BULETINUL Universităţii Petrol – Gaze din Ploieşti	Vol. LXV No. 3/2013	28 - 36	Seria Tehnică

The Utility of Using S-Functions to Define Complex Elements

Costin Daniel Lupu, Boris Siro, Cornel Ianache

Universitatea Petrol-Gaze din Ploieşti, Bd.Bucureşti, nr. 39, Ploieşti
e-mail: costindaniellupu@yahoo.com; bsiro@upg-ploieşti.ro; cianache@upg-ploiesti.ro

Abstract

In Simulink/SPS simulation diagrams are complex blocks such as electrical motor drives. Usually, these blocks have a mask and relatively simple dialog boxes, but behind them are hidden a multitude of Simulink blocks that, based on specific functioning equations, simulate (with a few simplifying assumptions) the operation of that diagram element. Using S-Functions bring a significant simplification to the definition of some of these complex diagram elements.

Key words: *simulation, simulation block definition, simplifying complex elements.*

Introduction

The utility of simulating, in the case of complex process diagrams, is well known, and as simulation environments, Simulink is the dedicated program for this from the MATLAB programming environment. All the Simulink derivatives such as SPS (SimPowerSystems), SimMechanics and others, have very well equipped block libraries, but Simulink (of a higher version) has a specific library named User-Defined Functions that can be used to define, among others, Embedded MATLAB Functions and S-Functions, that can allow the user to customize a specific block, such as the block for an electrical drive motor, although it is predefined in a certain form, in the Machines library of SPS.

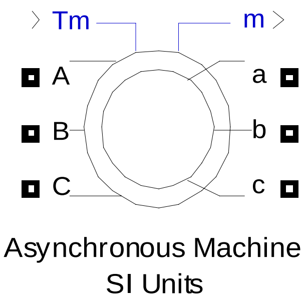


Fig. 1. The asynchronous machine icon

Because the example of an electrical drive motor was considered, maybe because of the authors' specialty, this example will be attended in the following, although as a diagram element, it can be replaced with a complex mechanical, hydraulic, thermic etc. element in a corresponding simulation diagram.

For example, the icon of an asynchronous machine, extracted from the "Machine" SPS library is shown in Figure 1, but behind it there is a differential equations system that defines, with a few simplifying assumptions (generally admissible in studies regarding this type of electrical machine), such as:

$$
\begin{aligned}
U_{ds} &= R_s i_{ds} + \frac{d\Psi_{ds}}{dt} - \omega\Psi_{ds};\\
U_{qs} &= R_s i_{qs} + \frac{d\Psi_{qs}}{dt} + \omega\Psi_{qs};\\
0 &= R_r i_{dr} + \frac{d\Psi_{dr}}{dt} + (\omega - \omega_r)\Psi_{qr};\\
0 &= R_r i_{qr} + \frac{d\Psi_{qr}}{dt} + (\omega - \omega_r)\Psi_{dr},
\end{aligned}
\tag{1}
$$

in which the first 2 equations refer to the asynchronous machine stator ("s" indices), and the last 2 equations refer to the rotor circuit ("r" indices), the machine's rotor circuit being considered a closed circuit (so $U_r = 0$); the stator/rotor equations system is completed with the equation for the electromagnetic torque (in instantaneous values):

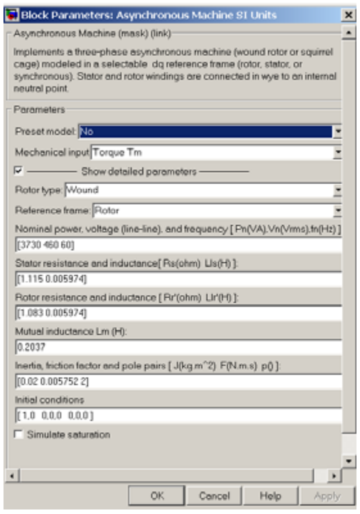
$$m_e = 1.5p\left(\Psi_{ds} i_{qs} - \Psi_{qs} i_{ds}\right), \tag{2}$$

and eventually with the linking equations between the magnetic flow and currents (admitting for a linear relation between the two measurements):

$$
\begin{aligned}
&\Psi_{ds} = L_s i_{ds} + L_m i_{dr}; -- \Psi_{qs} = L_s i_{qs} + L_m i_{qr};\\
&\Psi_{dr} = L_r i_{dr} + L_m i_{ds}; -- \Psi_{qr} = L_r i_{qr} + L_m i_{qs};\\
&L_s = L_{\sigma s} + L_m; -- L_r = L_{\sigma r} + L_m.
\end{aligned}
\tag{3}
$$

All equations are written in an orthogonal system of axis (d, q, 0) that rotates with an angular velocity ω; ω_r = p* ω_m , where p represents the number of pole pairs of the asynchronous machine and the other parameter is the rotor's angular velocity; the rest of the notations are specific to the asynchronous machine and are of no particular importance to the following (the rotor measurements are considered reported to the machine's stator).

Usually, to the previous equations, that are considered to be part of the "electrical component" of a drive, must be added the "mechanical component" of the drive, but the detailed presentation of the previous relations shows that the icon from Figure 1, that has input/output ports for yielding the values of the output measurements, must have the input signals travelling a complex chain of Simulink blocks that will always respect the equations (1) … (3). Forming such a chain of blocks is always complicated, and after forming it, some optimisations/adjustments must be made (regarding the necessary memory, computing time etc.) that can prove to be more complex than the first phase. The given block icon is associated, usually, with a dialog box that allows for entering the required work parameters, but also some specific elements (the initial values of the parameters, if the asynchronous machine is with a wound rotor or a squirrel-cage rotor, etc.). For example, the asynchronous machine has the dialog box presented in Figure 2. Basically, it is not required to detail other aspects from the point of view of the subject, but the most important fact that must be taken into consideration is that bringing the block in the work (simulation) space, the user is free from the work of constructing the asynchronous machine according to a certain equations/relations system.


Block Parameters: Asynchronous Machine SI Units

Asynchronous Machine (mask) (link)

Implements a three-phase asynchronous machine (wound rotor or squirrel cage) modeled in a selectable dq reference frame (rotor, stator, or synchronous). Stator and rotor windings are connected in wye to an internal neutral point.

Parameters

Preset model: No

Mechanical input: Torque Tm

[✓] Show detailed parameters

Rotor type: Wound

Reference frame: Rotor

Nominal power, voltage (line-line), and frequency [Pn(VA),Vn(Vrms),fn(Hz)]: [3730 460 60]

Stator resistance and inductance[Rs(ohm) Lls(H)]: [1.115 0.005974]

Rotor resistance and inductance [Rr'(ohm) Llr'(H)]: [1.083 0.005974]

Mutual inductance Lm (H): 0.2037

Inertia, friction factor and pole pairs [J(kg.m^2) F(N.m.s) p()]: [0.02 0.005752 2]

Initial conditions: [1,0 0,0,0 0,0,0]

[ ] Simulate saturation

Fig. 2 The dialog box for the icon in Figure 1

On the other hand, once the block is brought to the works space (meaning, taken from the corresponding library of blocks), basically, it can no longer be modified and put back in the library. So, if the asynchronous machine needs to have the addition of the current discharge effect (this element is specific to the asynchronous machine with squirrel-cage rotor with tall bars) by properly modifying the equations system (1)…(3), or the transversal magnetic saturation effect for more in depth studies or a more realistic modelling of the machine, then these modifications cannot be done on that specific block. Additionally, as an example, inside the same block there is a possibility to consider the magnetic saturation of the machine's iron, but this element is indecipherable (especially designed by the MathWorks corporation), although all other elements can be readable, after a few preliminary operations.

Finally, a customisation by reformatting the asynchronous machine block leads to the idea of forming a user created block for the asynchronous machine that can have the desired elements. Earlier, as a talking point, the block of an asynchronous machine was considered, but any other type of element (mechanical, hydraulic etc.) can replace it, without any of the conclusions needing any major modifications.

Using S-Functions

Basically, there are two options to form customised blocks inside the Simulink simulation environment: creating Embedded MATLAB blocks or S-Function blocks, both found in the User-Defined Functions library. An S-Function block allows the user to embed a certain algorithm that can be used in conjunction with other Simulink blocks.

In fact, specifically speaking, an S-Function block represents a description of a programming language for a Simulink block, written in Matlab, C, C++, Ada or Fortran, each model with its own specific syntax.

The User-Defined Function library (in MATLAB, version 7.4) contains 3 types of S-Function blocks:

- Level-2 M-File S-Function;
- S-Functions;
- S-Functions Builder.

The last 2 types allow for the use of the C programming language, while the last type allows for the automated generation of C language code for S-Functions; the first type refers to generating code in M language (MATLAB's proprietary language). Because C cannot be processed by MATLAB, the program compiles this type of S-Functions as MEX-files using the MEX utility from MATLAB.

If there is a program using C language, saved with the name NUME.c, then for it to be compatible with a S-Function block from Simulink, the command mex NUME.c must be run inside the command window from the MATLAB interface, thus creating an additional file named NUME.mexw32 that makes the link between C language and M language.

An advantage of using S-Functions is that it can construct a multipurpose block, being used several times inside a model, modifying the block parameters each time.

It was mentioned previously that the customizable blocks follow closely the programming language of Simulink blocks. To this extent, the following phases are found:

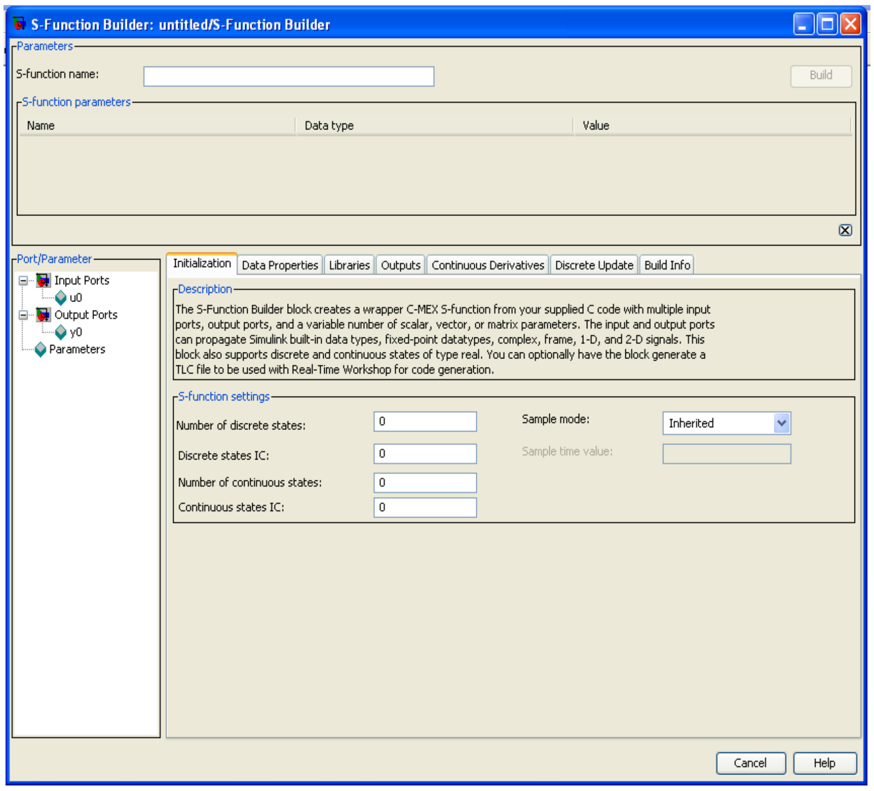
- initialization phase;
- setting the correct sample-time;
- calculating the outputs at each major time-step;
- refreshing the discrete states at each major time-step;

- the integration phase, which is applied to the models with continuous states and/or zero-crossings without samples; if there are continuous states, then Simulink takes in the output and the derivative formation of the S-Function at the minor time-step.

Level-2 M-File S-Functions represent a syntax that resembles with the C language syntax, while the S-Function block, although it operates in stages similar to the first type, its code is generated with C language syntax.

The S-Function Builder block creates a C-MEX S-Function starting from C code, with multiple input/output ports and a number of scalar, vector and matrix parameters. The input/output ports can propagate Simulink signals and one-dimensional or bidimensional complex signals. This block also supports discrete and continuous real states, so this block can be considered as the most evolved S-Function block.

The Builder allows for the fast and simple creation of an S-Function, the user setting the input/output ports and the system states, the type and dimensions and the mathematical equations that define the S-Function. The graphical interface of the S-Function Builder is shown in Figure 3.



S-function name:

The S-Function Builder block creates a wrapper C-MEX S-function from your supplied C code with multiple input ports, output ports, and a variable number of scalar, vector, or matrix parameters. The input and output ports can propagate Simulink built-in data types, fixed-point datatypes, complex, frame, 1-D, and 2-D signals. This block also supports discrete and continuous states of type real. You can optionally have the block generate a TLC file to be used with Real-Time Workshop for code generation.

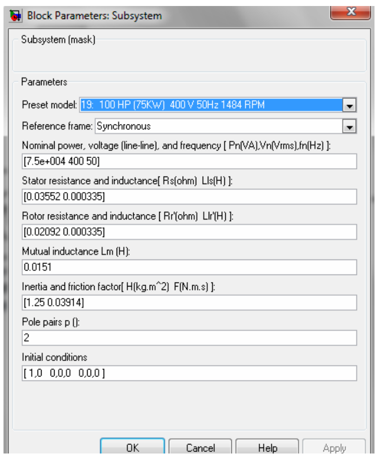
Number of discrete states: 0
Discrete states IC: 0
Number of continuous states: 0
Continuous states IC: 0
Sample mode: Inherited
Sample time value:

Fig.3 The Graphical interface for the S-Function builder block.

This block contains (in the MATLAB 7.4 version) several tabs (Initialization, Data Properties, Outputs, Continuous Derivatives, Discrete Update, among the most important ones) that allow for defining the S-Function by completing and operating the fields found within each tab. After the complete definition of the S-Function, it can be used as any other block inside a Simulink model. This option eliminates any difficulty of editing an S-Function by using the correct language and syntax in accordance with a specific programming language. The Build button (fig. 3, top-right corner) leads to the creation of the S-Function.

Creating the Asynchronous Machine Block Using S-Functions

At its core, an S-Function has a set of call-back methods (their number is relatively large) that execute the tasks for every stage of simulation. During a model's simulation, at each stage of the simulation, Simulink calls the most appropriate method for each S-Function block from inside that model. Based on this general work program, template S-Functions can be created for M and MEX file types.



Subsystem (mask)

Parameters

Preset model: 19: 100 HP (75KW) 400 V 50Hz 1484 RPM

Reference frame: Synchronous

Nominal power, voltage (line-line), and frequency [Pn(VA),Vn(Vrms),fn(Hz)]:
[7.5e+004 400 50]

Stator resistance and inductance[Rs(ohm) Lls(H)]:
[0.03552 0.000335]

Rotor resistance and inductance [Rr'(ohm) Llr'(H)]:
[0.02092 0.000335]

Mutual inductance Lm (H):
0.0151

Inertia and friction factor[H(kg.m^2) F(N.m.s)]:
[1.25 0.03914]

Pole pairs p ():
2

Initial conditions
[1,0 0,0,0 0,0,0]

Fig.4 Asynchronous machine dialog box.

As with the case of a normal SPS block, creating an S-Function block for an asynchronous motor can start with creating the mask or the dialog box, which is very useful to the user and can be seen in Figure 4.

Creating this dialog box is the same as for any other Simulink block, and it isn't different from the mask of the asynchronous machine block take out from the SPS library (done especially so it can be easier to be compared), but it was constructed according to the user's needs and specifications. Thus, there are present some pre-set motors and some fields used to insert some motor parameters and initial conditions.

The created S-Function can use a pre-set type of asynchronous machine, or any other asynchronous machine with specific parameters.

The source code for the S-Function contains more specific details for the type of the asynchronous motor, according to a specific template. For example, the setup function for the block has the following code:

```
function setup(block)
  block.NumInputPorts = 3;
  block.NumOutputPorts = 21;
  block.SetPreCompPortInfoToDefaults;
  block.SampleTimes = [-1 0];
  block.NumContStates = 6;
  block.RegBlockMethod('Outputs', @Output);
  block.RegBlockMethod('InitializeConditions',    @InitConditions);
  block.RegBlockMethod('Derivatives',             @Derivative);
```

and converting the parameters in pu has the following code :

```
%% Transformation in pu %%
web_psb = 2*pi*fn;
base_power = Pn/3;
base_voltage = Vn/sqrt(3);
base_current = base_power/base_voltage;
base_icurrent = base_current*sqrt(2);
base_impedance = base_voltage/base_current;
base_resistance = base_voltage/base_current;
base_inductance = base_impedance/(2*pi*fn);
base_speed = web_psb/PolePairs;
base_torque = (base_power*3)/base_speed;
inertia_constant = ((1/2)*H*(base_speed^2))/Pn;
F_pu = (F*base_speed^2)/Pn;
Nb_pu = base_speed;
Tb_pu = base_torque;
Vb_pu = 2*base_power/base_icurrent;
```

```
ib_pu = base_icurrent;
phib_pu = (sqrt(2)*base_voltage)/web_psb;
Rs = Rs0/base_resistance;
Rr = Rr0/base_resistance;
Lls = Lls0/base_inductance;
Llr = Llr0/base_inductance;
La_pu = ((Pn*fn)/(Vn^2))*((Llr0*Lls0*Lm)/(Llr0+Lls0+4*Lm)) *
        ((8*pi)/(Lls0+Llr0));
```

in which the equations (prepared above) are mentioned according to which the conversion is done, so that the code stages and sequences can be better highlighted; if need be, each line of code can be commented so that it contains all de necessary details. Finally, the connections of the input and output ports for the S-Function block that are hidden under the mask from Figure 4 can be seen in Figure 5.

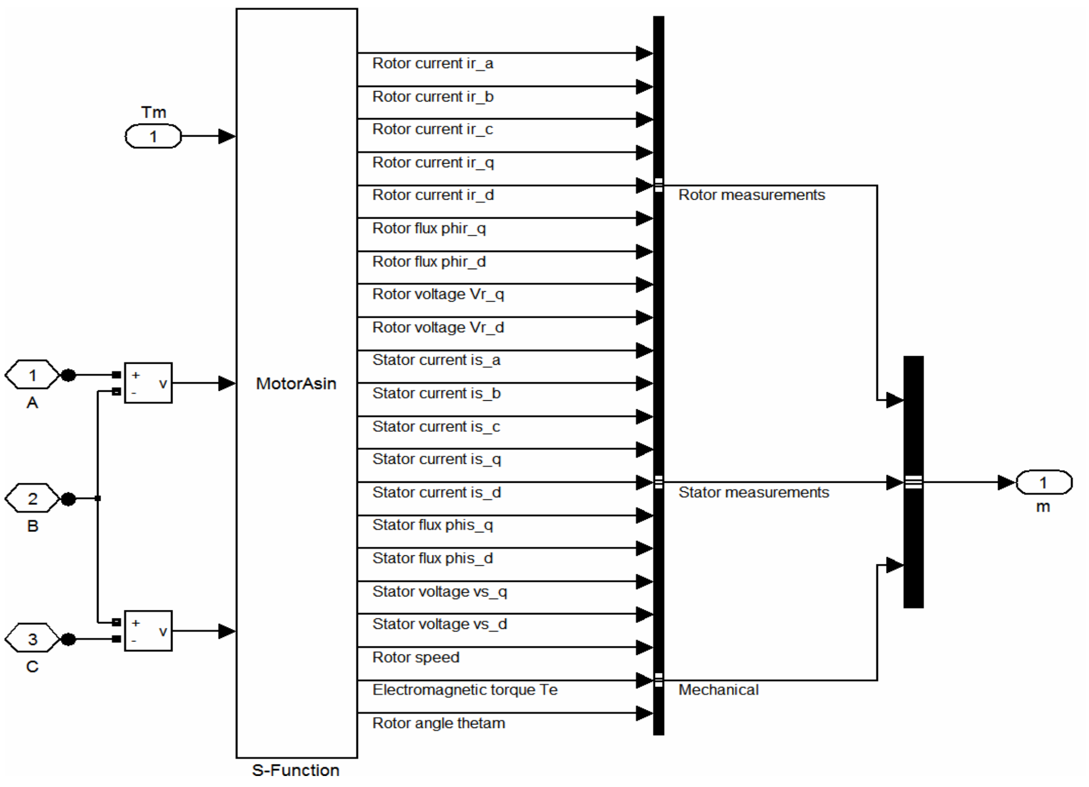


Fig. 5. Input/output connections diagram for the asynchronous machine S-Function

Figure 5 shows that there are 21 outputs for the asynchronous machine represented with an S-Function, identical to the number for the block from the SPS library; actually the diagram from Figure 5 is modelled (using an S-Function) after the asynchronous machine block from SPS library, so that a comparison between the two can be more easily drawn.

Comparison between the Asynchronous Machine Block from the SimPowerSystems (SPS) Library and the S-Function Block

The comparison between the asynchronous machine block from the SPS library and the one represented by an S-Function is done by simultaneously simulating the two blocks.
The model for that simultaneous simulation is shown in Figure 6.

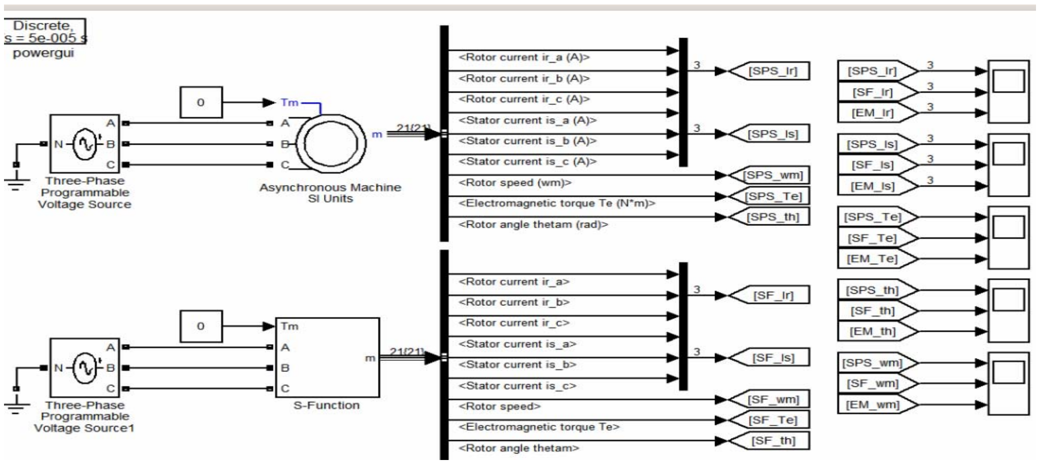


Fig. 6. The model used to simultaneously simulate the SPS block and S-Function for the asynchronous machine

The graphs can be shown on the scopes from the diagram in Figure 6 for multiple signals, but for the moment, only the graphs for the electromagnetic torque (fig. 7) and speed (fig. 8) are shown below.

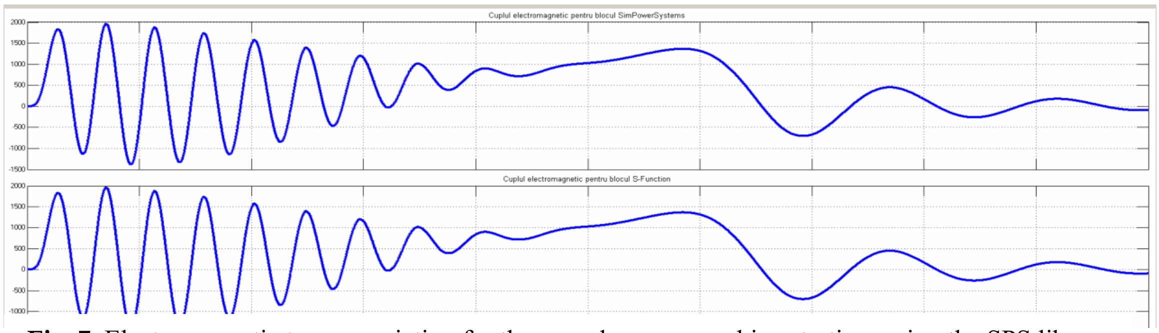


Fig. 7. Electromagnetic torque variation for the asynchronous machine starting, using the SPS library block and the S-Function block

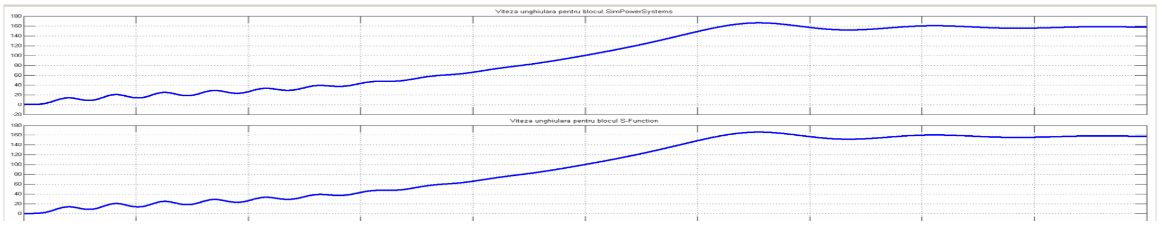


Fig. 8. The rotational speed graphs for the asynchronous machine according to the two types of block.

Figures 7 and 8 show that the results of simulating the 2 block versions for an asynchronous machine are practical identical, so the S-Function version can be used any time instead of the block from the SPS library. This can be considered as a very important concept, because the asynchronous machine block from the SPS library can be used exclusively with other blocks from the SPS library which, however vast it may be, it still has its limitations. Moreover, the customization of the asynchronous machine block of type S-Function can be extended as to consider the effect of current discharge (that can affect the rotor phase's resistance and reactance), the effect of longitudinal/transversal magnetic saturation, that are already not included (these are effects that happen inside a real asynchronous machine) in the block from the SPS library.

An Interconnection with Maple

The MapleSoft Company developed several packages to extend the use of their Maple products. Among these are: MapleSim, MapleSim Connector, BlockImporter, Maple Toolbox for MATLAB.

The first product is designed to model physical systems, being based on the symbolic technology used by Maple and can be considered an equivalent for Simulink from MATLAB, although some considerable differences can be noted in regards to the computational method used and from the point of view of the graphical user interface. Yet the two products are not completely incompatible, because a line of communication can be made between Simulink and MapleSim using the MapleSim Connector and BlockImporter products.

MapleSim Connector can create S-Functions compatible with Simulink and BlockImporter has a somewhat inverse function, importing a model from Simulink and converting it to mathematical expressions specific to Maple and MapleSim, that can be processed and simplified and returned to Simulink.

There already are transfer options for Maple ↔ LabVIEW, as there are corresponding communication channels for MATLAB ↔ LabVIEW, so real systems can be modelled, simulated and debugged in the three large programming environments on a wide range of domains.

Conclusions

The S-Function is a type of block for the Simulink simulation environment from MATLAB that can be conveniently customised for any dynamic element (electrical, mechanical, hydraulic or thermic), using a set of default S-Function call-back methods that execute each simulation stage's task for the given element. In previous examples, it was talked about an asynchronous electrical motor (a subject closer to the authors' expertise), but any dynamic element can be considered, defined by a system of relations and differential equations. The S-Function Builder block allows for the automatic redacting and editing of S-Functions, according to a necessary template, straight in the C environment, although an S-Function can generally be created in different environments, such as MATLAB, C, C++, Ada or Fortran.

The main aspect of using an S-Function is that it can use any Simulink block from a block diagram, however complex, and when using the Builder option, it would be even easier to create a block with multiple input and output ports and with a number of scalar, vector or matrix parameters.

Through a relatively recent breakthrough that allows for the possibility of communicating with the Maple environment, there is the option of passing an entire model from Simulink to Maple, and converting it into mathematical expressions specific to Maple and MapleSim, that can be returned to Simulink as S-Functions.

Customizing the definition of dynamic elements, of different types and eventually forming the user's own library with these elements can be considered the main advantage of using S-Functions.

References

1. L u p u , C . D . – *Introducere in folosirea funcţiilor Embedded MATLAB* şi *S-Function la simularea unor acţionări electomecanice cu motoare asincrone*, Proiect de diplomă, coordinator: prof.dr.ing. Boris Siro, UPG Ploieşti, 2011.
2. * * * – *Overview of S-Function/How S-Function work*, Simulink Documentation.
3. * * * – *Creating Block Mask/Mask Editor*, Simulink Documentation.
4. * * * – *Writing S-Function in C/S-Function Builder Dialog Box*, Simulink Documentation.
5. * * * – http:/www.Maplesoft.com/products/maple/features/feature_detail.aspx?fi d=6721.
6. * * * – *Blocks - by Category/Machines/Asynchronous Machine*, SimPowerSystems Documentation.
7. * * * – *External Interfaces/Calling C and Fortran Programs from MATLAB*, MATLAB Documentation.

Utilitatea folosirii *S-functions* la definirea unor elemente complexe

Rezumat

În schemele de simulare din Simulink/SPS există blocuri complexe cum ar fi motoarele electrice de acţionare. De regulă, aceste blocuri au masca şi boxa lor de dialog relativ simple, dar în „spatele” acestora se ascund o multitudine de blocuri Simulink, care pe baza ecuaţiilor de funcţionare specifice, realizează (cu anumite ipoteze simplificatoare) funcţionarea elementului de schemă respectiv. Folosirea S-functions aduce o simplificare semnificativă în definirea unor astfel de elemente complexe de schemă.