# C CODE SIMULATION BLOCKSET

*Andrzej Lara*

Motorola Software Development Center,
Jodłowa 13, 30-252 Kraków, Poland

**Abstract:** An approach offering a software developer an easy way of taking the full benefit of powerful Simulink/Matlab environment is presented. The method for connecting some actual C code with Simulink simulation without use of complex S-function is demonstrated. In addition a technique for combining a Simulink diagram and C code based model is presented. At the end the paper is supported by an example.

## I INTRODUCTION

The idea of taking benefit of simulation while developing software is one of the goals of Matlab/Simulink. The tool offers comprehensible and diverse functionality targeted toward efficient modeling of software execution. However the world is always on permanent move and everything, which has been consider as good, soon or later will be perceived as something to be improved, especially taking into consideration the most recent advances in the technical field. This paper proposes a methodology incorporated into the existing Matlab/Simulink tool set, which is aimed to help software developers working with Matlab/Simulink and to use the tools more efficiently.

The work originated as response to needs arisen from development of software in motor control field using DSP devices, see [6], [7]. Therefore the presented methodology is best suited for similar applications. However it seems to be a fairly easy task to adopt it to different field.

Let us consider a common methodology of developing software. Very often so-called V model is used and is characterized mainly by division of the whole software creation cycle into two major parts. The parts are often shown as a falling and rising line (Figure 1). The phases on both cycle lines correspond one to each other. The presented V model is very simple but sufficient for paper's purposes. More detailed information can be found in references, see [4] and [5].

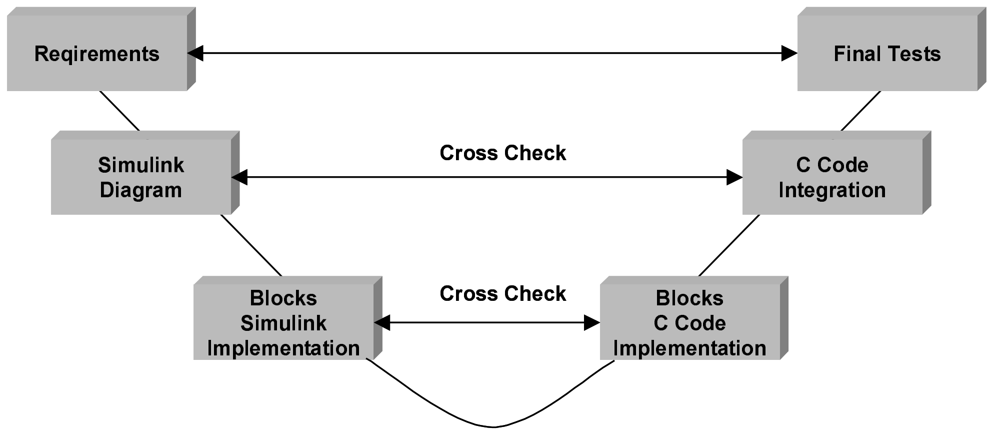


*Figure 1 Simple V Model*

From the above presented software process two issues arise. At first how to integrate C code in the simulation and at second how to arrange simulation/coding so that the backward cross check is done proficiently.

The current releases of Matlab/Simulink offer well defined and rich in functionality API, which allows whatever imaginable as far as connection between a user C code and the tool is taken into account. This capability is sacrificed by relative large amount of work that must be put into preparation of such a code for use with Matlab. This is not an issue for one or two times, however if more software pieces are in the workflow, then it is becoming tedious especially that the work to be done is the same or very similar all the times. Let's consider a very usual case when a Simulink block should represent a call to a function, having a defined number of input and output arguments. The work which is needed to port such a function with the Simulink block is very similar regardless of how the input and output port number of the function may differ and which method is used to invoke the function with regard to Matlab/Simulink sampling algorithm. This reasoning leads to a conclusion that another piece of software, which is able to perform this process automatically, would save a lot of time.

If at some point of time two models exist, for example one as Simulink diagram and another one as C code, a cross check which would ensure, that both implementation behave the same, is desirable. The check can be done efficiently if a developer can instantly switch between both models while leaving higher level Simulink diagram topology unchanged.

These simple ideas led to this work and resulted in the in the title mentioned C Code Simulation Blockset and the following sections treat these subjects in more detail.

## II CALLING C CODE

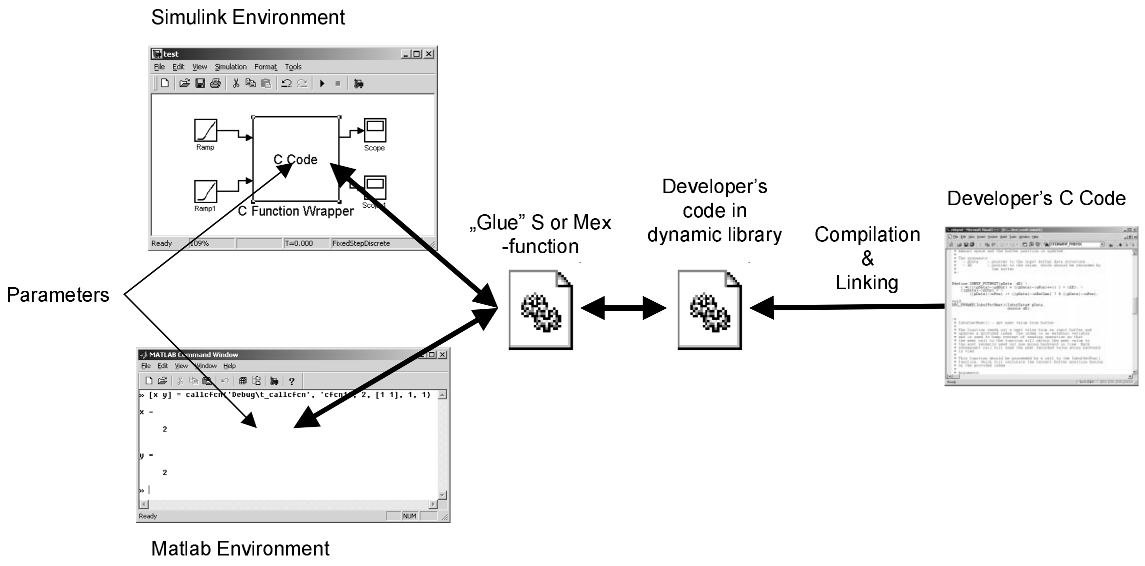
In section I above reasoning was provided, which justifies presence of some piece of software, which plays a role of a broker between a Simulink S-function and a developer C code. This software, acting as a "glue" code, is aimed to ease the task connected with adopting the rich functionality of the S-function API to a specific need.

The implementation of this "glue" code is straightforward and is depicted in Figure 2. A developer needs to provide a dynamic library containing compiled on a host platform functions under development. On the other side the C Code Simulation Blockset provides an S-function, whose task is to load the dynamic library, find a particular function and call it with appropriate arguments and at appropriate sample times.

The S-function can be placed on the Simulink diagram as a block. The S-function block can be then provided by arguments specifying among others the library, function, its inputs/outputs and the sampling time.

The Figure 2 shows also how a C function can be called from within a Matlab window (see left bottom corner). The working rule is very similar to that one described above.

It should be also noticed, that this method enables debugging the compiled C code under a debugger running on host platform. In this case the Matlab/Simulink needs to be run from within the debugger.



```
>> [x y] = callcfcn('Debug\t_callcfcn', 'cfcn1', 2, [1 1], 1, 1)

x =

     2


y =

     2

>> |
```

*Figure 2 Calling C Code From Simulink/Matlab*

## III COMBINING C CODE AND SIMULINK DIAGRAMS

In the V model it is essential to ensure the backward check between corresponding development phases. Continuing this thought and adopting it to the demonstrated approach a C implementation needs a cross check with models created as Simulink diagrams in previous phases. The check eases the task of ensuring correctness and consistency throughout the whole project.

The method, which is provided for this purpose, is based on a block, by which a developer can quickly switch between different model representations, like for example C code or Simulink diagram implementations. This feature is achieved through a parameter "view" of the block. The parameter determines a "view" of a block. A "view" is meant here as a block representations, which is established through a connection to another Simulink block. By choice of different views, a developer actually chooses different blocks arrangements. A view can be for example a Simulink diagram or a function scaled to +/- 1 value range or a function scaled to 16 bits length values or others. The example of applying the block in particular Simulink diagram is described in detail in section IV.

The Views Block is similar in term of operation to already available block from the Simulink library – Configurable Subsystem. However due to different rule of operation and due to some additional features it has been named differently.

## IV EXAMPLE – LMS ADAPTIVE PROCESSOR

To better illustrate the above description let us consider a development of a C implementation of LMS adaptive processor (see [3]). The whole process of developing a C code is depicted in Figure 3.

The starting point is the higher-level diagram, in which the model will be simulated and checked. It simply consists of a few sources supplying signals to the LMS processor and a few sinks for observation of output signals. The next steps are shown in circles with numbers inside. In the middle of the Simulink diagram a block can be found, which represent the LMS processor.

The first approach is to sketch quickly the LMS processor with use of basic Simulink blocks. This is marked with number 1. It is also clear from the beginning that the LMS processor model can be modularized by encapsulating adaptive coefficient into a separate model. The adaptive coefficient model is then instantiated several times within the higher level LMS processor model, as it is shown in step 1a. This first approach can be quickly finished and simulated.

Once a developer is satisfied with current result, he or she can proceed to the next step by implementing the adaptive coefficient model using C language. Here the C Code Simulation Blockset is coming in play by offering an easy and fast way of incorporating a developed C code into existing Simulink simulation. This can be seen in step marked with 2. To furthermore ease the use of the C code the Views Block is used to enable instant switching between the models created as Simulink diagram and the model created as C code.

After check the adaptive coefficient code can be reused in following development phases as well as can be used back within this step whenever a doubt appears regarding this piece of code suitability or correctness. Once the code is checked a developer may proceed to the next step.

As next the LMS processor is modeled as mix of C code and Simulink diagram. The higher level of the LMS processor is still simulated as Simulink diagram, but all the inside blocks, whenever applicable, are represented by a corresponding C code. This step is marked with number 3. Indeed what is happening here is partitioning of the whole software application into smaller modules.

It should be noticed that the code developed in step 2 (adaptive coefficient) is re-used here and is incorporated somewhere in the application.

Similarly to the step 2, the Views Block is used to instantly switch between the model created as Simulink diagram (see step 1 and 1a) and the model in the current step (mix of C code and Simulink diagram). Again after a developer is satisfied with results he or she may progress to the next step.

Finally the LMS processor can be implemented entirely in C code, which is marked with the number 4. The C code is easily linked with Simulink simulation and the Views Block is used to switch among all versions: (step 1, 1a and 2) the Simulink diagram, (step 3) the mix of Simulink diagram and C code and (step 4) full C code model.

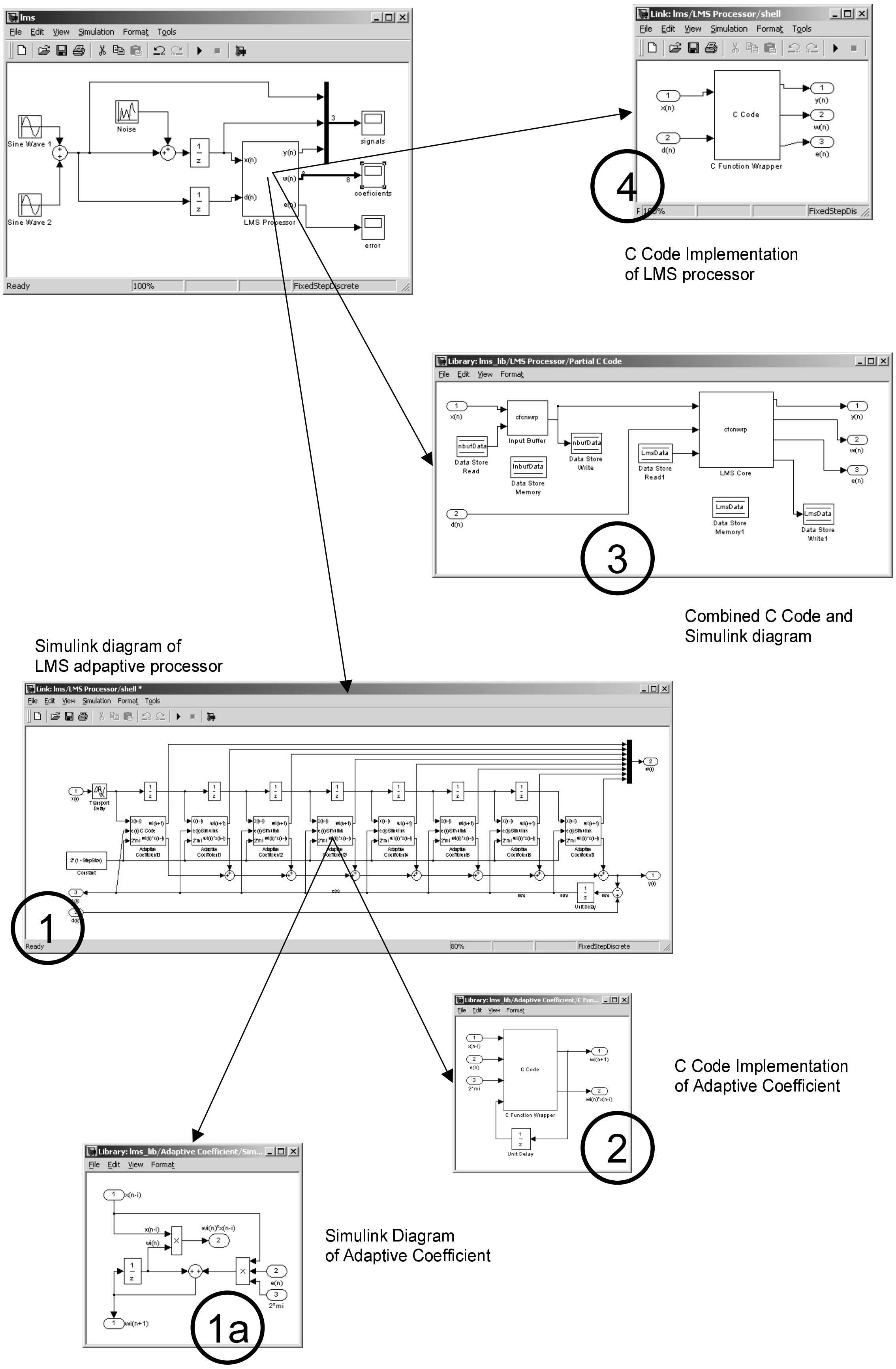

lms
Sine Wave 1
Sine Wave 2
Noise
LMS Processor
x(n)
d(n)
y(n)
wi(n)
e(n)
signals
coeficients
error
Ready
100%
FixedStepDiscrete
Link: lms/LMS Processor/shell
C Code
C Function Wrapper
4
C Code Implementation of LMS processor
Library: lms_lib/LMS Processor/Partial C Code
Input Buffer
Data Store Read
Data Store Memory
Data Store Write
Data Store Read1
LMS Core
Data Store Memory1
Data Store Write1
3
Combined C Code and Simulink diagram
Simulink diagram of LMS adpaptive processor
Link: lms/LMS Processor/shell *
Transport Delay
Constant
2*(1 - StepSize)
Unit Delay
1
80%
Library: lms_lib/Adaptive Coefficient/C Fun...
x(n-i)
2*mi
wi(n+1)
wi(n)*x(n-i)
Unit Delay
2
C Code Implementation of Adaptive Coefficient
Library: lms_lib/Adaptive Coefficient/Sim...
1a
Simulink Diagram of Adaptive Coefficient

*Figure 3 Development of LMS Adaptive Processor Using C Code Simulation Blockset*

The above-presented approach demonstrates a few advantages. At first a developer can: instantly switch among all versions and immediately check any discrepancies in operation. At second it is very easy to go back in development and re-check code. And at last but not the least, all the c code can be always reused through the whole simulation and also in the real application.

## V CONCLUSION

The presented approach of simulating C code within Simulink/Matlab environment is a powerful tool for software developers willing to use simulation in their work. By using the presented methodology the actual, developed C code and Simulink diagrams can be combined in any possible combination and at all software development phases. In addition by using the C Code simulation blockset a developer does not have to learn and use the Simulink S-function and Matlab Mex API, which greatly eases deployment of the tool.

## VI REFERENCES

[1] *Matlab - Using Matlab*. The Mathworks, Inc., Version 5, 1999.
[2] *Simulink - Writing S-functions*. The Mathworks, Inc., Version 4, 2000.
[3] *Adaptive Signal Processing*, Bernard Widrow, Samuel D. Stearns, Prentice Hall ISBN: 0130040290, March 15, 1985 (1 edition)
[4] *Automating software specification, design and synthesis for computer aided control system design tools* Ranville, S.; Bostic, D.; Toeppe, S. Digital Avionics Systems Conferences, 2000. Proceedings. DASC. The 19th, Volume: 1, 2000 Page(s): 4C3/1 -4C312 vol.1
[5] *A methodological approach to the requirement specification of embedded systems* Lattemann, F.; Lehmann, E. Format Engineering Methods, 1997. Proceedings, First IEEE International Conference on , 1997 Page(s): 183 –191
[6] http://www.motorola.com/SPS/DSP/documentation/DSP56800.html
[7] DSP 56800 application notes: AN1909/D, AN1911/D, AN1913/D, AN1916/D, see also web pages: http://e-www.motorola.com/brdata/PDFDB/docs/AN1909.pdf, (for other application notes the last part of the url path is changing, for example for AN1911/D the AN1909.pdf should be changed to AN1911.pdf)

*Authors:*
Andrzej Lara, MSc
Motorola Software Development Center
Jodłowa 13, 30-252 Kraków
Poland
Andrzej.Lara@motorola.com