Simulink®

基于模型与基于系统的设计

- 建模
- 仿真
- 实现

S-Function 的编写

Version 5

The MathWorks

目录

S-FUNCTION 概述 1

什么是 S-FUNCTION 2

在模型中使用 S-FUNCTION 3

向 S-Function 传递参数 3

何时使用 S-Function 4

S-FUNCTION 的工作原理 5

Simulink 块的数学关系 5

仿真过程 5

S-Function 回调程序 6

S-FUNCTION 的实现 8

M-文件的 S-Function 8

MEX 文件的 S-function 8

MEX 文件与 M-文件的 S-function 比较 9

S-FUNCTION 的概念 10

直接馈通 10

动态维矩阵 10

设置采样时间和偏移量 11

S-FUNCTION 范例 14

M 文件 S-function 范例 15

C S-Function 范例 16

Fortran S-Function 范例 18

C++ S-Function 范例 18

Ada S-Function 范例 18

编写 M S-FUNCTION 19

概述 20

S-Function 参数 20

S-Function 的输出 20

定义 S-FUNCTION 块特性 22

处理 S-FUNCTION 参数 22

M 文件的 S-FUNCTION 范例 23

范例 1——简单的 M 文件 S-Function 23

范例 2——连续状态 S-Function 25

范例 3——离散状态 S-Function 27

范例 4——混合系统 S-Function 28

范例 5——变步长 S-Function 31

使用 C 语言编写 S-FUNCTION 33

概述 34

创建 C MEX S-Function *35*

自动生成 S-FUNCTION 36

配置生成的 S-Function *37*

*S-Function Builder **如何生成** S-Function* *37*

设置 include 路径 *37*

S-FUNCTION BUILDER 的对话窗 39

*Initialization **选卡*** *39*

*Data Properties **选卡*** *40*

*Libraries **选卡*** *42*

*Outputs **选卡*** *43*

*Continuous Derivatives **选卡*** *45*

*Discrete Update **选卡*** *46*

*Build Info **选卡*** *47*

一个基本的 C MEX S-FUNCTION 范例 48

定义与包含 *49*

回调函数的实现 *50*

*Simulink/Real-Time Workshop **接口*** *51*

*Building Timestwo **范例*** *51*

C S-FUNCTION 模板 52

*S-Function **源文件必需的内容*** *52*

SimStruct *53*

编译 C S-Function *53*

SIMULINK 如何与 C S-FUNCTION 相互作用 54

进程层面 *54*

数据层面 *56*

编写回调函数 59

将 LEVEL 1 C MEX S-FUNCTION 转换到 LEVEL 2 60

创建 C++ S-FUNCTION **63**

创建 ADA S-FUNCTION **64**

创建 FORTRAN S-FUNCTION **65**

实现块特性 **67**

对话框参数 68

可调参数 *68*
运行参数 70
创建运行参数 *70*
更新运行参数 *71*
创建输入和输出端口 72
创建输入端口 *72*
创建输出端口 *73*
输入的标量扩展 *74*
掩码多端口 S-Function *75*
自定义数据类型 76
采样时间 77
基于块的采样时间 *77*
指定基于端口的采样时间 *79*
基于块与基于端口的混合采样时间 *81*
多速率 S-Function 块 *82*
多速率 S-Function 块的同步 *83*
工作向量 84
工作向量与过零检测 *85*
包括指针工作向量的范例 *85*
内存分配 *86*
FUNCTION-CALL 子系统 87
错误处理 89
防超程代码 *89*
SsSetErrorStatus 的终止条件 *90*
数组边界检查 *90*
S-FUNCTION 范例 91
连续状态的 S-Function 范例 *92*
离散状态的 S-Function 范例 *93*
混合系统的 S-Function 范例 *93*
变步长的 S-Function 范例 *94*
过零检测的 S-Function 范例 *94*
时变连续传递函数的 S-Function 范例 *94*

S-Function 概述

S-Function（系统函数）为扩展 Simulink®的性能提供了一个有力的工具。以下下节阐述了什么是 S-Function，为什么可以使用 S-Function，以及怎样编写自己的 S-Function。

什么是 S-Function

S-Function 可以使用 MATLAB®，C，C++，Ada，或 Fortran 语言来编写。使用 MEX 实用工具，将 C，C++，Ada，和 Fortran 语言的 S-Function 编译成 MEX-文件，在需要的时候，它们可与其它的 MEX-文件一起动态地连接到 MATLAB 中。

S-Function 使用一种特殊的调用格式让你可以与 Simulink 方程求解器相互作用，这与发生在求解器和内置 Simulink 块之间的相互作用非常相似。S-Function 的形式是非常通用的，且适用于连续、离散和混合系统。

S-function 为你提供了一种在 Simulink 模型中增加自制块的手段，你可以使用 MATLAB，C，C++，Ada，或 Fortran 语言来创建自己的块。按照下面一套简单的规则，你可以在 S-function 中实现自己的算法。在你编写一个 S-Functin 函数，并将函数名放置在一个 S-Functin 块中（在用户定义的函数块库中有效）之后，通过使用 masking 定制用户界面。

你可以与 Real-Time Workshop®（RTW）一起使用 S-function，也可通过编写目标语言编译器（TLC）文件来定制由 RTW 生成的代码。参看第八章‘ 对于 Real-Time Workshop 编写 S-Function ’和 Real-Time Workshop 文档资料以获取更多信息。

在模型中使用 S-Function

为了将一个 S-function 组合到一个 Simulink 模型中，首先从 Simulink 用户定义的函数块库中拖出一个 S-Function 块，然后在 S-Function 块对话框中的 S-Function name 区域指定 S-Function 的名字。如下图所示：

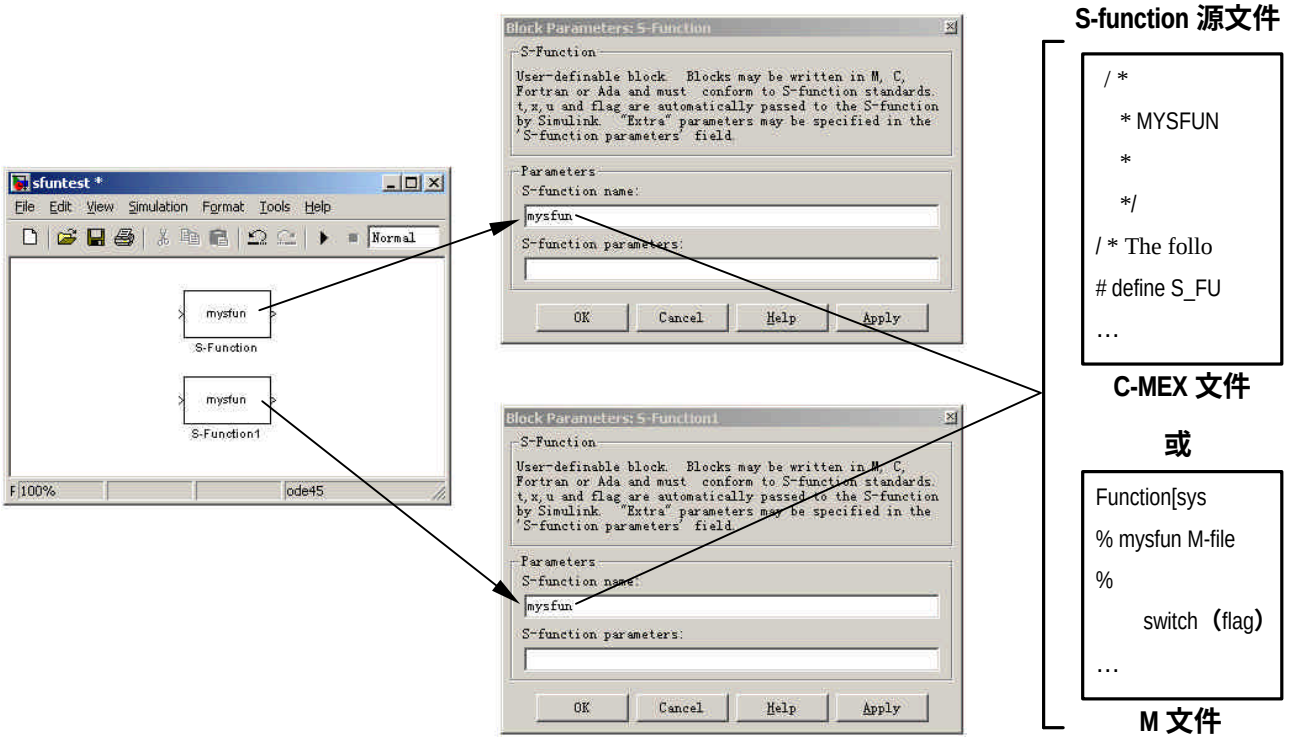


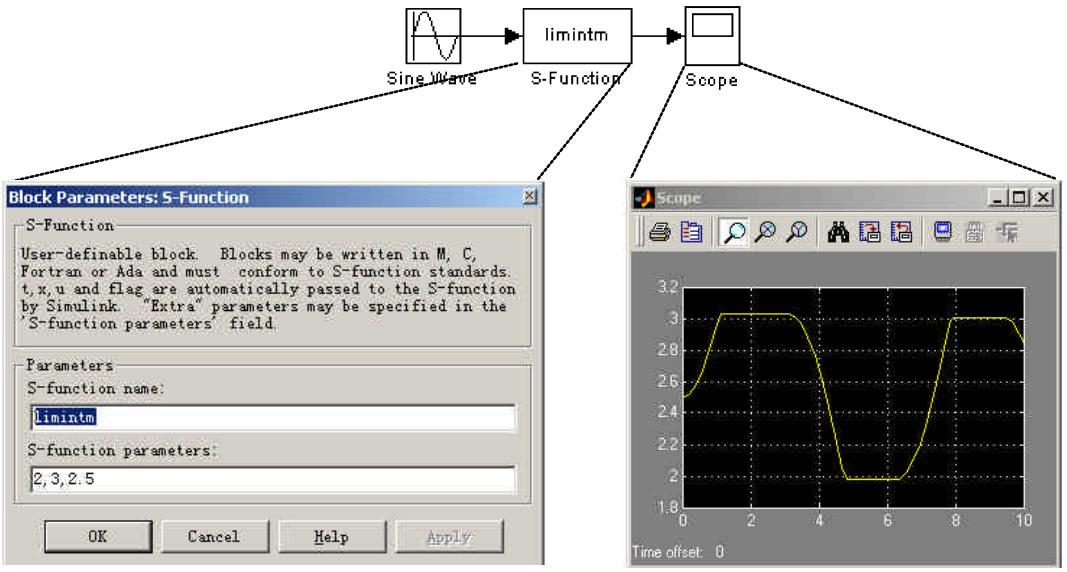
图 1-1　S-function 块、对话框、及决定块功能的源文件之间的关系

在本例中，模型包含了两个的 S-function 块，这两个块使用到同一个源文件（mysfun，可以是一个 C MEX 文件，或者是一个 M 文件）。如果一个 C MEX 文件与一个 M 文件具有相同的名字，则 C MEX 文件被优先使用，即在 S-function 块中使用的是 C MEX 文件。

向 S-Function 传递参数

在 S-function 块的 S-function parameters 区域可以指定参数值，这些值将被传递到相应的 S-function 中。要使用这个区域，必须了解 S-function 所需要的参数，及参数的顺序（如果不知道，应查询 S-function 的编制者、相关文件，或源代码）。输入参数值时，参数之间应使用逗号分隔，并按照 S-function 要求的参数顺序进行输入。参数值可以是常量、模型空做区间定义的变量名、或 MATLAB 表达式。

下面的图示使用 S-function parameters 区域输入用户自定义参数的用法：



在本例中，模型使用的是由 Simulink 提供的 S-function 范例，limintm。该 S-function 的源代码在目录 toolbox/simulink/blocks 下可以找到。函数 limintm 接受了三个参数：一个下边界，一个上边界，及一个初始条件。该函数将输入信号对时间进行积分，如果积分值在上下边界之间则输出积分值；如果积分值小于下边界值，则输出先边界值；如果积分值大于上边界值，则输出上边界的值。在本例的对话框中指定下边界值、上边界和初始条件分别为 2，3，和 2.5。图中示波器显示的曲线是当输入振幅为 1 的正弦波时的输出结果。

你可以使用 Simulink 的 masking 工具来为自己的 S-function 块创建用户对话框和图标。通过封装对话框可以使 S-function 附加参数的指定更清楚容易。关于附加参数及封装，参考文档《Using Simulink》。

何时使用 S-Function

S-function 的应用在大多数情况下是创建自定义的 Simulink 块。你可以使用 S-function 作为一些类型的应用，这些应用包括：

- 向 Simulink 增加一些新的通用块
- 增加作为硬件设备驱动程序的块
- 将已有的 C 代码组合到仿真中
- 使用一组数学方程式来对系统进行描述
- 使用可视化动作（参见倒立摆范例，penddemo）

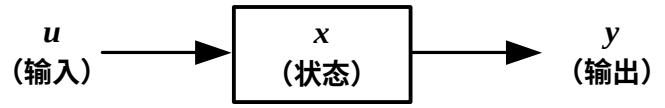
使用 S-function 的一个优点是你可以创建一个普通用途的块，在一个模型中多次使用，而且可单独改变模型中所使用的每个块的参数。

S-Function 的工作原理

要创建 S-function，你必须了解 S-function 是如何工作的。要了解 S-function 如何工作，则需要了解 Simulink 是如何进行模型仿真的，那么又需要了解块的数学公式。因此，本节首先从一个块的输入、状态和输出之间的数学关系开始介绍。

Simulink 块的数学关系

Simulink 块包含一组输入、一组状态和一组输出。其中，输出是采样时间、输入和块状态的函数。



下面的方程式表述了输入、输出和状态之间的数学关系：

$$y = f_0(t,x,u) \qquad （输出）$$

$$\dot{x}_c = f_d(t,x,u) \qquad （求导）$$

$$x_{d_{k+1}} = f_u(t,x,u) \qquad （更新）$$

其中，　$x = x_c + x_d$

仿真过程

Simulink 模型的执行分几个阶段进行。首先进行的是初始化阶段，在此阶段，Simulink 将库块合并到模型中来，确定传送宽度、数据类型和采样时间，计算块参数，确定块的执行顺序，以及分配内存。然后，Simulink 进入到“仿真循环”，每次循环可认为是一个“仿真步”。在每个仿真步期间，Simulink 按照初始化阶段确定的块执行顺序依次执行模型中的每个块。对于每个块而言，Simulink 调用函数来计算块在当前采样时间下的状态，导数和输出。如此反复，一直持续到仿真结束。

下图所示为一个仿真的步骤：

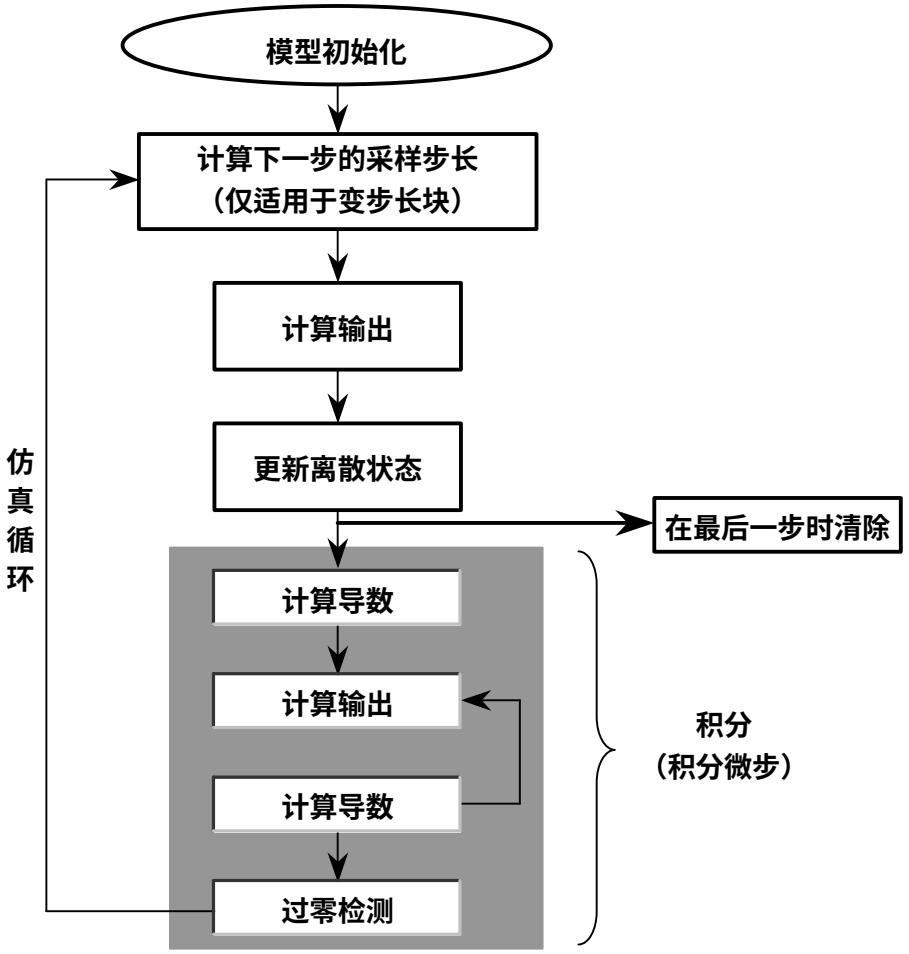


图 1-2　Simulink 执行仿真的步骤

S-Function 回调程序

一个 S-function 包含了一组 S-function 回调程序，用以执行在每个仿真阶段所必需的任务。在模型仿真期间，Simulink 对于模型中的每个 S-function 块调用适当的程序，通过 S-function 程序来执行的任务包括：

- 初始化——在仿真循环之前，Simulink 初始化 S-function。在该阶段期间，Simulink：
  - 初始化 SimStruct，这是一个仿真数据结构，包含了关于 S-function 的信息
  - 设置输入和输出端口的数量和宽度
  - 设置块的采样时间
  - 分配存贮区间和参数 sizes 的阵列
- 计算下步采样点——如果你创建了一个变步长块，那么在这里计算下一步的采样点，即计算下一个仿真步长。
- 计算主步长的输出——在该调用完成后，所有块的输出端口对于当前仿真步长有效。
- 按主步长更新离散状态——在这个调用中，所有的块应该执行“每步一次”的动作，如为下一个仿真循环更新离散状态，
- 计算积分——这适用于连续状态和/或非采样过零的状态。如果 S-function 中具有连续状态，

Simulink 在积分微步中调用 S-function 的输出和导数部分。这是 Simulink 能够计算 S-function 状态的原因。如果 S-function（仅对于 C MEX）具有非采样过零的状态，Simulink 在积分微步中调用 S-function 的输出和过零部分，这样可以检测到过零点。

注意：参考《Using Simulink》文档的“ Simulink 如何工作 ” 中关于主步长和微步的解释。

S-Function 的实现

S-Function 可以通过 M-文件或者 MEX 文件来实现。以下部分介绍了这些实现方法，并讨论各种实现方法各自的优缺点。

M-文件的 S-Function

一个 M-文件的 S-Function 由以下形式的 MATLAB 函数构成：

[sys，x0，str，ts] = f（t，x，u，flag，p1，p2，...）

其中，f 是 S-Function 的函数名，t 是当前时间，x 是相应 S-function 块的状态向量，u 是块的输入，flag 指示了需被执行的任务，p1，p2，... 是块参数。在模型仿真过程中，Simulink 反复调用 f，对于特定的调用使用 flag 来指示需执行的任务。S-function 每次执行任务都返回一个结构，该结构的格式在语法范例中给出。

在目录 matlabroot/toolbox/simulink/blocks 中给出了 M-文件 S-function 的模板，sfuntmpl.m。该模板由一个主函数和一组骨架子函数组成，每个子函数对应于一个特定的 flag 值。主函数通过 flag 的值分别调用不同的子函数。在仿真期间，这些子函数被 S-function 以回调程序的方式调用，执行 S-function 所需的任务。下表列出了按此标准格式编写的 M-文件 S-function 的内容。

仿真阶段	S-function 程序	flag
初始化	mdlInitializeSizes	flag = 0
计算下一步的采样步长（仅用于变步长块）	mdlGetTimeOfNextVarHit	flag = 4
计算输出	mdlOutputs	flag = 3
更新离散状态	mdlUpdate	flag = 2
计算导数	mdlDerivatives	flag = 1
结束仿真时的任务	mdlTerminate	flag = 9

当创建 M-文件的 S-function 时，推荐使用模板的结构和命名习惯。这可方便其他人读懂和维护你所创建的 M-文件的 S-function。

MEX 文件的 S-function

类似于 M-文件的 S-function，MEX 文件的 S-function 也由一组回调程序组成，在仿真期间，Simulink 调用这些回调程序来执行各种块相关任务。然而，两者之间也存在很大的不同。其一，MEX 文件的 S-function 的实现使用了不同的编程语言：C，C++，Ada，或 Fortran；其二，Simulink 直接调用 MEX 文件的 S-function 程序，而不象调用 M-文件的 S-function 程序通过 flag 值来选择。因为 Simulink 要直接调用这些函数，MEX 文件的 S-function 函数必须按照 Simulink 指定的标准命名规定来定义函数名。

还存在着其它一些关键的不同点。第一，MEX 函数能实现的回调函数比 M-文件能实现的回调函数要多得多；其次，MEX 函数直接访问内部数据结构 SimStruct，SimStruct 是 Simulink 用来保存关于 S-function 信息的一个数据结构；另外，MEX 函数也可使用 MATLAB MEX 文件 API 直接来访问 MATLAB 的工作空间。

在目录 matlabroot/simulink/src 中有一个 C 语言的 MEX 文件 S-function 的模板，sfuntmpl_basic.c。该模板包含了所有 C 语言的 MEX 文件 S-function 可执行的必需和可选的回调函数的基本结构。该目录下的另一个模板程序 sfuntmpl_doc.c 具有更详细的注释。

MEX 文件与 M-文件的 S-function 比较

MEX 文件与 M-文件的 S-function 都有各自的优点。M-文件的 S-function 的优点是开发速度。开发 M-文件的 S-function 避免了开发编译语言的编译-连接-执行所需的时间开销。M-文件的 S-function 还可以更容易地访问 MATLAB 和工具箱函数。

MEX 文件的 S-function 的主要优点是多功能性。更多数量的回调函数及对 SimStruct 的访问使 MEX 函数可实现 M-文件的 S-function 所不能实现的许多功能。这些功能包括，可处理除了 double 之外的数据类型、复数输入、矩阵输入等。

S-Function 的概念

了解以下几个关键概念有助于正确编写 S-function

- 直接馈通
- 动态宽度的输入
- 设置采样时间和偏移量

直接馈通

直接馈通 意味着输出（或者是对于变步长采样块的可变步长）直接受控于一个输入口的值。

有一条很好的经验方法来判断输入是否为直接馈通，如果：

- 输出函数（mdlOutputs 或 flag==3）是输入 u 的函数。即，如果输入 u 在 mdlOutputs 中被访问，则存在直接馈通。输出也可以包含图形输出，类似于一个 XY 绘图板。
- 对于一个变步长 S-function 的“下一步采样时间”函数（mdlGetTimeOfNextVarHit 或 flag==4）中可以访问输入 u。

例如，一个需要其输入的系统（也就是具有直接馈通）是运算 $y = k \times u$，其中，u 是输入，k 是增益，y 是输出。

又如，一个不需要其输入的系统（也就是没有直接馈通）是一种简单的积分运算：

输出：$y = x$

导数：$\dot{x} = u$

其中，x 是状态，$\dot{x}$ 是状态对时间的导数，u 是输入，y 是输出。注意 $\dot{x}$ 是 Simulink 的积分变量。

正确设置直接馈通标志是十分重要的，因为它影响模型中块的执行顺序，并可用来检测代数环。

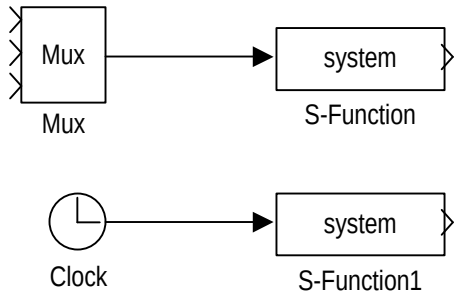
动态维矩阵

S-function 可编写成支持任意维的输入。在这种情况下，当仿真开始时，根据驱动 S-function 的输入向量的维数动态确定实际输入的维数。输入的维数也可用来确定连续状态的数量、离散状态的数量、以及输出的数量。

M-文件的 S-function 只可有一个输入端口，而且输入端口只能接受一维（向量）的信号输入。但是，信号的宽度是可以变化的。在一个 M-文件的 S-function 内，如果要指示输入宽度是动态的，必须在数据结构 sizes 中相应的域值指定为-1，结构 sizes 是在调用 mdlInitializeSizes 时返回的一个结构。当 S-function 通过使用 length(u)来调用时，你可以确定实际输入的宽度。如果指定为 0 宽度，那么 S-function 块中将不出现输入端口。

一个 C S-function 可以有多个 I/O 端口，而且每个端口可以具有不同的维数。维数及每维的大小可以动态确定。

例如，下图所示为在一个模型中，将一个相同的 S-function 块进行了两次使用：



图中上面的 S-function 块由一个带三元素输出的块来作为输入；下面的 S-function 块由一个标量输出块来驱动。通过指定该 S-function 块的输入端口具有动态宽度，那么同一个 S-function 可以适用于两种情况。Simulink 会自动地按照合适宽度的输入端口来调用该块。同样地，如果块的其它特性，如输出数量、离散状态数量或连续状态数量，被指定为动态宽度，那么 Simulink 会将这些向量定义为与输入向量具有相同的长度。

C S-function 在指定输入和输出宽度时提供了更多的灵活性。具体介绍可参考第七章中的“ 创建输入和输出端口 ”一节的内容。

设置采样时间和偏移量

M-文件与 C MEX 文件的 S-function 在指定 S-function 何时执行上都具有高度的灵活性。Simulink 对于采样时间提供了下列选项：

- 连续采样时间——用于具有连续状态和/或非过零采样的 S-function。对于这种类型的 S-function，其输出在每个微步上变化。
- 连续但微步长固定采样时间——用于需要在每一个主仿真步上执行，但在微步长内值不发生变化的 S-function。
- 离散采样时间——如果 S-function 块的行为是离散时间间隔的函数，那么可以定义一个采样时间来控制 Simulink 何时调用该块。你也可以定义一个偏移量来延迟每个采样时间点。偏移量的值不可超过相应采样时间的值。

  采样时间点 发生的时间按照下面的公式来计算：

  TimeHit = (n * period) + offset

  其中，n 是整数，当前仿真步。n 的起始值总是为 0。

  如果定义了一个离散采样时间，Simulink 在每个采样时间点时调用 S-function 的 mdlOutput 和 mdlUpdate。
- 可变采样时间——采样时间间隔变化的离散采样时间。在每步仿真的开始，具有可变采样时间的 S-function 需要计算下一次采样点的时间。
- 继承采样时间——有时，S-function 块没有专门的采样时间特性（即，它既可以是连续的也可以是离散的，取决于系统中其它块的采样时间）。你可以指定这种块的采样时间为 inherited（继承）。比如，一个增益块就是继承采样时间的例子，它从其输入块继承采样时间。

  一个块可以从以下几种块中继承采样时间：

- 输入块
- 输出块
- 系统中最快的采样时间

要将一个块的采样时间设置为继承，那么在 M-文件的 S-function 中使用-1 作为采样时间，在 C S-function 中使用 INHERITED_SAMPLE_TIME 作为采样时间。

S-function 可以是单速率的，也可以是多速率的，多速率系统有多个采样时间。

采样时间是按照固定格式成对指定的：[采样时间，偏移时间]。有效的采样时间对如下：

[CONTINUOUS_SAMPLE_TIME， 0.0]

[CONTINUOUS_SAMPLE_TIME，FIXED_IN_MINOR_STEP_OFFSET]

[*discrete_sample_time_period*，*offset*]

[VARIABLE_SAMPLE_TIME，0.0]

其中：

CONTINUOUS_SAMPLE_TIME = 0.0

FIXED_IN_MINOR_STEP_OFFSET = 1.0

VARIABLE_SAMPLE_TIME = -2.0

斜体字是必须给定的实际值。

另外，还可指定采样时间为从驱动块继承而来。在此情况下，S-function 只能有一个采样时间对：

[INHERITED_SAMPLE_TIME， 0.0]

或者，

[INHERITED_SAMPLE_TIME，FIXED_IN_MINOR_STEP_OFFSET]

其中：

INHERITED_SAMPLE_TIME = -1.0

以下指导方针也许有助于你指定采样时间：

- 一个在积分微步期间变化的连续 S-function 应该采用[CONTINUOUS_SAMPLE_TIME，0.0] 作为采样时间。
- 一个在积分微步期间不变化的连续 S-function 应该采用[CONTINUOUS_SAMPLE_TIME，FIXED_IN_MINOR_STEP_OFFSET] 作为采样时间。
- 一个在指定速率下变化的离散 S-function 应该采用离散采样时间对 [*discrete_sample_time_period*，*offset*]。其中：

  discrete_sample_period > 0.0 和 0.0 = *offset* < *discrete_sample_period*
- 一个在指定速率下变化的离散 S-function 应该采用离散采样时间对 [VARIABLE_SAMPLE_TIME，0.0]

  对于变步长的离散任务，mdlGetTimeOfNextVarHit 程序被调用以确定下一个采样时间点。

如果你的 S-function 没有本身特定的采样时间，你必须将采样时间指定为继承。有两种情况：

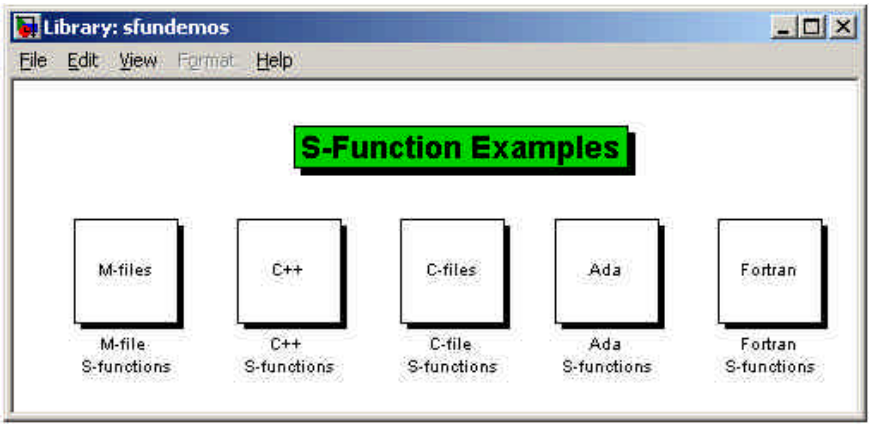
- 即使在积分微步内，S-function 的变化都是随着输入而变化的，应该采用 [INHERITED_SAMPLE_TIME，0.0]作为其采样时间。

- 如果一个 S-function 随着输入而变化，但在一个积分微步内不变化（即积分微步固定），应该采用[INHERITED_SAMPLE_TIME，FIXED_IN_MINOR_STEP_OFFSET] 作为采样时间。Scope 块就是一个这种类型的块。不管驱动块是连续还是离散，Scope 块以其驱动块的速率运行，但是在微步内它应该是不运行的。如果它在微步也运行的化，那么 Scope 应该显示求解器的中间计算结果，而不是每个时间点的最终结果。

S-Function 范例

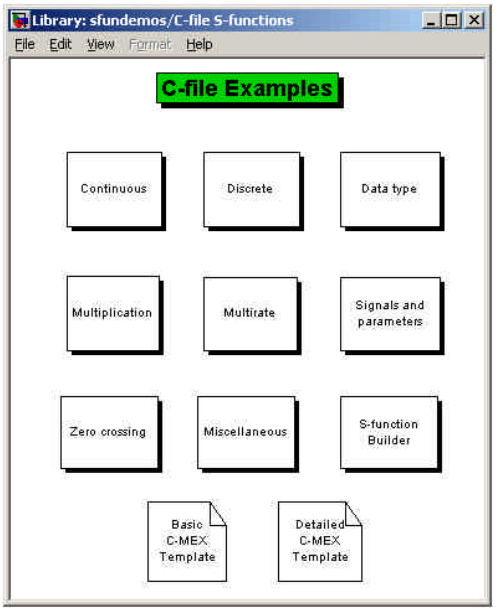
Simulink 提供了一个 S-function 范例库。要运行一个范例，按照以下步骤：

1、在 MATLAB 命令行输入 sfundemos，MATLAB 会显示如下图所示的 S-function 范例库：



库中的每个块代表了一种类别的 S-function 范例。

2、双击一个类别的块，可以显示出它所包含的范例。如下图所示：



3、双击一个块，选择范例并运行。

测试一些在下面所要读到的 S-function 样本是十分有好处的。这些范例的源码保存在 MATLAB 根目录下的以下几个子目录中：

M-files	toolbox/simulink/blocks
C, C++, and Fortran	simulink/src
Ada	simulink/ada/examples

M 文件 S-function 范例

子目录 simulink/blocks 下包含了许多 M 文件的 S-function。建议从通过看这些文件开始。

文件名	说 明
csfunc.m	在状态空间形式下定义一个连续系统
dsfunc.m	在状态空间形式下定义一个离散系统
vsfunc.m	图示说明如何建立一个变步长块。该块实现了一个变步长的延时，在块中第一个输入的延时取决于第二个输入。
mixed.m	由一个积分器与一个单位延迟串连，实现了一个混合系统
vdpm.m	实现范德蒙（Van der Pol）方程（类似于 demo 模型 vdp）
simom.m	内部含有 A,B,C,D 矩阵的状态空间 M 文件 S-function 范例。该 S-function 实现： dx/dt = Ax + By y = Cx + Du 其中：x 是状态向量，u 是输入向量，y 是输出向量；A,B,C,D 矩阵被植入到 M 文件中。
Simom2.m	带有外部 A,B,C,D 矩阵的状态空间 M 文件 S-function 范例。该状态空间的结构与 simom.m 相同，但 A,B,C,D 矩阵作为参数从外部提供。
limintm.m	实现一个连续的有限积分器，其输出被限定在上下限之间，并包含有初始条件。
sfun_varargm.m	示范了一个 M 文件 S-function 来说明如何使用 MATLAB vararg 工具。
vlimintm.m	一个连续有限积分器的 S-function 范例。它举例说明了如何使用宽度设置为-1 来创建一个适应动态输入/状态宽度的 S-function。
vdlimintm.m	一个离散有限积分器的 S-function 范例。该范例与 vlimintm.m 相同，只是其有限积分器是离散的。

C S-Function 范例

子目录 simulink/src 下还包含 C MEX S-function 范例，这些 S-function 中许多都有一个 M 文件的 S-function。这些 C MEX S-function 列表如下：

文件名	说　明
barplot.c	不使用标准块输入访问 simulink 信号
csfunc.c	定义一个连续系统的 C MEX S-function 范例
dlimint.c	实现一个离散时间的有限积分器
dsfunc.c	定义一个离散系统的 C MEX S-function 范例
fcncallgen.c	在指定的速率（采样时间）下，执行 function-call 子系统 n 次
limintc.c	实现一个有限积分器
mixedm.c	通过一个连续积分器（1/s）与一个单位延迟（1/z）串连来实现一个混合动态系统
mixedmex.c	由一个单输出和两个输入来实现一个混合动态系统
quantize.c	一个向量转换器块的 MEX 文件范例。将输入量化为按照量化间隔参数 q 指定的阶数。
resetint.c	复位积分器
sdotproduct.c	计算两个实数或复数向量的点乘（卷积）
sftable2.c	在 S-function 形式下的两维表查询
sfun_atol.c	为每个连续状态设置微分绝对误差
sfun_bitop.c	执行位操作：与、或、异或、左移、右移、以及 uint8，uint16，和 uint32 输入的补码。
sfun_cplx.c	复数信号与一个输入端口和一个参数相加
sfun_directlook.c	直接的一维查表
sfun_dtype_io.c	Simulink 数据类型用于输入和输出的应用范例
sfun_dtype_param.c	Simulink 数据类型用于参数的应用范例
sfun_dynsize.c	演示如何动态排列一个 S-function 输出大小的简单范例
sfun_errhdl.c	说明如何使用 S-function 程序 mdlCheckParams 检查参数的简单范例
sfun_fcncall.c	一个 S-function 被配置为在第一和第三输出元素上执行 function-call 子系统的范例
sfun_frmad.c	基于帧的 A/D 转换器
sfun_frmda.c	基于帧的 D/A 转换器
sfun_frmdft.c	多通道基于帧的离散傅立叶变换（及其反变换）
sfun_frmunbuff.c	基于帧无缓冲器的块

文件名	说　　明
sfun_multiport.c	具有多输入端口和多输出端口的 S-function
sfun_manswitch.c	手动开关
sfun_matadd.c	一个输入端口、一个输出端口和一个参数的矩阵相加
sfun_multirate.c	示范如何指定一个基于端口的采样时间
sfun_psbbreaker.c	实现 Power System Blockset 中 breaker 块的逻辑。
sfun_psbcontc.c	状态空间系统的连续实现
sfun_psbdiscc.c	状态空间系统的离散实现
sfun_runtime1.c	运行过程参数范例
sfun_runtime2.c	运行过程参数范例
sfun_zc.c	示范使用无过零采样来实现 abs (u)，该 S-function 使用变变长求解器。
sfun_zc_sat.c	使用过零检测实现饱和的范例
sfunmem.c	一个一积分步延迟和保持的函数
simomex.c	实现一个单输出，两个输入状态空间的动态系统，系统描述为： dx/dt = Ax + Bu y = Cx + Du 其中：x 是状态向量，u 是输入向量，y 是输出向量。
smatrxcat.c	矩阵串联
sreshape.c	输入信号整形
stspace.c	执行一组状态方程。你可以通过 S-function 块和 mask 功能将其转化为一个新块。该 MEX 文件实现了内置 State-Space 块相同的功能。该范例的输入、输出和状态的数量从 workspace 中获取。可将该例子用作其它 MEX 文件系统的模板。
stvctf.c	实现一个连续时间转移函数，其转移函数多项式通过输入向量来传递。它对于连续时间适应控制应用很有帮助。
stvdtf.c	实现一个离散时间转移函数，其转移函数多项式通过输入向量来传递。它对于离散时间适应控制应用很有帮助。
stvmgain.c	随时间变化的矩阵增益
table3.c	三维查表
timestwo.c	输入乘 2 的基本 C MEX S-function
vdlmint.c	实现一个时间离散向量化的有限积分器
vdpmex.c	实现一个范德蒙方程
vlimint.c	实现一个向量化的有限积分器
vsfunc.c	该范例说明了如何在Simulink中创建一个变步长的块。该块实现了一个变步长延迟，第一输入延迟的时间由第二输入来确定。

Fortran S-Function 范例

下表列出了示范用的 Fortran S-Function：

文件名	说　　明
sfun_timestwo_for.for	C S-function timestwo 的采样级别 1 的 Fortran 表示法。
sfun_atmos.for	1976 年 86 公里标准大气的计算
vdpmex.for	Van der Pol 系统

C++ S-Function 范例

下表列出了示范用的 C++ S-Function：

文件名	说　　明
sfun_counter_cpp.cpp	将一个 C++对象存贮在指针向量 Pwork 所指示的地址中。

Ada S-Function 范例

子目录 simulink/ada/examples 中包含了下面在 Ada 中执行的 S-Function 范例：

文件名	说　　明
matrix_gain	实现一个 Matrix Gain 块
multi_port	Multiport 块
simple_lookup	查表。示范了用单机 Ada 代码包装的特征 S-Function 的用法，该 S-Function 可以与 Simulink 一起使用，也可以直接与 RTW Ada Coder 生成的 Ada 代码一起使用。
times_two	输出为输入的 2 倍

编写 M S-Function

本章以下小节介绍了如何使用 M 编程语言创建一个 S-Function。

概述	介绍 M S-Function 的语法结构
定义 S-Function 块特性	介绍如何指定 M S-Function 所实现的块的状态、输入和输出的数量，以及块的其它属性。
处理 S-Function 参数	介绍如何处理传递到 M S-Function 的块参数
M 文件的 S-Function 范例	介绍了实现几种不同类型块的 M S-Function 范例

概述

一个 M 文件的 S-Function 由一个 MATLAB 函数组成，该函数形式如下：

[sys，x0，str，ts] = f（t，x，u，flag，p1，p2，...）

其中，f 是 S-function 的函数名。在模型的仿真过程中，Simulink 反复调用 f，并通过 flag 参数来指示每次调用所需完成的任务（或多个任务）。每次 S-function 执行任务，并将执行结果通过一个输出向量返回。

sfuntmpl.m 是实现 M 文件 S-function 的一个模板，存放在 *matlabroot*/toolbox/simulink/blocks 目录下。该模板由一个顶层的函数和一组骨架子函数组成，这些骨架函数被称为 S-Function 的回调函数，每一个回调函数对应着一个特定的 flag 参数值，顶层函数通过 flag 的指示来调用不同的子函数。在仿真过程中，子函数执行 S-function 所要求的实际任务。

S-Function 参数

Simulink 传递以下参数给 S-function：

t　　当前时间

x　　状态向量

u　　输入向量

flag　　用来指示 S-function 所执行任务的标志，是一个整数值。

下表给出了参数 flag 可以取的值，并列出了每个值所对应的 S-function 函数：

flag	S-function 程序	说　明
0	mdlInitializeSizes	定义 S-function 块的基本特性，包括采样时间，连续和离散状态的初始化条件，以及 sizes 数组。
1	mdlDerivatives	计算连续状态变量的导数
2	mdlUpdate	更新离散状态、采样时间、主步长等必需条件
3	mdlOutputs	计算 S-function 的输出
4	mdlGetTimeOfNextVarHit	计算下一个采样点的绝对时间。只有当在 mdlInitializeSizes 中指定了变步长离散采样时间时，才使用该程序。
9	mdlTerminate	执行 simulink 终止时所需的任何任务

S-Function 的输出

一个 M 文件返回的输出向量包含以下元素：

- sys，一个通用的返回参数。返回值取决于 flag 的值。例如：flag = 3，sys 则包含了 S-function 的输出。
- x0，初始状态值（如果系统中没有状态，则向量为空）。除 flag = 0 外，x0 被忽略。

- str，保留以后使用。M 文件 S-function 必须设置该元素为空矩阵，[]。
- ts，一个两列的矩阵，包含了块的采样时间和偏移量（参考在线帮助文件中的“ Specifying Sample Time ”可获取如何指定块的采样时间和偏移量的相关信息）。

  例如，如果你希望你的 S-function 在每个时间步（连续采样时间）都运行，则应设置为[0，0]；如果你希望你的 S-function 按照其所连接块的速率来运行，则应设置为[-1，0]；如果你希望它在仿真开始的 0.1 秒后每 0.25 秒（离散采样时间）运行一次，则应设置为[0.25，0.1]。

你可以创建一个 S-function 按照不同的速率来执行不同的任务（如：一个多速率 S-function）。在这种情况下，ts 应该按照采样时间升序排列来指定 S-function 所需使用的全部采样速率。例如，假设你的 S-function 每 0.25 秒执行一个任务，同时在仿真开始的 0.1 秒后每 1 秒执行另一个任务，那么你的 S-function 应设置 ts 为[0.25，0；1.0，0.1]。这将使 Simulink 按照这样的时间序列来执行 S-function：[0，0.1，0.25，0.5，0.75，1，1.1，...]。你的 S-function 必须确定在每一个采样时间点执行哪一个任务。

你也可以一个 S-function 来连续地执行一些任务（如：在每个时间步），同时按照离散间隔时间执行另一些任务。参考本章“ 范例-混合系统 S-function ”来了解相关信息。

定义 S-Function 块特性

为了使 Simulink 认识 M 文件 S-function，你必须提供给 Simulink 有关于 S-function 的一些特殊信息。这些信息包括输入、输出、状态的数量，以及其它块特性。

为了给 Simulink 提供这些信息，必须在 mdlInitializeSizes 的开头调用 simsizes：

```
sizes = simsizes；
```

该函数返回一个未初始化的 sizes 结构，你必须将 S-function 的信息装载在 sizes 结构中。下表列出了 sizes 结构的域，并对每个域所包含的信息进行了说明：

域　名	说　明
sizes.NumContStates	连续状态的数量
sizes.NumDiscStates	离散状态的数量
sizes.NumOutputs	输出的数量
sizes.NumInputs	输入的数量
sizes.DirFeedthrough	直接馈通标志
sizes.NumSampleTimes	采样时间的数量

在初始化 sizes 结构之后，再次调用 simsizes：

```
sys = simsizes(sizes)；
```

此次调用将 sizes 结构中的信息传递给 sys，sys 是一个保持 Simulink 所用信息的向量。

处理 S-Function 参数

当调用 M 文件 S-function 时，Simulink 总是传递标准块参数——t，x，u 和 flag 到 S-function 作为函数参数。Simulink 还可以传递用户另外指定的特定块参数给 S-function，这些参数在 S-function 块参数对话框的 S-function parameters 中，由用户指定。如果在对话框中指定了附加参数，那么 Simulink 将它们作为函数的附加参数传递给 S-function。这些附加参数在 S-function 的参数表中紧随标准参数之后，并以参数出现在对话框中的顺序作为 S-function 的参数表中附加参数的顺序。你可以使用 S-function 块指定参数的特性来实现一个 S-function 执行不同的处理选项。参考 toolbox/simulink/blocks 子目录内的 limintm.m 范例，就使用了块指定参数来实现多种处理。

M 文件的 S-Function 范例

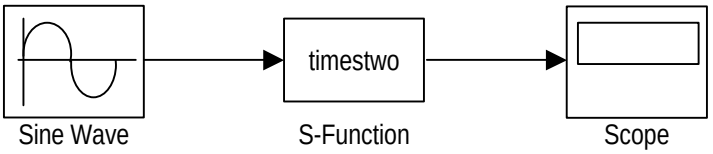
要了解 S-function 是如何工作的，最简单的方法就是通过学习 S-function 范例。本节从一个简单的范例——没有状态的 S-function（timestwo）入手，逐步深入。绝大多数 S-function 块需要处理状态，有连续的也有离散的。本节对四种公共类型的系统进行了讨论，你可以将这些 S-function 块在 Simulink 模型中使用。

- 连续
- 离散
- 混合
- 变步长

所有的范例都是基于 M 文件 S-Function 的模板——sfuntmpl.m 来编写的。

范例 1——简单的 M 文件 S-Function

该块输入一个标量信号，将信号加倍，然后输出到一个 scope 进行显示。



在 Simulink 中，我们模仿 S-function 结构建立了一个包含 S-function 功能的 M 代码模板文件——sfuntmpl.m。借助于该模板，你可以创建一个 M 文件的 S-function，其形式类似于 C MEX S-function。因而，这可使得从一个 M 文件到一个 C MEX 文件的转换变得很容易。

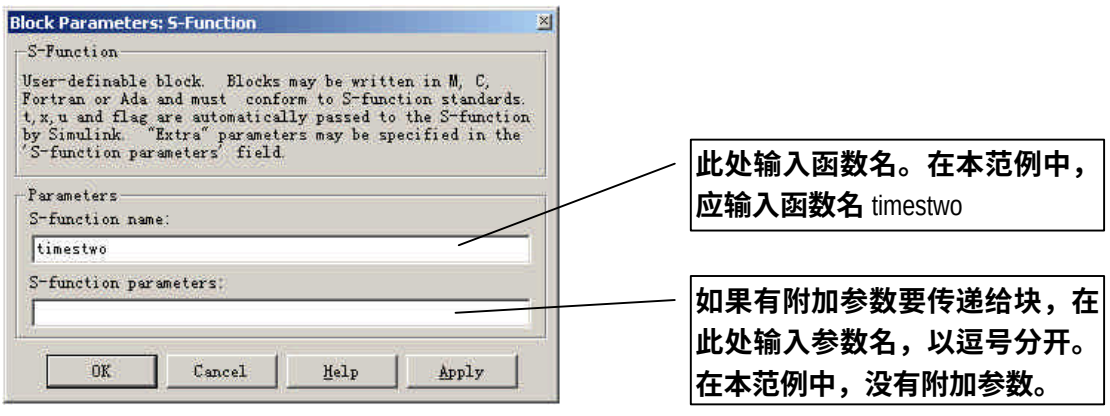
以下是 S-Function timestwo.m 的 M 文件代码：

```
function [sys,x0,str,ts] = timestwo(t,x,u,flag)
% Dispatch the flag. The switch function controls the calls to
% S-function routines at each simulation stage.

    switch flag,

    case 0
        [sys,x0,str,ts] = mdlInitializeSizes;          % Initialization

    case 3
        sys = mdlOutputs(t,x,u);                       % Calculate outputs

    case { 1, 2, 4, 9 }
        sys = [];                                      % Unused flags

    otherwise
        error(['Unhandled flag = ',num2str(flag)]);    % Error handling

end;                                                   % End of function timestwo.
```

下面是 timestwo.m **要调用的子程序：**

```
%=============================================================
% Function mdlInitializeSizes initializes the states, sample
% times, state ordering strings (str), and sizes structure.
%=============================================================
function [sys,x0,str,ts] = mdlInitializeSizes
% Call function simsizes to create the sizes structure.
sizes = simsizes;
% Load the sizes structure with the initialization information.
sizes.NumContStates= 0;
sizes.NumDiscStates= 0;
sizes.NumOutputs= 1;
sizes.NumInputs= 1;
sizes.DirFeedthrough=1;
sizes.NumSampleTimes=1;
% Load the sys vector with the sizes information.
sys = simsizes(sizes);
%
x0 = []; % No continuous states
%
str = []; % No state ordering
%
ts = [-1 0]; % Inherited sample time
% End of mdlInitializeSizes.
%=============================================================
% Function mdlOutputs performs the calculations.
%=============================================================
function sys = mdlOutputs(t,x,u)
sys = 2*u;

% End of mdlOutputs.
```

为了在 Simulink **中测试该** S-function**，将一个正弦波发生器连接到** S-function **块的输入，将** S-function **块的输出连接到一个** Scope**。双击** S-function **块打开对话窗：**



现在，你可以运行这个仿真了。

范例 2——连续状态 S-Function

Simulink 内包含了一个名为 csfunc.m 的函数，它是一个通过 S-function 模拟的连续状态系统的范例。下面是该 M 文件 S-function 的代码：

```
function [ sys，x0，str，ts ] = csfunc( t，x，u，flag )
% CSFUNC An example M-file S-function for defining a system of
% continuous state equations:
%    x'    =   Ax + Bu
%    y     =   Cx + Du
%
% Generate a continuous linear system:
A =  [     –0.09      –0.01
           1          0     ]；
B =  [     1          –7
           0          –2    ]；
C =  [     0          2
           1          –5    ]；
D =  [     –3         0
           1          0     ]；
%
% Dispatch the flag.
%
switch flag，
    case 0
        [sys，x0，str，ts] = mdlInitializeSizes(A，B，C，D)；      % Initialization
    case 1
        sys = mdlDerivatives(t，x，u，A，B，C，D)；                % Calculate derivatives
    case 3
        sys = mdlOutputs(t，x，u，A，B，C，D)；                    % Calculate outputs
    case { 2，4，9 }              % Unused flags
        sys = [ ]；
    otherwise
        error(['Unhandled flag = '，num2str(flag)])；              % Error handling
end
% End of csfunc.
%=============================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the
% S-function.
%=============================================================
%
function [sys,x0,str,ts] = mdlInitializeSizes(A,B,C,D)
%
% Call simsizes for a sizes structure, fill it in and convert it
% to a sizes array.
```

```
%
sizes = simsizes;
sizes.NumContStates = 2;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 2;
sizes.NumInputs = 2;
sizes.DirFeedthrough = 1; % Matrix D is nonempty.
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
%
% Initialize the initial conditions.
%
x0 = zeros(2,1);
%
% str is an empty matrix.
%
str = [];
%
% Initialize the array of sample times; in this example the sample
% time is continuous, so set ts to 0 and its offset to 0.
%
ts = [0 0];
% End of mdlInitializeSizes.
%
%=============================================================
% mdlDerivatives
% Return the derivatives for the continuous states.
%=============================================================
function sys = mdlDerivatives(t,x,u,A,B,C,D)
sys = A*x + B*u;
% End of mdlDerivatives.
%
%=============================================================
% mdlOutputs
% Return the block outputs.
%=============================================================
%
function sys = mdlOutputs(t,x,u,A,B,C,D)
sys = C*x + D*u;
% End of mdlOutputs.
```

上面的范例与本文最初介绍的仿真步骤一致。与 timestwo.m **不同的是，当** flag = 1 **时本范例调用了** mdlDerivatives **来计算连续状态变量的导数。系统状态方程如下：**

$$x' = Ax + Bu$$

$$y = Cx + Du$$

因此，非常通用的连续微分方程组可以通过 csfunc.m 来模拟。注意，csfunc.m 与 Simulink 内置的 State-Space 块十分类似。该 S-function 可以作为建立一个时变系数的状态空间系统的基础。

每次调用 mdlDerivatives 程序，它必须明确地设置所有导数的值，导数向量不保留上一次调用该程序所得到的值。在执行过程中，分配给导数向量的内存一直在变化。

范例 3——离散状态 S-Function

Simulink 包含了一个名为 dsfunc.m 的函数，它是一个通过 S-function 模拟的离散状态系统的范例。该函数与连续状态系统的范例 csfunc.m 十分相似，唯一的区别是在该函数中，调用的是 mdlUpdate，而不是调用 mdlDerivatives。当 flag = 2 时，mdlUpdate 更新离散状态。注意，对于一个单速率离散 S-function 而言，Simulink 只在采样点调用 mdlUpdate，mdlOutputs，以及 mdlGetTimeOfNextVarHit（如果需要）。以下是该 M 文件 S-function 的代码：

```
function [sys，x0，str，ts] = dsfunc( t，x，u，flag )
% An example M-file S-function for defining a discrete system.
% This S-function implements discrete equations in this form:
%    x(n+1) =   Ax(n) + Bu(n)
%    y(n)    =   Cx(n) + Du(n)
%
% Generate a discrete linear system:
A = [ –1.3839   –0.5097
      1.0000    0          ]；
B = [ –2.5559   0
      0         4.2382     ]；
C = [ 0         2.0761
      0         7.7891     ]；
D = [ –0.8141   –2.9334
      1.2426    0          ]；
switch flag，
    case 0
        sys = mdlInitializeSizes(A，B，C，D)；     % Initialization
    case 2
        sys = mdlUpdate(t，x，u，A，B，C，D)；% Update discrete states
    case 3
        sys = mdlOutputs(t，x，u，A，B，C，D)；% Calculate outputs
    case {1，4，9}         % Unused flags
        sys = [ ]；
    otherwise
        error(['unhandled flag = '，num2str(flag)])；% Error handling
end
% End of dsfunc.
%=============================================================
% Initialization
%=============================================================
function   [sys，x0，str，ts] = mdlInitializeSizes( A，B，C，D )
```

```
% Call simsizes for a sizes structure, fill it in, and convert it
% to a sizes array.
sizes = simsizes；
sizes.NumContStates = 0；
sizes.NumDiscStates = 2；
sizes.NumOutputs = 2；
sizes.NumInputs = 2；
sizes.DirFeedthrough = 1；     % Matrix D is non-empty.
sizes.NumSampleTimes = 1；
sys = simsizes(sizes)；
x0 = ones(2,1)；               % Initialize the discrete states.
str = [ ]；                    % Set str to an empty matrix.
ts = [ 1   0 ]；               % sample time: [period, offset]
% End of mdlInitializeSizes.
%=============================================================
% Update the discrete states
%=============================================================
function   sys = mdlUpdates( t，x，u，A，B，C，D )
sys = A*x + B*u；
% End of mdlUpdate.
%=============================================================
% Calculate outputs
%=============================================================
function   sys = mdlOutputs( t，x，u，A，B，C，D )
sys = C*x + D*u；
% End of mdlOutputs.
```

上面的范例与在第一章中介绍的仿真步骤一致。该系统离散状态方程如下：

x(n+1) = Ax(n) + Bu(n)

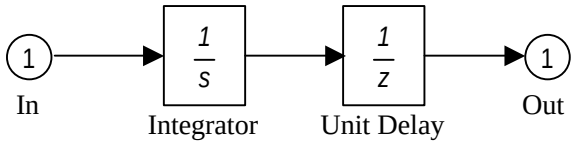
y(n) = Cx(n) + Du(n)

因此，使用 dsfunc.m **可以模拟一组通用的差分方程组。注意，**dsfunc.m **与** Simulink **内置的离散** State-Space **块十分类似。该** S-function **可以作为建立一个时变系数的离散状态空间系统的基础。**

范例 4——混合系统 S-Function

Simulink **包含了一个名为** mixed.m **的函数，它是一个通过** S-function **模拟的混合系统（组合了连续和离散状态）的范例。处理混合系统十分直接，通过参数** flag **来控制对于系统中的连续和离散部分调用正确的** S-function **子程序。混合系统** S-function **（或者任何多速率系统** S-function**）的一个差别之处就是在所有的采样时间上，**Simulink **都会调用** mdlUpdate**，**mdlOutputs**，以及** mdlGetTimeOfNextVarHit **程序。这意味着在这些程序中，你必须进行测试以确定正在处理哪个采样点以及哪些采样点只执行相应的更新。**

mixed.m **模拟了一个连续积分器及随后的离散的单位延迟。按照** Simulink **方块图的形式，该函数的功能可用如下模型来说明：**



以下是该 M 文件 S-function 的代码：

```
function    [sys，x0，str，ts] = mixedm( t，x，u，flag )

% A hybrid system example that implements a hybrid system
% consisting of a continuous integrator (1/s) in series with a
% unit delay (1/z).
%
% Set the sampling period and offset for unit delay.

dperiod = 1；
doffset = 0；
switch flag ，
    case 0              % Initialization
        [sys，x0，str，ts] = mdlInitializeSizes(dperiod，doffset)；
    case 1
        sys = mdlDerivatives(t，x，u)；                    % Calculate derivatives
    case 2
        sys = mdlUpdate(t，x，u，dperiod，doffset)；   % Update disc states
    case 3
        sys = mdlOutputs(t，x，u，doffset，dperiod)； % Calculate outputs
    case { 4，9 }
        sys = [ ]；                                        % Unused flags
    otherwise
        error(['unhandled flag = '，num2str(flag)])；       % Error handling
end
% End of mixedm.
%
%=============================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=============================================================
function    [sys，x0，str，ts] = mdlInitializeSizes( dperiod，doffset )
sizes = simsizes；
sizes.NumContStates = 1；
sizes.NumDiscStates = 1；
sizes.NumOutputs = 1；
sizes.NumInputs = 1；
sizes.DirFeedthrough = 0；
sizes.NumSampleTimes = 2；
sys = simsizes(sizes)；
x0 = ones(2，1)；
str = [ ]；
ts = [ 0，          0                              % sample time
      dperiod，  doffset ]；
```

```
% End of mdlInitializeSizes.
%
%=============================================================
% mdlDerivatives
% Compute derivatives for continuous states.
%=============================================================
%
function    sys = mdlDerivatives( t，x，u )
sys = u；
% end of mdlDerivatives.
%
%=============================================================
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step requirements.
%=============================================================
%
function    sys = mdlUpdate( t，x，u，dperiod，doffset )
% Next discrete state is output of the integrator.
% Return next discrete state if we have a sample hit within a
% tolerance of 1e-8. If we don't have a sample hit, return [] to
% indicate that the discrete state shouldn't change.
%
if abs(round((t -doffset)/dperiod) -(t -doffset)/dperiod) < 1e-8
    sys = x(1)；            % mdlUpdate is "latching" the value of the
                            % continuous state, x(1), thus introducing a delay.
else
    sys = [ ]；             % This is not a sample hit, so return an empty
                            % matrix to indicate that the states have not changed.
end
% End of mdlUpdate.
%
%=============================================================
% mdlOutputs
% Return the output vector for the S-function.
%=============================================================
%
function    sys = mdlOutputs( t，x，u，doffset，dperiod )
% Return output of the unit delay if we have a sample hit within a tolerance of 1e-8.
% If we don't have a sample hit then return [] indicating that the output shouldn't change.
%
if abs(round((t-doffset)/dperiod)-(t-doffset)/dperiod) < 1e-8
    sys = x(2)；
else
    sys = [ ]；         % This is not a sample hit, so return an empty
                        % matrix to indicate that the output has not changed
end
% End of mdlOutputs.
```

范例 5——变步长 S-Function

该 M 文件是一个 S-function 的范例，它的采样时间采用的是变步长。该范例是一个 M 文件——vsfunc.m，当 flag = 4 时，它调用了 mdlGetTimeOfNextVarHit 子程序。因为下一步采样时间点的计算取决于输入 u，该块具有直接馈通。一般情况下，所有使用输入计算下一采样步长（flag = 4）的块都要求直接馈通。以下是该 M 文件 S-function 的代码：

```
function    [sys，x0，str，ts] = vsfunc( t，x，u，flag )

% This example S-function illustrates how to create a variable step block in Simulink.
% This block implements a variable step delay in which the first input is delayed
% by an amount of time determined by the second input.
%
% dt = u(2)
% y(t+dt) = u(t)
%
switch flag ，
    case 0
        [sys，x0，str，ts] = mdlInitializeSizes；     % Initialization
    case 2
        sys = mdlUpdate(t，x，u)；                    % Update Discrete states
    case 3
        sys = mdlOutputs(t，x，u)；                   % Calculate outputs
    case 4
        sys = mdlGetTimeOfNextVarHit(t，x，u)；  % Get next sample time
    case { 1, 9 }
        sys = [ ]；                                   % Unused flags
    otherwise
        error(['Unhandled flag = '，num2str(flag)])；% Error handling
end
% End of vsfunc.
%=============================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=============================================================
%
function    [sys，x0，str，ts] = mdlInitializeSizes
%
% Call simsizes for a sizes structure, fill it in and convert it to a sizes array.
%
sizes = simsizes；
sizes.NumContStates = 0；
sizes.NumDiscStates = 1；
sizes.NumOutputs = 1；
sizes.NumInputs = 2；
sizes.DirFeedthrough = 1；          % flag=4 requires direct feedthrough
                                    % if input u is involved in calculating the next sample time hit.
%
```

```
sizes.NumSampleTimes = 1；
sys = simsizes(sizes)；
%
% Initialize the initial conditions.
%
x0 = [0]；
%
% Set str to an empty matrix.
%
str = [ ]；
%
% Initialize the array of sample times.
%
ts = [–2    0]；                          % variable sample time
% End of mdlInitializeSizes.
%
%=============================================================
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step requirements.
%=============================================================
%
function    sys = mdlUpdate( t，x，u )
sys = u(1)；
% End of mdlUpdate.
%
%=============================================================
% mdlOutputs
% Return the block outputs.
%=============================================================
%
function    sys = mdlOutputs(t，x，u)
sys = x(1)；
% end mdlOutputs
%
%=============================================================
% mdlGetTimeOfNextVarHit
% Return the time of the next hit for this block. Note that the result is absolute time.
%=============================================================
%
function    sys = mdlGetTimeOfNextVarHit( t，x，u )
sys = t + u(2)；
% End of mdlGetTimeOfNextVarHit.
```

mdlGetTimeOfNextVarHit **返回下一个采样点的时间，这个时间就是在仿真中下一次调用** vsfunc **的时间。这意味着直到下一次点之前，该** S-function **没有输出。在** vsfunc **中，下一采样时间点的时间被设置为** t + u(2)**，即意味着第二输入** u(2)**设置了下一次调用** vsfunc **的时间**

使用 C 语言编写 S-Function

本章以下小节介绍了如何使用 C 编程语言创建一个 S-Function。

概述	对于 C S-Function 的一般介绍
自动生成 S-Function	介绍如何使用 S-Function Builder 根据用户提出的要求自动生成 S-Function
S-Function Builder 的对话窗	介绍 S-Function Builder 的对话窗
一个基本的 C MEX S-Function 范例	举例说明了创建一个 C S-Function 所需的代码
C S-Function 模板	介绍代码模板，你可以使用该模板作为创建自己的 C S-Function 的基础。
Simulink 如何与 C S-Function 相互作用	介绍 Simulink 如何与 C S-Function 相互作用。了解这部分信息有助于创建和调试自己的 C S-Function。
编写回调函数	如何编写被 Simulink 调用来执行用户功能的函数
将 Level 1 C MEX S-Function 转换到 Level 2	介绍如何将针对于早期版本的 Simulink 编写的 S-Function 转换到当前版本上运行

概述

一个用来定义一个 S-Function 块的 C MEX 文件在仿真过程中必须向 Simulink 提供模型的相关信息。在仿真进程中，Simulink、ODE 求解器、以及 MEX 文件相互作用以完成特定的任务。这些任务包括定义初始化条件和块特性，以及计算导数、离散状态和输出。

与 M 文件的 S-Function 一样，Simulink 与一个 C MEX S-Function 之间的相互作用是通过调用 S-Function 中的回调函数来实现的。每个函数完成一个预先定义的任务，诸如计算块的输出，这些任务是 S-function 定义的仿真块功能必需的。Simulink 采用通用的方法来定义每个回调函数的任务，S-function 根据其实现的功能性自由地执行任务。例如，Simulink 指定 S-function 的 mdlOutput 函数在当前仿真时间计算块的输出，它不会指定输出应该是什么。因此，这个基于回调的 API 允许你创建任何希望功能的 S-function，并生成相应的块。

C MEX 文件实现的这组回调函数的功能性要比 M 文件 S-function 可实现的功能大得多。参考第九章“S-Function 回调函数”中关于 C MEX S-function 可实现的回调函数的介绍。与 M 文件 S-function 不同，C MEX 文件可以访问和修改 Simulink 内部使用的数据结构，用来存储 S-function 的有关信息。由于 C MEX 文件具有实现功能强大的回调函数和访问内部数据结构的能力，因而 C MEX 文件可实现范围更广的块特性，比如可处理矩阵信号和多数据类型。

C MEX 文件 S-function 所需实现的只是由 Simulink 定义的一组很少的回调函数。如果你的块不需要实现特殊性能，比如矩阵信号，你可以忽略实现这些特性所需的回调函数。因而，可以很快地创建一些简单的块。

一个 C MEX S-function 的一般格式如下：

```
#define S_FUNCTION_NAME your_sfunction_name_here
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"

static void mdlInitializeSizes(SimStruct   *S)
{
}
< 附加的S-function程序/代码 >

static void mdlTerminate(SimStruct *S)
{
}
#ifdef MATLAB_MEX_FILE      /* Is this file being compiled as a MEX-file? */
#include "simulink.c"       /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"        /* Code generation registration function */
#endif
```

当 Simulink 与 S-function 相互作用时，Simulink 所调用的第一个程序是 mdlInitializeSizes，随后调用的是其它 S-function 函数（所有的函数名均以 mdl 开头）。在仿真结束时，Simulink 调用 mdlTerminate 函数。

注意：与 M 文件 S-function 不同，C MEX S-function 函数均不需要 flag 参数。这是因为在仿真过程中的一定时刻，Simulink 直接调用每个 S-function 函数。

创建 C MEX S-Function

创建 C MEX S-Function 最简单的方法是使用 S-Function Builder（参考下节），该工具可以根据你提供的要求和部分代码来构建一个 C MEX S-Function。这省去了你从最初构想到程序流程设计整个工作。但是，S-Function Builder 只限于生成几种类型的 S-function。例如，它生成的 S-function 不能有一个以上的输入或输出，也不能处理除了 double 以外的其它数据类型。对于其它需要的 S-function，你必须从头开始创建。

以下各节提供了从头编写 C MEX S-function 的有关介绍：

- **“一个基本的 C MEX S-Function 范例”一步步地示范了如何从头编写一个简单的 S-function。**
- **“C S-Function 的模板”介绍了组成一个 C S-function 的完整框架，你可以以该模板为基础来创建自己的 S-function。**

自动生成 S-Function

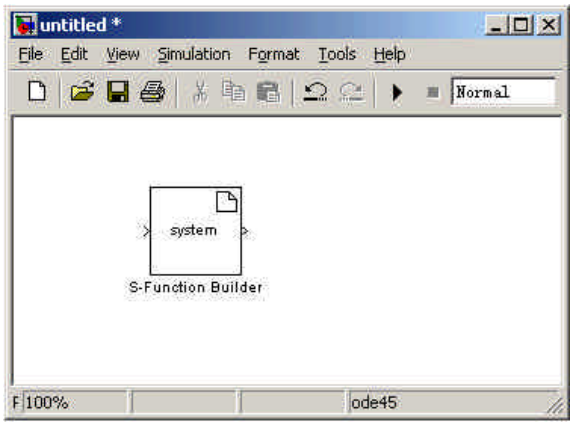
S-Function Builder 是一个根据你提供的要求和 C 代码来构建一个 S-function 的块，它也可以为在模型中使用的现成 S-function 进行包装。本节介绍了如何使用 S-Function Builder 来生成简单的 C MEX S-function。

要使用 S-Function Builder 生成一个 S-function，按照以下步骤进行：

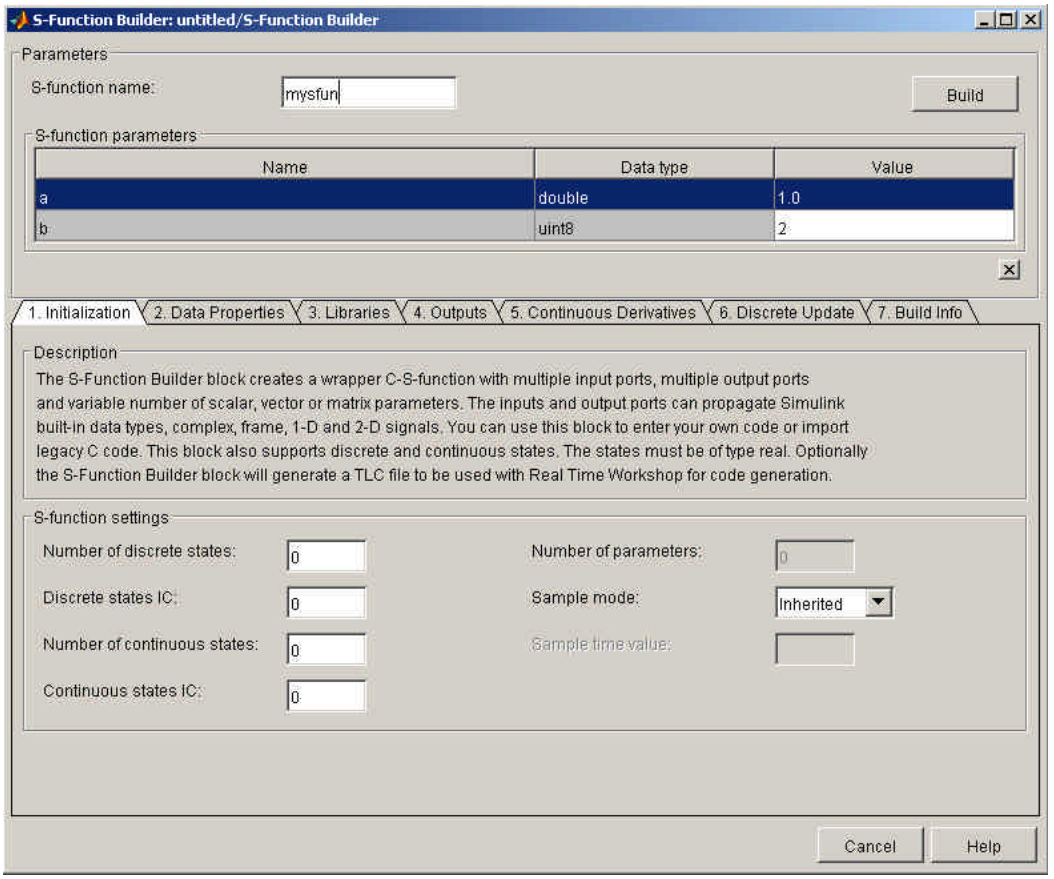
1、 将 MATLAB 当前目录设置为你想创建 S-function 的目录；

2、 创建一个新的 Simulink 模型；

3、 从 Simulink 的 User-Defined Functions 库中拖拽一个 S-Function Builder 块到新建的模型中。



4、 双击该块，打开 S-Function Builder 对话窗



5、 在 S-function name 区输入 S-function 的名字；

6、 如果该 S-function 带参数，在 S-function parameters 区输入参数的默认值；

7、 使用 S-Function Builder 对话框上的规范和代码输入窗输入所需的信息和用户源代码根据应用来裁减生成的 S-function。

8、 如果你还没有配置过 MATLAB mex 命令，那么必须在系统上配置 MATLAB mex 命令。要配置 mex 命令，只需在 MATLAB 命令行中输入 mex -setup 后，回车。

9、 在对话窗中点击 Build 按钮即可开始创建过程。Simulink 创建指定 S-function 的 MEX 文件，并将其保存在当前工作目录中。

10、 存包含了 S-Function Builder 块的模型。

配置生成的 S-Function

为了在另一个模型中使用所生成的 S-function，首先应检查并确保包含所生成 S-function 的目录是否在 MATLAB 路径下；然后，从刚才用来创建 S-function 的模型中复制 S-Function Builder 块到目标模型中，如有必要还应将参数设置为目标模型所要求的值。

S-Function Builder 如何生成 S-Function

创建 S-function 的过程如下。首先，它在当前目录下生成以下源文件：

- **sfun.c**

  其中，sfun 是你在 S-Function Builder 对话框的 S-function name 区指定的 S-function 名字。该文件包含了所生成的 S-function 标准部分的源代码。

- **sfun_wrapper.c**

  该文件包含了你在 S-Function Builder 对话框中输入的客户代码。

- **sfun.tlc**

  该文件允许 Simulink 在加速模式下运行所生成的 S-function，并允许 RTW 包含由该 S-function 生成的代码。

在生成 S-function 源代码之后，S-Function Builder 使用 MATLAB mex 命令创建 S-function 的 MEX 形式的文件，它包含了生成的源代码、任何外部的客户源代码、以及你指定的函数库。

设置 include 路径

S-Function Builder 按照 MATLAB 应用数据指定的目录来搜索客户头文件，MATLAB 的应用数据名称为 SfunctionBuilderIncludePath。该数据与你创建 S-Function Builder 块的模型联合起来。如果 S-function 使用了客户头文件，而这些头文件有不在当前目录下（例如，包含了生成 S-function 的目录），你必须更新 SfunctionBuilderIncludePath 以指定包含头文件的目录位置。SfunctionBuilderIncludePath 是一个三元素的元胞数组，可允许你指定三个包含目录。例如，下面的 MATLAB 命令可将 SfunctionBuilderIncludePath 设置为具有两个包含目录的路径：

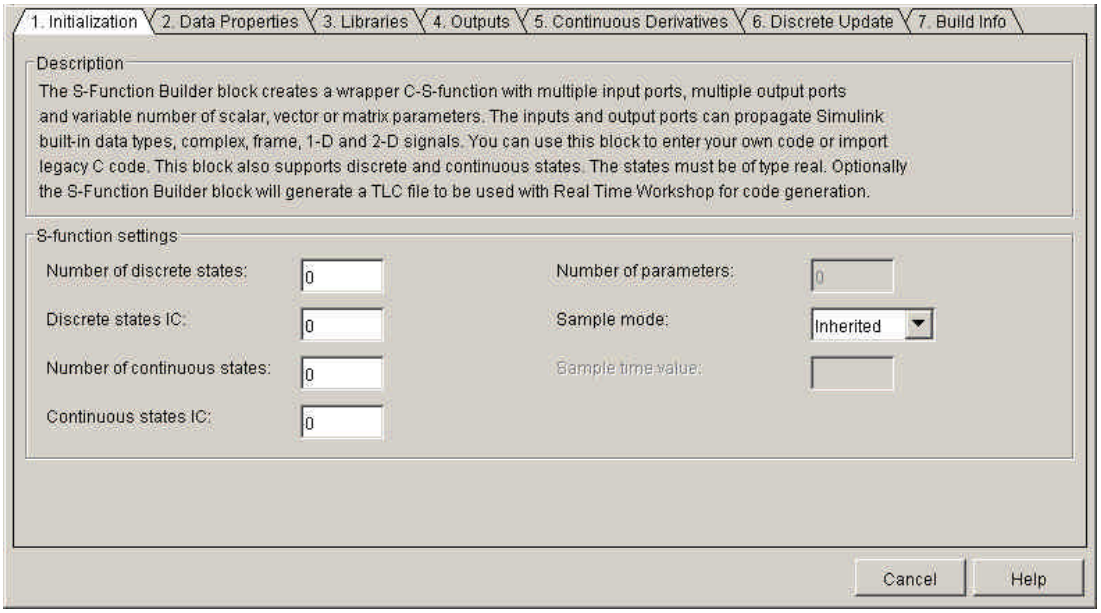
```
incPath = getappdata( 0， 'SfunctionBuilderIncludePath')；
incPath{1} =   '/home/jones/include'；
incPath{2} =   getenv( 'PROJECT_INCLUDE_DIR' )
setappdata( 0，'SfunctionBuilderIncludePath'，incPath )
```

S-Function Builder 的对话窗

S-Function Builder 对话窗的各标签选卡可使你输入信息和所需的客户化代码，以便让 S-function 实现一个特定的应用。该对话窗包含以下选卡：

Initialization 选卡

Initialization 选卡允许你指定 S-function 的基本特性，比如，输入和输出端口的宽度，采样时间等。



S-Function Builder 使用你在该选卡上输入的信息来生成 S-function 的 mdlInitializeSizes 回调函数。在模型仿真的初始化阶段，Simulink 调用该函数以获取 S-function 的基本信息。

Initialization 选卡包含以下区域：

Number of discrete states　S-function 中离散状态的数量

Discrete states IC　S-function 离散状态的初始条件。你可以输入离散状态的初始值，用逗号分隔，也可以用向量形式输入（如，[0　1　2]）。初始条件的数量必须与离散状态的数量相等。

Number of continuous states　S-function 中连续状态的数量

Continuous states IC　S-function 连续状态的初始条件。你可以输入连续状态的初始值，用逗号分隔，也可以用向量形式输入（如，[0　1　2]）。初始条件的数量必须与连续状态的数量相等。

Sample mode　S-function 的采样模式。采样模式决定了 S-function 更新输出的时间间隔。你可选择以下选项中的一种：

- Inherited ——S-function 从其输入端口所连接的块中继承采样时间。
- Continuous——块在每个仿真步长都更新输出。
- Discrete ——S-function 按照在 Sample time value 区中指定的速率更新输出。

Sample time value　S-function 更新输出的时间间隔。只有当 Sample mode 区选择为 Discrete 时，才激活该选择区。

Input port width　S-function 输入端口的宽度。此宽度是连接到输入端口向量信号的元素数

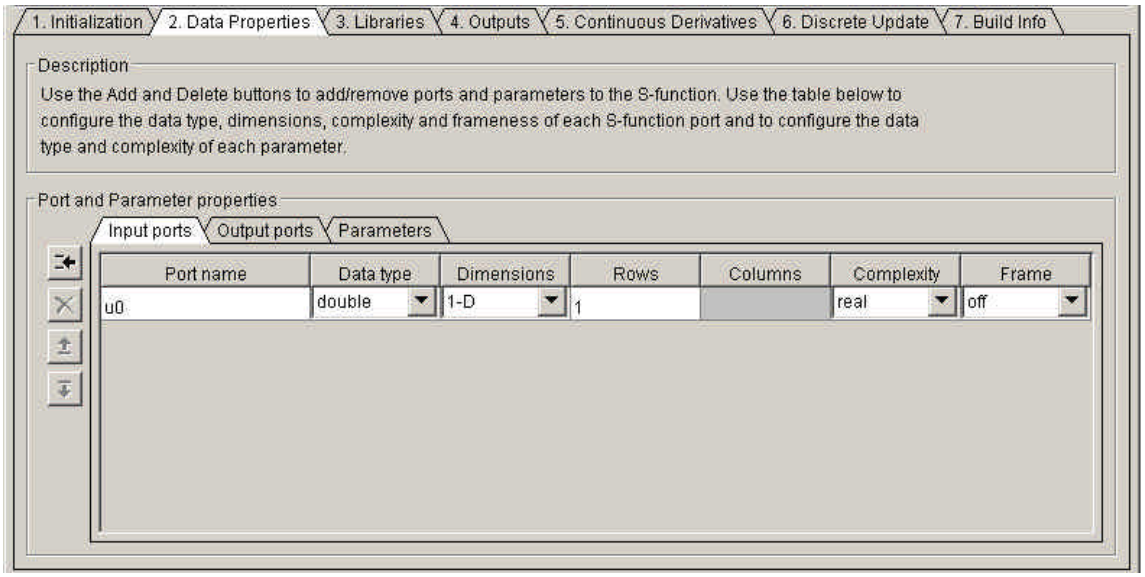
量。如果要在输入端口连接矩阵信号（2 维），应将输入口宽度设置为-1。

Output port width S-function 输出端口的宽度。此宽度是该 S-function 输出向量信号的元素数量。如果该 S-function 输出矩阵信号（2 维），应将输出口宽度设置为-1。

Number of parameters S-function 接受的参数数量。

Data Properties 选卡

Data Properties 选卡允许你为增加端口和参数。



该选卡本身又包含了三个子标签选卡，分别显示 S-function 的以下属性：

- 输入端口（参考“ Input ports 子选卡 ”的介绍）
- 输出端口（参考“ Output ports 子选卡 ”的介绍
- 参数（参考“ Parameters 子选卡 ”的介绍）

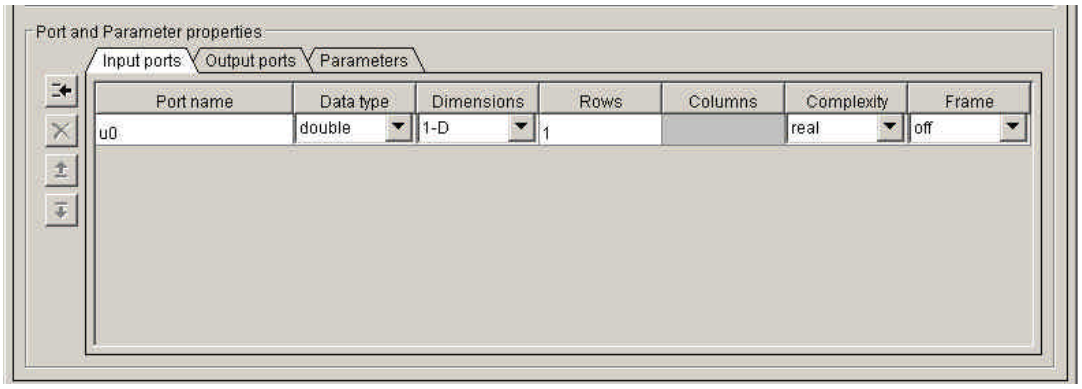
选卡左边的一列按钮允许你增加、删除、或重新排序端口或参数，具体是哪种端口或参数，取决于你选定的当前子选卡。

- 如果要增加一个端口或参数，点击 Add 按钮（按钮列的最上面的按钮）
- 如果要删除当前选定的端口或参数，点击 Delete 按钮（Add 按钮下面的按钮）
- 如果要将当前选定的端口或参数在相应的端口或参数队列中上移一个位置，点击 Up 按钮（Delete 按钮下面的按钮）
- 如果要将当前选定的端口或参数在相应的端口或参数队列中下移一个位置，点击 Down 按钮（Up 按钮下面的按钮）

Input ports 子选卡

Input ports 子选卡允许你检查和修改 S-function **输入端口的属性。**

该选卡包含了一个可编辑的输入端口属性列表，输入端口在表中的顺序与最终出现在 S-function 块上输入端口顺序一致。表中的每一行对应着一个端口，行中的每个条目显示了该端口的一种属性。下面对各属性分别进行说明：



Port name　端口名称。对该项进行编辑可改变端口名称。

Data type　列出了该端口可接受的数据类型。点击旁边的按钮可显示支持的数据类型列表，从列表中选择所需要的数据类型。

Dimensions　列出了该端口可接受的信号维数。点击旁边的按钮可显示支持的信号维数列表，从列表中选择所需要的信号维数（Simulink 信号最多只能是 2 维的）。

Row　指定输入信号第一维（或者只有一维）的宽度。

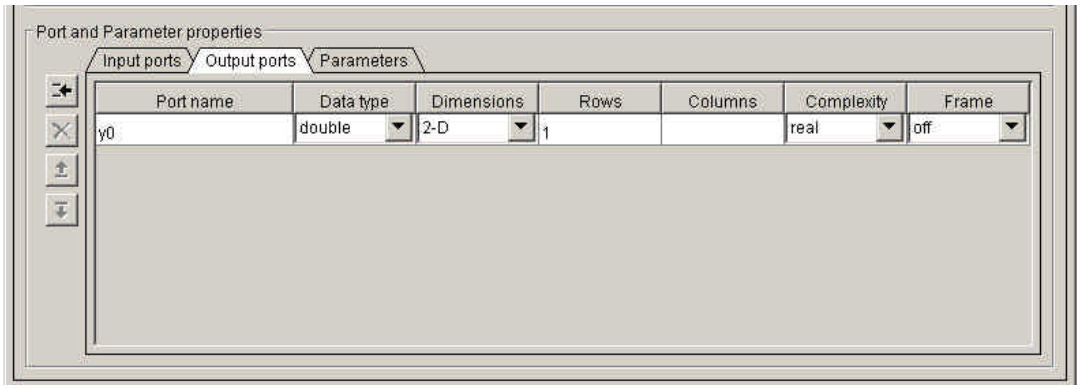
Column　指定输入信号第二维的宽度（只用于 2 维的输入信号）。

Complexity　指定输入端口接受的是实数值还是复数值的信号。

Frame　指定该端口接受的信号是否为 Communications Blockset 产生的基于帧的信号。

Output ports 子选卡

Output ports 子选卡允许你检查和修改 S-function 输出端口的属性。



该选卡包含了一个可编辑的输出端口属性列表，输出端口在表中的顺序与最终出现在 S-function 块上输出端口顺序一致。表中的每一行对应着一个端口，行中的每个条目显示了该端口的一种属性。下面对各属性分别进行说明：

Port name　端口名称。对该项进行编辑可改变端口名称。

Data type　列出了该端口可输出的数据类型。点击旁边的按钮可显示支持的数据类型列表，从列表中选择所需要的数据类型。

Dimensions　列出了该端口可输出的信号维数。点击旁边的按钮可显示支持的信号维数列表，从列表中选择所需要的信号维数（Simulink 信号最多只能是 2 维的）。

Row　指定输出信号第一维（或者只有一维）的宽度。

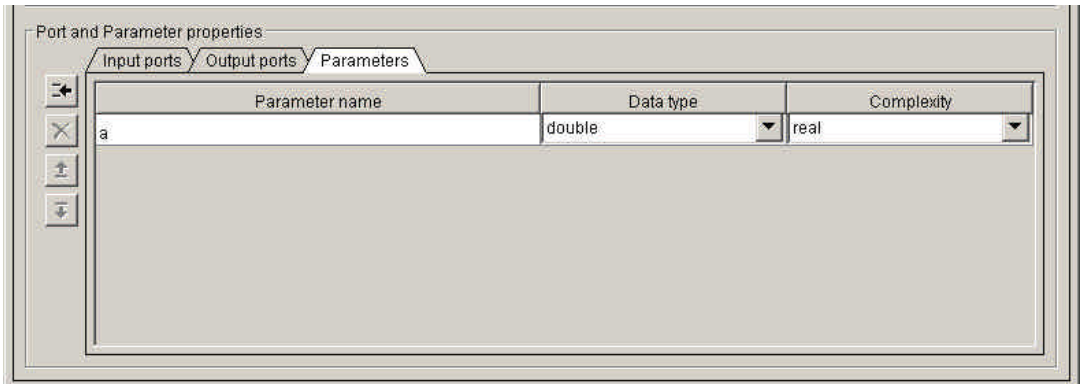
Column 指定输出信号第二维的宽度（只用于 2 维的信号输出）。

Complexity 指定输出端口输出的是实数值还是复数值的信号。

Frame 指定该端口输出的信号是否为 Communications Blockset 所接受的基于帧的信号。

Parameters 子选卡

Parameters 子选卡允许你检查和修改 S-function 参数的属性。



该选卡包含了一个可编辑的参数属性列表，用户在输入该 S-function 块的参数时，其顺序应与参数在表中排列的顺序一致。表中的每一行对应着一个参数，一行中的每个条目显示了该参数的一种属性。下面对各属性分别进行说明：

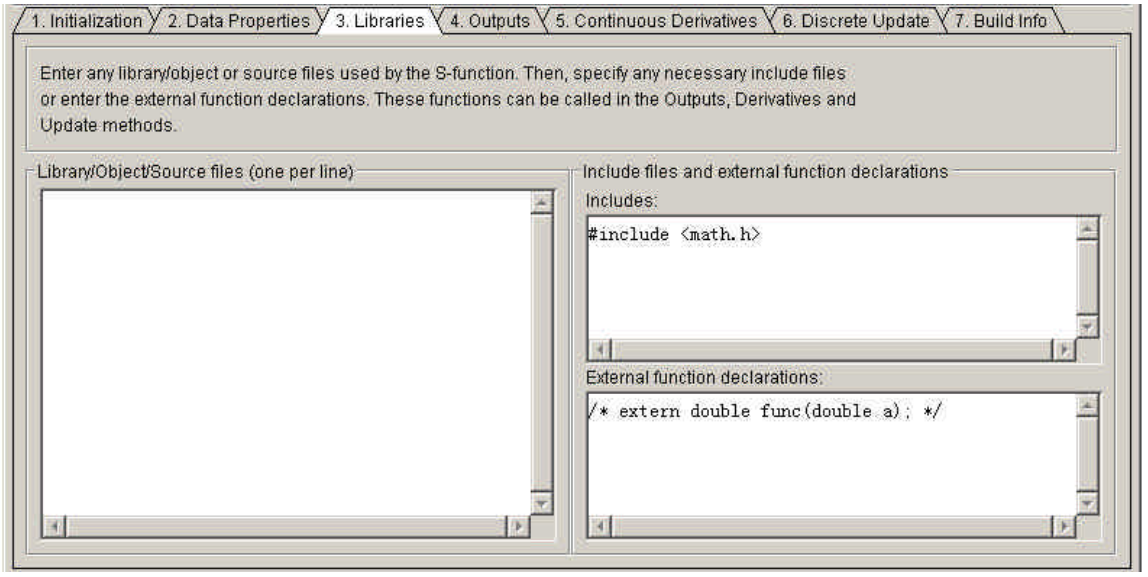
Parameter name 参数的名称。对该项进行编辑可改变参数名称。

Data type 列出了该参数可选的数据类型。点击旁边的按钮可显示支持的数据类型列表，从列表中选择所需要的数据类型。

Complexity 指定该参数是实数值还是复数值。

Libraries 选卡

Libraries 选卡允许你指定在其它选卡中输入的客户化代码所引用外部文件的位置。



Libraries 选卡包括以下区域：

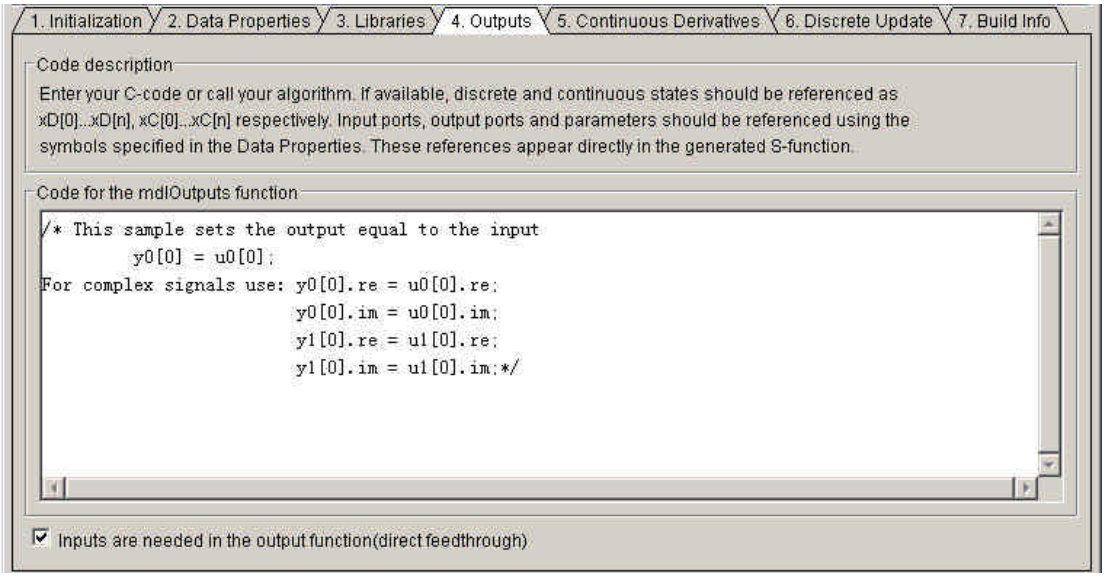
Library/Object/Source files　用户在 S-Function Builder 对话窗其它地方输入的客户化代码中所引用的外部函数库、目标代码和源文件。每个文件单独列写一行。如果文件在当前目录下，你只需要指定文件名即可；如果文件在其它目录下，你必须给出该文件的全部路径。

Includes　用户在 S-Function Builder 对话窗其它地方输入的客户化代码中所引用的包含函数申明、变量和宏定义的头文件。按照 #include 格式按单行列写每个文件。使用尖括号将标准的 C 头文件名括起（比如：#include <math.h>）；对于自定义的头文件采用引号标注头文件文件名（比如：#include "myutils.h"）。如果 S-function 中自定义的包含文件不在当前目录下，你必须将 S-Function Builder 的包含文件路径设置为存放自定义包含文件的目录。

External function declarations　如果 Includes 区列出的包含文件中对一些外部函数没有进行申明，必须在该区域对这些外部函数进行申明。每条申明单独占一行。S-Function Builder 将这些申明包含在它所生成的 S-function 源文件中。因而，这可以允许 S-function 代码调用外部函数来计算 S-function 的状态或输出。

Outputs 选卡

使用 Outputs 选卡来输入代码，可以在每步仿真时计算 S-function 的输出。



Outputs 选卡包含以下区域：

Code for the mdlOutputs function　在每步仿真(在离散 S-function 中为采样时间点)时计算 S-function 输出的代码。当生成 S-function 源代码时，将在此区域内输入的代码插入到一个 wrapper 函数中，形式如下：

```
void sfun_Outputs_wrapper(const real_T  *u,
                          real_T        *y,
                          const real_T  *xD,      /* optional */
                          const real_T  *xC,      /* optional */
                          const real_T  *param0,  /* optional */
                          int_T         p_width0, /* optional */
                          real_T        *param1,  /* optional */
```

```
                int_t           p_width1,     /* optional */
                int_T           y_width,      /* optional */
                int_T           u_width )     /* optional */
{
/* 你的代码被插入到这里 */
}
```

其中，sfun 是 S-function 的函数名。S-Function Builder 在回调函数 mdlOutputs 中插入一条调用该 wrapper 函数的代码。Simulink 在每步仿真（在离散 S-function 中为采样时间点）时调用 mdlOutputs 函数来计算 S-function 的输出。S-function 的 mdlOutputs 函数再调用包含有你的输出代码的 wrapper 函数，然后你的输出代码才实际地计算输出，并将结果返回给 S-function 的输出。

函数 mdlOutputs 传递某些或全部的参数给输出 wrapper 函数，这些参数是：

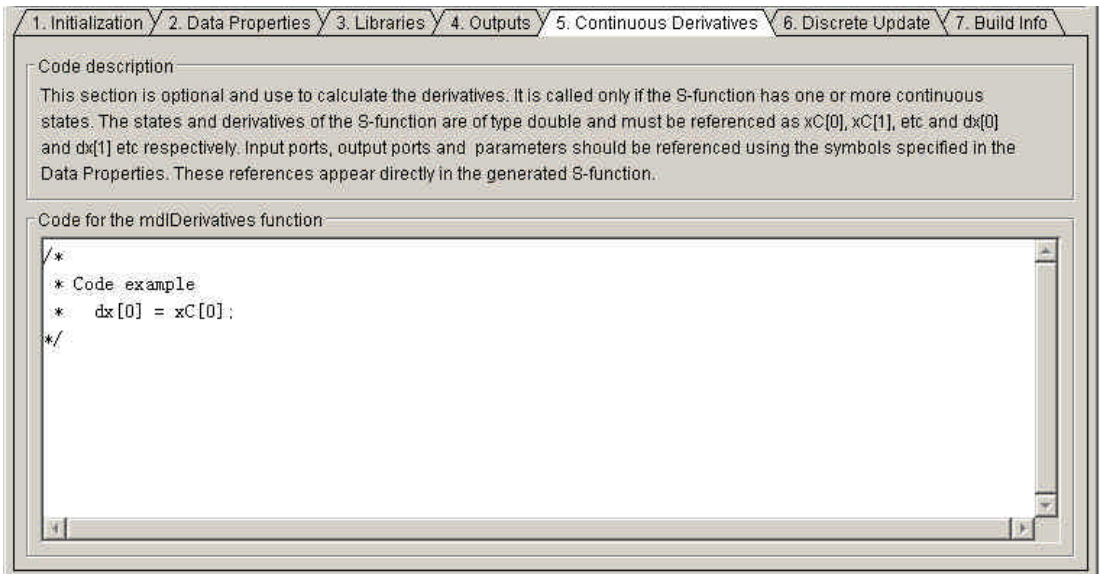
参数	说　明
u	包含 S-function 的输入数组的指针。数组的宽度与你在 Initialization 选卡中指定的输入宽度相同。如果你指定的输入宽度为-1，数组的宽度由 wrappe 函数的参数 u_width 来指定（见后面）。
y	包含 S-function 的输出数组的指针。数组的宽度与你在 Initialization 选卡中指定的输出宽度相同。如果你指定的输出宽度为-1，数组的宽度由 wrappe 函数的参数 y_width 来指定（见后面）。使用该数组向 Simulink 传递你的代码计算的输出。
xD	包含 S-function 离散状态的数组指针。如果你在 Initialization 选卡中指定了离散状态，该参数才出现。在第一步仿真时，离散状态使用你指定的初始值。在随后的采样步，状态值由上一采样步 S-function 计算获得。
xC	包含 S-function 连续状态的数组指针。如果你在 Initialization 选卡中指定了连续状态，该参数才出现。在第一步仿真时，连续状态使用你指定的初始值。在随后的采样步，状态值由上一采样步对状态导数进行积分来获取。
param0, p_width0, param1, p_width1, ... paramN, p_widthN	param0，param1，... paramN 是 S-function 参数数组的指针，其中 N 是你在 Initialization 选卡中指定的参数数目。p_width0，p_width1，... p_widthN 是参数的宽度数组。如果某个参数为矩阵，宽度则等于行列维数的乘积，例如 3 行 2 列矩阵参数的宽度是 6。如果你在 Initialization 选卡中指定了参数，这些参数才出现。
y_width	S-function 输出数组的宽度。如果你将 S-function 输出的宽度指定为-1，该参数才会出现在生成的代码中。如果输出为矩阵，y_width 则等于行列数的乘积。
u_width	S-function 输入数组的宽度。如果你将 S-function 输入的宽度指定为-1，该参数才会出现在生成的代码中。如果输入为矩阵，u_width 则等于行列数的乘积。

这些参数使得你计算一个块的输出类似调用一个带输入及可选状态和参数的函数。你在该选卡上输入的代码可调用外部函数，这些外部函数在 Libraries 选卡的头文件或外部申明中进行申明。这样，你可以使用现成的代码来计算 S-function 的输出。

Inputs are needed in the output function　　如果使用 S-function 的输入值来计算输出，则勾选该项。Simulink 通过该信息选项来检测是否存在由于 S-function 的输出直接或间接与输入连接而形成代数环。

Continuous Derivatives 选卡

如果 S-function 具有连续状态，则使用 Continuous Derivatives 选卡来输入计算状态导数所需的代码。



在该选卡的 Code for the mdlDerivatives function 区输入计算 S-function 连续状态导数的代码。在代码生成时，S-Function Builder 取出该选卡中的代码，将其按以下形式插入到 wrapper 函数中：

```
void    sfun_Derivatives_wrapper ( const real_T    *u,
                                   const real_T    *y,
                                   real_T          *dx,
                                   real_T          *xC,
                                   const real_T    *param0,     /* optional */
                                   int_T           p_width0,   /* optional */
                                   real_T          *param1,    /* optional */
                                   int_T           p_width1,   /* optional */
                                   int_T           y_width,    /* optional */
                                   int_T           u_width  )  /* optional */
{
    /* 此处插入你的代码 */
}
```

其中，sfun 是 S-function 的函数名。S-Function Builder 在回调函数 mdlDerivatives 中插入一条调用该 wrapper 函数的代码。Simulink 在每步仿真的最后调用 mdlDerivatives 函数来获取 S-function 连续状态的导数。Simulink 求解器通过对导数进行数字积分来确定下一采样步长内的连续状态。在下一采样步长中，Simulink 将更新过的状态传回给 S-function 的 mdlOutputs 函数。

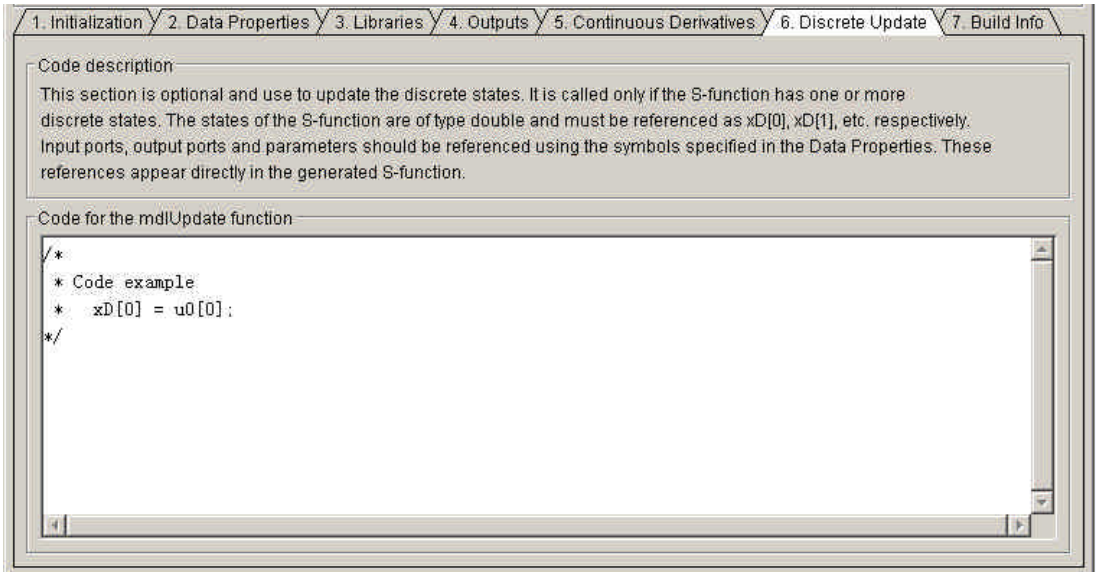
所生成的 S-function 回调函数 mdlDerivatives 传递以下参数到导数 wrapper 函数（壳函数）：

- u
- y
- dx
- xC
- param0, p_width0, param1, p_width1, ... paramN, p_widthN
- y_width
- x-width

其中，dx 是一个数组指针，其宽度与你在 Initialization 选卡中指定的连续导数的数目相同。你的代码应使用该数组来返回所计算的导数值。其它参数的含义参考第 44 页的说明。这些参数使得你计算导数就象调用一个带 S-function 的输入、输出及可选参数的函数。你在该选卡上输入的代码中可调用外部函数，这些外部函数在 Libraries 选卡的头文件或外部申明中进行申明。

Discrete Update 选卡

如果 S-function 具有离散状态，则须使用 Discrete Update 选卡来输入计算离散状态值所需的代码，该代码在当前采样步所计算的值是下一采样步使用的离散状态值。



在该选卡的 Code for the mdlUpdate function 区输入计算 S-function 离散状态导数值的代码。在代码生成时，S-Function Builder 取出该选卡中的代码，将其按以下形式插入到 wrapper 函数中：

```
void  sfun_Update_wrapper ( const real_T   *u,
                            const real_T   *y,
                            real_T         *xD,
                            const real_T   *param0,    /* optional */
                            int_T          p_width0,   /* optional */
                            real_T         *param1,    /* optional */
                            int_T          p_width1,   /* optional */
                            int_T          y_width,    /* optional */
                            int_T          u_width )   /* optional */
```

```
{
    /* 此处插入你的代码 */
}
```

其中，sfun 是 S-function 的函数名。S-Function Builder 在回调函数 mdlUpdate 中插入一条调用该 wrapper 函数的代码。Simulink 在每步仿真的最后调用 mdlUpdate 函数来获取 S-function 的离散状态值。在下一采样步长中，Simulink 将更新过的状态传回给 S-function 的 mdlOutputs 函数。

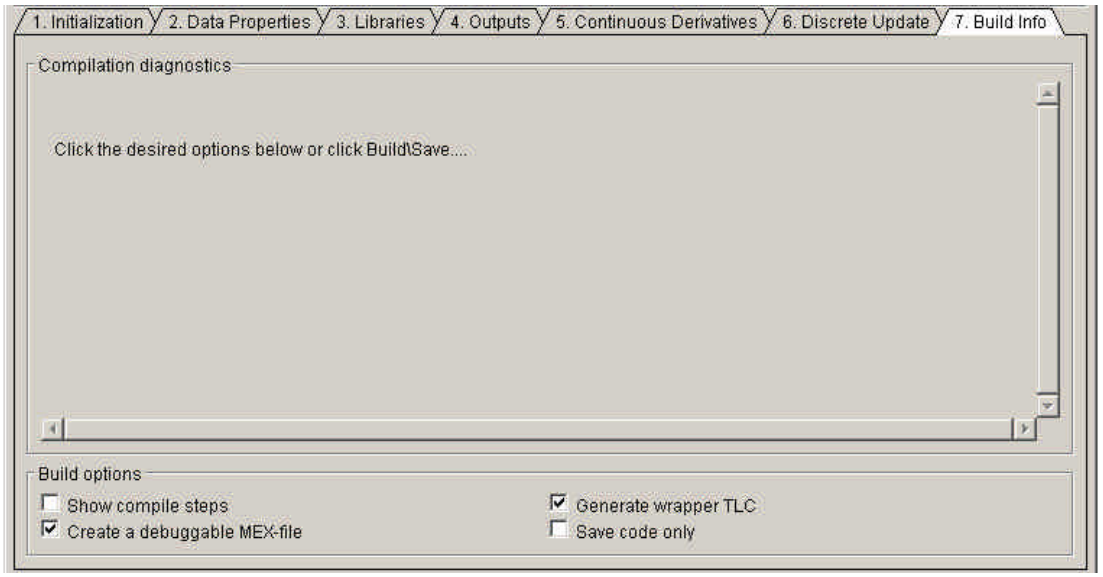
所生成的 S-function 回调函数 mdlUpdate 传递以下参数到状态更新 wrapper 函数（壳函数）：

- u
- y
- xD
- param0, p_width0, param1, p_width1, ... paramN, p_widthN
- y_width
- x-width

这些参数的含义参考第 44 页的介绍。你的代码应使用 xD（离散状态）变量来返回所计算的状态值。这些参数让计算离散状态值就象调用一个带 S-function 的输入、输出及可选参数的函数。在该选卡上输入的代码中可调用外部函数，这些外部函数在 Libraries 选卡的头文件或外部申明中进行申明。

Build Info 选卡

使用 Build Info 选卡可为创建 S-function 的 MEX 文件指定有关选项。



该选卡包括以下选项区：

Compilation diagnostics　在生成 S-function 时，该区域显示由 S-Function Builder 发布的诊断信息。

Show compile steps　勾选此项则在 Compilation diagnostics 区显示生成过程的每一步

Create a debuggable MEX-file　勾选此项则生成的 MEX 文件中包含调试信息

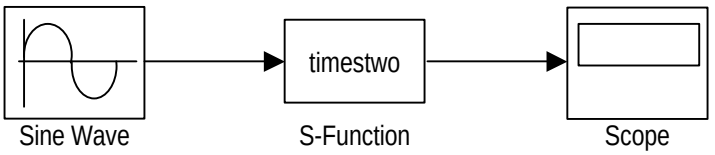
Generate wrapper TLC　勾选此项生成一个 TLC 文件。如果你不希望该 S-function 在加速模式下运行或用于 RTW 生成代码的模型中，可不必生成 TLC 文件。

Save code only　勾选此项则不需要从生成的源代码中 build 为一个 MEX 文件。

一个基本的 C MEX S-Function 范例

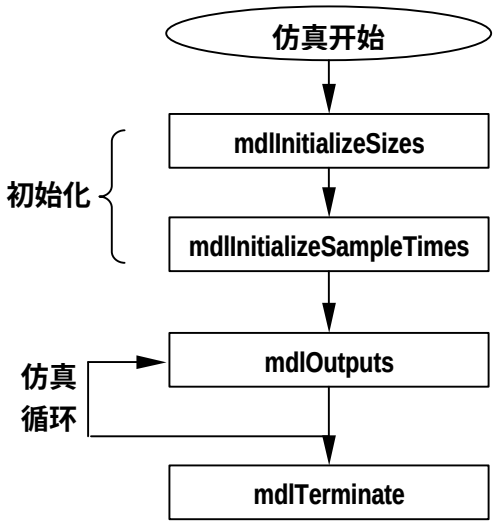
本节介绍一个 C MEX S-function 的例子，你可以将它作为创建简单 C S-function 的样板。该例子是来自于 Simulink 的 S-function 范例 timestwo（参考目录 *matlabroot*/simulink/src/timestwo.c）。该 S-function 的输出为其输入的 2 倍。

下面的模型使用了 timestwo S-function 来将一个正弦波的振幅加倍，并绘制在一个 scope 上。



在块对话框中指定 timestwo 作为 S-function 的函数名；参数区为空。

timestwo S-function 内包含的 S-function 回调函数如下图所示：



timestwo.c 的内容如下：

```
#define   S_FUNCTION_NAME   timestwo
#define   S_FUNCTION_LEVEL   2

#include   “simstruc.h”

static void   mdlInitializeSizes ( SimStruct   *S )
{
     ssSetNumSFcnParams( S，  0 )；
     if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount (S))   {
          return；            /* Parameter mismatch will be reported by Simulink */
     }
     if   ( !ssSetNumInputPorts( S，1 ))   return；
     ssSetInputPortWidth (S，0，DYNAMICALLY_SIZED )；
     ssSetInputPortDirectFeedThrough (S，0，1)；
     if ( !ssSetNumOutputPorts (S，1))     return；
     ssSetOutputPortWidth ( S，0，DYNAMICALLY_SIZED )；
```

```
    ssSetNumSampleTimes ( S，1)；
    /* Take care when specifying exception free code - see sfuntmpl.doc */
    ssSetOptions ( S，SS_OPTION_EXCEPTION_FREE_CODE )；
}
    static void    mdlInitializeSampleTimes ( SimStruct    *S )
{
    ssSetSampleTime (S，0，INHERITED_SAMPLE_TIME )；
    ssSetOffsetTime ( S，0，0.0 )；
}
    static void mdlOutputs(SimStruct *S, int_T tid)
{
    int_T    I；
    InputRealPtrsType    uPtrs = ssGetInputPortRealSignalPtrs ( S，0 )；
    real_T               *y = ssGetOutputPortRealSignal (S，0 )；
    int_T                width = ssGetOutputPortWidth ( S，0 )；
    for ( i=0；i<width；i++ )   {
        *y++    = 2.0 *( *uPtrs[ I ] )；
    }
}

static void    mdlTerminate (SimStruct    *S) {   }

#ifdef    MATLAB_MEX_FILE          /* Is this file being compiled as a MEX-file? */
#include    “simulink.c”           /* MEX-file interface mechanism */
#else
#include    “cg_sfun.h”            /* Code generation registration function */
#endif
```

该范例包括三个部分：

- **定义与包含**
- **回调函数的实现**
- Simulink**（或** Real-Time Workshop**）接口**

以下各小节对上述三个部分分别进行介绍。

定义与包含

该范例以以下的定义开头：

```
#define    S_FUNCTION_NAME    timestwo
#define    S_FUNCTION_LEVEL    2
```

第一条指定了 S-function **的名字（**timestwo**），第二条指定了该** S-function **是按照** *level 2* **的格式进行编写的。**

在定义了这两条之后，该范例包含了 simstruc.h **文件，这是一个头文件，它给出了** SimStruct **数据结构的访问通道，以及** MATLAB **应用程序接口（**API**）函数。**

```
#define   S_FUNCTION_NAME   timestwo
#define   S_FUNCTION_LEVEL   2
#include   “simstruc.h”
```

simstruc.h 文件定义了一个名为 SimStruct 的数据结构，Simulink 使用它来维护 S-function 的有关信息。simstruc.h 文件也定义了一些宏，可让你的 MEX 文件在 SimStruct 中设置某些值，或从 SimStruct 中获取某些值。

回调函数的实现

S-function timestwo 的下一部分包含了 Simulink 所需要的回调函数的实现。

mdlInitializeSizes

Simulink 调用 mdlInitializeSizes 来获取输入端口和输出端口的数量、端口宽度、以及 S-function 所需的任何其它对象（诸如状态数量）等有关信息。timestwo 在 mdlInitializeSizes 中指定了以下数量信息：

- 无参数——这意味着 S-function 对话框的 S-function parameters 区必须为空。如果在此处输入了任何参数，Simulink 将报告参数不匹配信息。
- 一个输入端口与一个输出端口——输入与输出端口的宽度是动态的。这告诉 Simulink 将输入 S-function 的信号中每个元素都乘以 2，并将结果放在输出信号中。注意对于动态宽度的 S-function 在这种情况（一个输入和一个输出）下的默认处理方法是输入与输出的宽度相同。
- 一个采样时间——timestwo 范例在程序 mdlInitializeSampleTimes 中指定了采样时间的实际值。
- 代码无异常检测——指定 exception-free 代码可以提高 S-function 的执行速度。在指定该选项时，必须特别小心。一般情况下，如果 S-function 与 MATLAB 之间没有相互作用，指定该选项是安全的。有关详细说明参考“ Simulink 如何与 C S-Function 相互作用 ”中的介绍。

mdlInitializeSampleTimes

Simulink 调用 mdlInitializeSampleTimes 来设置 S-function 的采样时间。只要驱动块执行，该 timestwo 块就必须执行。因此，它是单一的继承采样时间——SAMPLE_TIME_INHERITED。

mdlOutputs

在每个采样时间步长内，Simulink 调用 mdlOutputs 来计算块的输出。该 timestwo 块的 mdlOutputs 函数的实现是取出输入信号，将其乘以 2，并将结果写入输出中。

timestwo 的 mdlOutputs 函数使用了 SimStruct 的一个宏：

```
InputRealPtrsType    uPtrs = ssGetInputPortRealSignalPtrs ( S，0 )；
```

来获取输入信号。该宏返回一个向量的指针，必须使用 *uPtrs[i] 来访问它。

timestwo 的 mdlOutputs 函数使用了 SimStruct 的一个宏：

```
real_T      *y = ssGetOutputPortRealSignal (S，0 )；
```

来访问输出信号。该宏返回一个包含了输出向量的指针。

S-function 使用 int_T width = ssGetOutputPortWidth (S，0) 来获取通过块传递的信号宽度。最后，S-function 采用循环通过输入来计算输出。

mdlTerminate

执行仿真结束时的任务。这是一个托管 S-function **程序。但是，**timestwo S-function **不需要执行任何终止动作，所以该程序是空的。**

Simulink/Real-Time Workshop 接口

在 S-function **的结尾部分，使用代码判断该范例是植入** Simulink **中，或是** Real-Time Workshop **中。**

```
#ifdef      MATLAB_MEX_FILE
#include    "simulink.c"
#else
#include    "cg_sfun.h"
#endif
```

Building Timestwo 范例

要将此 S-function **结合到** Simulink **中，在** MATLAB **命令行输入下面的命令：**

```
mex timestwo.c
```

mex **命令将** timestwo.c **进行编译和连接以生成一个可动态下载的可执行文件，供** Simulink **使用。**

最后的可执行文件是一个 MEX S-function **文件，其中，**MEX **代表了** MATLAB EXecutable **（可执行文件）。**MEX **文件的扩展名根据不同的平台而有所不同，例如，在** Microsoft Windows **中，**MEX **文件的扩展名为** .dll**。**

C S-Function 模板

Simulink 提供了实现一个 C MEX S-function 的架构文件，这里称之为模板。它是专门为用户编写的用于指导用户编写自己的 S-function。该模板包含了实现回调函数的框架结构，并加以注释说明用法。该模板文件名为 sfuntmpl_basic.c，存放在 MATLAB 根目录下的 simulink/src 目录中，它适合于编写普通用途的 S-function 程序。另外一个模板包含了所有有效的程序（并有更详细的注释），存放在相同目录下，文件名为 sfuntmpl_doc.c。

注意：在开发 MEX S-function 时，我们推荐使用 C MEX 文件的模板。

S-Function 源文件必需的内容

本小节介绍每个 S-Function 进行正确编译的必需条件，这些条件在模板中都已给出。

S-Function 头部必需的申明

为了使 S-function 能够正确地进行运作，S-function 中需要访问 SimStruct 结构的每个源程序模块必须包含以下顺序的定义与包含：

```
#define    S_FUNCTION_NAME      your_sfunction_name_here
#define    S_FUNCTION_LEVEL     2
#include   "simstruc.h"
```

其中，*your_sfunction_name_here* 是 S-function 的函数名（如：在 Simulink S-Function 块的对话框所输入的名称）。这些申明使你能够访问 SimStruct 数据结构，该数据结构包含了仿真所用数据的指针。包含文件中还定义了宏用来在 SimStruct 中存储数据，或从 SimStruct 中获得数据。另外，该代码指定了你使用的是 S-function level 2 的格式。

注意：从 Simulink 1.3 到 2.1 版本的 S-function 都使用 level 1 格式，它们兼容 Simulink 3.0。但是，我们推荐使用 level 2 格式编写新的 S-function。

在编译为一个 MEX 文件时，应包含头文件 *matlabroot*/simulink/include/simstruc.h。

下表是在 S-function 编译为一个 MEX 文件时，被 simstruc.h 包含的头文件：

头文件	说　明
matlabroot/extern/include/tmwtypes.h	通用数据类型。如：real_T
matlabroot/extern/include/mex.h	MATLAB MEX 文件 API 程序
matlabroot/extern/include/matrix.h	MATLAB MEX 文件 API 程序

下表是在 S-function 编译为与 RTW 一起使用的文件时，被 simstruc.h 包含的头文件：

头文件	说　明
matlabroot/extern/include/tmwtypes.h	通用数据类型。如：real_T
matlabroot/extern/include/ rt_matrx.h	MATLAB API 程序的宏

S-Function 结尾必需的申明

仅在 C MEX S-function **主模块的最后包含这些结尾代码。**

```
#ifdef    MATLAB_MEX_FILE    /* Is this being compiled as MEX-file? */
#include  "simulink.c"       /* MEX-file interface mechanism */
#else
#include  "cg_sfun.h"        /* Code generation registration func */
#endif
```

这些申明用来为特定的应用选择合适的代码：

- **如果被编译为一个** MEX **文件，则包含** simulink.c **文件**
- **如果该文件被用来与** Real-Time Workshop **连接以产生一个单机或实时可执行文件，则包含** cg_sfun.h **文件**

注意：该结尾代码不能放置在任何一个 S-function **的函数体内**

SimStruct

matlabroot/simulink/include/simstruc.h **是一个** C **语言的头文件，它定义了** Simulink **的数据结构和** SimStruct **访问宏，它封装了所有与模型或** S-function **相关的数据，包括块参数和输出。**

对于 Simulink **分配了一个数据结构** SimStruct**。模型中的每个** S-function **都有一个独立的** SimStruct **与之联系。这些** SimStruct **的结构非常类似一个目录树，与模型关联的** SimStruct **是“根”** SimStruct**，与** S-function **关联的** SimStruct **是“子”** SimStruct**。**

注意：根据协定，端口的编号从 0 **开始，最后一个端口的编号为总端口数减** 1**。**

Simulink **提供了一组宏，**S-function **可以使用它们来访问** SimStruct **的域。**

编译 C S-Function

根据下面定义的条件，S-function **可按照三种模式之一进行编译：**

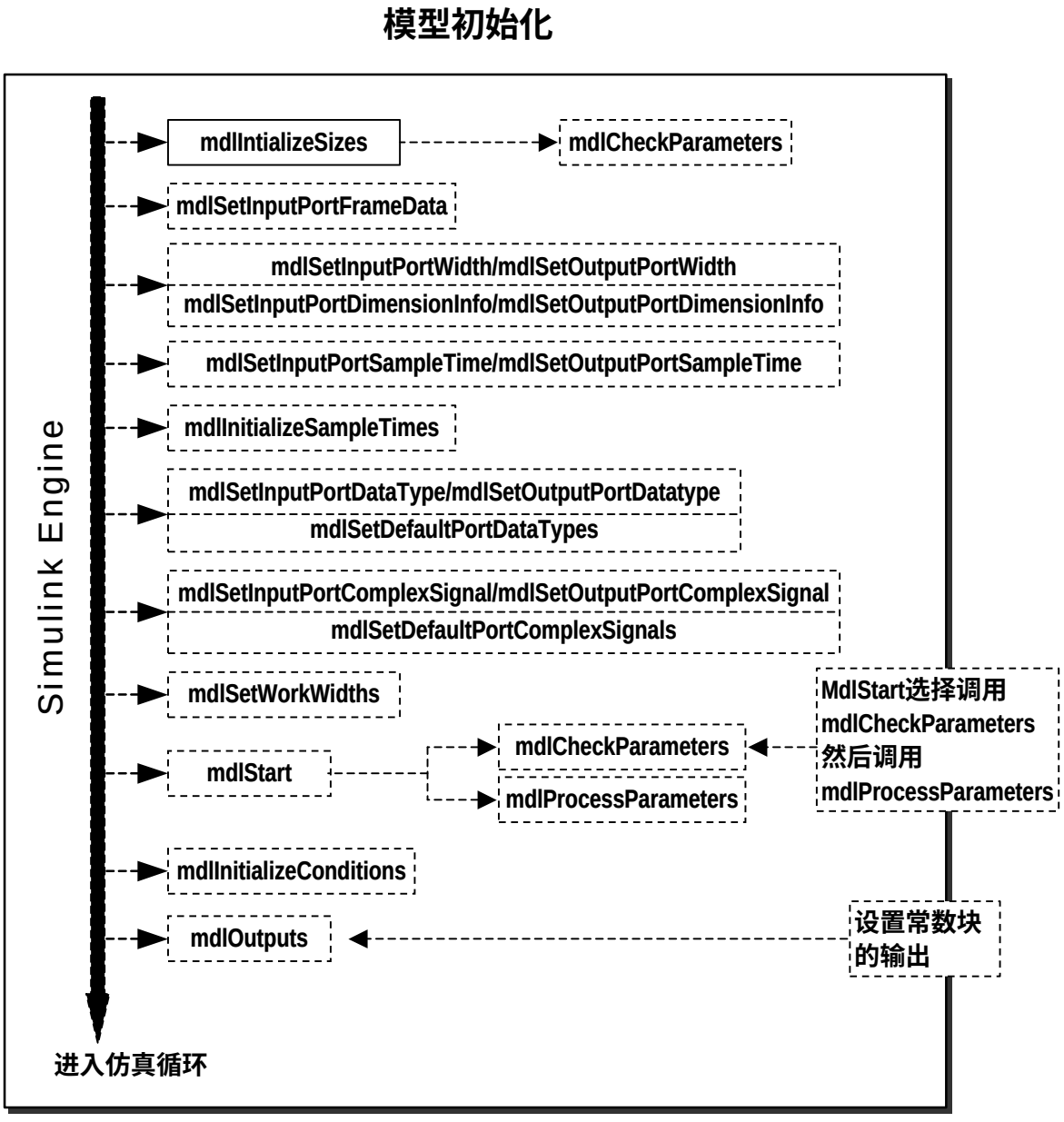
- MATLAB_MEX_FILE**——指示** S-function **被编译成用于** Simulink **的** MEX **文件**
- RT**——指示** S-function **与** Real-Time Workshop **一起编译生成使用定步长求解器的实时应用**
- NRT**——指示** S-function **与** Real-Time Workshop **一起编译生成使用变步长求解器的非实时应用**

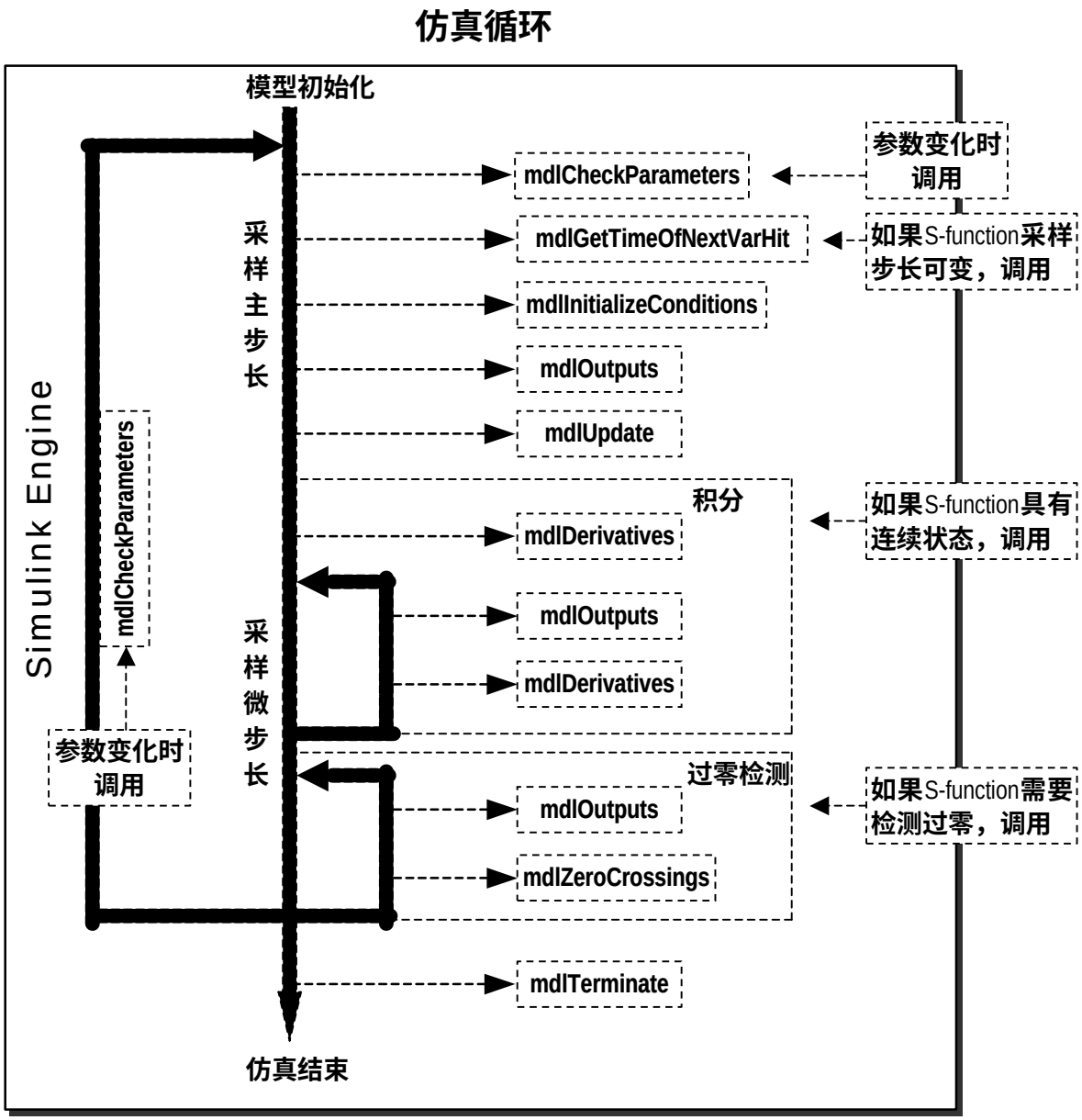
Simulink 如何与 C S-Function 相互作用

了解 Simulink 如何与 C S-Function 相互作用对于编写 C MEX 文件的 S-function 是十分有帮助的。本节从两个层面来介绍这种相互作用：进程层面，比如在仿真的哪一时刻，Simulink 调用 S-function；数据层面，比如在仿真过程中，Simulink 与 S-function 何如交换信息。

进程层面

下图所示为 Simulink 调用 S-function 回调函数的顺序。实线框部分表示在模型初始化和/或每个仿真步长内需使用的回调函数；虚线框部分表示在初始化阶段和/或在仿真循环的一些或所有采样步长内使用的回调函数。





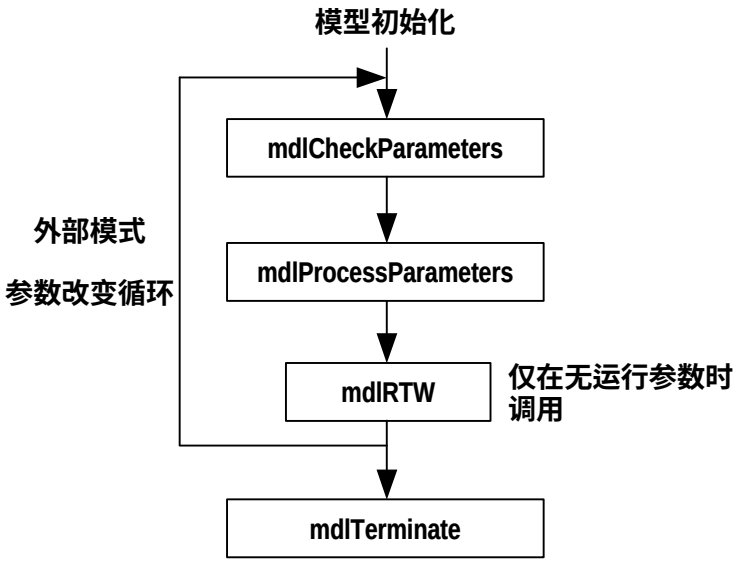
对于 Real Time Workshop（RTW）的调用结构

生成代码时，RTW 不需要按上面所述的顺序调用全部函数。在上一小节的模型初始化中，Simulink 只调用 mdlRTW、mdlTerminate，然后退出，其中 mdlRTW 是一个专用于 RTW 的 S-function 程序。

关于 RTW 的更多信息以及它是如何与 S-function 相互作用的，参考 Real Time Workshop 文档和 Language Compiler Reference Guide 文档。

对于外部模式的交替调用结构

当在外部模式下运行 Simulink 时，对于 S-function 程序的调用顺序会发生变化。下图所示为外部模式下的正确调用顺序。



当进入外部模式时，Simulink 调用 mdlRTW 一次，当某个参数发生变化时再调用一次，或者在模型的 Edit 菜单下选择了 Updata Diagram 命令，也会调用一次。

注意：在外部模式下运行 Simulink 需要 Real-Time Workshop 的支持。关于外部模式的更多信息，参考 Real Time Workshop 文档。

数据层面

S-function 块具有输入和输出信号、参数、以及内部状态，再外加其它一般的工作域。在一般情况下，对于块输入和输出信号的读写是通过一个块 I/O 向量来实现的。输入信号也可来源于：

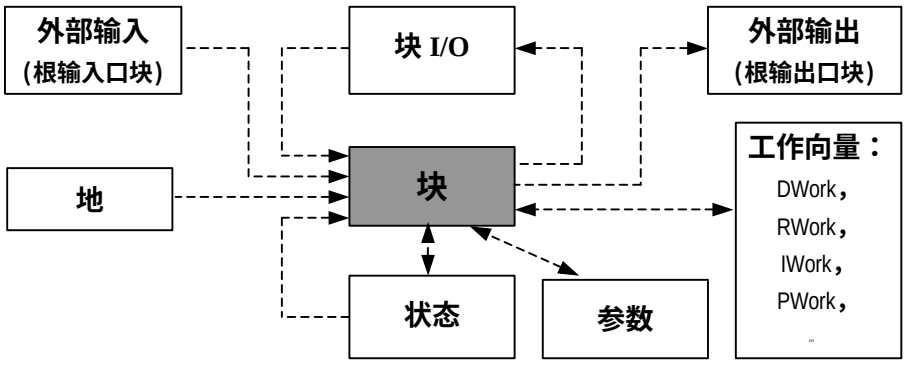
- 通过根输入端口块而来的外部输入
- 如果输入信号未连接或接地，作为接地输入

块输出也可以通过根输出端口块传递到外部输出。除了输入和输出信号之外，S-function 还具有：

- 连续状态
- 离散状态
- 诸如实数的、整数的或指针的工作向量等其它工作域。

通过使用 S-function 块的对话框进行参数传递，可以对 S-function 块的参数进行赋值。

下图所示为各种类型的数据之间一般的映射关系：



S-function 的 mdlInitializeSizes 程序用来设置各种信号和向量的宽度。在仿真循环中调用的 S-function 函数能够确定信号的宽度和值。

S-function 函数可以通过两种渠道来访问输入信号：

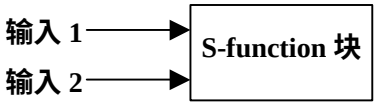
- 通过指针
- 使用相邻的输入

使用指针访问信号

在仿真循环中，对于输入信号的访问通过下面的函数来完成：

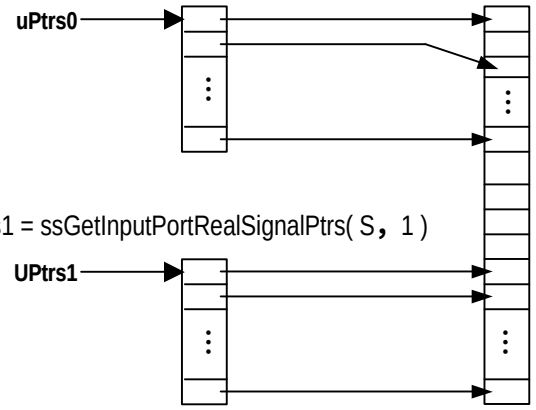
InputRealPtrsType　uPtrs = ssGetInputPortRealSignalPtrs(S，portIndex)

这是一个指针数组，其中 *portIndex* 从 0 开始，对于每个输入端口都有一个索引号。要访问该信号的一个元素，应使用 *uPtrs[element]。如下图所示：



要访问输入 1：

InputRealPtrsType　uPtrs0 = ssGetInputPortRealSignalPtrs(S，0)



要访问输入 2：

InputRealPtrsType　uPtrs1 = ssGetInputPortRealSignalPtrs(S，1)

请注意，输入数组指针能够指向内存中非相邻的单元。

你可使用下面的代码获取输出信号：

real_T　*y = ssGetOutputPortSignal(S，outputPortIndex)；

访问相邻的输入信号

通过使用 ssSetInputPortRequiredContiguous 宏，S-function 的 mdlInitializeSizes 函数可以指定其输入信号的元素必须占据相邻的内存区间。如果输入是相邻的，其它函数可以使用 ssGetInputPortSignal 来访问输入。

访问独立端口的输入信号

本部分介绍如何访问特定端口的所有输入，以及如何写输出端口。上面的图中显示，输入指针数组能够指向块 I/O 向量中的非相邻入口。特定端口的输出信号组成了一个相邻的向量。因此，访问输入元素并将它们写入输出元素（假设输入与输出端口的宽度相同）的正确方法是使用该代码：

```
int_T       element；
int_T       portWidth = ssGetInputPortWidth( S，inputPortIndex )；
InputRealPtrsType   uPtrs = ssGetInputPortRealSignalPtrs( S，inputPortIndex )；
real_T      *y = ssGetOutputPortSignal( S，outputPortIdx )；
for (element=0；element<portWidth；element++ )  {
      y[ element ] = *uPtrs[ element ]；
}
```

一个很常见的错误是试图通过指针赋值来访问输入信号。例如，如果使用这样的代码：

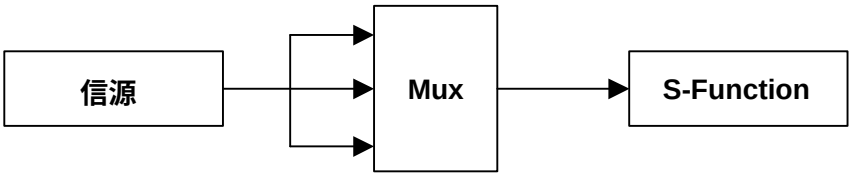
```
real_T      *u = *uPtrs；   /* 不正确的用法 */
```

仅在 uPtrs 初始化之后，使用下面的语句代替上面代码中的 for 循环也是错误的：

```
*y++ = *u++；  /* 不正确的用法 */
```

该代码可以进行编译，但是 MEX 文件可能会与 Simulink 冲突，这是因为它可能回访问无效的内存（取决于你是如何构建模型的）。当访问输入信号不正确时，一旦该信号输入非相邻的 S-function 块就会发生冲突。非相邻数据一般发生在通过虚拟连接块传递的数据，比如通过 Mux 块或 Selector 块传递过来的数据。

要检验你访问宽输入信号是否正确，应将信号复制后传递给你的 S-function 的每个输入端口，你可以通过创建一个输入端口与你的 S-function 输入信号宽度相同的 Mux 块来实现。然后将驱动信源连接到 Mux 块的每个输入端口，如下图所示：



编写回调函数

编写一个 S-Function 的主要内容就是创建在仿真过程中由 Simulink 调用的回调函数。关于实现特定功能的回调函数的编写指导参考第九章“ S-Function 回调函数 ”中的介绍；有关如何使用回调函数实现特定的块特性，如参数或采样时间，参考第七章“ 实现块特性 ” 中的介绍。

将 Level 1 C MEX S-Function 转换到 Level 2

Level 2 S-function 是与 Simulink 2.2 一起发布的，Level 1 S-function 是在 Simulink 2.1 及以前版本上运行的。Level 1 S-function 兼容 Simulink 2.2 及以后版本，你可以不作任何代码修改将它们用于新的模型。尽管如此，为了获得 S-function 新特性的优点，Level 1 S-function 必须更新到 Level 2 S-function。以下给出了一些指导：

- 通过学习 simulink/src/sfunctmpl_doc.c 模板文件入手。该 S-function 模板文件对 Level 2 S-function 进行了简明地概述。
- 在你的 S-function 文件头部增加定义语句：#define S_FUNCTION_LEVEL 2
- 更新 mdlInitializeSizes 的内容。在特殊情况下，对于 S-function 参数增加以下错误处理操作：

```
ssSetNumSFcnParams(S, NPARAMS); /*Number of expected parameters*/
if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
    /* Return if number of expected != number of actual parameters */
    return;
}
```

使用下面的语句构建输入：

```
if (!ssSetNumInputPorts(S, 1))    return;          /*Number of input ports */
ssSetInputPortWidth(S, 0, width);                  /* Width of input port one (index 0)*/
ssSetInputPortDirectFeedThrough(S, 0, 1);          /* Direct feedthrough or port one */
ssSetInputPortRequiredContiguous(S, 0);
```

使用下面的语句构建输出：

```
if (!ssSetNumOutputPorts(S, 1))    return;
ssSetOutputPortWidth(S, 0, width);                 /* Width of output port one (index 0) */
```

- 如果你的 S-function 的 mdlInitializeConditions 为非空，将它更新为下面的形式：

```
#define    MDL_INITIALIZE_CONDITIONS
static void mdlInitializeConditions(SimStruct *S)
{
}
```

否则，删去该函数。

- 使用 ssGetContStates 访问连续状态，ssGetX 宏已经作废
- 使用 ssGetRealDiscStates(S)访问离散状态，ssGetX 宏已经作废
- 对于连续状态和离散状态混合的 S-function，状态向量不再由先连续状态再离散状态组合而成，而是作为分开的向量，因而在内存中可能不是相邻的。

- mdlOutputs 函数原型已经改变。将原来的：

```
static void    mdlOutputs ( real_T *y, const real_T *x, const real_T *u, SimStruct *S, int_T tid )
```

改变为：static void mdlOutputs(SimStruct *S, int_T tid)

因为 y，x 和 u 不再是直接被传递进 level-2 S-function，而必须使用：

- ssGetInputPortSignal 来访问输入
- ssGetOutputPortSignal 来访问输出

- ssGetContStates 或 ssGetRealDiscStates 来访问状态

■ 函数原型已经改变。将原来的：

```
void  mdlUpdate( real_T *x,  real_T *u,  Simstruct *S,  int_T  tid )
```

改变为：void mdlUpdate(SimStruct *S, int_T tid)

■ 如果你的 S-function 的 mdlUpdate 为非空，将它更新为下面的形式：

```
#define MDL_UPDATE
static void mdlUpdate(SimStruct *S, int_T tid)
{
}
```

否则，删去该函数。

■ 如果你的 S-function 的 mdlDerivatives 为非空，将它更新为下面的形式：

```
#define MDL_DERIVATIVES
static void mdlDerivatives(SimStruct *S, int_T tid)
{
}
```

否则，删去该函数。

■ 替换所有作废的 SimStruct 宏。参见下面的作废宏的列表，该表列出了所有作废的宏。

■ 当将 level 1 S-function 转换为 level 2 S-function 时，你应该在 S-function 中附带完全的（比如：最高的）警告级。例如，如果你在 UNIX 系统上有 gcc，在使用 mex 时应使用如下选项：

```
mex  CC=gcc  CFLAGS= -Wall sfcn.c
```

如果你的系统里有 Lint，可使用下面的代码：

```
lint  -DMATLAB_MEX_FILE  -I<matlabroot>/simulink/include
      -Imatlabroot/extern/include sfcn.c
```

■ 在 PC 机上，如要使用最高警告级，你必须在集成开发环境（IDE）中为所使用的编译器创建一个项目（project）文件。在项目文件里，定义 MATLAB_MEX_FILE，并增加：

matlabroot/simulink/include

matlabroot/extern/include

到路径中（构建时应保证将队列设置为 8）。

作废宏

下面的宏已经作废。每个作废的宏应使用指定的宏来替代。

作废宏	替换为
ssGetU(S), ssGetUPtrs(S)	ssGetInputPortSignalPtrs(S,port)
ssGetY(S)	ssGetOutputPortRealSignal(S,port)
ssGetX(S)	ssGetContStates(S), ssGetRealDiscStates(S)
ssGetStatus(S)	一般不用，也可用 ssGetErrorStates(S)替换
ssSetStatus(S,msg)	ssSetErrorStatus(S,msg)
ssGetSizes(S)	按需要指定调用，如 ssGetNumContStates(S)

作废宏	替换为
ssGetMinStepSize(S)	**不再支持**
ssGetPresentTimeEvent(S,sti)	ssGetTaskTime(S,*sti*)
ssGetSampleTimeEvent(S,sti)	ssGetSampleTime(S,sti)
ssSetSampleTimeEvent(S,t)	ssSetSampleTime(S,sti,t)
ssGetOffsetTimeEvent(S,sti)	ssGetOffsetTime(S,sti)
ssSetOffsetTimeEvent(S,sti,t)	ssSetOffsetTime(S,sti,t)
ssIsSampleHitEvent(S,sti,tid)	ssIsSampleHit(S,sti,tid)
ssGetNumInputArgs(S)	ssGetNumSFcnParams(S)
ssSetNumInputArgs(S, numInputArgs)	ssSetNumSFcnParams(S,numInputArgs)
ssGetNumArgs(S)	ssGetSFcnParamsCount(S)
ssGetArg(S,argNum)	ssGetSFcnParam(S,argNum)
ssGetNumInputs	ssGetNumInputPorts(S)**和** ssGetInputPortWidth(S,port)
ssSetNumInputs	ssSetNumInputPorts(S,nInputPorts)**和** ssSetInputPortWidth(S,port,val)
ssGetNumOutputs	ssGetNumOutputPorts(S)**和** ssGetOutputPortWidth(S,port)
ssSetNumOutputs	ssSetNumOutputPorts(S,nOutputPorts)**和** ssSetOutputPortWidth(S,port,val)

创建 C++ S-Function

创建 C++ S-Function 的步骤与创建 C S-Function 基本一样。本章各节阐述了两者的区别：

源文件格式	介绍了 C++ S-Function 与 C S-Function 源文件结构之间的差别
Making C++ Objects Persistent	如何创建支持 S-Function 调用的 C++对象
Building C++ S-Functions	如何 build C++ S-Function

本章内容参考英文资料。

创建 Ada S-Function

以下各节介绍了何如使用 Ada 编程语言来创建 S-Function：

概述	从总体上介绍 Ada S-Function 的创建
Ada S-Function 的源文件格式	Ada S-Function 源代码的结构
编写 Ada 回调函数	如何使用 Ada 来实现 S-Function 回调函数
Building an Ada S-Function	Ada S-Function 的编译与连接
Ada S-Function 范例	timestwo S-function 范例的 Ada 版

本章内容参考英文资料。

创建 Fortran S-Function

以下各节介绍了何如使用 F ortran 编程语言来创建 S-Function：

概述	从总体上介绍 Ada S-Function 的创建
创建 Level 1 Fortran S-Function	介绍一个纯粹使用 Fortran 创建 S-Function 的方法
创建 Level 2 Fortran S-Function	介绍使用 C/Fortran 混合方式编写 S-Function，以实现更多功能的块
Porting Legacy Code	介绍如何将现成的 Fortran 代码包装为 S-function。

本章内容参考英文资料。

实现块特性

以下各节介绍如何使用 S-Function 回调函数实现不同的块特性：

对话框参数	介绍如何通过 S-function 块的对话框来传递处理参数
运行参数	介绍如何创建和使用运行参数
创建输入和输出端口	介绍如何在块中创建输入和输出端口
自定义数据类型	介绍如何对一个块的信号和参数创建自定义数据类型
采样时间	介绍如何指定速率或者块操作的速率
工作向量	介绍如何创建与使用工作向量
Function-Call 子系统	介绍如何创建一个可调用子系统的 S-function。
错误处理	介绍在 S-function 中如何进行错误处理
S-Function 范例	介绍几个 S-function 的实际例子

对话框参数

用户可以在仿真开始时和仿真过程中使用块对话框的 S-Function parameters 域将参数传递给一个 S-function。这样的参数被称为对话框参数，以区别运行参数——由 S-function 使用代码生成。Simulink 将对话框参数的值存储在 S-function 的 SimStruct 结构中。Simulink 提供了回调函数和 SimStruct 宏，可以让 S-function 访问和检查参数，以及将参数应用于计算块的输出中。

如果希望 S-function 能够使用对话框参数，那么在创建 S-function 时必须完成以下步骤：

1. 确定在块对话框中参数的输入顺序
2. 在函数 mdlInitializeSizes 中，使用 ssSetNumSFcnParams 宏告诉 Simulink 本 S-function 接受多少参数。指定 S 为第一个参数，并将通过交互定义的参数数量作为第二个参数。如果在 S-function 中需要编写 mdlCheckParameters 函数，那么 mdlInitializeSizes 程序应该调用 mdlCheckParameters 来检查参数初始值的有效性。
3. 在 S-function 中使用 ssGetSFcnParam 宏来访问这些输入参数。指定 S 为第一个参数，将在块对话框参数输入队列的相对位置（第一个位置为 0）作为第二参数。宏 ssGetSFcnParam 返回一个指向 mxArray 的指针，mxArray 中包含了参数值。你可以使用 ssGetDTypeIdFromMxArray 来获取参数的数据类型。

当仿真运行时，用户必须在块对话框的 S-Function parameters 域按照上面第一步的顺序输入参数值。用户可以输入任何有效的 MATLAB 表达式作为参数值，包括数字值、workspace 内的变量名、函数调用、或者算术表达式等。Simulink 将计算出表达式的值并将它们传递给 S-function。

例如，以下代码为一个设备驱动 S-function 的一部分，使用了四个输入参数：BASE_ADDRESS_PRM，GAIN_RANGE_PRM，PROG_GAIN_PRM 和 NUM_OF_CHANNELS_PRM。该代码使用#define 语句将特定的输入参数与参数名联系起来。

```
/* Input Parameters */
#define    BASE_ADDRESS_PRM(S)          ssGetSFcnParam(S, 0)
#define    GAIN_RANGE_PRM(S)            ssGetSFcnParam(S, 1)
#define    PROG_GAIN_PRM(S)             ssGetSFcnParam(S, 2)
#define    NUM_OF_CHANNELS_PRM(S)       ssGetSFcnParam(S, 3)
```

当仿真运行时，用户在块对话框的 S-Function parameters 域中输入四个变量名或数值。第一个输入对应第一个预期的参数 BASE_ADDRESS_PRM(S)；第二个输入对应下一个预期参数；依次类推。

函数 mdlInitializeSizes 中应包含语句：ssSetNumSFcnParams(S, 4);

可调参数

对话框参数既可以是可调的，也可以是不可调的。可调参数是在仿真运行中用户可以改变的参数。在 mdlInitializeSizes 中使用 ssSetSFcnParamTunable 宏指定由宏使用的对话框参数的可调性。

注意：在默认情况下，对话框参数都是可调的。然而，尽管参数都是可调的，通过对每个参数进行可调性设置可获得很好的编程经验。如果用户激活了仿真诊断 S-function upgrade needed，只要 Simulink

遇到一个指定所有参数可调性失败的 S-function，就会发布诊断。

在仿真运行时，mdlCheckParameters 函数能够让你确认对于可调参数的改变。只有在仿真循环过程中，用户改变了参数的值，Simulink 就会调用 mdlCheckParameters 函数。该函数将检查 S-function 的对话框参数以确保改变有效。

注意：S-function 的 mdlInitializeSizes 程序也应调用 mdlCheckParameters 函数以确保参数的初始值是有效的。

可选的 mdlProcessParameters 回调函数允许 S-function 对于可调参数的改变进行处理。只有在上一采样步时发生了有效的参数改变，Simulink 才会调用该函数。该函数的一个典型用法是执行那些只依赖于参数值的计算，因而仅在参数值发生变化时才需要进行计算。该函数能够把参数计算结果在工作向量中进行高速缓存，或者更好的办法是作为运行参数。

在外部模式下调节参数

当用户在仿真过程中调节参数时，Simulink 调用 S-function 的 mdlCheckParameters 函数来确认改变，然后调用 S-function 的 mdlProcessParameters 函数提供给 S-function 按照某种方法处理参数的机会。在外部模式下运行仿真时，Simulink 也可调用这些函数，但它传递未经处理的改变给 S-function 目标。因此，如果你的 S-function 对于参数改变的处理是必需的，那么在代码生成进程中，你必须为 S-function 创建一个目标语言编译器（Target Language Compiler—TLC）文件，该文件包含了参数处理代码。关于内嵌 S-function 的有关信息，请参考《*Target Language Compiler Reference Guide*》。

运行参数

Simulink 允许 S-function 创建和使用外部对话框参数的内部表示法，这种方法表示的参数称为运行参数。每个运行参数对应着一个或多个对话框参数，并可与其对应的外部参数具有相同的值或数据类型，或者不同的值或数据类型。如果一个运行参数与其对应的外部参数具有不同的值或数据类型，则该对话框参数可能已经被转化以创建运行参数。与多个对话框参数对应的运行参数值一般是对话框参数值的函数。Simulink 负责运行参数存贮内存的分配与释放，并提供函数来更新与访问它们，因此，避免了 S-function 来执行这些任务。

运行参数的使用可方便以下的 S-function 操作：

- 需要计算的参数

  一个块的输出往往是几个对话框参数值的函数。例如，假设一个块有两个参数：某个物体的体积与密度，输出是输入信号与物体重量的函数。在这种情况下，重量可以看做从两个外部参数——体积和密度计算出来的第三个内部参数。S-function 可以对应于计算出的重量创建一个运行参数，从而避免了在输出计算中提供重量的处理。

- 数据类型的转换

  块往往需要对于对话框参数进行数据类型的转换以方便内部处理。例如，假设块的输出是输入与一个参数的函数，而输入与该参数是不同的数据类型。在这种情况下，S-function 可以创建一个运行参数，使之与对话框参数的值相同，与输入信号的数据类型相同，这样将该运行参数用于输出的计算中。

- 代码生成

  在代码生成过程中，Real-Time Workshop 自动将所有的运行参数写入 model.rtw 文件中，这样避免了 S-function 使用 mdlRTW 函数来执行这些任务。

创建运行参数

S-function 可以一次创建所有的运行参数，也可以一个一个地创建运行参数。

一次创建所有的运行参数

使用 mdlSetWorkWidths 中的 SimStruct 函数 ssRegAllTunableParamsAsRunTimeParams 来对所有可调参数创建对应的运行参数。该函数需要一个参数名数组，每个参数名对应着一个运行参数，RTW 在代码生成过程中使用这个名字作为参数名。

这种创建运行参数的方法使得 S-function 的运行参数与可调对话框参数之间具有一一对应的关系。它不能应用于这种情况：比如，S-function 需要一个经计算的参数，它的值是几个对话框参数值的函数。那么，在这种情况下，S-function 必须单独地创建运行参数。

分开创建运行参数

要分开创建运行参数，S-function 的 mdlSetWorkWidths 函数必须：

1. 使用 ssSetNumRunTimeParams 指定需要使用的运行参数的数量；

2. 使用 ssRegDlgParamAsRunTimeParam 来注册与单个未转换对话框参数相对应的一个运行参数，或者使用函数来设置与多个对话框参数相对应的、或与单个转换对话框参数相对应的一个运行参数的属性。

注意：块运行参数名字的头四个字符必须是唯一的。否则，Simulink 会发出一个错误信号。例如，如果一个名字为 param1 的参数已经存在，那么试图采用 param2 作为参数名来注册一个参数将会触发错误。这种限制保证了 RTW 在预定义的字符数目内生成的变量名是唯一的。

更新运行参数

在仿真进行过程中，用户一旦改变了一个 S-function 对话框参数的值，Simulink 就会调用函数 mdlCheckParameters 来确认改变。如果改变有效，在下一个仿真步的开头 Simulink 就会调用 S-function 的 mdlProcessParameters 函数。该函数将会按照对话框参数的改变更新 S-function 的运行参数。

一次性更新所有参数

如果 S-function 的可调对话框参数与运行参数之间具有一一对应的关系，S-function 可使用 SimStruct 函数 ssUpdateAllTunableParamsAsRunTimeParams 来完成该更新任务。该函数将运行参数更新为与对应的对话框参数具有相同的值。

分别更新参数

如果 S-function 的可调对话框参数与运行参数之间不具有一一对应的关系，或者运行参数是经过对话框参数转换而来的，那么 mdlProcessParameters 必须分别更新每个参数。

如果运行参数与其对应的对话框参数仅值不同，可使用 ssUpdateRunTimeParamData 来更新运行参数。该函数使用新值更新参数属性记录中的数据域——ssParamRec。如果运行参数与其对应的对话框参数仅数据类型不同，可用 ssUpdateDlgParamAsRunTimeParam 更新运行参数。否则，mdlProcessParameters 必须自己更新参数的属性记录。要更新属性记录，该函数必须：

1. 使用 ssGetRunTimeParamInfo 来获取参数属性记录的指针
2. 映射对应对话框参数的改变来更新属性记录
3. 使用 ssUpdateRunTimeParamInfo 来注册改变

创建输入和输出端口

Simulink 允许 S-function 创建和使用任意数量的块 I/O 端口。本节介绍如何创建和初始化 I/O 端口，以及如何基于所连接的其它块来改变 S-function 块端口的特性，诸如维数、数据类型。

创建输入端口

要创建和配置输入端口，mdlInitializeSizes 函数首先必须使用 ssSetNumInputPorts 指定该 S-function 输入端口的数量；然后，对于每个输入端口，该函数必须：

- 指定输入端口的维数

  如果你希望 S-function 从该端口所连接的块继承其维数，应该在 mdlInitializeSizes 函数中将该端口的维数指定为动态宽度。

- 指定该输入端口是否允许输入的标量扩展
- 使用 ssSetInputPortDirectFeedThrough 指定该输入端口是否具有直接馈通

  如果该输入被用于 mdlOutputs 函数，或者 mdlGetTimeOfNextVarHit 函数，则该端口具有直接馈通。每个输入端口的直接馈通标志可以为设置为 1 = yes，或者 0 = no 。如果输入 *u* 用于 mdlOutputs 函数或 mdlGetTimeOfNextVarHit 函数中，直接馈通标志应设置为 1。将直接馈通标志应设置为 0，则等于告诉 Simulink 输入 *u* 没有用于这些 S-function 的程序中。违反此规定将导致不可预见的结果。

- 如果该输入端口的数据类型不是默认的 double 型，应指定数据类型

  使用 ssSetInputPortDataType 来设置输入端口的数据类型。如果你希望数据类型取决与该端口所连接端口的数据类型，应将该输入端口的数据类型指定为 DYNAMICALLY_TYPED。在这种情况下，你必须提供 mdlSetInputPortDataType 与 mdlSetDefaultPortDataTypes 函数的实现方法以使在信号传播过程中数据类型得以正确设置。

- 指定输入端口的数字类型

  如果该端口接受复数值信号，使用 ssSetInputComplexSignal 来设置该输入端口的数字类型。如果你希望数字类型取决与该端口所连接端口的数字类型，将数字类型指定为 inherited。在这种情况下，你必须提供 mdlSetInputPortComplexSignal 与 mdlSetDefaultPortComplexSignal 函数的实现方法以使在信号传播过程中数字类型得以正确设置。

注意：mdlInitializeSizes 函数在设置任何特性之前，必须指定端口的数量。如果试图设置一个不存在端口的属性，则会访问无效的内存空间，并且 Simulink 会产生冲突。

初始化输入端口的维数

以下选项用来设置输入端口的维数：

- 如果输入信号是一维的，且输入端口的宽度为 w，使用 ssSetInputPortVectorDimension(S, inputPortIdx, w)语句进行设置
- 如果输入信号是一个 m×n 的矩阵，使用 ssSetInputPortMatrixDimensions(S, inputPortIdx, m, n) 语句进行设置

- 否则，使用 ssSetInputPortDimensionInfo(S, inputPortIdx, dimsInfo) 进行设置

你可以使用该函数全部或部分地初始化端口的维数（参考下面的介绍）

动态确定输入端口的大小

如果你的 S-function 不需要输入信号具有特定的维数，你可能希望该输入端口的维数与其所连接的端口维数相匹配。要使输入端口具有动态维数，那么 S-function 必须：

- 在 mdlInitializeSizes 中，指定输入端口的某些或所有维的大小为动态的

  如果输入端口可以接受任意维数的信号，使用 ssSetInputPortDimensionInfo(S, inputPortIdx, DYNAMIC_DIMENSION) 来设置该端口的维数；

  如果输入端口只能接受向量（一维）信号，且可以是任意长度，使用 ssSetInputPortWidth(S, inputPortIdx, DYNAMICALLY_SIZED) 来设置该端口的维数；

  如果输入端口只能接受矩阵信号，且行和列可以是任意长度，使用 ssSetInputPortMatrixDimensions(S, inputPortIdx, m, n)。其中，m 和/或 n 是 DYNAMICALLY_SIZED。

- 提供一个 mdlSetInputPortDimensionInfo 函数来设置输入端口的维数为其所连接的信号维数

  当 Simulink 可以确定输入端口所连接信号的维数时，在信号传递过程中调用该函数。

- 提供一个 mdlSetDefaultPortDimensionInfo 函数来设置输入端口的维数为默认值

  当 Simulink 不能确定输入端口所连接信号的某些或全部维数时，在信号传递过程中调用该函数。这种情况是可能发生的，比如，如果该输入端口未连接。如果 S-function 不提供这个函数，Simulink 将块的输入维数设置为一维标量。

创建输出端口

要创建和配置输出端口，mdlInitializeSizes 函数首先必须使用 ssSetNumOutputPorts 指定该 S-function 输出端口的数量；然后，对于每个输出端口，该函数必须：

- 指定输出端口的维数

  Simulink 提供了以下几个宏用于设置输出端口的维数：

  - ssSetOutputPortDimensionInfo
  - ssSetOutputPortMatrixDimensions
  - ssSetOutputPortVectorDimensions
  - ssSetOutputWidth

  如果你希望端口的维数取决于块的连通性，应将维数设置为 DYNAMICALLY_SIZED。然后，S-function 必须提供 mdlSetOutputPortDimensionInfo 与 ssSetDefaultPortDimensionInfo 函数，以使得在代码生成时输出端口的维数被设置为正确的值。

- 指定输出端口的数据类型

  使用 ssSetOutputPortDataType 来设置输出端口的数据类型。如果你希望端口的维数取决于块的连通性，应将数据类型设置为 DYNAMICALLY_TYPED。在这种情况下，你必须提供 mdlSetOutputPortDataType 与 mdlSetDefaultPortDataTypes 函数的实现方法，以便在信号传递过程中正确设置数据类型。

■ **指定输出端口的数字类型**

如果输出端口为复数值信号，应使用 ssSetOutputComplexSignal **来设置输出端口的数字类型。如果该端口的数字类型取决于所连接端口的数字类型，应指定数字类型为** inherited**。在这种情况下，你必须提供** mdlSetOutputPortComplexSignal **与** mdlSetDefaultPortComplexSignal **函数，以便在信号传递过程中正确设置数字类型。**

输入的标量扩展

输入的标量扩展从概念上讲就是将输入端口的标量输入信号进行处理扩展成与其所连端口的信号具有相同维数。通过将扩展信号的每个元素设置为标量输入的值来实现。S-function **的** mdlInitializeSizes **函数通过使用** ssSetOptions **设置** SS_OPTION_ALLOW_INPUT_SCALAR_EXPANSION **选项使得对其输入端口能够实现输入的标量扩展。**

要了解标量扩展规则的最好办法是参考一个两输入的 Sum**（加法）块例子，其中第一输入为标量，第二输入为** w **个元素（**w>1**）的一维向量，输出为** w **个元素的一维向量。在这种情况下，在输出函数中，标量输入被扩展成具有** w **个元素的一维向量，并且扩展信号的每个元素被设定为标量输入的值。**

```
Outputs
    <snip>
    u1inc = (u1width > 1);
    u2inc = (u2width > 1);
    for (i=0;i<w;i++) {
        y[i] = *u1 + *u2;
        u1 += u1inc;
        u2 += u2inc;
    }
```

如果该块具有两个以上的输入，每个输入信号必须是标量，或者相同元素数量的宽信号。另外，宽信号是由一维和两维向量，输出是两维向量信号，而且标量输入被扩展为两维的向量信号。
标量扩展的实际工作方式取决于S-function**是否采用了**mdlSetInputPortWidth**和**mdlSetOutputPortWidth**，或者**mdlSetInputPortDimensionInfo、mdlSetOutputPortDimensionInfo**和**mdlSetDefaultPortDimensionInfo**来管理输入与输出端口的维数。**

如果 S-function **不采用以上函数来指定/控制输入与输出端口的维数，那么** Simulink **采用默认函数来设置输入与输出端口。**

在 mdlInitializeSizes **函数中，**S-function **通过采用** ssSetOptions **设置** SS_OPTION_ALLOW_INPUT_SCALAR_EXPANSION **选项来对于输入端口进行标量扩展。**Simulink **的默认函数使用上述选项来允许或禁止对于块的输入端口进行标量扩展。如果没有通过** S-function **设置上述选项，**Simulink **默认所有的端口（输入和输出端口）必须具有相同的维数，并将所有端口的维数设置为由一个驱动块所指定的维数。**

如果 S-function **指定/控制其输入和输出端口的维数，**Simulink **则忽略** SCALAR_EXPANSION **选项。**

参考 *matlabroot*/simulink/src/sfun_multiport.c **这个范例。**

掩码多端口 S-Function

如果你需要开发一个端口数量可变的多端口 S-function 块，其端口数量取决于某个参数，并且希望将该块放置在 Simulink 库中，那么，你必须指定掩码修改块的外形。要实现这个目的，保存在库中之前，在 MATLAB 提示符下执行命令：set_param('block','MaskSelfModifiable','on')。如果没有指定掩码修改块的外形，则意味着在使用该块的模型一旦被装载或更新库连接时，块端口数量将回复到其保存在库中时的端口数量。

自定义数据类型

S-function 可接受和输出用户定义的和 Simulink 内置的数据类型。要使用一个用户定义的数据类型，S-function 的 mdlInitializeSizes 程序必须：

1. 使用 ssRegisterDataType 注册该数据类型
2. 使用 ssSetDataTypeSize 指定该数据类型在内存中存贮的字节数
3. 使用 ssSetDataTypeZero 该数据类型的 0 值

采样时间

Simulink 支持执行不同速率的块。S-function 可以将其速率（例如，采样时间）指定为：

- 基于块的采样时间
- 基于端口的采样时间
- 基于块与端口的混合采样时间

在基于块的采样时间下，S-function 在仿真的初始化阶段，将块作为一个整体为其指定一组操作速率；在基于端口的采样时间下，S-function 在初始化阶段，为每个输入和输出端口分别指定采样时间。在执行阶段，对于基于块的采样时间，S-function 在块的一个采样点处理所有的输入和输出一次；与之不同的是基于端口的采样时间，当针对于某个端口的采样点发生时，块只对相应的端口进行处理。

例如，以一个两速率的采样为例，采样步长分别为 0.5 秒和 0.25 秒：

- 在基于块采样的方式下，通过对 0.5 秒和 0.25 秒进行选择，每个 0.25 秒引导块执行一次输入和输出。
- 在基于端口采样方式下，你可以将输入端口设置为 0.5 秒，将输出端口设置为 0.25 秒，那么块将按照 2Hz 的频率处理输入，按照 4Hz 的频率处理输出。

如果你的应用对于输入与输出的执行需要不同的采样速率，或者你不希望输入与输出端口的运行与最高采样速率的块相联系，那么你必须使用基于端口的采样时间。

在一些应用中，一个 S-function 内部可能需要执行一种或多种采样速率，同时输入或输出信号在其它采样速率下执行，那么采用基于块和端口的采样方式可以创建这样的块。

在一般应用中，你只需指定一个基于块的采样时间。高级的 S-function 可能需要基于端口或多种块采样速率。

基于块的采样时间

接下来的两个小部分讨论如何指定基于块的采样时间。你必须在下面的函数中指定有关信息：

- mdlInitializeSizes
- mdlInitializeSampleTimes

第三部分给出了一个简单的例子，说明如何在 mdlInitializeSampleTimes 中指定采样时间。

在 mdlInitializeSizes 中指定采样时间的数量

为了将你的 S-function 块配置为基于块的采样时间，使用 ssSetNumSampleTimes(S,numSampleTimes);

其中，numSampleTimes> 0，这告诉 Simulink 你的 S-function 具有基于块的采样时间。Simulink 会调用 mdlInitializeSampleTimes 依次来设置采样时间。

在 mdlInitializeSampleTimes 中设置采样时间及指定函数调用

mdlInitializeSampleTimes 用来指定两条执行信息：

- 采样时间和偏移时间——在 mdlInitializeSizes 中，通过使用宏 ssSetNumSampleTimes 指定了你希望在 S-function 中所使用的采样速率的数量。在 mdlInitializeSampleTimes 中，你必须指

定每个采样速率的周期和偏移。

采样时间可以是输入/输出端口宽度的函数。在 mdlInitializeSampleTimes 中，你可以指定采样时间为 ssGetInputPortWidth 和 ssGetOutputPortWidth 的函数。

- 函数调用——在 ssSetCallSystemOutput 中，指定执行函数调用的输出元素。参考范例 matlabroot/simulink/src/sfun_fcncall.c

你必须采用时间对[*sample_time*，*offset_time*]来指定一个采样时间，使用下面的宏：

ssSetSampleTime(S, *sampleTimePairIndex*, *sample_time*)

ssSetOffsetTime(S, *offsetTimePairIndex*, *offset_time*)

其中，*sampleTimePairIndex* 以 0 开始编号。

有效的时间对如下（大写字母的值是在 simstruc.h 中定义的宏）：

[CONTINUOUS_SAMPLE_TIME， 0.0]

[CONTINUOUS_SAMPLE_TIME， FIXED_IN_MINOR_STEP_OFFSET]

[*discrete_sample_period*， *offset*]

[VARIABLE_SAMPLE_TIME， 0.0]

当然，你也可以指定为从驱动块继承采样时间。在这种情况下，S-function 只有一个采样时间对：

[INHERITED_SAMPLE_TIME， 0.0]

或者，

[INHERITED_SAMPLE_TIME， FIXED_IN_MINOR_STEP_OFFSET]

以下几条指导思想可能会对你指定采样时间有帮助：

- 在积分微步内改变的连续函数的采样时间应指定为[CONTINUOUS_SAMPLE_TIME，0.0]；
- 在积分微步内不改变的连续函数的采样时间应指定 [CONTINUOUS_SAMPLE_TIME，FIXED_IN_MINOR_STEP_OFFSET]；
- 在特定速率在改变的离散函数的采样时间应指定为[*discrete_sample_period*，*offset*]，其中，*discrete_sample_period* > 0.0，并且 0.0 <= *offset* < *discrete_sample_period*
- 在可变速率在改变的离散函数的采样时间应指定为变步长离散采样时间 [VARIABLE_SAMPLE_TIME， 0.0]。应调用函数 mdlGetTimeOfNextVarHit 为变步长离散任务来获取下一步的采样时间点。只有 VARIABLE_SAMPLE_TIME 可与变步长求解器一起使用。

如果你的 S-function 没有固定的采样时间，你必须根据以下原则来指定采样时间的继承：

- 如果函数即使在积分微不内也会随着其输入的变化而变化，应指定采样时间为 [INHERITED_SAMPLE_TIME， 0.0]
- 如果一个函数随其输入的改变而改变，但在积分微步内不变化（即积分微步内保持），应指定采样时间为[INHERITED_SAMPLE_TIME， FIXED_IN_MINOR_STEP_OFFSET]

要在执行过程中(在 mdlOutputs 或 mdlUpdate 中)检查采样点,使用 ssIsSampleHit 或 ssIsContinuousTask 宏。例如，如果你的第一个采样时间是连续的，那么使用下面的代码片段来检查采样点。注意，如果你使用 ssIsSampleHit(S,0,tid)将得到不正确的结果：

```
if (ssIsContinuousTask(S,tid)) {
```

```
}
```

如果你希望确定第三个任务（比如为离散任务）是否具有采样点，使用下面的代码片段：

```
if (ssIsSampleHit(S,2,tid) {
}
```

范例：mdlInitializeSampleTimes

该范例指定两个离散的采样时间，周期分别为 0.01 秒和 0.5 秒：

```
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, 0.01);
    ssSetOffsetTime(S, 0, 0.0);
    ssSetSampleTime(S, 1, 0.5);
    ssSetOffsetTime(S, 1, 0.0);
}   /* End of mdlInitializeSampleTimes. */
```

指定基于端口的采样时间

如果希望 S-function 使用基于端口的采样时间，你必须在 mdlInitializeSizes 中指定作为基于端口采样时间的数量：ssSetNumSampleTimes(S, PORT_BASED_SAMPLE_TIMES)

在 mdlInitializeSizes 中，你还必须指定每个输入和输出端口的采样时间，使用以下宏：

```
ssSetInputPortSampleTime(S, idx, period)
ssSetInputPortOffsetTime(S, idx, offset)
ssSetOutputPortSampleTime(S, idx, period)
ssSetOutputPortOffsetTime(S, idx, offset)
```

注意：当使用基于端口的采样时间时，mdlInitializeSizes 中不应该任何 ssSetSampleTime 或 ssSetOffsetTime 调用

对于任何给定的端口，你可以指定：

- 一个特定的采样时间和偏移量

  例如，以下代码将 S-function 第一个输入端口的采样时间设置为仿真开始时每 0.1 秒采样一次：

  ```
  ssSetInputPortSampleTime(S, 0, 0.1);
  ssSetInputPortOffsetTime(S, 0, 0);
  ```

- 继承采样时间。比如，端口从其所连接的端口继承采样时间；
- 恒定采样时间。比如，端口的输入或输出从不变化

注意：如果希望 S-function 适用于触发子系统，那么 S-function 所有端口的采样时间必须是继承或连续的。

对于端口指定继承采样时间

要指定端口的采样时间是继承的，，在 mdlInitializeSizes 中应将采样周期设置为-1，偏移量设置为 0。例如，以下代码将 S-function 的第一输入端口的采样时间指定为继承：

```
ssSetInputPortSampleTime(S, 0, -1);
ssSetInputPortOffsetTime(S, 0, 0);
```

当基于端口指定采样时间时，Simulink 调用 mdlSetInputPortSampleTime 和 mdlSetOutputPortSampleTime 来确定继承信号的速率。

所有速率一旦被确定，Simulink 就调用 mdlInitializeSampleTimes。即使此时没有基于端口的采样时间需要初始化，Simulink 也会调用该函数以提供 S-function 一个配置函数调用关系的机会。因此，不论你的 S-function 是否使用了基于端口的采样时间或函数调用关系，你在 S-function 中都必须编写该函数的内容。虽然你可以让该函数执行空操作，但你还是可能需要使用该函数来检查在采样时间传递过程中块继承的采样时间是否合适。

对于端口指定恒定采样时间

如果你的 S-function 使用基于端口的采样时间，可指定任何端口具有恒定的采样时间。这意味着输入该端口或从该端口输出的信号从仿真开始起一直保持初始值不变。

在对于一个输出取决于 S-function 参数的输出端口指定恒定的采样时间之前，S-function 应该使用 ssGetInlineParameters 来检查用户是否在 Simulation parameters 对话框的 Advanced 选卡中选择了 Inline parameter 选项。如果用户没有选择该选项，那么对用户而言，在仿真中是可能会改变 S-function 参数的值，继而改变到输出的值。在这种情况下，S-function 不应该对任何输出取决于参数的端口指定恒定的采样时间。

要对于端口指定恒定的采样时间，S-function 必须执行以下任务：

- 在 mdlInitializeSizes 中，通知 Simulink 它支持恒定的采样时间

```
ssSetOptions(S, SS_OPTION_ALLOW_CONSTANT_PORT_SAMPLE_TIME);
```

注意：通过设置该选项，你的 S-function 实际上等于告诉 Simulink 其所有端口均支持恒定的采样时间，包括那些从其它块继承采样时间的端口。如果 S-function 的任一继承采样时间的端口不可有恒定采样时间，那么 S-function 中的函数 mdlSetInputPortSampleTime 和 mdlSetOutputPortSampleTime 必须检查这样的端口是否继承了恒定的采样时间。如果继承了恒定的采样时间，S-function 应该给出错误提示。

- 设置端口采样周期为 inf，偏移量为 0，例如：

```
ssSetInputPortSampleTime(S, 0, mxGetInf());
ssSetInputPortOffsetTime(S, 0, 0);
```

- 在 mdlOutputs 中检查该函数的 tid 参数是否等于，如果相等，对于输出端口设置端口输出值

通过范例 sfun_port_constant.c 了解如何创建恒定采样时间的端口，该范例是 sfcndemo_port_constant 演示程序的源文件。

对于触发子系统的应用配置基于端口的采样时间

为了使 S-function 能够应用于触发子系统，基于端口采样时间的 S-function 必须执行以下任务：

- 在 mdlInitializeSizes 中通知 Simulink 本 S-function 能够运行于触发子系统中：

  ssSetOptions(S, SS_OPTION_ALLOW_PORT_BASED_SAMPLE_TIME_IN_TRIGSS);

- 在 mdlInitializeSizes 中设置 S-function 块的所有端口的采样时间为继承或恒定的采样时间；
- 在 mdlSetInputPortSampleTime 和 mdlSetOutputPortSampleTime 中按照下面的方式处理触发采样时间的继承性。

如果 S-function 放置在一个触发子系统中，在采样时间传递过程中，Simulink 会调用 mdlSetInputPortSampleTime 或 mdlSetOutputPortSampleTime。无论调用哪一个函数，必须将对应端口的采样时间和偏移量设置为 INHERITED_SAMPLE_TIME（-1）。例如：

```
ssSetInputPortSampleTime(S, 0, INHERITED_SAMPLE_TIME);
ssSetInputPortOffsetTime(S, 0, INHERITED_SAMPLE_TIME);
```

将一个端口的采样时间和偏移量均设置为 INHERITED_SAMPLE_TIME，指示该端口的采样时间是被触发的，比如只有当它存放的子系统被触发时，端口才产生输出或接受输入。该函数也必须将 S-function 其它所有的输入和输出端口的采样时间设置为触发的或恒定的采样时间，都可以。

对于触发子系统中的 S-function 而言，无法预知 Simulink 是否会调用 mdlSetInputPortSampleTime 或 mdlSetOutputPortSampleTime 来设置其采样时间。因为这个原因，这两个函数能够正确地设置采样时间。

- 在 mdlUpdate 和 mdlOutputs 中，使用来检查 S-function 是否存放于触发子系统中，如果是，则可采用适当的数学运算来计算其状态和输出。

通过范例 sfun_port_triggered.c 了解如何创建恒定采样时间的端口，该范例是 sfcndemo_port_triggered 演示程序的源文件。

基于块与基于端口的混合采样时间

指定采样时间的混合方式组合了基于块与基于端口两种采样时间。在 mdlInitializeSizes 中，首先使用 ssSetNumSampleTimes 指定块操作速率的数量有多少种，包括内部的、输入和输出的速率；然后，使用 ssSetOptions 设置 SS_OPTION_PORT_SAMPLE_TIMES_ASSIGNED 选项，用来告诉 Simulink 引擎将使用基于端口的方式分别指定输入和输出端口的速率；接下来，按照基于块的方式指定所有块速率的采样周期和偏移量，其中包括内部和外部，使用的函数是：

```
ssSetSampleTime
ssSetOffsetTime
```

最后，按照基于端口方式来指定每个端口的速率，使用的函数是：

```
ssSetInputPortSampleTime(S, idx, period)
ssSetInputPortOffsetTime(S, idx, offset)
ssSetOutputPortSampleTime(S, idx, period)
```

```
ssSetOutputPortOffsetTime(S, idx, offset)
```

注意，对于每个端口所指定的速率必须与上一步按照基于块指定的采样速率中的一种速率相同。

多速率 S-Function 块

在一个多速率 S-function **块中，通过一条判定采样点是否发生的语句，可以将那些在** mdlOutputs **和** mdlUpdate **函数中定义行为的操作代码组合起来。**ssIsSampleHit **就是这样的一个宏，它可以判断当前时刻是否是一个指定的采样点。该宏的语法如下：**

```
ssIsSampleHit(S, st_index, tid)
```

其中，S **是** SimStruct**，**st_index **标识了一个特定采样速率的索引号，**tid **是任务** ID **(**tid **是函数** mdlOutputs **和** mdlUpdate **的一个参数）。**

例如，以下这些语句指定了三个采样时间：一个是连续行为，两个是离散行为：

```
ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
ssSetSampleTime(S, 1, 0.75);
ssSetSampleTime(S, 2, 1.0);
```

在 mdlOutputs **函数中，下面的语句封装了** 0.75 **秒的采样时间所定义的操作：**

```
if (ssIsSampleHit(S, 1, tid)) {
/*  需要进行的操作  */
}
```

其中，第二个参数为 1**，对应着采样周期为** 0.75 **秒的第二个速率。**

一个定义连续块采样时间的范例

该例子是对于一个连续块定义一个采样时间：

```
/*  初始化采样时间和偏移量  */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}
```

你必须在 mdlInitializeSizes **函数中增加一条语句：**ssSetNumSampleTimes(S, 1);

一个定义混合块采样时间的范例

该例子是对于一个混合 S-function **块定义采样时间：**

```
/*  初始化采样时间和偏移量  */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    /*  连续状态采样周期和偏移量  */
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
```

```
        ssSetOffsetTime(S, 0, 0.0);
        /* 离散状态采样周期和偏移量 */
        ssSetSampleTime(S, 1, 0.1);
        ssSetOffsetTime(S, 1, 0.025);
    }
```

第二个采样速率的偏移量为 0.025 秒，因此 Simulink 调用 mdlUpdate 函数的时刻是：0.025 秒，0.125 秒，0.225 秒，…，时间间隔为 0.1 秒。

下面这条语句定义了采样速率的数量，必须增加在 mdlInitializeSizes 函数中：

```
ssSetNumSampleTimes(S, 2);
```

多速率 S-Function 块的同步

如果在不同速率下运行的任务需要共享数据，必须确保一个任务产生的数据可以被另一个速率运行的任务有效访问。可以在多速率 S-function 的 mdlOutputs 和 mdlUpdate 函数中使用 ssIsSpecialSampleHit 宏，以确保共享数据的有效性。如果一种速率的采样点已经发生，并且在同一时间步长内另一速率的采样点也发生了，那么这个宏返回 True。因此，允许一个高速率的任务在低速率任务能适应的速率下向其提供所需要的数据。

例如，假设你的模型中一个输入端口在速率 0 在操作，一个输出端口在较低的速率 1 下操作。再假设你需要输出端口将输入端口当前的数值输出，下面的程序示范了 ssIsSpecialSampleHit 的用法：

```
if (ssIsSampleHit(S, 0, tid) {
if (ssIsSpecialSampleHit(S, 0, 1, tid) {
    /* 将输入传输到输出内存 */
    ...
    }
}
if (ssIsSampleHit(S, 1, tid) {
/* 执行输出 */
...
}
```

在本例中，第一个块按照输入速率在采样点发生时运行，如果按照输出速率在此时也有采样点发生，第一个块就会将输入数据传输到输出的内存中；第二个块按照输出速率在采样点发生时运行，它将内存区域中的内容传输到块的输出端口。

注意，高速率的任务总是在低速率任务之前运行。因此，上面例子中的输入任务总是在输出任务之前运行，保证了出现在输出端口的数据总是有效的。

工作向量

如果你的 S-function 需要固定的内存存贮，应使用 S-function 的工作向量来取代静态或全程变量。如果你使用的是静态或全程变量，它们会在你的 S-function 的多个副本中使用。当一个 Simulink 模型中具有多个 S-function 块，并且每个块都指定了同一个 S-function C MEX 文件，这种情况就会发生。这种保持对一个 S-function 的多个副本进行跟踪的能力被称为 ***reentrancy（重入）***。

你可以使用工作向量创建一个可以重入的 S-function。这些工作向量是 Simulink 为 S-function 管理的固定的内存空间，它支持整数、浮点（实数）、指针、以及通用数据类型。每个向量的元素数量可以作为 S-function 输入数量的函数而动态确定。

工作向量具有以下几点优势：

- 用于块变量的特定副本存贮
- 整数、实数、指针、以及通用数据类型
- 避免了静态和全程变量，以及多副本变量关联的问题

例如，假设你想跟踪前一次输入到 S-function 输入端口 1 的每个输入信号元素，无论是离散状态向量或是实型工作向量都可以实现这种需要，主要取决于上次的输入值是否是一个离散状态（即比较单位延迟块与记忆块）。如果你不希望在保存状态时记录上次的值，那么就使用实型工作向量 rwork。要实现该操作，首先要在 mdlInitializeSizes 中使用 ssSetNumRWork 设置该向量的长度；然后，在 mdlStart 或 mdlInitializeConditions 中，初始化 rwork 工作向量 ssGetRWork。在 mdlOutputs 中，你可以使用 ssGetRWork 获取以前的输入值 ;在 mdlUpdate 中，可以使用 ssGetInputPortRealSignalPtrs 更新以前的 rwork 工作向量值。

在 mdlInitializeSizes 中使用下表提供的宏来为 S-function 的每个副本指定工作向量的宽度：

宏	说　明
ssSetNumContStates	连续状态向量的宽度
ssSetNumDiscStates	离散状态向量的宽度
ssSetNumDWork	数据类型工作向量的宽度
ssSetNumRWork	实型工作向量的宽度
ssSetNumIWork	整型工作向量的宽度
ssSetNumPWork	指针工作向量的宽度
ssSetNumModes	模式工作向量的宽度
ssSetNumNonsampledZCs	非采样过零检测向量的宽度

在 mdlInitializeSizes 中指定向量的宽度，有三种选择：

- 0（默认）。这表示 S-function 不使用该向量
- 一个非 0 的正整数。这是工作向量的有效宽度值，这些向量可在 mdlStart，mdlInitializeConditions，及在仿真循环中调用的 S-function 程序中使用。
- DYNAMICALLY_SIZED。这是对于动态宽度向量的默认设置，它将向量宽度设置为全部的块宽度。Simulink 在传递线宽度和采样时间之后，才进行设置。块宽度是通过块传递的信号宽度，

一般情况下它等同输出端口的宽度。

如果动态宽度向量的默认设置不符合你的需求，那么应使用 mdlSetWorkWidths 和上表中的宏来明确地设置工作向量的宽度。mdlSetWorkWidths 也允许你将工作向量的宽度设置成块采样时间和/或端口宽度的函数。

当状态需要使用 Simulink 求解器进行积分的时，应使用连续状态。当你指定了连续状态时，必须在 mdlDerivatives 中返回状态的导数。离散状态向量用来保持在固定间隔变化的信息。典型做法是对于离散状态向量的更新被放置在 mdlUpdate 中。

在仿真过程中，Simulink 不记录整数、实数和指针工作向量的存储位置，在对于 S-function 的调用之间，这些工作向量固定地保持数据。

工作向量与过零检测

模式工作向量和非采样的过零检测向量一般与过零检测一起使用。模式向量的元素是整数值。你可以在 mdlInitializeSizes 中使用 ssSetNumModes(S, num) 指定模式向量元素的数量，然后可以使用 ssGetModeVector 访问模式向量。当求解器在引导过零检测时，模式向量用来确定 mdlOutputs 应该何如操作。过零检测或一些信号（通常是 S-function 输入的函数）的状态事件（比如，在一阶导数中的非连续性）由求解器通过查看非采样过零检测而进行跟踪。要登记非采样过零检测，首先在 mdlInitializeSizes 中使用 ssSetNumNonsampledZCs(S, num)来设置非采样过零检测的数量；然后，定义 mdlZeroCrossings 程序以返回非采样过零检测。参考范例 *matlabroot*/simulink/src/sfun_zc.c。

包括指针工作向量的范例

该范例打开一个文件，并将 FILE 指针存储在指针工作向量中。

下面的语句包含于 mdlInitializeSizes 函数中，所说明的指针工作向量中只包含一个元素：

```
ssSetNumPWork(S, 1)       /* pointer-work vector */
```

下面的代码使用指针工作向量来存储 FILE 指针，该指针是从标准的 I/O 函数 fopen 中返回的：

```
#define MDL_START        /* Change to #undef to remove function. */
#if defined(MDL_START)
static void mdlStart(real_T *x0, SimStruct *S)
{
    FILE *fPtr;
    void **PWork = ssGetPWork(S);
    fPtr = fopen("file.data", "r");
    PWork[0] = fPtr;
}
#endif /* MDL_START */
```

下面的代码实现通过指针工作向量获得 FILE 指针，并将其传递给函数 fclose 以关闭文件：

```
static void mdlTerminate(SimStruct *S)
{
    if (ssGetPWork(S) != NULL) {
    FILE *fPtr;
    fPtr = (FILE *) ssGetPWorkValue(S,0);
    if (fPtr != NULL) {
        fclose(fPtr);
        }
    ssSetPWorkValue(S,0,NULL);
    }
}
```

注意：如果你使用的是 mdlSetWorkWidths，**在** S-function **的** mdlInitializeSizes **函数中所使用的任何工作向量将被设置为** DYNAMICALLY_SIZED，**即使在** mdlInitializeSizes **被调用之前已经知道了准确值。用于** S-function **的工作向量长度应该在** mdlSetWorkWidths **中指定。**

格式如下：

```
#define MDL_SET_WORK_WIDTHS          /* Change to #undef to remove function. */
#if defined(MDL_SET_WORK_WIDTHS) && defined(MATLAB_MEX_FILE)
static void mdlSetWorkWidths(SimStruct *S)
{
}
#endif                                          /* MDL_SET_WORK_WIDTHS */
```

参考范例 *matlabroot*/simulink/src/sfun_dynsize.c。

内存分配

在创建 S-function **时，可用的工作向量可能不足以提供变量存储空间，在这种情况下，必须为** S-function **的每个副本分配内存。**MATLAB **标准的内存分配函数** mxCalloc **和** mxFree **不能够与** C MEX S-function **一起使用，因为这些程序设计为从** MATLAB **调用与** MEX **文件一起使用，而不能从** Simulink **调用。分配内存的正确方法是使用使用** stdlib.h **库中的** calloc **和** free **程序。在** mdlStart **中，进行内存的分配与初始化，并地址指针或者放置在指针工作向量的元素中：**

```
ssGetPWork(S)[i] = ptr;
```

或者将其附着作为用户数据：

```
ssSetUserData(S,ptr);
```

在 mdlTerminate **中，释放分配的内存空间。**

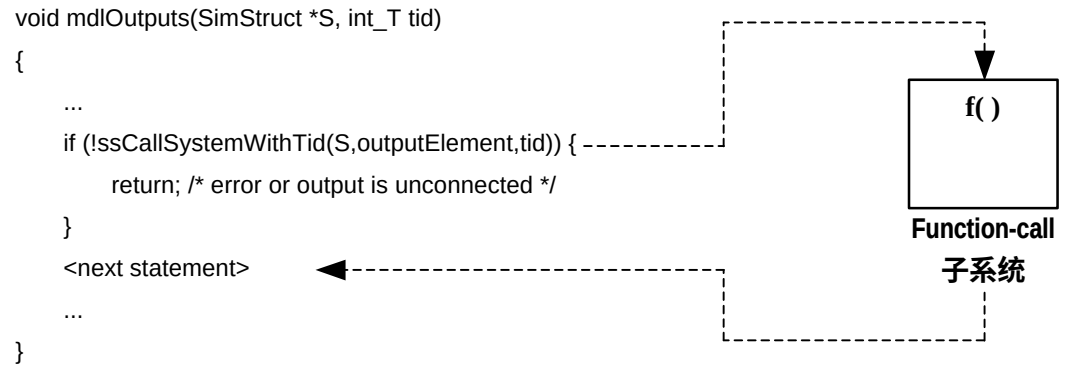
Function-Call 子系统

一个触发子系统的执行决定于对于 S-function 的内部逻辑，而不是 S-function 的信号值。如此配置的一个子系统被称为 Function-Call 子系统。要实现一个 function-call 子系统，必须：

- 在 Trigger 块中，将 Trigger type 的参数选择为 function-call；
- 在 S-function 中，使用 ssEnableSystemWithTid 宏和 ssDisableSystemWithTid 宏来激活或禁止触发子系统，使用 ssCallSystemWithTid 宏来调用触发字系统；
- 在模型中，将 S-function 块的输出直接连接到触发端口。

注意：Function-call 的连接只能在第一输出端口上进行

Function-call 子系统不能直接被 Simulink 执行，更准确地讲，S-function 决定了何时执行子系统。当子系统执行完成时，控制权返还给 S-function。下面的图表示了 function-call 子系统与 S-function 之间这种相互作用的关系：



在上图中，ssCallSystemWithTid 执行连接到第一个输出端口元素的 function-call 子系统。如果在 function-call 子系统时发生错误，或者输出未连接，ssCallSystemWithTid 返回 0。在 function-call 子系统执行完成之后，控制权交还给 S-function。

Function-call 子系统只能与 S-function 连接，S-function 应正确配置以保证能够调用 function-call 子系统。

要配置一个可调用 function-call 子系统的 S-function，必须：

- 在 mdlInitializeSampleTimes 中指定执行 function-call 子系统的元素。例如：

```
ssSetCallSystemOutput(S,0); /* 通过第一个元素调用 */
ssSetCallSystemOutput(S,2); /* 通过第三个元素调用 */
```

- 在 mdlInitializeSampleTimes 中指定是否需要 S-function 具有能够激活或禁止 function-call 子系统的能力。例如：

```
ssSetExplicitFCSSCtrl(SimStruct *S, TRUE);
```

- 在 S-function 的 mdlOutputs 或 mdlUpdate 程序中执行子系统。例如：

```
static void mdlOutputs(...)
{
    if (((int)*uPtrs[0]) % 2 == 1) {
```

```
            if (!ssCallSystemWithTid(S,0,tid)) {
            /* Error occurred, which will be reported by Simulink */
               return;
            }
        } else {
            if (!ssCallSystemWithTid(S,2,tid)) {
            /* Error occurred, which will be reported by Simulink */
               return;
            }
        }
        ...
    }
```

参考范例 simulink/src/sfun_fcncall.c

Function-call **子系统具有一个强大的模块化结构。你可以通过配置** Stateflow®**块来执行** function-call **子系统，因此扩展了块的能力。**

错误处理

当使用 S-function 工作时，正确处理意想不到的事件，比如无效参数值，是非常重要的。

如果你的 S-function 的参数内容需要确认，使用下面的方法来报告遇到的错误：

```
ssSetErrorStatus(S, "error encountered due to ...");
return;
```

注意，ssSetErrorStatus 的第二个参数必须是固定存储的，在你的程序中，该参数不能是一个局部变量。例如，下面的代码会带来不可预知的错误：

```
mdlOutputs()
{
    char msg[256];          /* 非法： should be "static char msg[256];" */
    sprintf(msg,"Error due to %s", string);
    ssSetErrorStatus(S,msg);
    return;
}
```

因为 ssSetErrorStatus 不产生超程错误，因而在 S-function 中使用它来报告错误比使用 mexErrMsgTxt 更合适。mexErrMsgTxt 函数使用了超程处理来终止 S-function 的执行，并将控制权交还给 Simulink。为了在 S-function 中支持超程处理，Simulink 必须在每个 S-function 调用之前建立超程处理器。这也给仿真引入了额外的开销。

防超程代码

通过保证 S-function 包含完整的防超程代码，你可以避免建立超程处理器带来的额外开销。防超程代码是指代码中绝对不包含长转移。如果你的 S-function 在调用时含有任何潜在的长转移，那么该 S-function 就不是防超程的。例如，在调用 mexErrMsgTxt 时，它采用了一个超程（比如，长转移），从而终止了 S-function 的执行。使用 mxCalloc 在遇到错误内存分配事件时，会带来一个不可预知的结果，因为 mxCalloc 进行了长转移。如果必须进行内存分配，直接使用 stdlib.h 的 calloc 程序，并执行你自己的错误处理。

如果你没有调用 mexErrMsgTxt 或其它带来超程错误的 API 程序，可以使用 S-function 的选项 SS_OPTION_EXCEPTION_FREE_CODE。你可以通过在 mdlInitializeSizes 函数中添加下面的这条语句，完成该选项的设置：

```
ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
```

设置该选项可以提供 S-function 的性能，因为它允许 Simulink 绕过超程处理的建立，一般超程处理的建立往往是在调用每个 S-function 之前都需要进行。在使用 SS_OPTION_EXCEPTION_FREE_CODE 时应特别小心检查，确保你的代码是防超程的。如果设置了该选项，一旦发生了超程错误，结果是不可预料的。

所有的 mex* 程序都有潜在的长转移，几个 mx* 程序也有潜在的长转移。为了避免任何问题，只使用 API 程序来获取指针或确定参数所占的内存大小。例如，下面的函数是绝不会产生超程的：mxGetPr，mxGetData，mxGetNumberOfDimensions，mxGetM，mxGetN，以及 mxGetNumberOfElements。

运行程序中的代码也会产生超程。运行程序是指在仿真循环中 Simulink 调用的某个 S-function 程序。运行程序包括：

- mdlGetTimeOfNextVarHit
- mdlOutputs
- mdlUpdate
- mdlDerivatives

如果在 S-function 中所有的运行程序都是防超程的，你可使用下面的选项进行说明：

```
ssSetOptions(S, SS_OPTION_RUNTIME_EXCEPTION_FREE_CODE);
```

这样的设置不必保证 S-function 中的其它程序也是防超程的。

SsSetErrorStatus 的终止条件

当你从 S-function 中调用 ssSetErrorStatus 函数并返回时，Simulink 停止仿真，并发出错误。要确定仿真是如何关闭的，参考第三章中的“ Simulink 如何与 C S-function 相互作用 ”这一节中的流程图（第 55 页）。如果 ssSetErrorStatus 在 mdlStart 之前调用，则不会调用其它的 S-function 程序 如果 ssSetErrorStatus 在 mdlStart 之中或之后调用，mdlTerminate 会被调用。

数组边界检查

如果你的 S-function 出现了其它无法解释的错误，原因可能是 S-function 的编写超出了分配的内存区域。你可以通过激活 Simulink 数组边界检查特性来检查这种可能性。这种检查特性可以监测任何通过 S-function 块进行的超边界操作，检查针对以下几种块数据的类型：

- 工作向量（R、I、P、D、以及模式工作向量）
- 状态（连续和离散）
- 输出

要激活数组边界检查，应在 Simulation Parameters 对话框的 Bounds checking 选项列表中选择 warning 或 error；或者在 MATLAB 命令行中输入下面的命令：

```
set_param(modelName, 'ArrayBoundsChecking', 'none' | 'warning' | 'error')
```

S-Function 范例

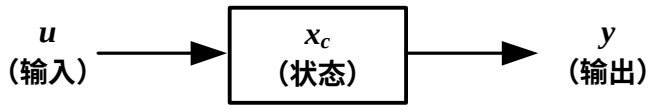
大多数 S-function 块需要处理连续或离散的状态。本节主要讨论几种通用类型的系统，你可以通过 S-function 在 Simulink 建立模型：

- 连续状态
- 离散状态
- 混合状态
- 变步长采样时间
- 过零检测
- 时变连续传递函数

所有的范例都是基于 C MEX 文件 S-function 模板 sfuntmpl_basic.c 和 sfuntmpl_doc.c 来编写的。

连续状态的 S-Function 范例

matlabroot/simulink/src/csfunc.c 范例示范了如何在一个 C MEX S-function 中模拟一个带状态的连续系统。在连续状态的积分中，通过 Simulink 求解器进行积分的一组状态满足以下方程：



$$y = f_0(t, x_c, u) \quad （输出）$$

$$\dot{x}_c = f_d(t, x_c, u) \quad （求导）$$

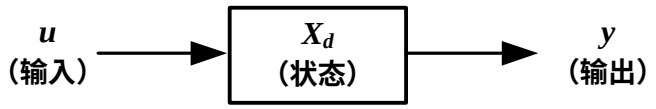
包含连续状态的 S-function 满足一个状态空间方程。输出部分的实现放置在 mdlOutputs 中，导数部分的实现放置在 mdlDerivatives 中。为了形象化地表述积分过程，可参考第三章中的“ Simulink 如何与 C S-function 相互作用 ”这一节中的流程图（第 55 页），上面的输出方程对应着采样主步长中的 mdlOutputs。接着，例子进入到流程图积分部分，这时 Simulink 执行许多采样微步，Simulink 在每个微步中调用 mdlOutputs 和 mdlDerivatives 函数。每次调用输出和求导可以认为是一个积分阶段。积分返回更新过的连续状态和仿真时间，如果状态的积分误差不符合要求，时间会一直延续。最大的时间步长受到事件的限制，比如仿真停止时间和用户设置的限制等。

注意，csfunc.c 指定了输入端口具有直接馈通，这是因为矩阵 D 被初始化为非零阵。如果 D 在状态空间表达式中被设置为 0 矩阵，那么输入信号就不用于 mdlOutputs 中。在这种情况下，直接馈通可以设置为 0，这表明 csfunc.c 在执行输出时不需要输入信号。

该范例 S-function 保存在 *matlabroot*/simulink/src/csfunc.c，源程序省略。

离散状态的 S-Function 范例

matlabroot/simulink/src/dsfunc.c 范例示范了如何在一个 C MEX S-function 中模拟一个带状态的离散系统。该离散系统可用以下方程进行描述：



$$y = f_0(t, x_d, u) \quad （输出）$$

$$x_{d+1} = f_u(t, x_d, u) \quad （更新）$$

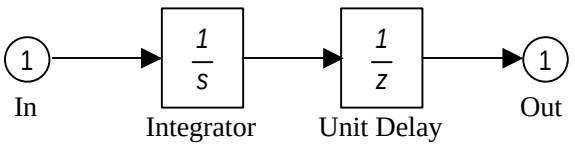
S-function dsfunc.c 满足一个离散的状态空间方程。输出部分的实现放置在 mdlOutputs 中，更新部分的实现放置在 mdlUpdate 中。为了形象化地表述仿真过程，借用第三章中的“ Simulink 如何与 C S-function 相互作用 ”这一节中的流程图（第 55 页），上面的输出方程对应着采样主步长中的 mdlOutputs，更新方程对应着采样主步长中的 mdlUpdate。如果你的模型不包含连续元素，可跳过积分阶段，时间前移到下一个离散采样时刻。

该范例 S-function 保存在 *matlabroot*/simulink/src/dsfunc.c，源程序省略。

混合系统的 S-Function 范例

matlabroot/simulink/src/mixedm.c 是一个混合系统（组合了连续与离散状态）的范例，它组合了 csfunc.c 和 dsfunc.c 中的元素。如果你有一个混合系统，应将你的连续方程放置在 mdlDerivatives 中，离散方程放置在 mdlUpdate 中。另外，你需要对采样点进行检查以确定在什么时刻调用你的 S-function。

按照 Simulink 方块图的形式，该 S-function mixedm.c 类似于以下方块图：



它模拟了一个连续积分器及随后的离散的单位延迟。

因为在仿真终止时不需要执行其它任务，mdlTerminate 是一个空函数。mdlDerivatives 计算连续状态向量 x 的导数，mdlUpdate 中包含了用于更新离散状态向量 x 的方程。

该范例 S-function 保存在 *matlabroot*/simulink/src/ mixedm.c，源程序省略。

变步长的 S-Function 范例

S-function 范例 vsfunc.c 使用了变步长的采样时间。变步长函数需要调用 mdlGetTimeOfNextVarHit，它是一个计算下一步采样时间的 S-function 程序。使用变步长采样时间的 S-function 只能使用变步长的求解器。vsfunc.c 是一个离散的 S-function，它将第一个输入端口的输入进行延时后输出，延迟时间取决于第二个端口的输入。

vsfunc.c 的输出是将输入 u 进行延时。在 mdlOutputs 中设置输出 y 等于状态 x，在 mdlUpdate 中设置状态 x 等于输入向量 u。该范例调用 mdlGetTimeOfNextVarHit 来计算下一步调用 vsfunc.c 的时间。在函数 mdlGetTimeOfNextVarHit 中，宏 ssGetU 用来获取输入 u 的指针，然后调用：

```
ssSetTNext(S, ssGetT(S)(*u[1]));
```

宏 ssGetT 获得仿真时间 t。块的第二输入与当前仿真时间 t 相加，宏 ssSetTNext 将下一步采样时刻设置为 t+(*u[1])，即延迟时间设置为(*u[1])。

该范例 S-function 保存在 *matlabroot*/simulink/src/vsfunc.c，源程序省略。

过零检测的 S-Function 范例

S-function 范例 sfun_zc_sat.c 示范了如何实现一个饱和块。该 S-function 被设计为可使用定步长或变步长求解器。当该 S-function 继承了一个连续的采样时间，并使用变步长求解器时，使用了一个过零检测来定位准确的饱和发生时刻。

该范例 S-function 保存在 *matlabroot*/simulink/src/sfun_zc_sat.c，源程序省略。

时变连续传递函数的 S-Function 范例

stvctf.c 是一个时变连续传递函数的 S-Function 范例，它示范了如何使用求解器维持仿真的连续性，意味着即使用于积分的方程在不断变化，该块仍然对积分器保持平滑和一致的信号。

该范例 S-function 保存在 *matlabroot*/simulink/src/stvctf.c，源程序省略。