Integration of C code in Simulink via S-Functions

PN153 | Posted on March 25, 2021 | Updated on May 7, 2025

Julien ORSINGER
Power Applications Specialist
imperix

François LEDENT
Development Engineer
imperix

Table of Contents

- Creating an S-function
- Good to know about S-Functions
  - Auxiliary functions
  - Arrays as arguments
  - Path issues with the .mat file
  - Reuse the S-Function
  - Example
- C code on PLECS

This note provides instructions for integrating C code into a control algorithm developed using the ACG SDK via S-Functions, which is probably the best way to include C code into a Simulink model.

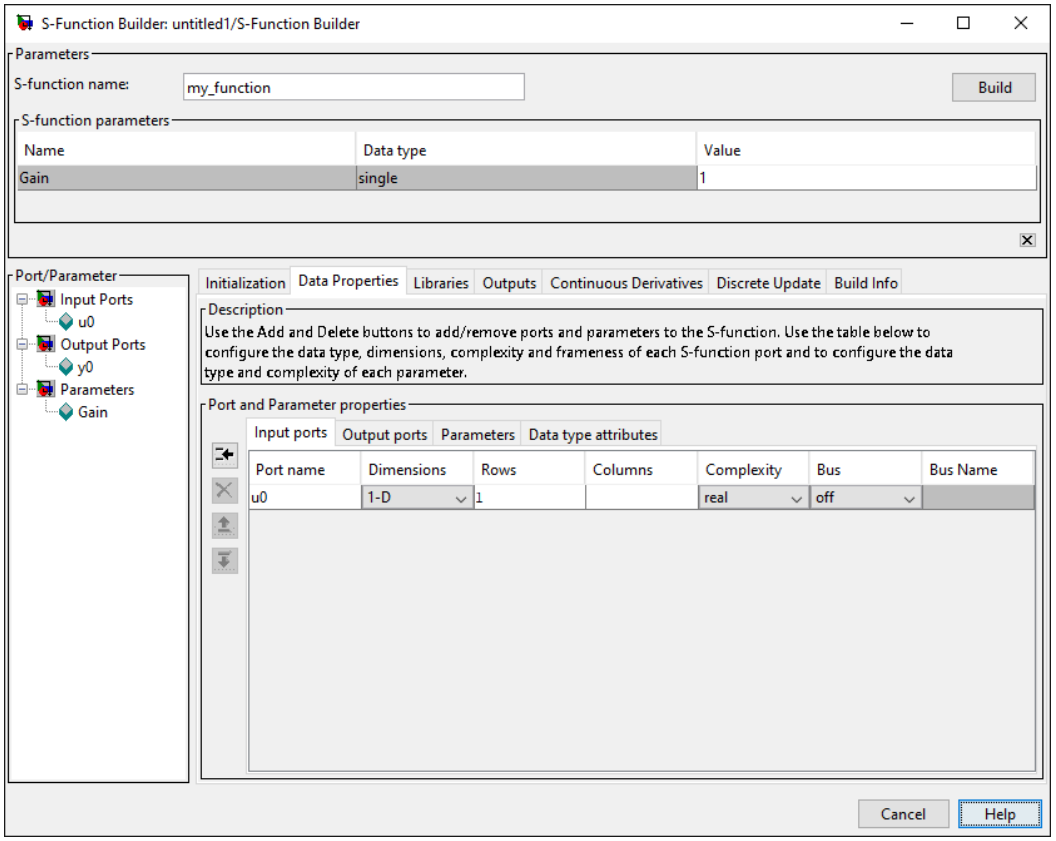
With S-Functions, users can create a custom Simulink block, whose behavior is defined by code (C, C++, Fortran or MATLAB), and thus implement advanced algorithms that would be complex to develop with the graphical approach. However, S-Functions have specificities that are good to have in mind, several are discussed in this article.

The integration of C code in PLECS is straightforward and thus briefly discussed at the end.

Creating an S-function

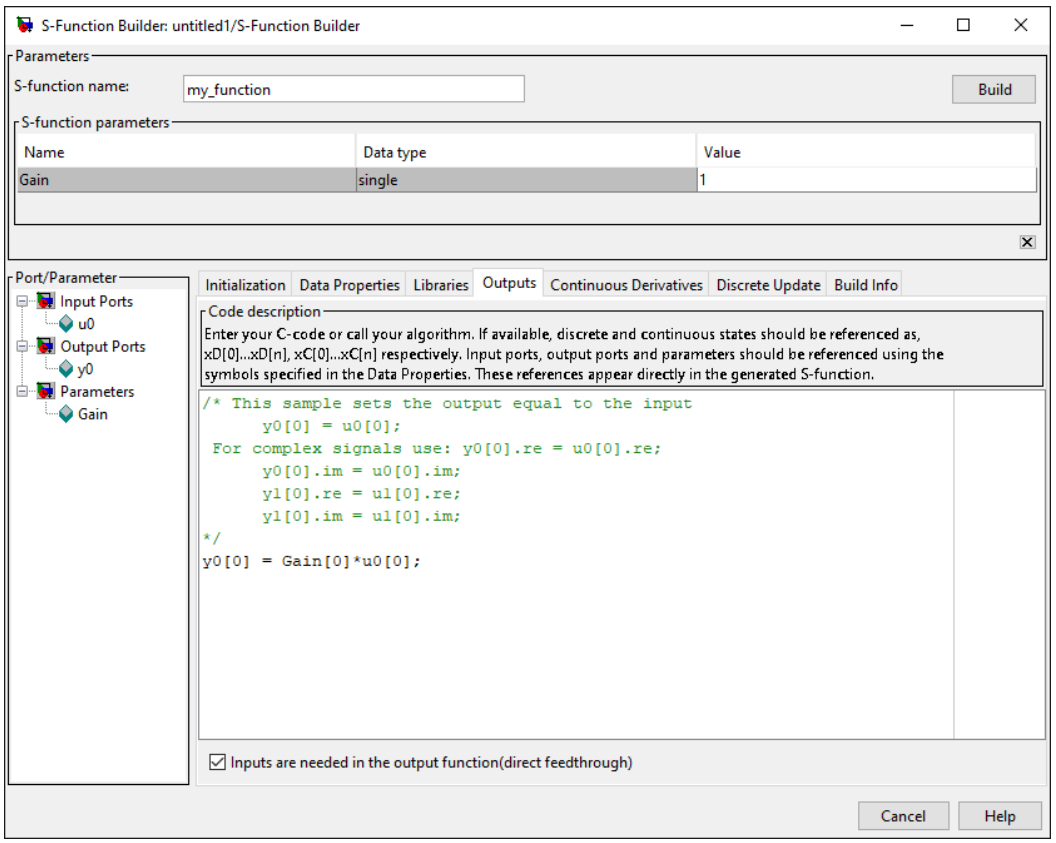
Simulink provides an S-function Builder block that generates the necessary files and helps define the inputs, outputs and parameters of the S-function.

The inputs, outputs and parameters can be defined in the Data Properties tab:



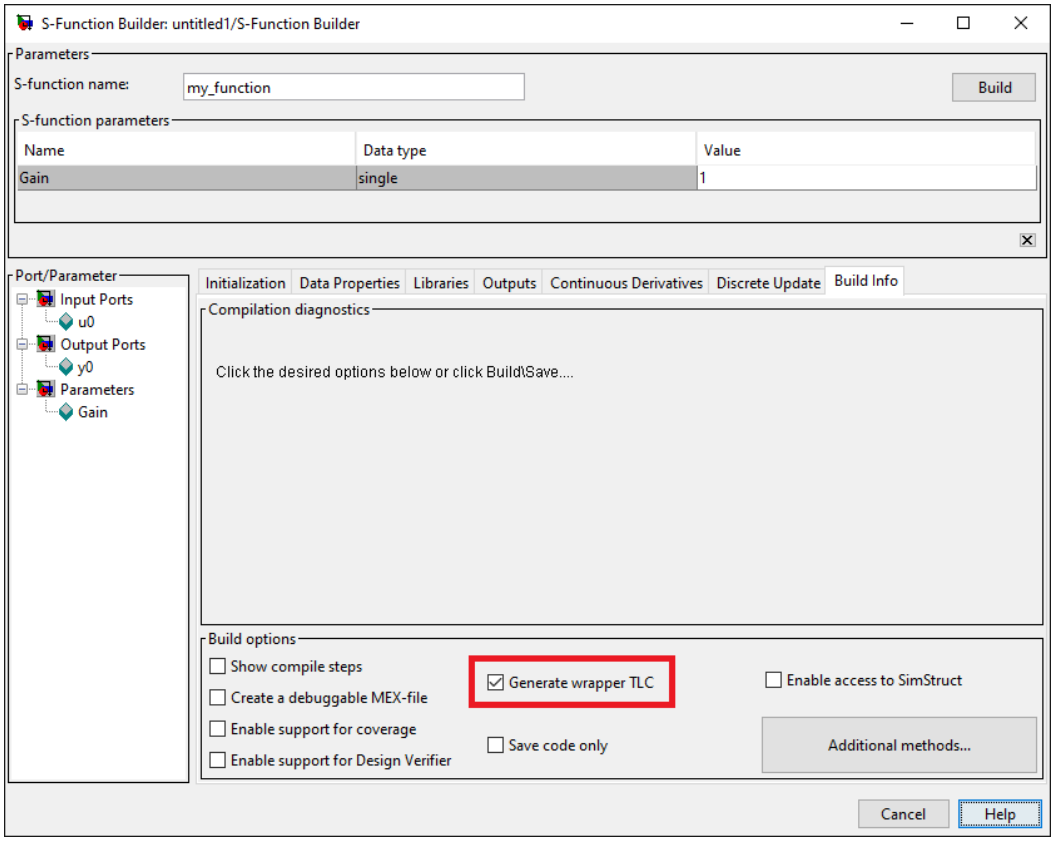
S-function builder input/output configuration

Then, your C code can be written in the Outputs tab.



S-function builder output code

Finally, the following build options can be configured in the Build Info tab:



S-function build options

When finished, the Build button on the top right corner generates the necessary files. These include:

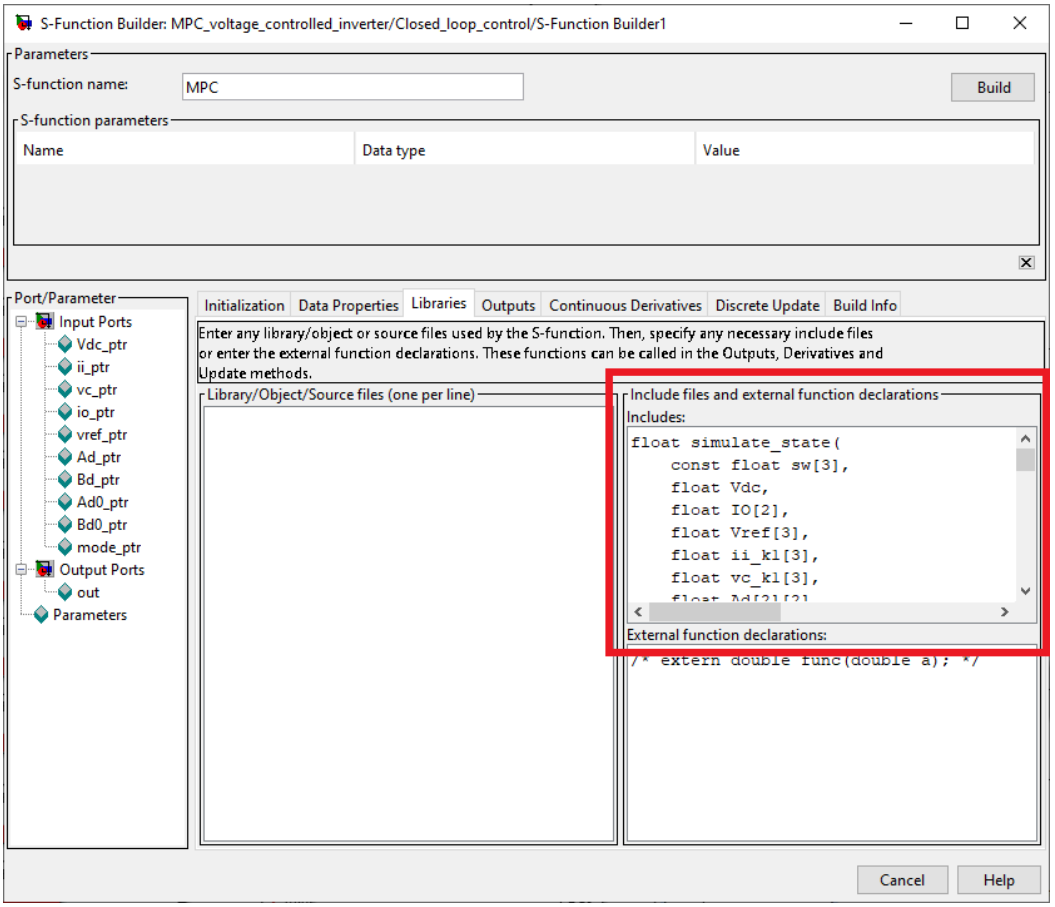
name.cpp	Is the S-function files that define the function inputs, outputs and parameters and call the wrapper function.
*name*_wrapper.cpp	Is a function that contains the user-written code of the S-function. This file is the only file compiled by the imperix toolchain in code generation.
name.tlc	Is an interface defining how the code should be called by Simulink Coder.
name.mexw64	Is the compiled version of the S-function.

Make sure that MATLAB is in the same folder as your model when you click the Build button. This ensures that the generated files will have the same path as your model and that the compiler will be able to find them when generating code for the whole model.
A MEX compiler is required to use S-functions. If none is installed, please follow the instructions of the section “Mex compiler add-on” from Installation guide for imperix ACG SDK (PN133).

Good to know about S-Functions

Auxiliary functions

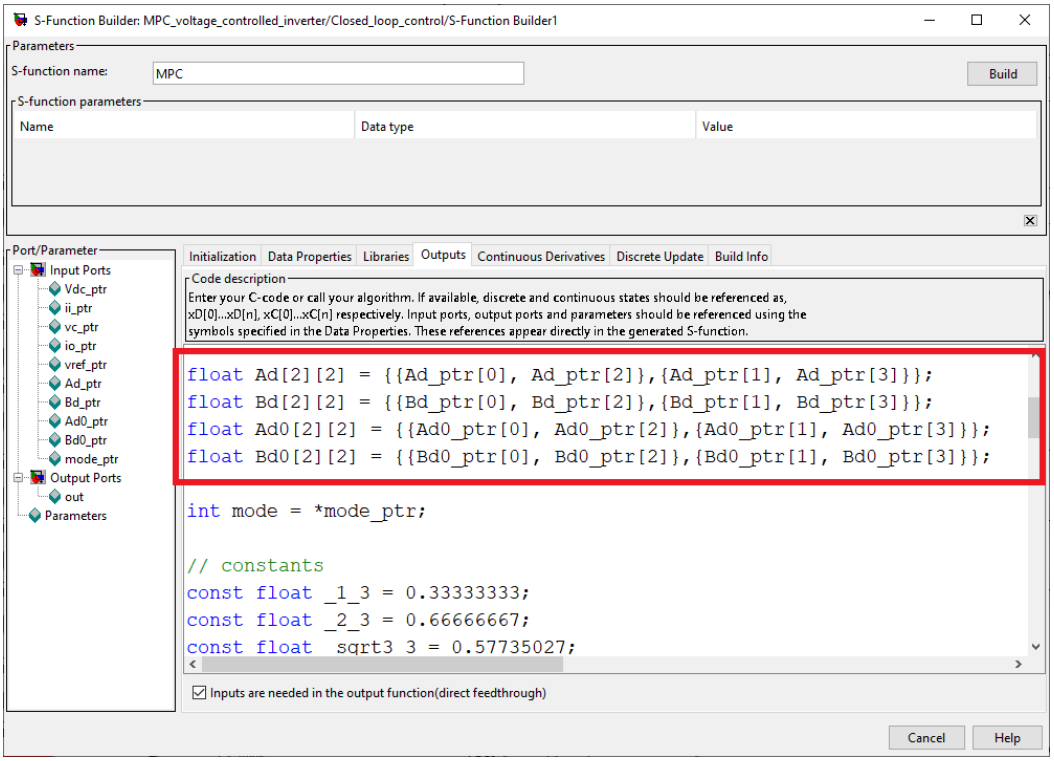
Auxiliary functions can be placed in the **Includes** input box, along with any potential **define** directives (as shown with the *simulate_state* function in the figure below).



S-Functions : auxiliary functions via the **Includes** input box

Arrays as arguments

The S-function supports arrays as arguments. However, arrays of dimension higher than 1 are flattened and must be resized in the C script to recover the desired shape, as in the following example :



Reshaped array arguments in a S-Function

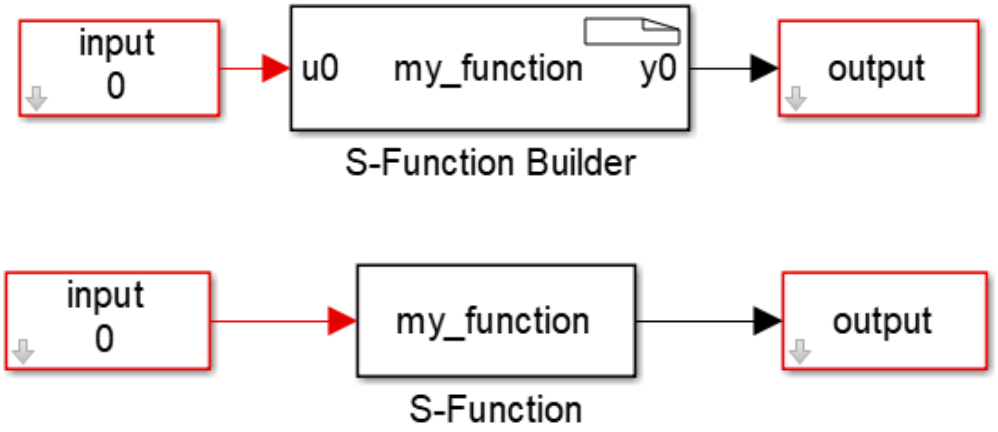
Note that Matlab is based on the column-major ordering. That means that arrays are transposed compared to the row-major ordering usually used in C, C++ and Python, for example. This explains the position of the terms *[...]_ptr[1]* and *[...]_ptr[2]* in the figure above.

Path issues with the .mat file

Path issues can occur when the project is located after the build of the S-function. Those issues can be avoided by removing the *SFB__[s-function name]__SFB.mat* file before transferring the project folder. Removing the .mat file does not affect the simulation or the code generation of the model.

Reuse the S-Function

The S-Function Builder block also serves as a wrapper for the generated S-function, which means that it can be used inside a model as any other Simulink block. Alternatively, an S-function block can be used to integrate the generated files into a Simulink model.



Example

A working example is available in the Simulink code (CPU version) of TN162.

C code on PLECS

PLECS provides a handy C-Script block that lets you write and compile custom C code to integrate it into your model. It works in both simulation and code generation modes.