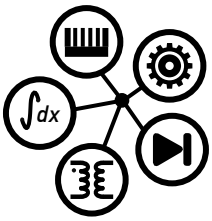

PLECS

DEMO MODEL

Buck Converter with Controls in Co-Simulation

Template for co-simulations of PLECS with other simulation tools

Last updated in PLECS 4.7.1

1 Overview

This demonstration shows the implementation of a digital controller for a simple buck converter. The controller block uses a configurable subsystem that can be toggled between a co-simulation with an external tool and a discrete digital control with a Z-domain controller. The latter implementation is already known from the demo model "Buck Converter with Digital Controls". The implementation of the controller block is accessible by looking under the mask (**Ctrl+U**).

Note This model contains model initialization commands that are accessible from:
PLECS Standalone: The menu **Simulation + Simulation Parameters... + Initializations**
PLECS Blockset: Right click in the **Simulink model window + Model Properties + Callbacks + InitFcn***

---

2 Model

Figure 1 shows a schematic of a buck converter using a MOSFET. The circuit is clocked with a fixed frequency of $100\,\mathrm{kHz}$. The output voltage of the converter is regulated to the voltage reference by a proportional-integral-derivative (PID) controller. The controller is configured as a discrete digital controller or as a co-simulation with Python. The configurations for co-simulation will be shown in the following sections.

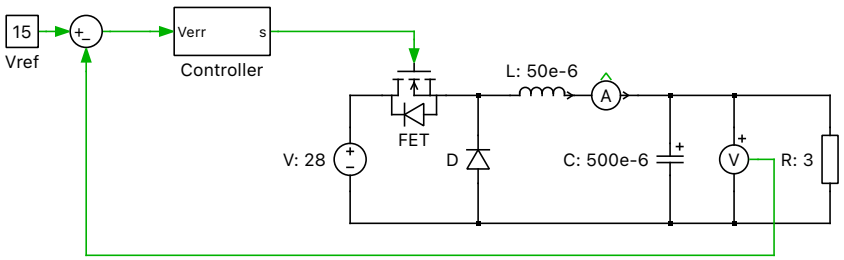

Figure 1: Buck converter with configurable digital controls.

2.1 Discrete Digital Control

This discrete proportional-integral-differential (PID) control method uses a Z-domain controller. While the controller operates with a fixed sampling time of $10\,\mu s$, the PWM block is still evaluated with a variable time step solver.

2.2 Co-Simulation

This implementation of the co-simulation runs as a C-Script in PLECS. In this case, the co-simulation is done with an external Python script `buck_converter_with_co_simulation.py` that runs the controller.

At each clock of $100\,\mathrm{kHz}$, PLECS writes the voltage error signal V_{err} to the file `outputPLECS.txt` and deletes the file `outputPyhton.txt` to let Python know that PLECS is done. Python reads the value, updates the PID controller parameters, and writes a PWM pattern with a time resolution of $7.5\,ns$. This may be a realistic resolution for an FPGA. The Python script deletes the `outputPLECS.txt` file to tell PLECS that Python is done. PLECS reads the file `outputPython.txt` and simulates until the next clock cycle is reached. An integer is added to the end of each file to represent the simulation step. This ensures that no incorrect files are read and avoids any `wait`-statements. Detailed explanations are given in Sections 4.2 and 5.

The Python code is derived from the generic C code generated from the discrete digital control configuration using the PLECS Coder.

3 Simulation

While the simulation of the discrete digital control can be run like all other PLECS simulations, the co-simulation can only be activated through the simulation scripts (**Simulation + Simulation scripts...**).

Note This only works if your operating system is able to run Python scripts directly from the console! Make sure that the `wait`-statement on line 9 is long enough to start the Python script.

Alternatively, the Python script `buck_converter_with_co_simulation.py` can be started with any suitable runtime environment. To do this, first start the Python script and then the simulation script with line 9 deactivated.

The simulation results show similar behavior and differ only in the evaluation of the PWM block. While the Python implementation runs with a fixed time step of $7.5\,ns$, the PLECS PWM block runs with a variable time step. These small differences are particularly visible when a transient occurs, as shown in Fig. 2.

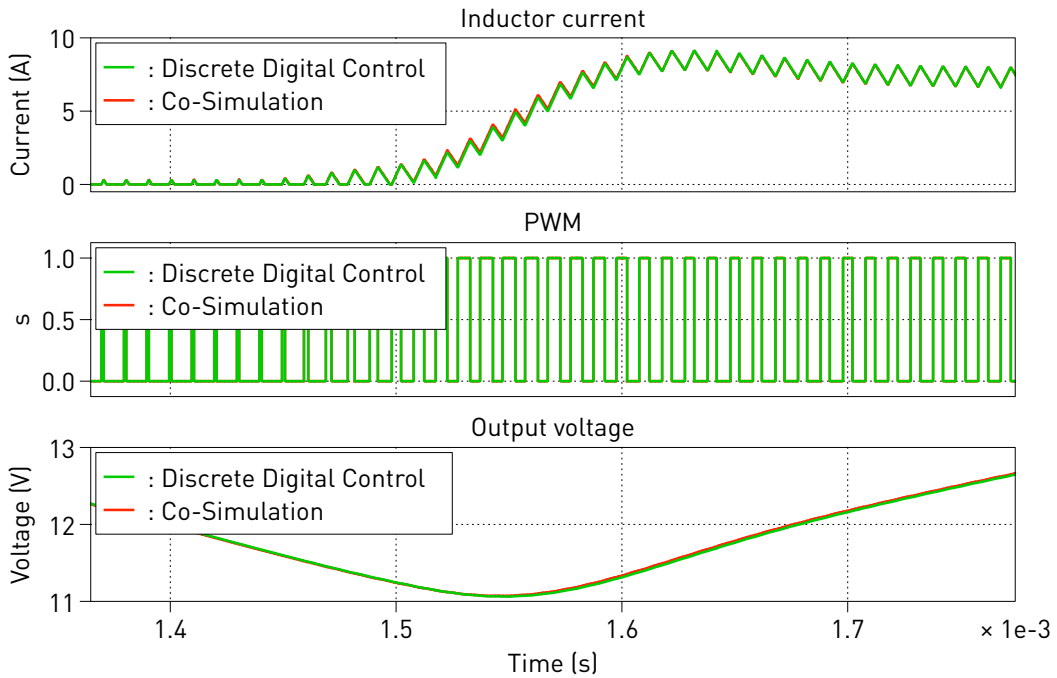

Figure 2: Load change in the simulation. Since the discrete digital control uses a variable time step for the PWM generator, a small discrepancy to the PWM generation with fixed time step is visible in the co-simulation.

4 Overview Templates

The following sections provide simple templates for co-simulating with PLECS. A total of three dummy examples are shown, two in Python and one in ModelSim. This concept can be adapted to any conceivable software capable of writing data to a file, such as FPGA or FEM simulators.

4.1 Model

The PLECS model consists mainly of a C-Script where the communication with another simulation tool is handled. This communication channel is realized by writing `.txt` or `.csv` files. This very simple way of communicating can be easily adapted for different simulation tools. However, when increasing the length of the simulation interval it needs to be considered that this communication scheme is very slow.

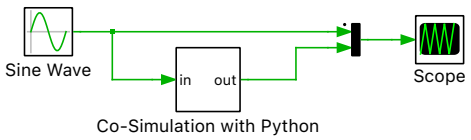

Figure 3: Main schematic of PLECS model

Note This model contains model initialization commands that are accessible from:
PLECS Standalone: The menu **Simulation + Simulation Parameters... + Initializations**
PLECS Blockset: Right click in the **Simulink model window + Model Properties + Callbacks + InitFcn***

4.2 Simulation: TimesTwo with Python

This co-simulation focuses on synchronization with Python and PLECS. A signal from PLECS is read into Python, multiplied by 2 and sent back to PLECS. The co-simulation is started using the simulation scripts (**Simulation + Simulation Scripts...**) or manually by running `co_simulation.py` in the demo model folder and then starting the PLECS simulation.

The Python script runs following steps:

1. The Python script creates a dummy file `outputPython_0.txt`
2. Python then waits until the file `outputPython_0.txt` is deleted by PLECS
3. Python reads `outputPLECS_1.txt`
4. Python multiplies the value by two
5. Python writes the result to `outputPython_1.txt`
6. Python deletes the `outputPLECS_1.txt` file so PLECS knows it can read `outputPython_1.txt`
7. Goto 2 (the index number is toggling between 2 and 3 in order to avoid an overflow)

In parallel PLECS makes the following steps:

1. PLECS checks if `outputPython_0.txt` exists
2. PLECS runs one time step
3. PLECS creates a file `outputPLECS_1.txt` with the signal from the Sine Wave block
4. PLECS deletes the file `outputPython_0.txt` so that Python notices that it is allowed to read `outputPLECS_1.txt`.
5. PLECS waits until Python deletes the file `outputPLECS_1.txt`
6. PLECS reads the file `outputPython_1.txt` and continues with the simulation
7. Goto 2 (the index number is toggling between 2 and 3 in order to avoid an overflow)

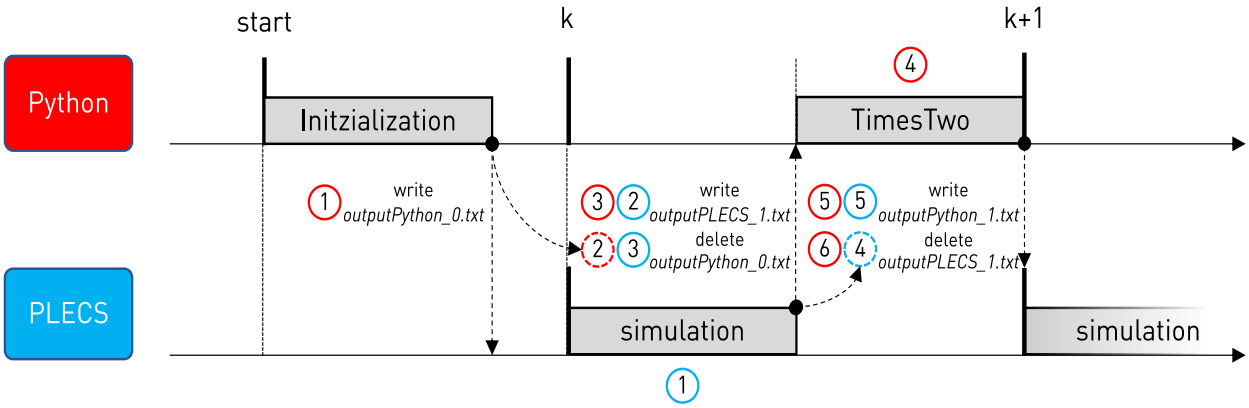

Figure 4: Flow diagram of the co-simulation between Python and PLECS.

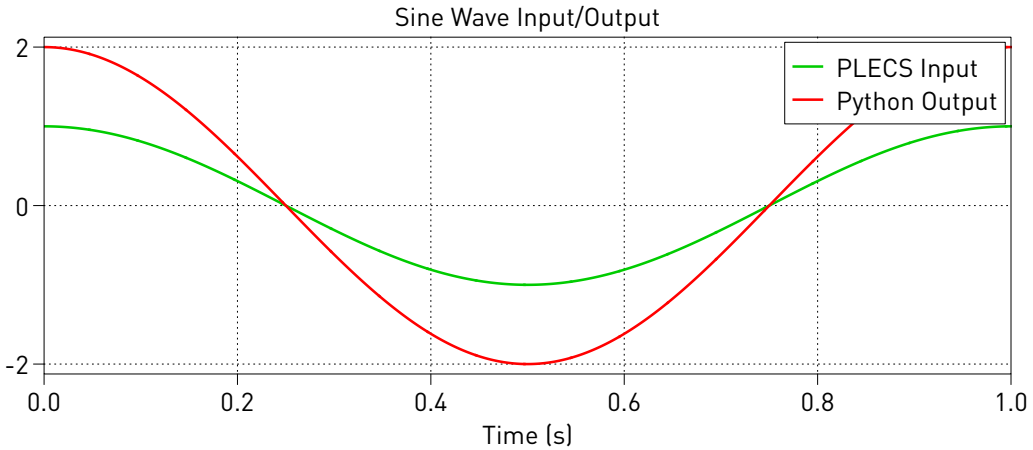

Figure 5: Result of co-simulation. The chosen sample time of $0.005\ s$ in the C-Script is clearly visible in the lower plot.

4.3 Simulation: TimesTwo with ModelSim

This co-simulation focuses on synchronization between ModelSim and PLECS. ModelSim is a simulation tool for testing hardware description languages such as VHDL or Verilog. This tool is included in the Lattice Diamond software package, which can be downloaded for free. As with the co-simulation presented above, a signal is read from PLECS into ModelSim, multiplied by 2, and sent back to PLECS. To prevent confusion, the file written by ModelSim is called `modelsim.csv` and the one written by PLECS is called `plecs.csv`.

Setting up and running the simulation can be done with the following steps:

1 Open Lattice Diamond

2 File + New + Project...

- Next
- Set name and path
- Next
- Next (as this is only a generic testbench, FPGA type is irrelevant)
- Change to `Synplify Pro` and click Next
- Finish

3 File + Add + Existing File...

- select `tv.sv` and `top.sv` from the demo model source folder
- save the project and close Lattice Diamond

4 Copy `co_simulation_modelSim.plecs` into the project folder defined above

5 Open the project again in Lattice Diamond

6 Click **Simulation Wizard**

- Next
- Enter “tb” as project name (tb := testbench) and click Next and Yes
- Next
- Next
- Next
- Finish

7 Simulation will start and writes “waiting for PLECS to be started...” to the ModelSim Console

8 Open the PLECS model in the project folder and start PLECS simulation

9 Co-simulation runs through and terminates after everything is done

10 You can restart the ModelSim Simulation with **Simulation + Restart... + OK** and then click **Run – All**

This co-simulation generates the same results as already depicted in Fig. 5.

5 Simulation: Pattern with Python

Both presented co-simulation templates use a communication exchange after each simulation step. Writing to a file and reading from a file is a bottleneck and therefore this co-simulation setup is time consuming. However, depending on the use case, this approach may not be necessary and a much larger step size could be chosen. An example is given right at the beginning with a buck converter where an exact PWM signal is derived for each switching period. Therefore, a pattern could be calculated in the controller and then sent as a packet to PLECS for solution.

The co-simulation `co_simulation_pattern` (and `co_simulation_pattern.py`) is able to do this. For a time step of $0.005\ s$ a pattern is generated respectively loaded from the file `pwm.csv`. This pattern is written in Python to the file `outputPythonPattern.txt`, which in turn is loaded into PLECS and executed for the length of this pattern with the commands `NextSampleHit` and `IsSampleHit`. The sample time setting of `[-2, 0]`[1]. Moreover, in both files `ouputPython.txt` and `outputPLECS.txt` the same signal manipulation is performed for the same time step as in the other co-simulation templates.

6 Conclusion

In this demonstration, a simple solution for how to set up a co-simulation with two different simulation tools is shown. Furthermore, two different possibilies for selecting the necessary time steps are presented.

[1] The solver of *PLECS Standalone* is able to hit the selected point exactly with the `NextSampleHit` macro. Unfortunately, the s-function interface of Simulink only offers to resolve this selected point in time with the zero crossing method. The solver may miss a sample as this is not exactly on but only numerically close to the chosen time point. To mitigate this problem, the next time step is chosen. This results in a maximum time shift error of $7.5\,ns$.

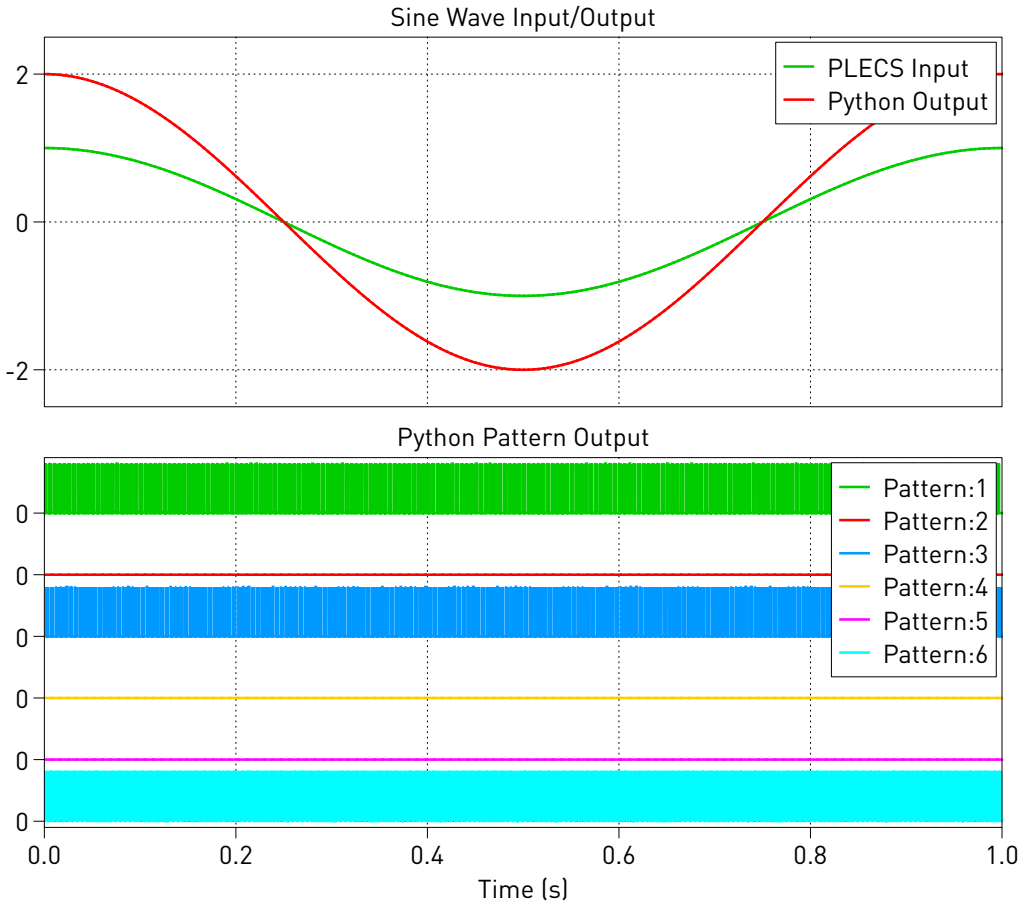

Figure 6: Result of co-simulation with a pattern. The chosen sample time of $0.005\ s$ in the C-Script is clearly visible in the lower plot.

Revision History:

PLECS 4.7.1	First release

How to Contact Plexim:

☎	+41 44 533 51 00	Phone
	+41 44 533 51 01	Fax
✉	Plexim GmbH Technoparkstrasse 1 8005 Zurich Switzerland	Mail
@	`info@plexim.com`	Email
	`http://www.plexim.com`	Web

PLECS Demo Model