

# The Development of a MATLAB Instrumentation Tutor\*

FASIL MUDDEEN and KESTER GABRIEL

Department of Electrical and Computer Engineering, University of the West Indies, Trinidad, West Indies. E-mail: fmuddeen@eng.uwi.tt

*This paper describes a MATLAB/Simulink blockset called the MATLAB Instrumentation Tutor, currently being developed in the Department of Electrical and Computer Engineering at the University of the West Indies. The Tutor is being designed to support the teaching of a final year Instrumentation Systems course. The system will enable students to simulate various scenarios involving sensors, instrumentation amplifiers, filters, analog to digital and digital to analog conversions, sampling and quantizing, interference, error budget calculations, and the effect of random and coherent noise on system performance. Real parameter values taken from component data sheets will be used in this system so as to form an environment that will enable simulation of typically encountered real world instrumentation problems. The development of the necessary S-functions needed to accomplish this will also be discussed.*

## INTRODUCTION

THE FINAL YEAR Instrumentation Systems Course, EE35E, at the University of the West Indies (UWI), concentrates on the performance aspect of Instrumentation Systems (IS). At this point in the curriculum, students are already adequately familiar with the operation of the components of an IS—the transducers, signal conditioning elements, signal processing elements and the display elements. EE35E looks at the effect of each of these elements on the overall IS performance. Specifically, characteristics such as signal to noise ratios, signal bandwidth, quantization and error rates are discussed, leading to the development of an overall system error budget.

In covering the course material, many examples depicting various instrumentation scenarios are solved. Most of the problems can be solved with a calculator and ‘what-if’ scenarios are explored with a spreadsheet. However it was recognized that neither approach was flexible enough to allow an entire system to be quickly tested and simulated under conditions where components, inputs, noise levels and sources, signals, filters characteristics and so on were changing.

This type of evaluation required an environment which would allow easy manipulation of parameters and components, while simultaneously permitting complex calculations to be performed. UWI has had long experience with MATLAB, especially its use in control system applications. In addition, students are trained in MATLAB from their first year and are quite comfortable with it by the time the final year courses are taken. One particularly attractive feature of

MATLAB was its Simulink environment, which provided an intuitive, icon-based work area for designing and simulating systems. It was therefore decided to develop a blockset for use with Simulink that would support and enhance our IS teaching.

The following discussion is based on the results of a project conducted by Mr. Kester Gabriel, a final year Electrical engineering student at UWI, over a period of nine months under the supervision of Mr Fasil Muddeen.

## WHAT IS A SIMULINK BLOCKSET?

A Simulink Blockset is a collection of block libraries, defined later on, grouped according to specific applications. For example, the Signal Processing Toolbox contains blocksets specific to signal processing applications—filters, transforms, sources and so on. Also included with most blocksets are comprehensive help files, examples and where possible, demos.

In developing the prototype of the IS blockset, it was decided that as a minimum, the blockset should contain blocks for typical sensors, differential and instrumentation amplifiers, filters, general signal and noise sources, data conversion blocks and data presentation blocks.

### *System development*

Simulink builds up a system model using objects called blocks, interconnected by lines [1]. A block comprises a set of inputs, states and outputs as shown in Fig. 1. Note that this figure uses some blocks from the prototype IS blockset. *Block parameters* define properties such as the name, colour, handle or number of ports of the block.

\* Accepted 2 April 2005.

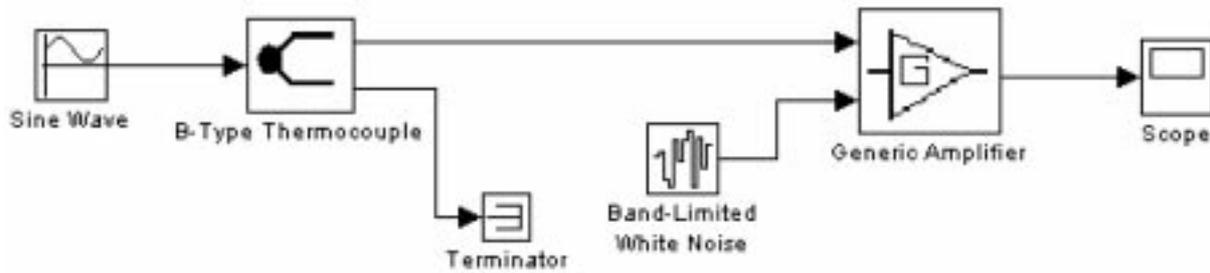


Fig. 1. Typical Simulink blocks from the IS blockset

Signals are the values that appear at the output of the block when a simulation model is run.

At this point, it should be noted that Simulink handles signal information differently from parameter information. Signal values are updated every time step; there are no MATLAB defined functions to automatically give a parameter the value of a signal. The mask parameter values are not updated every time step. They are updated when the user double-clicks on the block to bring up the mask interface or when the user selects ‘OK’ or ‘Apply’ on that interface. The updating of the mask parameters are event-triggered. Mask parameters can also be easily read and set within the MATLAB workspace using MATLAB defined functions.

*Development of a block*

The first step in the development of any block is to analyse the inputs, outputs and required functionality for it. If the output is dependent on input signals, then the block must be implemented as an *S-Function*. An *S-Function* is a computer language description of a Simulink block and can be written as a MATLAB M-file or in C, C++, Ada or Fortran [1]. If the output is dependent on parameters only, then the user may choose to implement the solution via a block mask or *S-Function*. A set of selection criteria is given in Table 1.

For example, in Fig. 1, the generic amplifier model has an error performance [2] given by the following equation:

Table 1. Criteria for deciding if to implement required functionality as an S-Function or in a Block Mask

Selection criterion	Function implemented	
	In a block mask	As an S-function
Any input is a signal	<b>No</b> M-code does not have real-time access to the signal data. Signal data from the previous simulation may be obtained but a real-time solution was desired.	<b>Yes</b> Signal information read in through the input, parameter information passed from mask through the S-Function block. All inputs are parameters
<b>All inputs are parameters</b>	<b>Yes</b> Parameter data is easily manipulated in the mask; the most recent value of the output parameter is obtained when the user accesses the mask prompts.	<b>Yes</b> Parameter information can be passed to the S-Function and used to drive the output signal.
The block functionality is dependent on MATLAB functions eg. Laplace Transform expression for a filter.	<b>Yes</b> The mask uses M-code and therefore has easy access to MATLAB-defined functions.	<b>Avoid</b> S-Functions do have the capability to call some MATLAB functions using the mexCallMatlab external reference, however, the preparation of the data for use in this form makes this option prohibitive.
Output parameters to be set, for example, in an edit box in the mask.	<b>Yes</b> If the output is a parameter, it can be displayed in the block mask using the set_param command.	<b>Avoid</b> There are three possible ways to attempt this; 1. Using mxSetPr—difficult 2. Using _ssSetScnParam—This is not publicly supported and shouldn't be used, 3. A combination of the set_param and mexCallMatlab commands—Passing the pointers to the the set_param command for the actual parameters to set proved difficult.
The output must be in signal form	<b>No</b> Signal data cannot be set using MATLAB functions	<b>Yes</b> The S-Function allows easy manipulation of signal data.

$$\varepsilon_{amp} = V_{os} + I_{os}R_s + \frac{f(A_v)V_{FS}}{A_{v,diff}} + \sqrt{\left[\left(\frac{d(V_{os})}{dt} \cdot \Delta T\right)^2 + \left(\frac{V_{cm}}{CMRR}\right)^2 + (6.6V_n\sqrt{f_{hi}})^2 + \left(\frac{d(A_v)}{dt} \cdot \Delta T \cdot \frac{V_{FS}}{A_{v,diff}}\right)^2\right]} \quad (1)$$

The variables are defined in Table 2. This equation was implemented as an S-function.

#### Simulink workspaces

Four different workspaces are defined in the MATLAB/Simulink environment. The MATLAB software is installed over an operating system, Windows or Linux for example. The variables defined in the MATLAB workspace, visible in the workspace window, are completely separate from the operating system environment, and can only be accessed using methods provided by MATLAB. Likewise, the Simulink workspace is separate from the MATLAB workspace. The Simulink workspace can be seen when running the model in the debug mode. Within Simulink, the Mask workspace is separate from the Block workspace. Separating the variables into locally defined workspaces makes the overall software more efficient, allows repetition of variable names and allows for control of how each workspace is accessed.

Figure 2 illustrates the workspaces. MATLAB provides methods for accessing its different locally defined workspaces called Callback Functions. These functions are readily seen if the user creates a GUI, or passes information between an S-Function, Simulink block or a block mask.

Figure 3 illustrates some of the typical data communication that occurs between Simulink models using a two-model example. The communication between the models is facilitated through parameters, signals and callback functions:

- **Communicating parameter information.** Parameter information is passed through what may be described as the Mask Workspace. Model 2 obtains the block name of Model 1 using the *Port Connectivity* struct, and then accesses Model 1's parameters using the 'get\_param' command and the block name.

Table 2. Description of variables used in Equation (1)

$V_{os}$	Input offset voltage
$\frac{dV_{os}}{dt}$	Temperature drift coefficient
$I_{os}$	Input offset current
$V_n$	Input referred Noise voltage
$f(A_v)$	Gain Non-linearity
$\frac{dA_v}{dt}$	Gain temperature drift coefficient
CMRR	Common-mode rejection ratio
$R_{i,cm}$	Common Mode input resistance
$R_{i,diff}$	Differential Mode input resistance

- **Communicating signal information.** The user does not have to implement signal data access as MATLAB routes the signal data depending on how the user connects the signal lines. Even in the S-Function, there are set methods for accessing the input and output pointers to the signal data.
- **Callback functions.** Simulink has to keep track of several parameters such as block position, name and connectivity. For the simple six-block system shown in Fig. 1, the generic amplifier alone has 193 parameters that fully describe it: 99 parameters for the unmasked S-Function block, 35 due to the presence of a mask and 59 user-defined prompts. The S-Function contains callback functions as well as SimStruct macros, which allow Simulink to reference the required information and set the block's output signal data. Each Simulink model and S-function has a SimStruct associated with it, defining all the data types contained within the model. The SimStruct (short for Simulink Struct) [3] is a large yet well organised construct that contains the vast amount of information required for Simulink.

## OVERALL SYSTEM DESIGN

The overall system design was carried out using a three-tiered approach illustrated in Fig. 4. The conceptual design began at the middle tier. The selection process summarised in Table 1, identified the functionalities implemented in masks or in

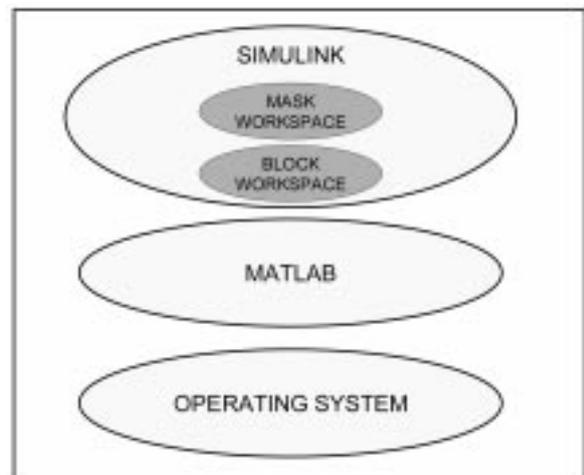


Fig. 2. MATLAB Simulink environment.

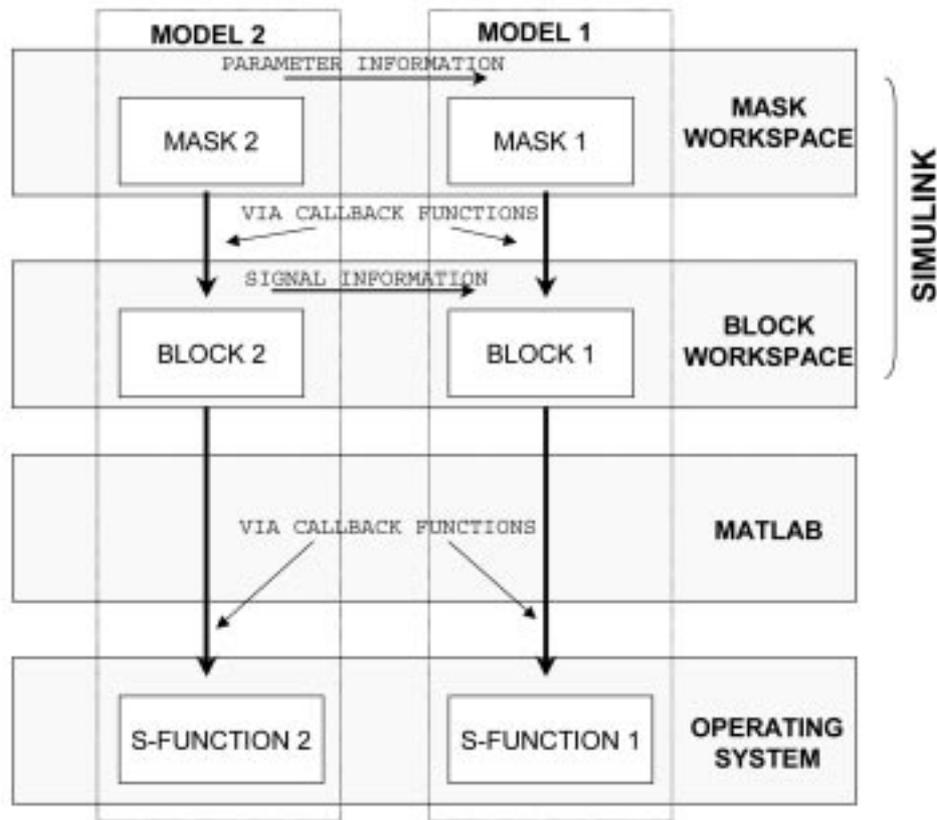


Fig. 3. Simulink data communication.

S-Functions. This was a complex process since information for the output signals of the blocks may have come from both signal as well as parameter data and that the blocks themselves may eventually be MATLAB S-Functions.

#### *Middle tier design: routing and filtering signals*

There were two main components of the middle tier (the signal workspace). The first component was responsible for producing the output signal from the input signal and possibly the input parameters. This was accomplished using a one-block S-Function (for example the B-Type Thermocouple) or using a combination of blocks (for example the Generic Amplifier). The sensors and signal conditioning models were all bandlimited by including a first-order, low-pass filter in each. Low frequency effects were not considered.

#### *Bottom tier design: the S-function*

Most of the functionality of an S-function is devoted to ensuring that the data entering, leaving and used in processing within the function is correct, that is of the required type and dimension. C-MEX S-Functions [4] were written to create the blockset. MEX files were used since they provided a large number of available callback functions and direct access to the SimStructs. The C language was chosen since engineering students were familiar with it. The first MEX-file was created using the Automatic S-Function Builder block provided

in the Simulink library. This file was coded in C and provided an easy template to learn from. The thermocouple function in Fig. 1 and the amplifier error budget Equation (3), were implemented using an S-function written in C.

#### *Top tier: masking the blocks*

Having completed the 'first-pass' block design at the signal workspace level, it was then masked. A mask, as defined in [5], is a custom user interface for a subsystem that hides the subsystem's contents, making it appear to the user as an atomic block with its own icon and parameter dialog box. Through the Simulink Mask editor, Fig. 5, the parameter list for the subsystem was entered, as well as any initialization commands. Recall that the mask workspace is part of the Simulink block object model (see Fig. 2) and all of its parameters and their values are listed as members of the particular block.

The use of dynamic dialogs for masked blocks allowed the user to make visible those mask prompts which were needed and to hide those which were to be used later. The mask prompts were grouped in a logical way to increase efficiency and user comfort. For example, the Generic Amplifier block had 59 (user-defined) prompts, 54 of which were grouped under six categories: Amplifier Parameters, Type of Amplifier, Edit Circuit Impedances, SNR Analysis and Channel Error Analysis. When the SNR Analysis prompt

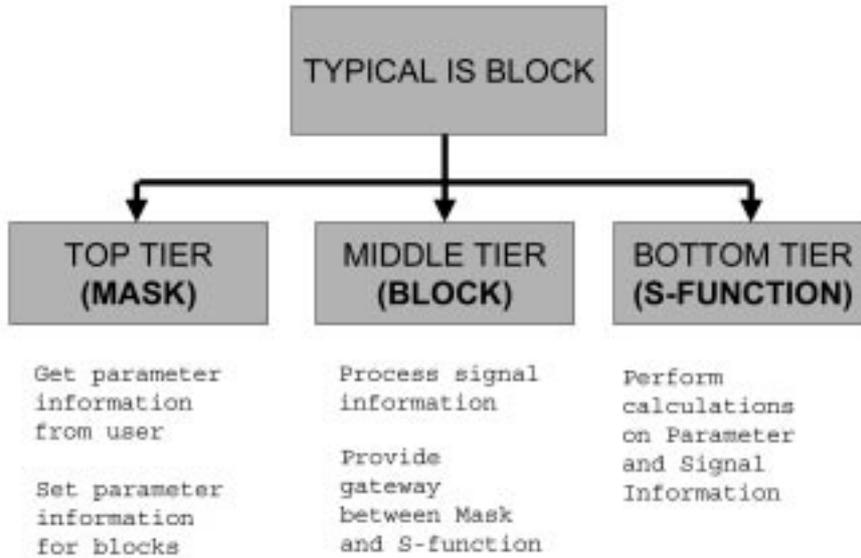


Fig. 4.

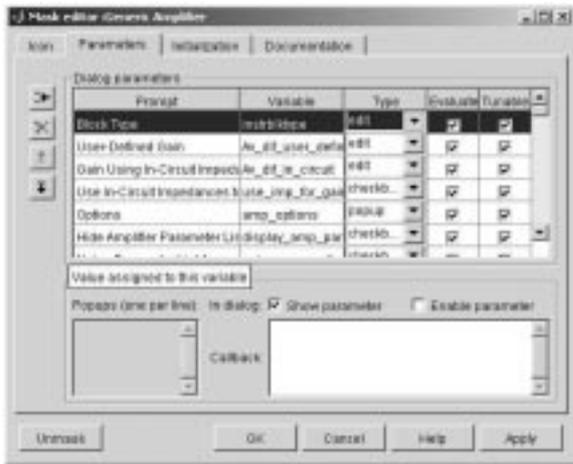


Fig. 5. Generic amplifier mask editor views.

was selected for instance, all of the prompts containing data about the input and output signal powers as well as the input and output SNR would become visible. Available screen size and the type of prompts also limited the number of visible prompts. For example edit boxes took up more screen space than checkboxes and popup windows. A typical 15-inch screen may hold between 12 and 16 prompts on average.

**SYSTEM EVALUATION RESULTS**

The prototype Instrumentation Blockset was tested under several operating conditions to evaluate its performance and to compare the results with those obtained by other techniques such as manual calculation.

For example, as one of the tests, a system as shown in Fig. 6 was configured on the Blockset for the parameter values shown in the diagram. This was actually a past final examination question of the type that students of the IS course would be expected to solve. The question required the students to be able to calculate several error values for the depicted circuit, based on component and other information given in the question. The value, for instance would be calculated using Equation (1). It represents the type of scenario that a student would be expected to encounter as a real IS design and evaluation problem. Table 3 presents a comparison of the results obtained from the IS blockset with those calculated manually, showing the close agreement between the two sets of answers, thus confirming the accuracy of the blocks developed in this project.

**FEATURES OF THE IS BLOCKSET**

The following blocks have so far been developed:

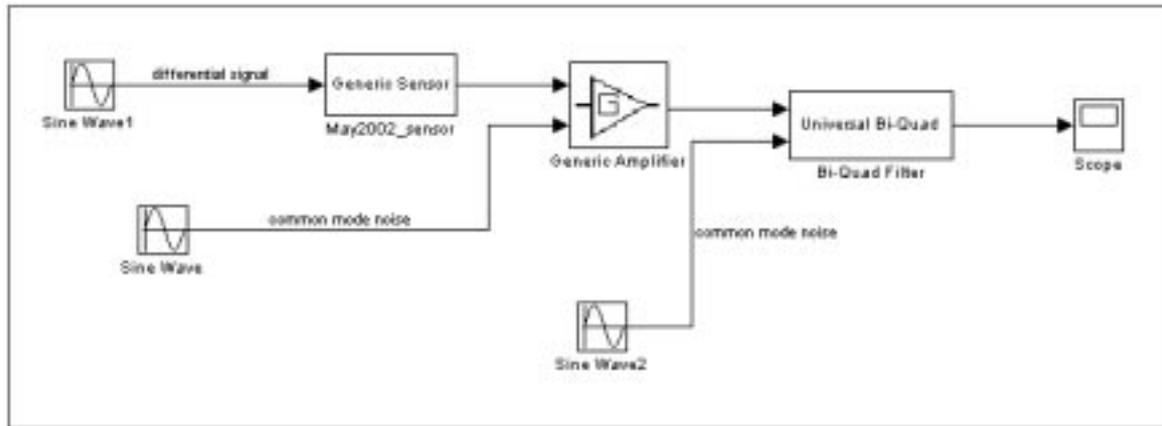


Fig. 6. Typical instrumentation systems examination question.

Table 3. Parameters calculated for the analysis of Fig. 6.

Parameter	Hand calculated output	Actual output
$\varepsilon_{amp,RTI}$	17.819 $\mu$ V	17.819 $\mu$ V
$\varepsilon_{amp,RTI}$ %	0.089095%	0.089095%
$\varepsilon_{SNR,coh}$ %	$7.5000 \times 10^{-3}$ %	$7.4999 \times 10^{-3}$ %
$\varepsilon_{SNR,in}$	$1.7778 \times 10^{-4}$	$1.7778 \times 10^{-4}$
$\varepsilon_{SNR,out}$	$1.7778 \times 10^{-8}$	$1.7778 \times 10^{-8}$
$\varepsilon_{channel}$	0.2394%	0.2394%
$\varepsilon_{filter}$	0.0000%	0.0018%
$\varepsilon_{channel}$	0.2394%	0.2412%

- sensors—thermocouples and a strain gauge;
- signal conditioning—resistance bridge, differential amplifier and instrumentation amplifier;
- signal processing—2-pole and 3-pole Butterworth filters;
- A to D blocks—a sample and hold block.

The Uniform Encoder and Decoder blocks (originally from the Simulink DSP blockset) were included in the new IS blockset and adjusted to add information relevant to the Instrumentation course. Most of the blocks in the Instrumentation Tutor blockset were of a generic nature since obtaining the correct functionality was the most important goal at this stage of the system development. Changing the generic amplifier into an INA106 differential amplifier, for example, will only be a matter of setting the amplifier parameters such as voltage offset and disabling the edit boxes so that the user cannot change those parameters.

Help and demo files, along with tutorials, have been developed and included in the appropriate MATLAB directories. These files include examples and instrumentation theory from the IS course so that it forms a possible self-tutoring environment for engineering students. A student seeking help for a particular block may access the help files

using all the methods provided for the default MATLAB blocks. A comprehensive manual on the development of the blockset has been written and is being edited.

## CONCLUSION

The initial evaluation of the MATLAB Instrumentation Tutor has proven that it can facilitate the solving of problems at the level of the IS course material, including past final examination questions and that the models used are performing correctly. The next development stage is to include a full range of datasheet specifications for the various devices and sensors to enable real-world modelling of instrumentation systems. In this regard a series of lab exercises is being developed to make full use of this blockset and other Simulink capabilities. These lab exercises will also serve to evaluate the usefulness of the Tutor as an educational tool.

*Acknowledgements*—We would like to especially thank Alejandra Villegas of Mathworks for the guidance and advice provided on MATLAB and Simulink during the course of this project.

## REFERENCES

1. *Simulink: Writing S-Functions*, 3rd Edn., The Mathworks (2002).
2. P. H. Garrett, *Multisensor Instrumentation 6 $\sigma$  Design*, John Wiley & Sons, Inc (2002).
3. *Matlab Release 13 Help: Writing S-Functions*, The Mathworks.

4. [www.mathworks.com/access/helpdesk/help/techdoc/apiref/apiref.html](http://www.mathworks.com/access/helpdesk/help/techdoc/apiref/apiref.html)
5. *Matlab Release 13 Help: Using Simulink: Creating Masked Subsystems*, The Mathworks.
6. S. L. Pfleeger, *Software Engineering Theory and Practice*, 2nd Edn., Prentice-Hall Inc. (2001).

**Fasil Muddeen** is a lecturer in instrumentation and signal processing at the University of the West Indies Department of Electrical and Computer Engineering in Trinidad. He has several years experience in standards laboratory operation, instrumentation and electronic calibration. His current research interests are in low cost embedded instrumentation and acoustic signal processing.

**Kester Gabriel** is a recent graduate of the University of the West Indies Department of Electrical and Computer Engineering in Trinidad in Electrical Engineering. This paper was based on work done in his final year project at UWI in 2003–2004.