Mastering
SIMULINK 2®

James B. Dabney
Thomas L. Harman

The MATLAB® Curriculum Series

Prentice Hall
Upper Saddle River, New Jersey 07458

10

This material describes what I call S-function protocol.

S-Functions

In this chapter we'll show how you can use S-Functions to build custom Simulink blocks. S-Functions allow you to incorporate existing code into a Simulink model. They can also be used in situations where it's easier to describe a subsystem algorithmically than in block diagram notation. You can write S-Functions either as MATLAB function M-files or by using the C programming language via the MATLAB MEX-file mechanism.

10.1 Introduction

S-Functions allow you to define custom Simulink blocks using either MATLAB or C language code. S-Functions are useful in several situations. If there is existing MATLAB or C language code that models a portion of a system, it may be desirable to reuse that code in a Simulink model. For example, suppose you wish to try several different control system designs for a plant for which you have developed a dynamics model using an M-file. You can include the dynamics model in an S-Function, then use standard Simulink blocks to model the control system. S-Functions are also useful where it is easier to describe a portion of a dynamical system algorithmically than to describe it graphically via block diagram notation. S-Functions also might improve the efficiency of a simulation, particularly in models involving algebraic loops (see Chapter 12 for a discussion of algebraic loops). Finally, S-Functions provide a straightforward mechanism for adding animations to a Simulink model, a capability to be discussed in Chapter 11.

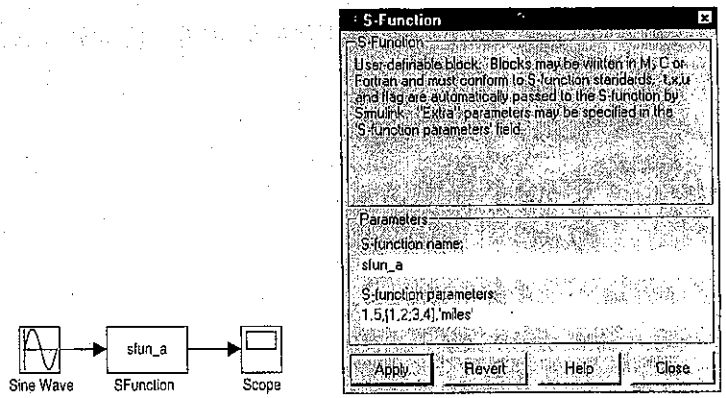
In this chapter, we will start with a general description of S-Function structure. Next we will discuss M-file S-Functions and provide examples of M-file S-Functions that have no states, that have continuous states, and that have discrete states. Last, we will discuss C S-Functions and provide C examples of the same three example subsystems.

10.2 S-Function Block

An S-Function is included in a Simulink model using the S-Function block in the Nonlinear block library. Figure 10-1(a) shows a simple model that includes an S-Function. The block dialog box (Figure 10-1(b)) has two fields. **S-function**

name contains the filename of the S-function, without an extension. This field must not be empty. **S-function parameters** contains any parameters required by the S-Function. If parameters are present, they must be in the form of a list, with the elements separated by commas, and without brackets. For example, suppose the S-Function requires three parameters: 1.5, a matrix [1,2;3,4], and the string "miles". A suitable entry in **S-function parameters** would be:

```
1.5,[1,2;3,4],'miles'
```



(a) Simulink model with S-Function (b) S-Function block dialog box

Figure 10-1 Simple model with S-Function block

10.3 S-Function Overview

An S-Function represents a general Simulink block with input vector $\boldsymbol{u}$, output vector $\boldsymbol{y}$, and state vector $\boldsymbol{x}$ consisting of continuous states $\boldsymbol{x}_c$ and discrete states $\boldsymbol{x}_d$. Every S-Function must include code to set the initial values of all elements of the state vector, and to define the sizes of the input vector, the output vector, and the continuous and discrete components of the state vector. Also, every S-Function must be able to compute the output

$$\boldsymbol{y} = \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u}, t, \boldsymbol{p}) \tag{10-1}$$

update the discrete states

$$\boldsymbol{x}_d(k+1) = \boldsymbol{f}_d(\boldsymbol{x}, \boldsymbol{u}, t, \boldsymbol{p}) \tag{10-2}$$

and compute the derivatives of the continuous states

$$\dot{\boldsymbol{x}}_c = \boldsymbol{f}_c(\boldsymbol{x}, \boldsymbol{u}, t, \boldsymbol{p}) \tag{10-3}$$

Here, t is the current value of simulation time and $\boldsymbol{p}$ is the optional list of parameters specified in S-Function block dialog box field **S-function parameters**. If an S-Function has variable sample time, it must also compute the time of the next sample hit.

The size of each of these vectors is completely arbitrary, and in particular, may be zero. So, for example, it is common for an S-Function to have inputs and outputs, but no states (behavior exhibited by Gain blocks and most blocks in the Nonlinear block library). An S-Function could also be used to create a custom source or sink, or a block with only continuous states (integrators) or only discrete states (delays).

In equations (10-1) to (10-3), the functions have access to all elements of the state vector. If there are n continuous states and m discrete states, the first n elements of the state vector $\boldsymbol{x}$ are the current values of the continuous states ($\boldsymbol{x}_c$) and the remaining m components of $\boldsymbol{x}$ are the current values of the discrete states ($\boldsymbol{x}_d$). Simulink may call the S-Function to evaluate equations (10-1) and (10-3) at any value of simulation time (t), but equation (10-2) is evaluated only at sample times. If there are multiple sample times, the S-Function must include logic that determines which components of $\boldsymbol{x}_d$ to update.

The S-Function state vector contains only the states defined by the S-Function—not all of the states in the model. Thus, if an S-Function implements a scalar integrator, its state vector will have one state. If it implements a transfer function of a third-order system, the state vector will have three components.

10.4 M-file S-Functions

An M-file S-function is a function M-file with a prescribed set of calling arguments. The first executable statement in the M-file is the function statement:

```
function [sys,x0,str,ts] = sfunc_name(t,x,u,flag,p1,p2,...,pn)
```

where *sfunc_name* is the name of the S-Function. For example, if an S-Function is stored in file `sfun1.m`, *sfunc_name* would be `sfun1` (the same as the filename, without the .m extension). The input arguments are defined in Table 10-1; and the output arguments in Table 10-3.

The MathWorks includes a template M-file S-Function in `sfuntmpl.m`, located in the `matlab/toolbox/simulink/blocks` directory in the standard Simulink installation. It is advisable to use the template as the starting point for

Table S-Function input arguments

Argument	Definition
`t`	Current value of simulation time.
`x`	Current value of S-Function state vector. If there are n continuous states and m discrete states, the first n components of `x` are the continuous states, and the remaining m are the discrete states.
`u`	Current value of input vector.
`flag`	Flag set by Simulink each time the S-Function is called. The S-Function must test the value of `flag` and perform the action indicated in Table 10-2.
`p1,...,pn`	List of optional parameters. The number of parameters should be the same as the number of parameters in S-Function dialog box field **S-function parameters**.

Table 10-2 S-Function flag definition

Flag value	Action required of the S-function
0	Initialize the `sizes` structure, and assign the value of the `sizes` structure to `sys`. Set `x0` (initial conditions) and `ts` (matrix of sample times and offsets). Set `str = []`.
1	Compute the values of the derivatives of the continuous states. Set `sys` to the value of the derivative vector (that is, the derivative of the continuous portion of the S-Function state vector). `sys` is the only output argument.
2	Update the discrete states. Set `sys` to the new value of the discrete portion of the S-Function state vector. If there are multiple sample times, all components of the discrete state vector must be set each time `flag` is 2, including those that don't change. `sys` is the only output argument.
3	Compute outputs. Set `sys` to the value of the output vector. `sys` is the only output argument.
4	Compute the next sample time. Set `sys` to the value of the next sample time. `sys` is the only output argument.
9	Perform any necessary end-of-simulation tasks. There are no output arguments.

Table 10-3 S-Function output argument definitions

Argument	Definition
`sys`	Multipurpose output argument. The definition of `sys` depends on the value of `flag`.
`x0`	Initial value of S-Function state vector. `x0` contains the initial value of both the continuous and discrete states. `x0` is needed only if `flag` is 0.
`str`	`str` is a placeholder output argument. It should be set to an empty matrix. `str` is needed only if `flag` is 0.
`ts`	Matrix of sample time, offset pairs. The matrix must have two columns. There must be at least one row. `ts` is needed only if `flag` is 0.

building an S-Function, as this will eliminate some typing, and allow you to start with an S-Function that follows The MathWorks conventions.

An S-Function M-file must test the value of input argument `flag`, and perform the corresponding operation indicated in Table 10-2. The approach taken in The MathWorks template (and here) is to use a switch-case block that calls internal functions based on the value of `flag`. `flag` has six possible values. We will discuss the required behavior of the S-Function for each possible value.

10.4.1 Initialization (flag = 0)

Initialization entails four operations. The first is to set up a `sizes` structure, and to assign this structure to the multipurpose return argument `sys`. Create a `sizes` structure using the statement

```
sizes = simsizes ;
```

Next, assign values to each member of `sizes`. Each member of `sizes` must be assigned a value, even if the value is zero. Table 10-4 lists the members of the `sizes` structure.

Once the sizes structure is defined, assign it to output `sys` using the statement

```
sys = simsizes(sizes) ;
```

The second initialization step is to set the initial condition vector `x0`. For example, if there are two continuous states with initial values of 1.0, and two discrete states with initial value 0.0, use the statement

```
x0 = [1.0,1.0,0.0,0.0] ;
```

Table 10-4 Sizes structure

Member	Definition
`sizes.NumContStates`	Number of continuous states. For example, if there are two continuous states, set `sizes.NumContStates = 2 ;`
`sizes.NumDiscStates`	Number of discrete states.
`sizes.NumOutputs`	Number of outputs.
`sizes.NumInputs`	Number of inputs.
`sizes.DirFeedthrough`	Set to 1 if there is direct feedthrough, otherwise 0. Direct feedthrough means that the block output (Equation (10-1)) is a function of the input. A block for which the output is an algebraic function of the input (a Gain block, trigonometric function, etc.) has direct feedthrough. A block for which the output is a function only of the states does not have direct feedthrough. Simulink uses the value of this flag to determine whether there are any algebraic loops. For more information on algebraic loops, see Chapter 12. A block that has a variable sample time has direct feedthrough.
`sizes.NumSampleTimes`	Number of sample time, offset pairs. Must be at least 1, even for purely continuous S-Functions.

The third initialization step is to set `str` using the statement

```
str = [] ;
```

The final initialization step is to create the sample time, offset matrix, `ts`. There must be one row in `ts` for each sample time, offset pair, and there must be at least one row, even for continuous S-Functions. For a continuous S-Function, use the statement

```
ts = [0,0] ;
```

As another example, suppose there are two discrete sample times. The first has a period of 0.5 s and no offset, so the sample times are at 0.0, 0.5, 1.0, ... The second has a period of 0.25 s with a 0.1 s offset, resulting in sample times of 0.1, 0.35, 0.60, ... In this case, set

```
ts = [0.5,0 ; 0.25, 0.1] ;
```


If the sample time is to be inherited, set the sample time to –1. The sample time will usually be inherited from the block connected to the S-Function block input. In certain situations, Simulink will detect that a longer sample time will not affect the simulation results, and the sample time (if inherited) will be set accordingly.

If the sample time is to be variable, set the value of sample time to –2. In this case, the S-Function will be called with `flag = 4` to compute the time of the next sample hit.

10.4.2 Continuous State Derivatives (flag = 1)

If `flag` is `1`, assign to `sys` the value of the derivative of the continuous portion of the state vector. Note that if there are also discrete states, the size of `sys` will not be the same as the size of `x`, as `x` includes both the continuous and discrete states.

Example 10-1

Suppose an S-Function models the nonlinear system

$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = x_1 - 3x_2^2 + u_1$$

The following statements will set `sys` appropriately:

```
sys(1) = x(2) ;
sys(2) = x(1) - 3*x(2)^2 + u(1) ;
```

10.4.3 Discrete State Updates (flag = 2)

If `flag` is 2, assign to `sys` the updated value of the discrete part of the state vector. If there are multiple sample times (including hybrid systems which have a sample time of 0 for the continuous states), the S-Function must test for a sample time hit and only update those components of the discrete portion of the state vector for which the current simulation time is a sample time hit. But note that all components of the discrete portion of the state vector must be assigned a value.

Example 10-2

First, consider the single-rate first-order discrete subsystem

$$x_1(k+1) = x_1(k) + u_1(k)$$

Use the following statement to set the new value of sys if flag is 2:

```
sys = x(1) + u(1) ;
```

Now, consider a second-order discrete subsystem with two sample times. The first discrete component is updated every 0.3 s, and the second component every 0.5 s, both with zero offset. The first state is to be updated according to the equation

$$x_1(k+1) = x_1(k) + 0.5x_2(k)$$

and the second state is to be updated according to the equation

$$x_2(k+1) = x_2(k) + u_1(k)$$

The following statements test the simulation time and update sys appropriately if the current time is a sample time hit.

```
period_1 = 0.3 ;
offset_1 = 0.0 ;
period_2 = 0.5 ;
offset_2 = 0.0
sys = x ;
if abs(round(t-offset_1)/period_1-((t-offset_1)/period_1))< 1.0e-8
  sys(1) = sys(1) + 0.5*x(2) ;
end
if abs(round(t-offset_2)/period_2-((t-offset_2)/period_2))< 1.0e-8
  sys(2) = sys(2) + u(1) ;
end
```

Notice that we did not use an `if..else` control structure when testing for sample time hits. If we had done that, the S-Function would produce incorrect results whenever both sample times hit simultaneously.

10.4.4 Block Outputs (flag = 3)

If flag is 3, assign to sys the value of the S-Function output (Equation (10-1)).

Example 10-3

Suppose the output of a second-order system is

$$y = x_1 + x_2$$

The following statement will set sys appropriately when flag is 3.

```
sys = x(1) + x(2) ;
```

10.4.5 Next Sample Time (flag = 4)

If flag is 4, assign to sys the value of the next sample time. The S-Function will be called with flag = 4 only if the sample time is variable (set to –2).

10.4.6 Terminate (flag = 9)

When the simulation is complete for any reason (**Stop time** reached, **Simulation:Stop** selected), the S-Function is called with flag set to 9. The S-Function should perform any necessary end-of-simulation tasks. There is no need to assign a value to sys.

10.4.7 Programming Considerations

Local data storage

Frequently, S-Functions require local data storage. Since an S-Function is a MATLAB function M-file, local variables must be initialized each time the S-Function is called. It is possible to preserve data between calls using global variables, but that is not a good idea. If global variables are used, there can be only one instance of a particular S-Function in a model. Otherwise, multiple instances of the S-Function would share the same storage locations. The preferred approach is to use the UserData parameter for the S-Function block that references the S-Function. Using the block UserData, there may be multiple instances of a particular S-Function in a model, because each instance of the S-Function block has its own UserData. UserData can be a scalar, a matrix, or even a cell array or structure. Therefore, there is no limit to the amount or types of data that can be stored in a block UserData parameter.

Example 10-4

Suppose that an S-Function needs to store the value of time and the state vector for use the next time the S-Function is called. The following statements will save the data:

```
u_dat.time = t ;
u_dat.state = x ;
set_param(gcb,'UserData',u_dat) ;
```

The most recent values may be read from the block `UserData` using the statement

```
old_data = get_param(gcb,'UserData') ;
```

Dynamic sizing

Many Simulink blocks can accept inputs of varying dimension. For example, a Gain block with scalar gain can accept scalar or vector input signals, and the vector input signals can be of any dimension. To configure an S-Function to adapt to different size input vectors, set `sizes.NumInputs` to –1. The S-Function can determine the size of the input vector when `flag` is 1, 2, 3, or 4 using the statement

```
size_input = size(u) ;
```

If the size of the output vector or the number of continuous or discrete states is dependent on the size of the input vector, set the appropriate member of `sizes` to –1 as well. The dimension of any of these vectors specified to be –1 is defined to be the same as the size of the input vector.

Hybrid and multirate S-Functions

S-Functions can be hybrid (containing both discrete and continuous states). Discrete S-Functions can have multiple sample times. There are occasions where these capabilities are useful. In general, however, it is preferable to build S-Functions that have a single purpose. This makes the S-Functions easier to develop and maintain, and also makes the S-Functions more likely to be reusable for other projects.

10.4.8 M-file S-Function Examples

Three example M-file S-Functions are presented next. These examples show how to create algebraic, continuous, and discrete M-file S-Functions. Several other helpful examples can be found in the `matlab/toolbox/simulink/sim-demos` directory of the standard MATLAB/Simulink installation.

Example 10-5

This example demonstrates an S-Function that has no states. The S-Function represents the algebraic equation

$$y = u_1 + u_2^2$$

The S-Function will have two inputs, one output, and no states. Since the input is passed directly to the output, the S-Function has direct feedthrough. A Simulink model that uses the S-Function is shown in Figure 10-2. S-Function block dialog box field **S-function name** contains `s_xmp1`. The S-Function listing is shown in Figure 10-3.

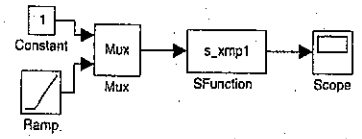


Figure 10-2 Simulink model with no states

```
function [sys,x0,str,ts] = s_xmp1(t,x,u,flag)
%  S-file example 1
%  This is an S-file subsystem with no states. It performs
%  the algebraic function y = u(1) + u(2)^2. There are two
%  inputs and one output.
%
%  Based on sfuntmpl.m, supplied with Simulink
%  Copyright (c) 1990-96 by The MathWorks, Inc.
%
switch flag,
  case 0,              % Initialization
    [sys,x0,str,ts]=mdlInitializeSizes;
```

Figure 10-3 M-file S-Function with no states

```
  ca                    % Compute derivatives of continuous states
    sys=mdlDerivatives(t,x,u);
  case 2,
    sys=mdlUpdate(t,x,u);
  case 3,
    sys=mdlOutputs(t,x,u);  % Compute output vector
  case 4,                   % Compute time of next sample
    sys=mdlGetTimeOfNextVarHit(t,x,u);
  case 9,                   % Finished. Do any needed
    sys=mdlTerminate(t,x,u);
  otherwise                 % Invalid input
    error(['Unhandled flag = ',num2str(flag)]);
end
%*************************************************************
%*                  mdlInitializeSizes                       *
%*************************************************************
function [sys,x0,str,ts]=mdlInitializeSizes()
% Return the sizes of the system vectors, initial conditions,
% and the sample times and offets.
sizes = simsizes;   % Create the sizes structure
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 1;
sizes.NumInputs      = 2;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;   % at least one sample time needed
sys = simsizes(sizes);      % load sys with the sizes structure
x0  = [];   % Specify initial conditions for all states
str = [];   % str is always an empty matrix
ts  = [0 0]; %initialize the array of sample times
%*************************************************************
%*                  mdlDerivatives                           *
%*************************************************************
function sys=mdlDerivatives(t,x,u)
% Compute derivatives of continuous states
sys = [];   % Empty since this S file has no continuous states
%*************************************************************
%*                  mdlUpdate                                *
%*************************************************************
function sys=mdlUpdate(t,x,u)
% Compute update for discrete states. If necessary, check for
% sample time hits.
```

Figure 10-3 M-file S-Function with no states (Continued)


```
sys = [];   % Empty since this model has no discrete states.
%*************************************************************
%*                  mdlOutputs                               *
%*************************************************************
function sys=mdlOutputs(t,x,u)
% Compute output vector given current state, time, and input
sys = [u(1) + u(2), 2];
%*************************************************************
%*                  mdlGetTimeOfNextVarHit                   *
%*************************************************************
function sys=mdlGetTimeOfNextVarHit(t,x,u)
% Return the time of the next hit for this block.  Note that
% the result is absolute time.  Note that this function is only
% used when you specify a variable discrete-time sample time
sampleTime = [];
%*************************************************************
%*                  mdlTerminate                             *
%*************************************************************
function sys=mdlTerminate(t,x,u)
% Perform any necessary tasks at the end of the simulation
sys = [];
```

Figure 10-3 M-file S-Function with no states (Continued)

Example 10-6

This example illustrates using an S-Function to model a continuous system. It also illustrates the process of using a masked subsystem to supply parameters to an S-Function.

Consider the inverted pendulum model from Example 8-9, shown in Figure 8-14. The equations of motion of the cart and pendulum are

$$(M+m)\ddot{x} - ml\dot{\theta}^2\sin\theta + ml\ddot{\theta}\cos\theta = u$$

$$m\ddot{x}\cos\theta + ml\ddot{\theta} = mg\sin\theta$$

where g is the acceleration due to gravity. In Example 8-9, we manipulated the equations of motion such that $\ddot{x}$ and $\ddot{\theta}$ each appears in only one equation. An alternative approach is to rewrite the equations of motion in the form of a

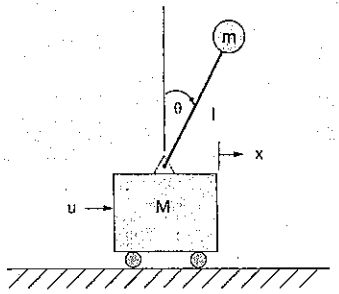


Figure 10-4 Cart with inverted pendulum

linear system, and solve the linear system for $\ddot{x}$ and $\ddot{\theta}$. In matrix notation, the equations of motion can be written

$$\begin{bmatrix} (M+m) & ml\cos\theta \\ m\cos\theta & ml \end{bmatrix} \begin{bmatrix} \ddot{x} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} ml\dot{\theta}^2\sin\theta + u \\ mg\sin\theta \end{bmatrix}$$

There are three parameters in the equations of motion: the cart mass (M), the pendulum mass (m), and the pendulum length (l). An S-Function that implements the equations of motion is shown in Figure 10-5. The function statement includes the three parameters.

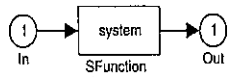
The state derivatives are computed by solving the linear system for $\ddot{x}$ and $\ddot{\theta}$. Next, the value of the state vector time derivative is assigned to sys as follows:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} x_2 \\ \ddot{x} \\ x_4 \\ \ddot{\theta} \end{bmatrix}$$

The S-Function output is the state vector.

This S-Function can be used to build a subsystem to replace the inverted pendulum subsystem shown in Figure 8-15. To build the new subsystem, proceed as follows. Make a copy of the model in Example 8-9 and delete the `Cart Model` subsystem.

Drag a Subsystem block to the model window from the Connections block library. Open the Subsystem block, and drag Inport and an Outport blocks to the subsystem window. Also drag an S-Function block to the subsystem window. Connect the blocks as shown.



This is the prototype upon which I based the m-file called pendcom.m used to model the pendulum.

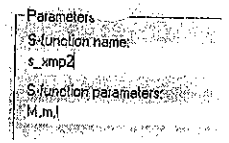
```
function [sys,x0,str,ts] = s_xmp2(t,x,u,flag,M,m,l)
%   S-file example 2
%   This is an S-file subsystem that models a cart with
%   inverted pendulum. The cart and pendulum masses and
%   pendulum length are parameters that must be set in
%   the block dialog box.
%
%   Based on sfuntmpl.m, supplied with Simulink
%   Copyright (c) 1990-96 by The MathWorks, Inc.
%
switch flag,
  case 0,          % Initialization
    [sys,x0,str,ts]=mdlInitializeSizes;
  case 1,          % Compute derivatives of continuous states
    sys=mdlDerivatives(t,x,u,M,m,l);
  case 2,
    sys=mdlUpdate(t,x,u);
  case 3,
    sys=mdlOutputs(t,x,u);  % Compute output vector
  case 4,                   % Compute time of next sample
    sys=mdlGetTimeOfNextVarHit(t,x,u);
  case 9,                   % Finished. Do any needed
    sys=mdlTerminate(t,x,u);
  otherwise                 % Invalid input
    error(['Unhandled flag = ',num2str(flag)]);
end
%*************************************************************
%*                    mdlInitializeSizes                     *
%*************************************************************
function [sys,x0,str,ts]=mdlInitializeSizes
% Return the sizes of the system vectors, initial
% conditions, and the sample times and offets.
sizes = simsizes;   % Create the sizes structure
sizes.NumContStates  = 4;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 4;
sizes.NumInputs      = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;   % at least one sample time
sys = simsizes(sizes);      % load sys with the sizes structure
x0  = [0,0,0,0];    % Specify initial conditions for all states
```

Figure 10-5 S-Function model of cart with inverted pendulum

```
str = [];  % str is always an empty matrix
ts  = [0 0]; %initialize the array of sample times
%*************************************************************
%*                     mdlDerivatives                        *
%*************************************************************
function sys=mdlDerivatives(t,x,u,M,m,l)
% Compute derivatives of continuous states
g = 9.8 ;
Mass = [(M+m),m*l*cos(x(3));m*cos(x(3)),m*l] ;
x_dot_dot = Mass\[m*l*x(4)^2*sin(x(3))+u ; m*g*sin(x(3))] ;
sys = [x(2),x_dot_dot(1),x(4),x_dot_dot(2)] ;
%*************************************************************
%*                     mdlUpdate                             *
%*************************************************************
function sys=mdlUpdate(t,x,u)
% Compute update for discrete states. If necessary, check for
% sample time hits.
sys = [];    % Empty since this model has no discrete states.
%*************************************************************
%*                     mdlOutputs                            *
%*************************************************************
function sys=mdlOutputs(t,x,u)
% Compute output vector given current state, time, and input
sys = x ;
%*************************************************************
%*                  mdlGetTimeOfNextVarHit                   *
%*************************************************************
function sys=mdlGetTimeOfNextVarHit(t,x,u)
% Return the time of the next hit for this block.  Note that
% the result is absolute time.  Note that this function is
% only used when you specify a variable discrete-time sample
% time [-2 0] in the sample time array in sampleTime = 1;
sys = [] ;
%*************************************************************
%*                     mdlTerminate                          *
%*************************************************************
function sys=mdlTerminate(t,x,u)
% Perform any necessary tasks at the end of the simulation
sys = [];
```

Figure 10-5 S-Function model of cart with inverted pendulum (Continued)

Open the S-Function block dialog box and configure the Parameters as shown.

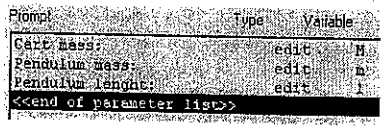


Close the Subsystem block dialog box and the subsystem window.

Rename the Subsystem block `Cart Subsystem`.

Select the Cart Subsystem block, then choose **Edit:Mask Subsystem** from the model window menu bar.

Select the Initialization page and define the prompts and associated variables as shown. Note that the variables (M,m,l) are used in the S-Function block dialog box as the parameters sent to the S-Function.



Connect the output of the Sum block to the input of the new masked subsystem, and the output of the new masked subsystem to the Demux block. Configure the Gain blocks as shown in Figure 10-6. This model will produce results identical to the model in Example 8-9.

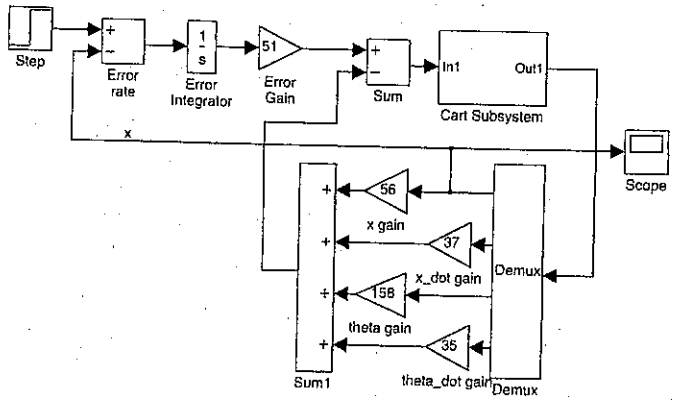


Figure 10-6 Inverted pendulum model using an S-Function