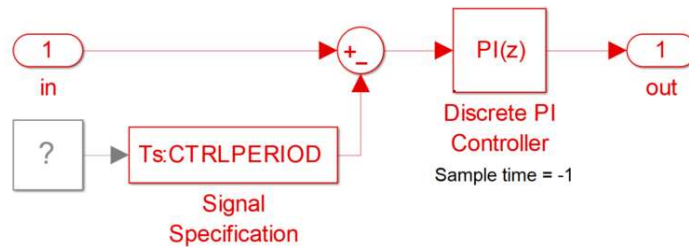
To solve this, there are basically two options to force the sample time of a particular signal or block.

- 1) Set the sample time of the “faulty” block or the discrete block to CTRLPERIOD:



Step block parameters

- 2) If the faulty block cannot be found or its sample time cannot be specified, use a **Signal specification** block with CTRLPERIOD as sample time



Further readings

- [Programming and operating imperix controllers \(PN138\)](#)
- [Cockpit – User guide \(PN140\)](#)
- [Speeding up Simulink simulation \(PN131\)](#)

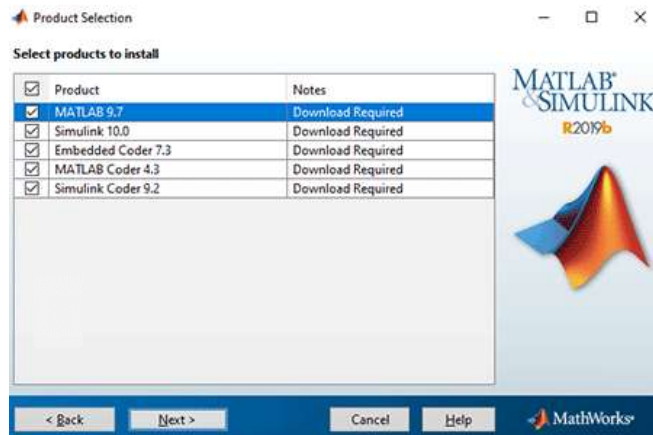


Julien Orsinger

Julien is a power electronics engineer. On the knowledge base, he is acting as the editorial manager and is therefore the co-author of numerous articles on a broad variety of topics.

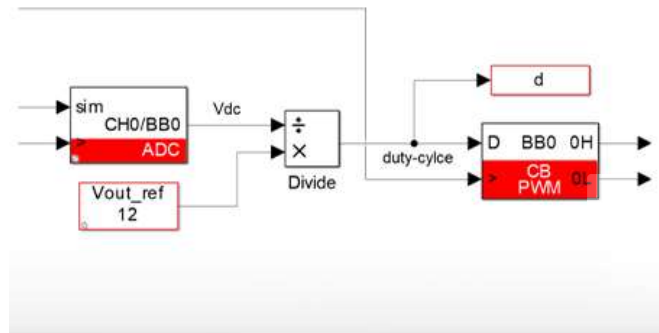


Similar Posts



Installation guide for imperix ACG SDK

By Julien Orsinger March 25, 2021 PN133



Getting started with ACG SDK on Simulink

By Julien Orsinger March 24, 2021 PN134

RELATED PRODUCTS

B-Box RCP

[learn more](#)

B-Board PRO

[learn more](#)

B-Box Micro






























[learn more](#)

DOCUMENTATION SPACES

[Application notes](#)[Product notes](#)[Technical notes](#)[Software reference](#)

PRODUCT NOTES

- PN133 Installation guide for imperix ACG SDK
- PN134 Getting started with ACG SDK on Simulink
- PN135 Simulation essentials with Simulink
- PN136 Getting started with ACG SDK on PLECS
- PN137 Simulation essentials with PLECS
- PN139 Getting started with BB Control
- PN151 Basic examples for ACG SDK on PLECS
- PN130 Graphical User Interface with MATLAB App Designer

-  PN158 Multi-master feature for distributed networked control systems
-  PN131 Speeding up Simulink simulation
-  PN143 Starting from a blank Simulink model
-  PN144 Simulink External Mode with ACG SDK
-  PN145 Multi-rate control on Simulink with ACG SDK
-  PN153 Integration of C code in Simulink via S-Functions
-  PN155 Multi-rate control on PLECS with ACG SDK
-  PN146 Installation and use of the CPP SDK
-  PN147 CPP SDK troubleshooting
-  PN120 Setting up the FPGA development toolchain
-  PN116 Imperix firmware IP product guide
-  PN104 Using the angle decoder modules
-  PN105 Analog front-end configuration on B-Box RCP
-  PN115 Dead time selection for imperix power modules
-  PN121 Variable frequency operation with the B-Box/B-Board
-  PN142 Discrete control delay identification
-  PN152 Applying pre-recorded profiles as setpoints
-  PN154 Oversampling configuration and utilization
-  PN156 Timing info tab in BB Control
-  PN200 Expanding an open-frame rack with PEB modules
-  PN157 LCD display information of closed converter racks
-  PN159 Getting started with FPGA control development
-  PN163 Xilinx Model Composer introduction
-  PN161 Xilinx System Generator introduction
-  PN164 Xilinx Vitis HLS introduction
-  PN162 MATLAB HDL Coder introduction
-  PN166 Installing the Xilinx Blockset for Simulink
-  PN169 FPGA development on imperix controllers
-  PN170 How to build a 3 phase inverter
-  PN168 Xilinx Vivado Design Suite installation
-  PN119 How to build a buck converter
-  PN180 How to build a variable speed drive
-  PN171 Power electronic bundle – quick start guide
-  PN177 OPC UA: the communication protocol for industrial automation applications
-  PN178 Making an OPC UA client with MATLAB using the Industrial Communication Toolbox
-  PN140 Cockpit – User guide
-  PN116 Imperix firmware IP product guide
-  PN175 Build a custom user interface to operate Imperix power converters
-  PN138 Programming and operating imperix controllers
-  PN181 Motor Testbench quick start guide
-  PN202 Real time communication protocols for B-Box RCP
-  PN190 Getting started with the TPI 8032
-  PN124 Synchronous averaging
-  PN201 Custom carrier board design for B-Board PRO
-  PN107 Getting started with B-Box Micro
-  PN106 Analog inputs configuration on B-Box Micro
-  PN136 Getting started with ACG SDK on PLECS
-  PN172 MMC bundle – quick start guide

NEWSLETTER

First Name

Last Name

Email address

Subscribe

[Previous newsletters](#)

CONTROL PRODUCTS

- Power electronic controllers
- Rapid prototyping controller
- Power inverter controller
- Inverter control board
- Software
- Knowledge base
- Accessories

POWER PRODUCTS

- Half-bridge module
- SiC power module
- IGBT power module
- Full-bridge module
- NPC converter module
- Kits & bundles
- MMC test bench

COMPANY

- Company
- Upcoming events
- News & press releases
- Customers
- Scientific publications
- Request a demo
- Contact