第5章 子系统与S函数

规模较大的系统都包含大量的各种模块。如果将所有模块都直接显示在同一个 Simulink 模型窗口，将显得拥挤杂乱，不便于分析调试和仿真建模。为此，Simulink 提供了子系统用于将若干功能模块组合在一起，用一个模块来替代，从而简化模型。

此外，MATLAB 中还提供了 S 函数，以便用户根据需要对 Simulink 模块库进行扩充，实现模块的定制。

子系统和 S 函数是 MATLAB/Simulink 系统建模仿真的高级技术，本章将介绍子系统的基本概念、分类及创建方法，以及 S 函数的概念和编写方法。

微课视频

5.1 子系统的基本概念

子系统(Subsystem)是用单个子系统模块替换的一组模块。利用子系统模块可以创建多层次结构模型，使功能相关的模块集中在一起，有助于减少仿真模型窗口中显示的模块个数。当仿真模型比较大并且复杂时，通过将模块分组为不同的子系统，可以极大地简化仿真模型的布局。

此外，需要仿真的实际系统可能本身就是由若干不同的环节构成，例如一个基本的点到点通信系统可以划分为 3 个环节，即发送端、接收端和信道。在建模仿真时，可以将这 3 个环节分别用一个子系统实现。

5.1.1 子系统的分类

在 Simulink 中，所有子系统分为两大类：虚拟子系统和非虚拟子系统。

1. 虚拟子系统

虚拟子系统(Virtual Subsystem)为仿真模型提供图形层次结构，在仿真模型中用细边框表示。虚拟子系统不影响执行。在执行模型仿真运

行之前，Simulink 首先将子系统进行扩展，类似于 C 语言或 C++语言中的宏。

2. 非虚拟子系统

在仿真模型中，非虚拟子系统（Nonvirtual Subsystem）用粗边框模块表示。执行仿真时，非虚拟子系统被视为一个单元（原子模块）。

在 Simulink 中，非虚拟子系统又分为条件子系统和控制流子系统。

在条件子系统中，子系统的执行取决于输入控制信号。只有当输入控制信号满足指定的条件，例如触发、使能、函数调用、某个操作发生时，才执行子系统。

根据控制信号控制作用的不同，条件子系统又分为如下几种。

（1）使能子系统（Enabled Subsystem）：当控制信号为正时才执行的子系统。当控制信号过零（从负方向到正方向）时启动执行。只要控制信号保持为正，就继续执行。

（2）触发子系统（Triggered Subsystem）：当发生触发事件时执行。触发事件可以发生在触发信号的上升沿或下降沿，可以是连续的，也可以是离散的。

（3）使能触发子系统（Enabled and Triggered Subsystem）：当触发事件发生同时控制信号为正时，触发和启动子系统执行。

控制流子系统（Control Flow Subsystem）是通过控制流模块启动，在当前时间步骤内执行一次或多次的子系统。其中，控制流模块实现控制逻辑，类似于编程语言中的流程控制语句，例如 if-then 语句、switch 语句、while 语句、do-while 语句和 for 语句。根据控制逻辑的不同，又分为动作子系统和循环迭代子系统等。

5.1.2 虚拟子系统的创建和基本操作

Simulink 模型编辑器中提供了多种方法以创建虚拟子系统，创建得到的虚拟子系统还可以重新还原，对子系统还可以进行各种操作等。

1. 子系统的创建

虚拟子系统具体有如下 3 种典型的创建方法。

1）利用 Subsystem 模块创建

在仿真模型中放置一个 Subsystem 模块，该模块位于 Ports & Subsystems 库中。然后双击打开该模块，将构成子系统的模块放置到其中，并相互连接起来。可以根据需要放置若干输入端子（In1）和输出端子（Out1）模块，这两个模块都在 Ports & Subsystems 库中。

2）利用 MULTIPLE 选项卡创建

如果仿真模型已经搭建好，要将其中的某些模块组合为一个子系统，可以利用 Simulink 模型编辑器中的相关按钮创建虚拟子系统。

在仿真模型中选中需要构成子系统的所有模块，在 Simulink 模型编辑器中将出现 MULTIPLE 选项卡，该选项卡中的 Create 按钮组如图 5-1 所示，分别实现虚拟子系统

(Create Subsystem)、原子子系统(Atomic Subsystem)、使能子系统(Enabled Subsystem)、触发子系统(Triggered Subsystem)和函数调用子系统(Function-call Subsystem)的创建。

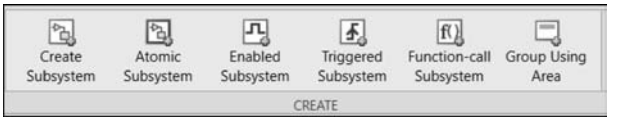


图 5-1 MULTIPLE 选项卡下的 Create 按钮组

为了将选中的模块创建为子系统,只需单击 Create Subsystem 按钮。然后,被选中的所有模块及相互之间的连接信号线将合并为一个名为 Subsystem 的模块。

3) 通过快捷菜单创建

在仿真模型中选中需要构成子系统的所有模块后,右键单击,在弹出的快捷菜单中选择 Create Subsystem from Selection 菜单命令。然后,原来被选中的那些模块将被一个子系统模块替代,并自动添加上必要的输入端子和输出端子(In * 和 Out *),各端子模块及其名称上显示的数字代表端子的编号。

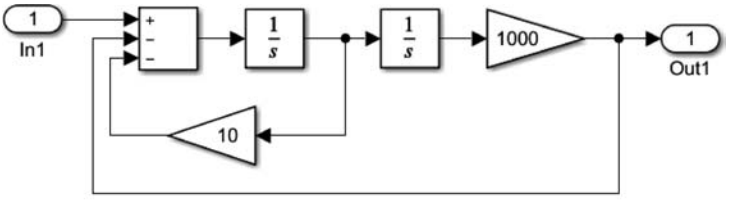
【例 5-1】 已知二阶电路的微分方程为

$$y''(t)+10y'(t)+1000y(t)=1000f(t)$$

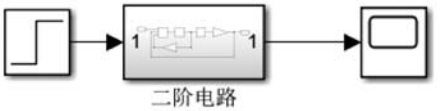
将其创建为一个子系统,并添加相关模块,观察该电路的单位阶跃响应。

方法 1:

(1) 向仿真模型中添加一个 Subsystem 模块,双击打开该模块,在其中搭建如图 5-2(a)所示的模型。注意,双击打开时,子系统内部已经存在 In1 和 Out1 模块,并且这两个模块直接连接,表示子系统默认的功能是将从 In1 模块送入的信号由 Out1 模块直接输出。



(a) 子系统



二阶电路

(b) 上层仿真模型

图 5-2 例 5-1 方法 1 图

(2) 单击编辑区左上角的 Navigate Up to Parent 按钮,返回上层模型窗口。双击 Subsystem 名,将该子系统重新命名为“二阶电路”。

(3) 在上层模型中添加一个 Step 模块和一个 Scope 模块,并分别与前面创建的子系统

的输入端子和输出端子相连接，得到如图 5-2(b)所示的仿真模型。

方法 2：

(1) 搭建如图 5-3 所示的仿真模型。

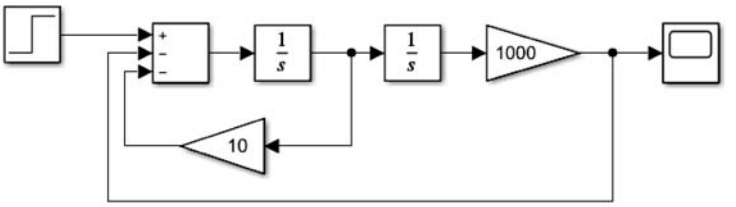


图 5-3　例 5-1 方法 2 图

(2) 按住鼠标左键，拖动选中除 Step 模块和 Scope 模块以外的所有模块。此时，屏幕显示需要创建为子系统的所有模块，如图 5-4 所示。

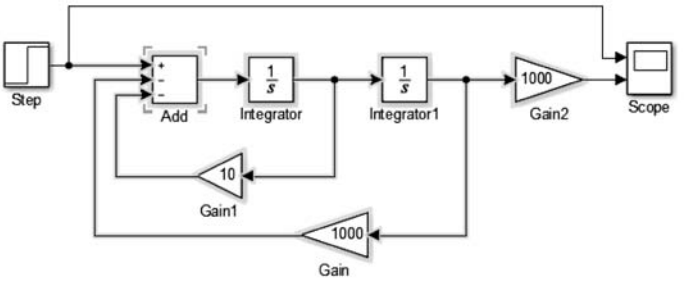


图 5-4　选中多个模块

(3) 右键单击选中的任一模块，在弹出的菜单中选择 Create Subsystem from Selection，即可将所有这些模块创建为子系统。

(4) 在原来的模型中，所有被选中的模块及其连线替换为一个 Subsystem 模块。单击模块名，将其修改为合适的子系统名称。

2. 子系统的展开

将仿真模型中的某些模块创建为子系统后，整个仿真模型成为两级层次结构，即上层模型和下层子系统。在上层模型中，双击 Subsystem 模块，可以进入子系统内部。在子系统内部模型中，单击模型编辑区左上角的 Navigate Back 和 Navigate Up to Parent 按钮，可以返回上层模型。

在上层模型中，单击 Subsystem 模块，将在 Simulink 编辑器窗口顶部出现 SUBSYSTM BLOCK 选项卡。在该选项卡的 COMPONENT 按钮组中，单击 Expand 按钮，则可以将被选中的子系统模块重新展开，使整个仿真模型变为一层结构。例如，在例 5-1 中，选中 Subsystem 模块后，单击 Expand 按钮，则整个仿真模型如图 5-5 所示。

在展开后得到的仿真模型中，原来位于子系统内部的所有模块将用一个阴影方框包围。单击该方框（注意不要选中其中模块），按 Del 键可以将该方框删除。也可以通过鼠标拖动选中这些模块，然后通过 MULTIPLE 选项卡 Create 按钮组中的 Group Using Area 将这些模块重新用阴影框包围。

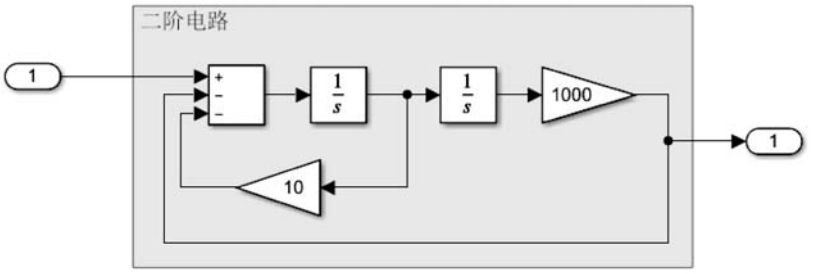


图 5-5　子系统的展开

一个子系统也能够通过复制在仿真模型中被多次使用。具体复制步骤是：右键单击某子系统，在弹出的快捷菜单中选择 Copy 命令，然后在仿真模型中任意空白位置右键单击，再一次弹出快捷菜单，选择 Paste 命令，即可将被选中的子系统进行复制。如果在一个仿真模型中有若干功能相同的模块需要多次使用，通过这种方法可以极大地提高建模效率。

5.1.3　模型浏览器

对于复杂的仿真模型，利用上述方法可以创建多层结构。此时，在搭建、检查、修改和调试仿真模型的过程中，需要逐层打开子系统，这将是非常烦琐的。为此，Simulink 编辑器提供了模型浏览器（Model Browser）。

利用模型浏览器，可以在层次结构模型中实现导航，确定模块在层次结构所处的位置和层次，或者直接打开指定层次的子系统模型。

要打开模型浏览器，可以在 Simulink 模型编辑区左侧的选项板中，单击最下面的 Hide/Show Model Browser（隐藏/打开模型浏览器）按钮。此时，将在选项板左侧显示一个导航栏，导航栏中显示模型的分层视图以及编辑区中的仿真模型在层次结构中所处的层次，如图 5-6 所示。再次单击该按钮，可关闭导航栏。

模型浏览器面板采用树形结构显示模型的层次结构。在图 5-6 中，树形结构的根节点为 smart_braking，这是顶层仿真模型的文件名。根节点下面有 3 个子节点，从上往下依次为 Alert system、Proximity sensor 和 Vehicle，代表该仿真模型中共有 3 个子系统。

单击节点名称左侧的箭头，可以展开该子节点，继续查看该子节点对应子系统的下层模型。例如，图 5-6 中展开了 Vehicle 节点，可以看到下面还有 Brake system、Engine 和 Vehicle Dynamic 三个子节点。

单击某个子节点名称，将在右侧的模型编辑区立即显示该子系统的内部模型。

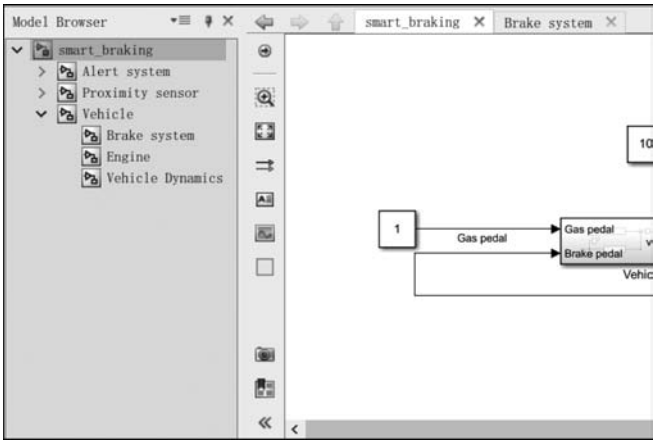


图 5-6　模型浏览器面板

5.2　条件子系统

微课视频

条件子系统属于非虚拟子系统，常用的条件子系统有使能子系统、触发子系统和触发使能子系统，这些子系统都在 Simulink/Ports & Subsystem 库中利用不同的模块实现。

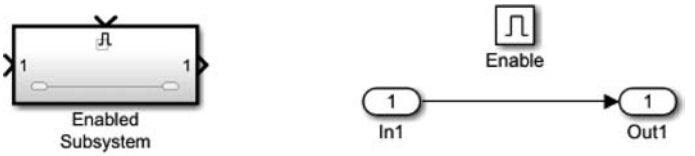
5.2.1　使能子系统

使能子系统是在控制信号为高电平期间才被执行的子系统；当控制信号从高电平翻转为低电平后，将停止子系统的执行。

1. Enabled Subsystem 模块

使能子系统通过 Enabled Subsystem 模块实现。该模块的图标和内部模型如图 5-7 所示。与由 Subsystem 模块创建的虚拟子系统一样，使能子系统内部有一个输入端 In1 和一个输出端 Out1。此外，增加了一个 Enable(使能)模块，对应该子系统模块图标上的使能控制输入端，用于从该端子接收外部送来的使能控制信号。

如果控制信号为低电平，使能子系统将不会执行。在此期间，子系统输出信号的状态可以通过 Out1 模块的参数进行设置。图 5-8 为 Out1 模块参数对话框。



(a) 图标　　(b) 内部模型

图 5-7　Enabled Subsystem 模块

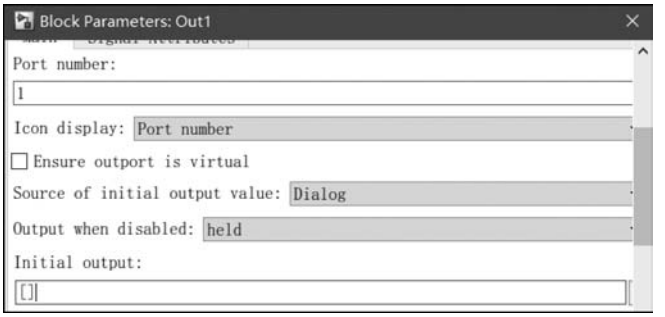


图 5-8　使能子系统内部 Out1 模块的参数对话框

与普通的 Out1 模块相比，使能子系统内部的 Out1 模块增加了如下 3 个参数：

(1) Output when disabled：设置当子系统禁止执行时对应输出端子的输出状态，可以选择 held(保持)或者 reset(复位)。当选择 held 时，当控制信号从高电平变为低电平后，端子的输出状态将保持不变；当选择 reset 时，端子的输出将复位为初始状态。

(2) Initial output：指定端子输出的初始值或端子复位时的状态。

(3) Source of initial output value：指定初始输出值的来源。如果通过下拉列表选择 Dialog(对话框)选项，则指定初始输出值等于 Initial output 参数的设置值；选择 Input signal，则指定从输入信号继承初始输出值。

2. 使能子系统的创建

创建使能子系统，将 Enabled Subsystem 模块从 Ports & Subsystems 库中拖入模型编辑区即可。双击进入该模块对应的子系统内部，即可根据需要修改内部模型，实现仿真模型指定的功能。下面举例说明。

【例 5-2】 创建如图 5-9 所示的仿真模型。在该仿真模型中，有两个使能子系统，内部都采用默认结构，当控制信号有效时，将输入的正弦信号直接输出。

设置模型中 Sine Wave 模块产生 2Hz 正弦波，其他参数取默认值。设置 Pulse Generator 模块产生幅度为 1V、周期为 1s、宽度为 50%的周期方波脉冲。

第一个子系统输出的波形如图 5-10 中的第二个波形所示。在脉冲的每个高电平期间，输出相同时间范围内一个周期的正弦波。在脉冲的低电平期间，输出保持为 0。

第二个子系统的控制信号是由来自于同一个脉冲发生器输出的周期方波经过反向而得到的。因此，在方波为低电平期间，该子系统的控制信号为高电平，子系统输出该时间范围内一个周期的正弦波。

与第一个子系统不同的是，设置第二个子系统内部 Out1 模块的 Output when disabled 参数为 reset，并且设置 Initial output 参数为－1。因此，在方波为高电平期间，第二个子系统禁止执行，其输出端被复位，输出恒定的－1。

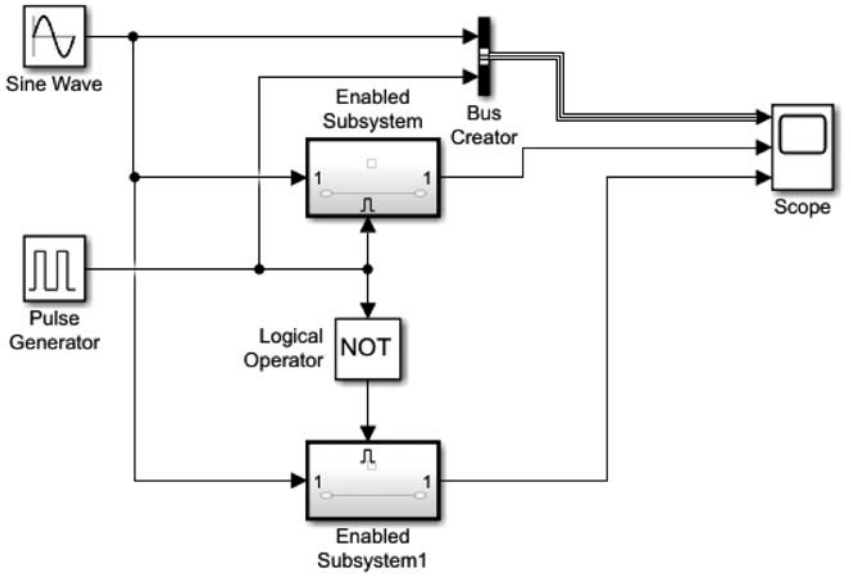


图 5-9　使能子系统仿真模型

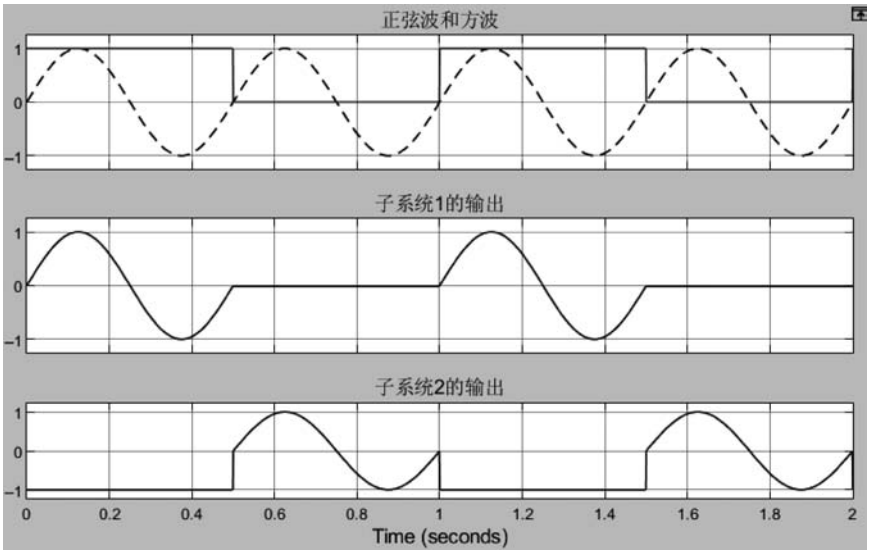


图 5-10　例 5-2 模型的运行结果

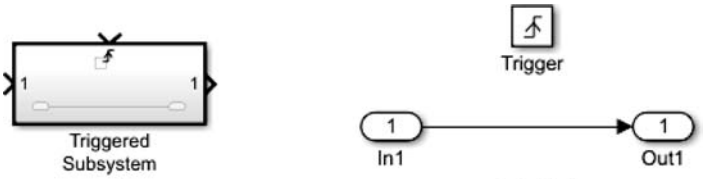
5.2.2　触发子系统

使用触发子系统可以实现微机系统中来自 I/O 硬件的中断和异常或错误处理请求等过程的建模和仿真。与使能子系统类似，触发子系统也有一个控制输入，以决定子系统是否

执行。不同的是，触发子系统是在控制输入的跳变时刻触发执行，在两次触发之间保留前一次触发的输出值。控制输入可以选择是上升沿触发、下降沿触发或者双边沿触发。此外，触发子系统不能在执行时重置模块状态。

1. Triggered Subsystem 模块

触发子系统一般使用 Triggered Subsystem 模块实现，图 5-11 为其图标和内部模型。在内部模型中，默认仍然有一个输入端模块 In1 和一个输出端模块 Out1，分别对应该模块图标上的输入端子和输出端子。此外，内部另外有一个 Trigger(触发)模块，用于接收从模块的控制端子送入的触发控制信号。



(a) 图标　　(b) 内部模型

图 5-11　Triggered Subsystem 模块

图 5-12 是 Trigger 模块的参数对话框。在参数对话框中，通过 Trigger type(触发类型)下拉列表可以选择 rising(上升沿)、falling(下降沿)、either(边沿)或者 function-call(函数调用)触发。在子系统中，Trigger 模块图标上会显示不同的符号，以直观表示触发类型，如图 5-13 所示。

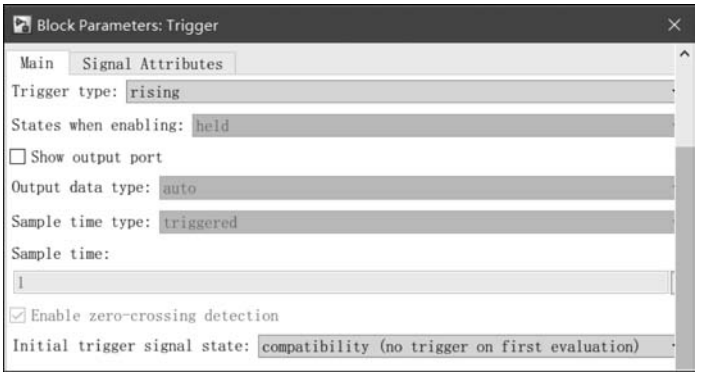


图 5-12　Trigger 模块的参数对话框

(a) 上升沿触发　(b) 下降沿触发　(c) 边沿触发　(d) 函数调用触发

图 5-13　不同触发类型时端子的符号

对于上升沿、下降沿和边沿触发，模块的参数 States when enabling(触发时的状态)固定设置为 held(保持)。因此，在触发时刻，模块的状态将保持为当前值。如果选择触发类型为 function-call，则 States when enabling 参数还可以设为 reset 或 inherit，分别表示重置模块状态值和继承函数调用上层模型的 held 或 reset 设置。

参数 Sample time 用于指定 Trigger 模块所在子系统的时间间隔。如果子系统的实际调用速率与此参数指定的时间间隔不同，Simulink 将报错。当触发类型设置为上升沿、下降沿和边沿触发时，该参数固定设置为 1。

2. 触发子系统的创建

创建触发子系统的基本方法是向仿真模型中添加一个触发子系统模块 Triggered Subsystem。然后，双击打开该模块，根据需要设置其中 Trigger 模块和 In1、Out1 模块的参数，并添加实现子系统功能的其他模块。

此外，也可以利用 Ports & Subsystem 库中的 Trigger 模块创建触发子系统。该模块与 Triggered Subsystem 模块子系统内部的 Trigger 模块功能和参数设置完全相同。利用该模块创建触发子系统时，首先利用 Subsystem 模块创建虚拟子系统，然后向其中添加 Trigger 模块。此时，在原来的虚拟子系统图标上将出现一个触发符号，并且自动添加一个触发输入端子，从而成为一个触发子系统。

【例 5-3】 搭建如图 5-14 所示的触发子系统仿真模型，运行观察波形。

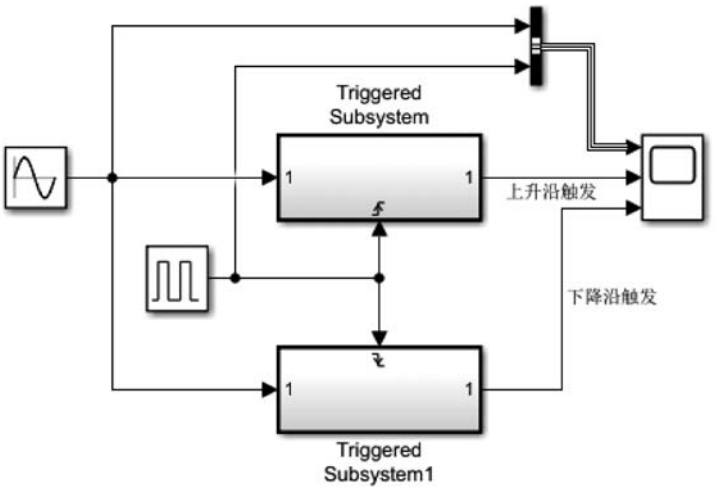


图 5-14 触发子系统仿真模型

模型中，两个触发子系统的触发控制信号分别设置为上升沿和下降沿。Pulse Generator 模块产生频率为 1Hz 的方波脉冲，作为两个触发子系统的触发控制信号。

运行后，示波器上的波形如图 5-15 所示。在方波脉冲的每个上升沿和下降沿时刻，两个子系统分别输出当前正弦信号的幅度，并且每个触发时刻的输出都保持到下一个触发时

刻。读者可以将该图与图 5-10 所示的使能子系统的输出波形进行对比，以便清楚地理解触发子系统和使能子系统的区别。

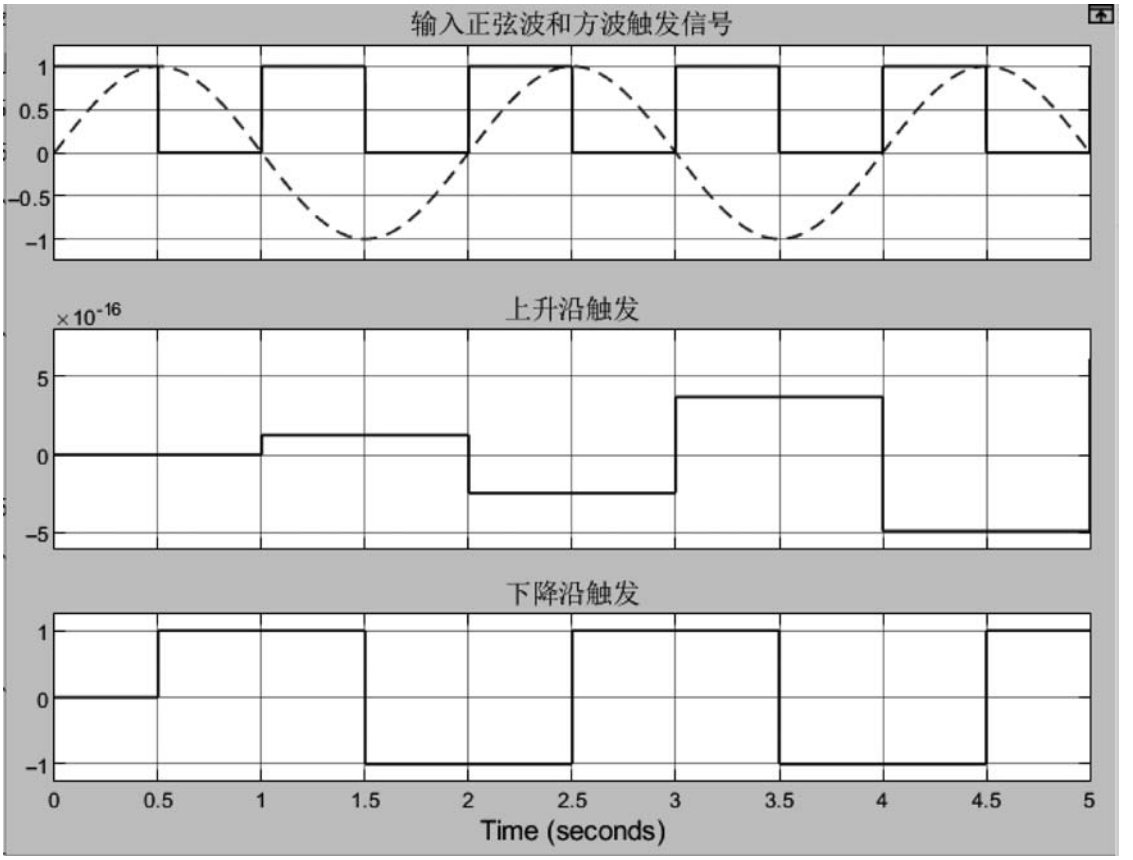


图 5-15　例 5-3 运行结果 1

在该例中，如果设置触发控制方波信号的频率足够高，例如频率提高到 100Hz，此时的运行结果如图 5-16 所示。由此可见，该仿真模型可以实现零阶保持器的功能，从而对实际系统中的采样过程进行建模仿真。

5.2.3　使能触发子系统

使能触发子系统是上述两种条件子系统的综合，用 Enabled and Triggered Subsystem 模块实现。该模块的图标及子系统内部模型如图 5-17 所示，其中同时有 Trigger 模块和 Enable 模块，这两个模块的参数设置与上述两种子系统相同。当使能控制输入端输入的使能控制信号保持为高电平时，在触发输入控制信号的每个上升沿或者下降沿，子系统将当前时刻的输入由输出端子输出。相邻两个触发时刻之间，子系统输出保持不变。在使能控制

信号为低电平期间，不管触发输入信号是否有跳变，子系统的输出都保持不变或者复位。

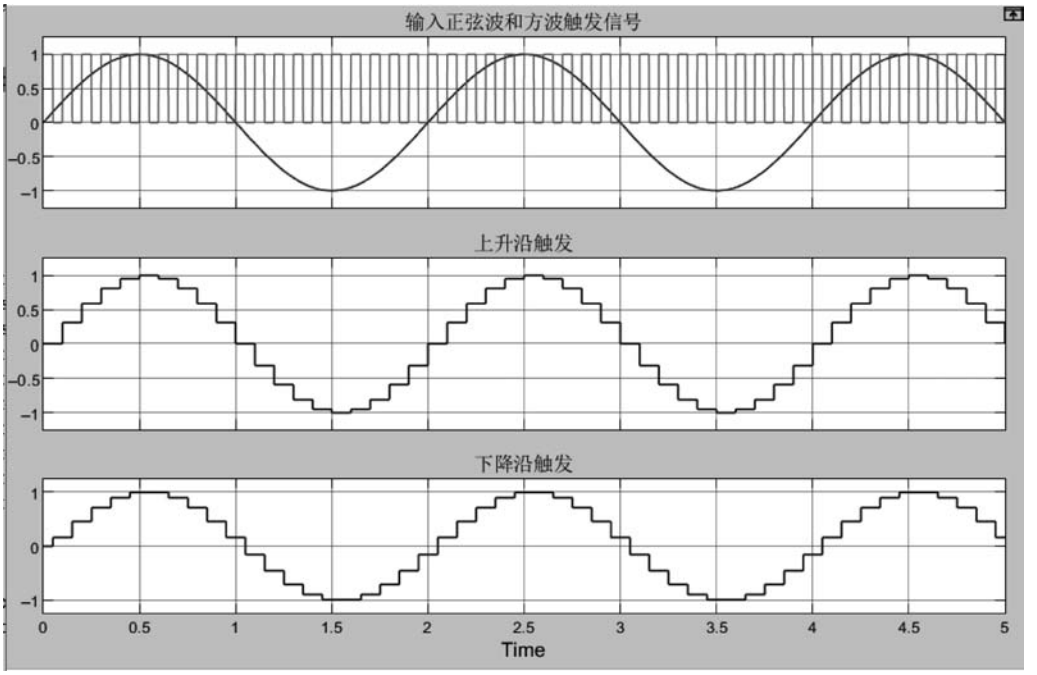
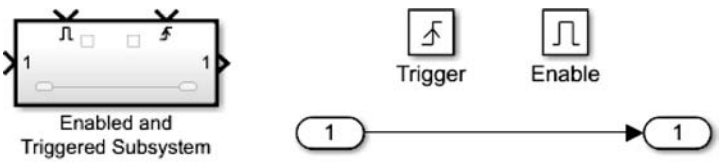


图 5-16　例 5-3 运行结果 2



(a) 图标　　(b) 内部模型

图 5-17　Enabled and Triggered Subsystem 模块

下面举例说明该模块的用法。

【例 5-4】　搭建如图 5-18 所示的使能触发子系统仿真模型，运行后观察波形。

在图 5-18 所示的顶层模型中，两个脉冲发生器 Pulse Generator 和 Pulse Generator1 模块产生的周期方波脉冲分别作为子系统模块 Enabled and Triggered Subsystem 的使能输入和触发输入。设置两个脉冲发生器模块的 Period 参数分别为 1s 和 0.1s。

使能触发子系统内部采用如图 5-17(b)所示的默认结构，其中 Out1 模块的参数 Output when disabled 设置为 reset，Initial condition 参数设置为 0。仿真运行后，示波器上显示的波形如图 5-19 所示。在使能控制信号为高电平期间，每个触发控制信号的上升沿输出正弦波当前时刻的幅度，并保持到下一个触发时刻。在使能控制信号为低电平期间，子系统的输出复位为 0，并保持到使能控制信号重新变为高电平。

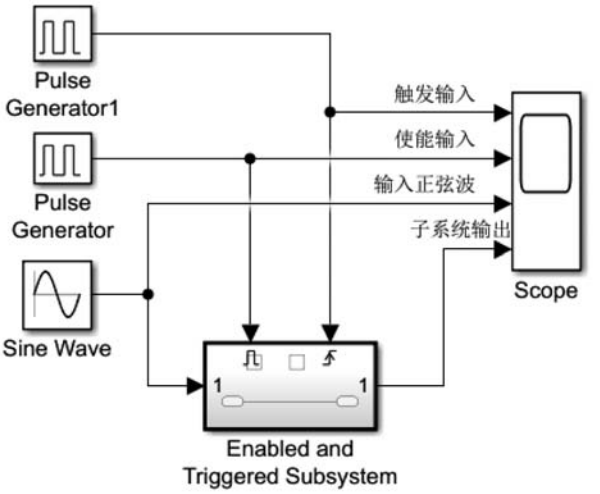


图 5-18　使能触发子系统仿真模型

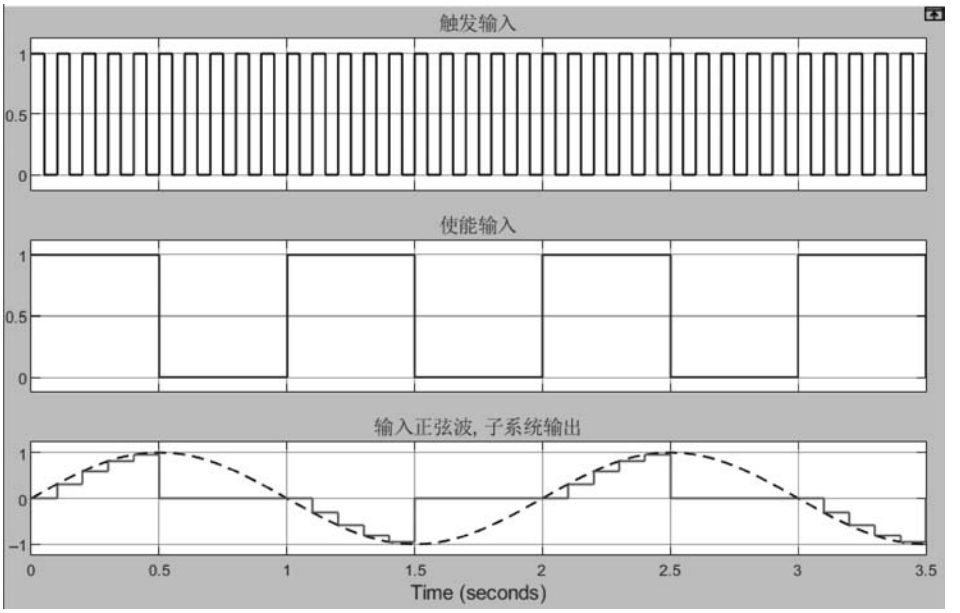


图 5-19　例 5-4 运行结果

微课视频

5.3　控制流子系统

控制流子系统(Control Flow Subsystem)是另外一种典型的非虚拟子系统。这种子系统通过专门的控制流模块启动,在当前时间步骤执行一次或多次。其中,控制流模块可以是

If、Switch Case、While Iterator 或 For Iterator 模块，用于实现执行流程的控制，类似于编程语言中的 if-then、switch-case、while-do 和 for 流程控制语句。

在控制流子系统中，用 If Action Subsystem 和 Switch Case Action Subsystem 模块实现的控制流子系统又称为动作子系统（Action Subsystem），含有 While Iterator 的 While Iterator Subsystem 和含有 For Iterator 的 For Iterator Subsystem 又统称为循环迭代子系统。

5.3.1 动作子系统

动作子系统包括 If 动作子系统和 Switch 动作子系统，分别类似于 MATLAB 和 C 语言编程中的 if-then-else 和 switch-case 语句。

1. If 动作子系统

一个典型的 If 动作子系统结构如图 5-20 所示。图中，If 模块的输入 u_1 决定该模块两个输出端子的状态，每个输出端子分别接到不同的 If Action Subsystem 模块。当 If 模块某个端子的条件成立时，该端子输出触发信号，触发执行对应的子系统。两个子系统的输出通过 Merge 模块合并为一路，由 Out1 端子输出。

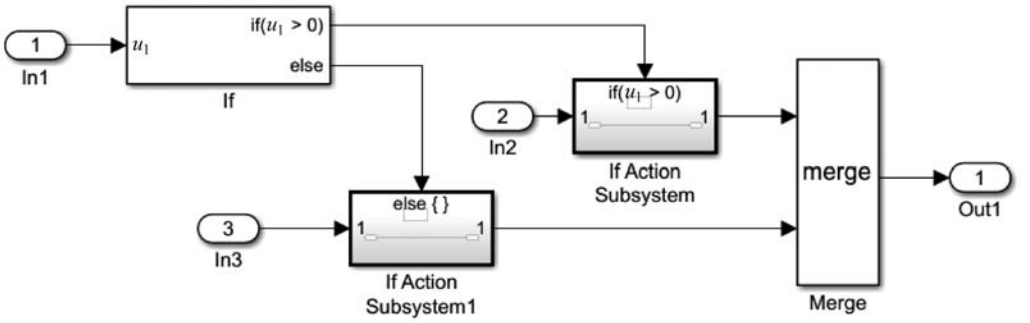


图 5-20 If 动作子系统的结构示意图

上述过程可以用如下程序流程表示：

```
if (u1 > 0) {
    If Action Subsystem 1;
}
else {
    If Action Subsystem 2;
}
```

1）If 模块及其参数设置

在 If 动作子系统中，If 模块起控制作用。该模块的参数对话框如图 5-21 所示，不同的参数设置将决定 If 模块图标的显示。

(1) Number of inputs：设置 If 模块输入信号的个数，这些信号用于确定 if-else 流程中的条件。

(2) If expression：设置 if 条件的表达式，此表达式将显示在 If 模块图标中 if 输出端口的旁边。表达式只能包含关系运算符，不允许使用算术运算符，表达式中不能包含数据类型表达式，也不能引用除 double 和 single 以外其他数据类型的工作区变量。

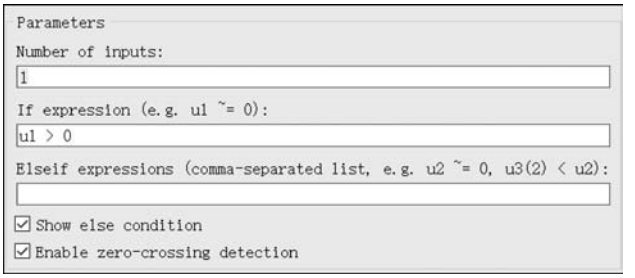


图 5-21　If 模块的参数对话框

(3) Elseif expressions：设置 if-else 语句中的 elseif 条件表达式，可以用逗号分隔列出多个条件。这些表达式将在模块上出现对应的若干个输出端，各输出端都命名为“elseif(条件表达式)”。

(4) Show else condition：控制是否在 If 模块图标上显示 else 输出端。

2) If 动作子系统的创建

为了创建 If 动作子系统，将 If 模块添加到当前仿真模型中，并根据需要设置其参数。同时，分别创建各 If Action Subsystem，向其中添加实现具体功能的模块。然后，将各子系统的 Action Port 输入端子与 If 模块上相应的 if、else 或 elseif 输出端相连接。注意，启动仿真运行后，这些连接线将变为虚线。

【例 5-5】 搭建如图 5-22 所示的 If 动作子系统仿真模型。

模型中，信号发生器(Signal Generator)模块输出频率为 10Hz、幅度为 1V 的锯齿波(Sawtooth)，作为 If 模块的 u_1。

If 模块的参数设置如图 5-23 所示。根据图中的设置，模块图标上显示一个输入端和 3 个输出端。3 个输出端分别作为 If Action Subsystem、If Action Subsystem1 和 If Action Subsystem2 模块的控制输入。

3 个 If 动作子系统的内部结构如图 5-24 所示。在 If Action Subsystem 和 If Action Subsystem1 内部，将原来的 In1 模块替换为 Constant 模块，分别设置其 Constant value 参数为 1 和－1。If Action Subsystem2 的内部结构保持不变，即 In1 模块直接与 Out1 模块相连接。在仿真模型中，Sine Wave 模块产生 2Hz 的正弦波，送入该子系统。

此外，3 个子系统内部 Out1 模块的参数 Output when disabled 都设置为 reset，参数 Initial condition 都设置为 0，则当子系统没有触发执行时，通过 Out1 端子输出为 0。3 个子系统的输出最后通过 Add 模块相加，得到总的输出，送入示波器。

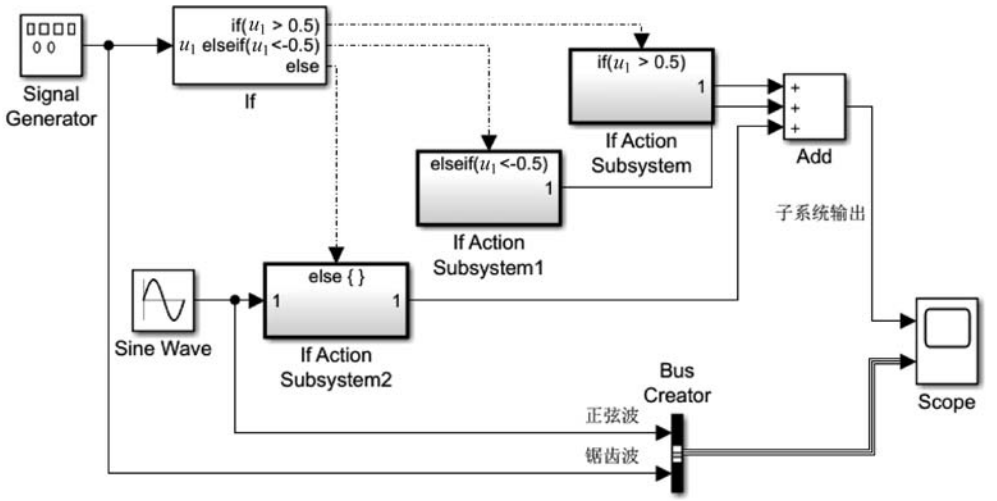


图 5-22　例 5-5 仿真模型

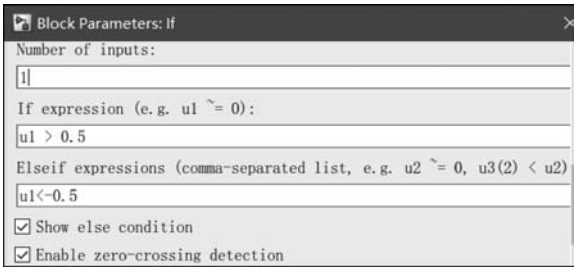


图 5-23　If 模块参数设置

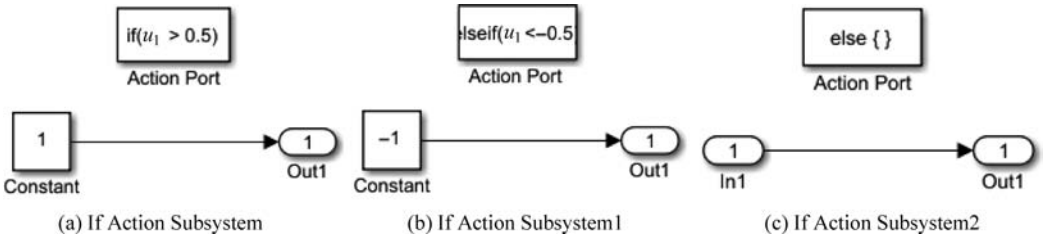


图 5-24　If 动作子系统内部结构

运行后示波器上的时间波形如图 5-25 所示。在 0～0.15s 期间，锯齿波幅度大于 0.5，因此子系统输出 1V；在 0.47～0.63s 期间，锯齿波幅度小于－0.5，因此子系统输出－1；在 0.15～0.47s 期间，锯齿波幅度位于－0.5～0.5，因此输出该时间范围内的正弦波。

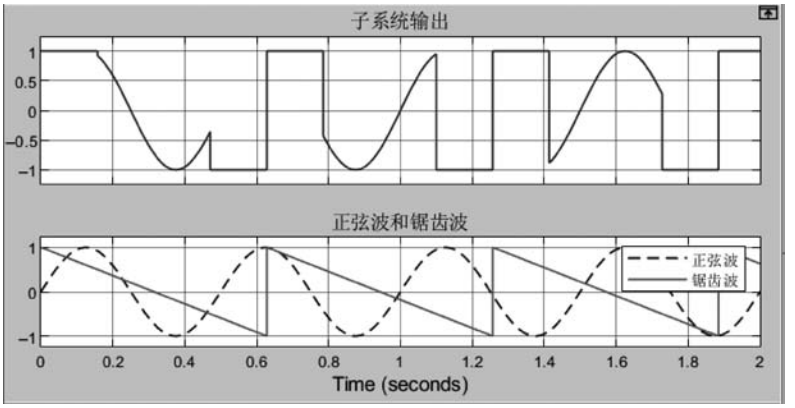


图 5-25　例 5-5 运行结果

2. Switch 动作子系统

Switch 动作子系统用 Switch Case 模块和 Switch Case Action Subsystem 模块实现，其连接示意图如图 5-26 所示。

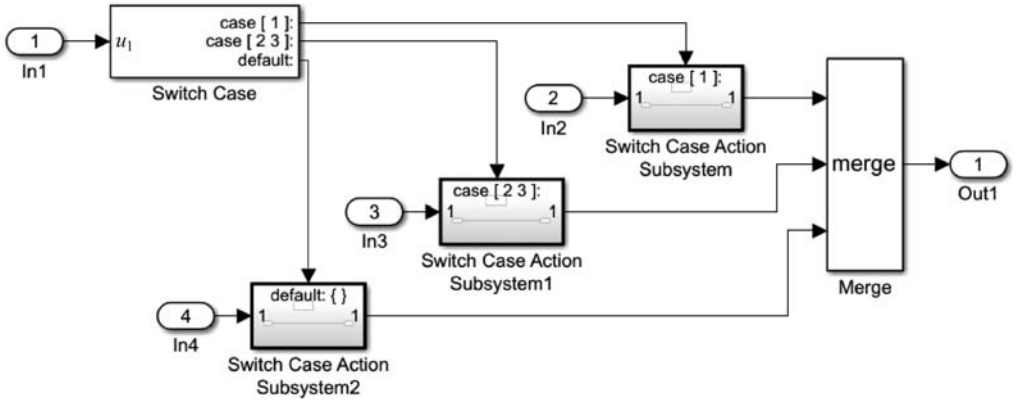


图 5-26　Switch 动作子系统的结构示意图

图 5-26 中，Switch Case 模块只有一个输入端子，根据输入 u_1 的取值决定其 3 个输出端子的状态，每个输出端子分别接到不同的 Switch Case Action Subsystem 模块。当 Switch Case 模块某个端子的条件成立时，该端子输出触发信号，触发执行对应的子系统。

下面举例说明 Switch 动作系统的创建和使用方法。

【例 5-6】 搭建如图 5-27 所示的 Switch 动作子系统仿真模型。

模型中，Random Integer Generator（随机整数发生器）模块位于 Communications Toolbox/Comm Sources/Random Data Sources 库中，设置其 Set size 参数为 4，Sample time 参数为 0.5，其他参数取默认值。根据这些参数设置，该模块产生 0～3 的均匀分布的随机整数，输出的每个整数（又称为代码或码元）保持 0.5s 的时间。

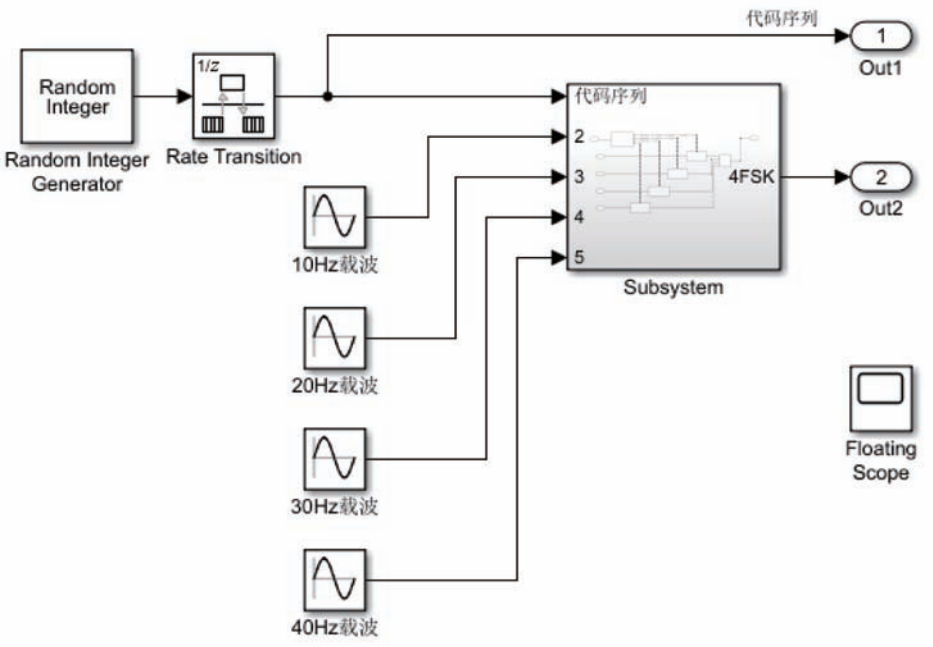


图 5-27　例 5-6 仿真模型

模型中的 Rate Transition(速率转换器)位于 Simulink/Signal Attributes 库中,设置其 Output port sample time 参数为 1e-3,则将 Random Integer Generator 模块输出整数序列的采样速率提高到 1kHz,再输出到后面的子系统。4 个 Sine Wave 模块分别产生频率为 10Hz、20Hz、30Hz 和 40Hz 的正弦波,也送入子系统。

模型中的 Subsystem 模块为虚拟子系统,其内部结构如图 5-28 所示,其中共有 5 个输入端子模块。从 In1 端子输入的整数代码序列送入 Switch Case 模块,该模块的参数设置如图 5-29 所示。当输入整数为 0、1 和 2 时,模块相应的端子输出有效的触发控制信号,控制执行相连接的 Switch Case Action Subsystem。当输入整数为 3 时,default 端子输出有效的控制信号,控制执行 Switch Case Action Subsystem3。最后,4 个子系统的输出通过 Merge 模块合并为一路,由 Out1 端子输出。

在图 5-27 中,Rate Transition 模块输出的代码和由虚拟子系统 Subsystem 输出的信号分别由两个 Out1 端子送出。注意到该模型中还用到了一个浮动示波器 Floating Scope 模块,以显示两个信号波形。双击打开该模块,在打开的浮动示波器窗口中,按照第 3 章介绍的方法对其属性、布局和显示样式等进行设置,并选择需要送到示波器显示的信号。

运行后在浮动示波器中得到信号波形如图 5-30 所示。在运行结果波形中,随机整数发生器以 0.5s 的间隔依次输出代码序列 00231……整个仿真模型输出的 4FSK 波形对应每个不同的代码,输出不同频率的正弦波。这就是通信中称为 4FSK 的调频信号,因此该仿真模型实现了 4FSK 调制。

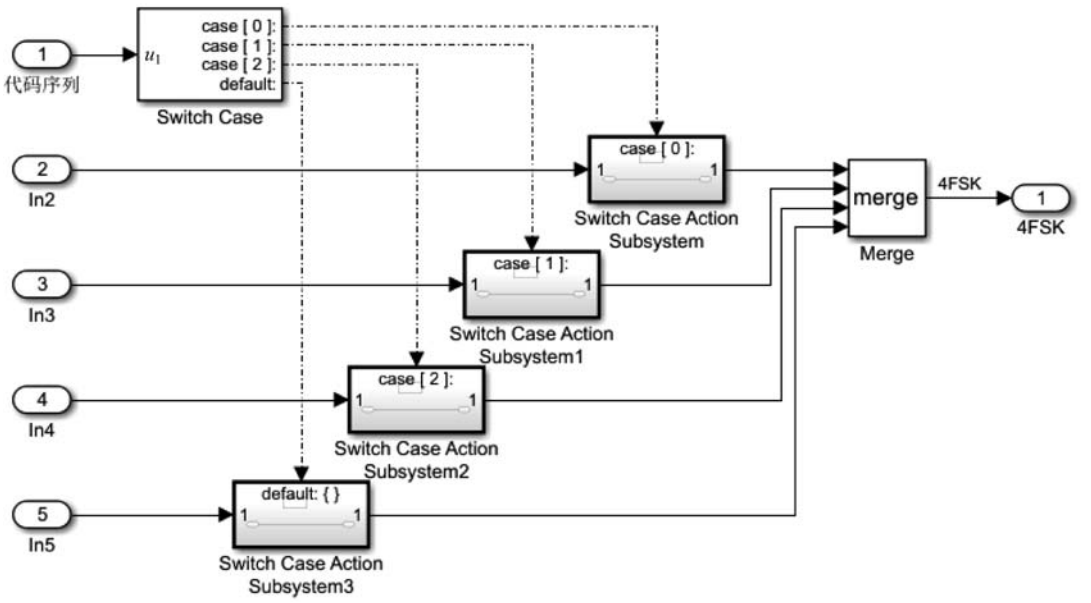


图 5-28　图 5-27 中的 Subsystem 虚拟子系统

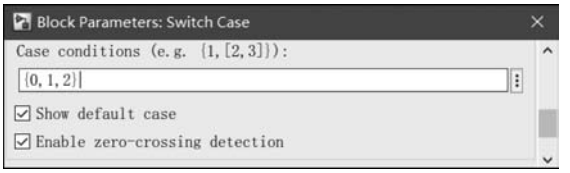


图 5-29　Switch Case 模块参数设置

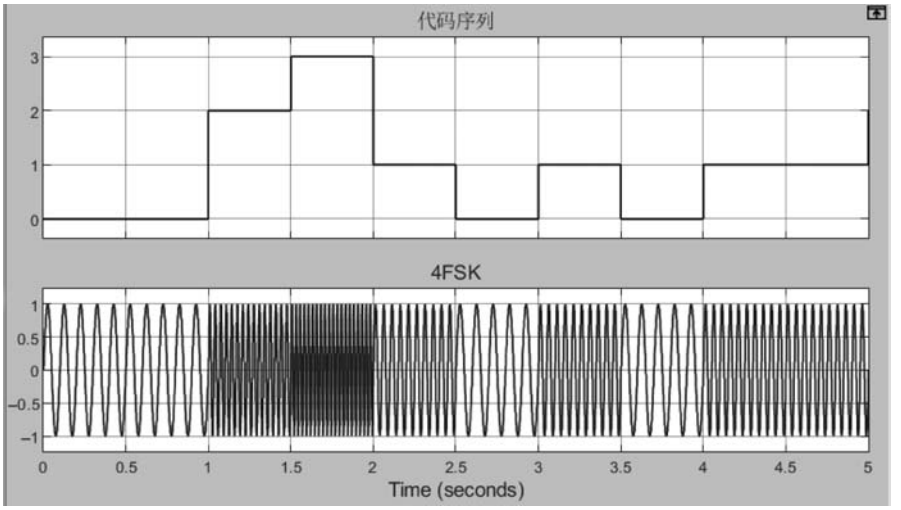


图 5-30　例 5-6 运行结果

5.3.2 循环迭代子系统

循环迭代子系统在每个仿真步内重复执行指定的次数，主要包括 For 循环迭代子系统(For Iterator Subsystem)和 While 循环迭代子系统(While Iterator Subsystem)。两种子系统内部结构如图 5-31 所示。

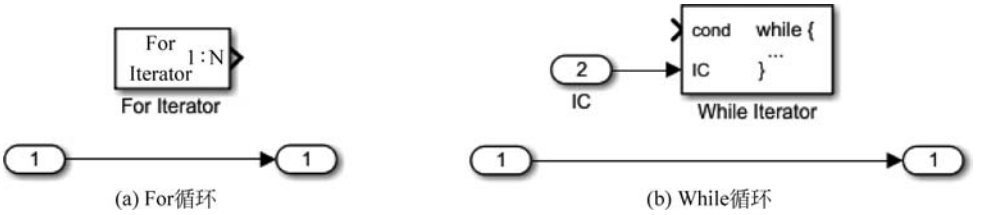


图 5-31 循环迭代子系统内部结构

与前面的各种子系统类似，循环迭代子系统默认都有一个输入端子 In1 和一个输出端子 Out1，实现子系统具体功能的模块放在输入/输出端子之间。不同的是，循环迭代子系统内部分别还有一个 For Iterator(For 迭代器)或 While Iterator(While 迭代器)模块，对子系统循环执行的次数进行控制。

1. For 循环迭代子系统

在 For 循环迭代子系统中，用 For 迭代器控制在当前时间步内重复执行子系统，直到迭代变量超过指定的迭代次数限制。这一控制过程类似于程序中的 for 循环。

1) For 迭代器模块

默认情况下，For 迭代器只有一个输出端子，该端子在运行过程中将迭代器变量(计数变量)的值在每次循环时输出。

图 5-32 为 For 迭代器的参数对话框。如果设置 Iteration limit source(迭代限制值的来源)参数为 external(外部)，则模块上将出现一个输入端子，通过该端子由外部模块设置循环迭代次数。如果设置该参数为 internal(内部)，则由对话框中的 Iteration limit(迭代限制值)参数设置循环次数。

在参数对话框中，如果不勾选 Show iteration variable(显示循环计数变量)选项，则迭代器图标上不显示输出端子。如果在实现子系统功能时需要用到计数变量，可以勾选该选项，则模块图标上将出现一个输出端子，可以将运行过程中的计数变量从输出端子输出。

此外，还可以设置模块的 States when starting 参数为 held 或者 reset，以确定在相邻的两个仿真运行时间步之间子系统的状态。如果选择 held，则在相邻两个时间步之间子系统的状态保持不变；如果选择 reset，则在每个时间步的开始，将子系统状态重置为其初始值。

2) For 循环迭代子系统的使用

下面举例说明 For 循环迭代子系统的创建和使用方法。

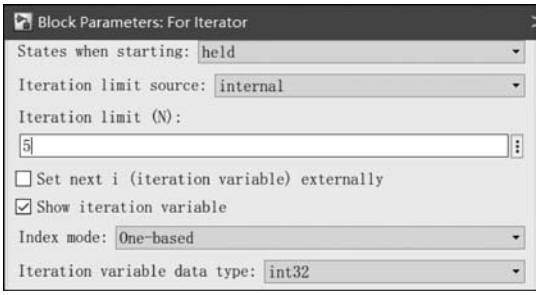
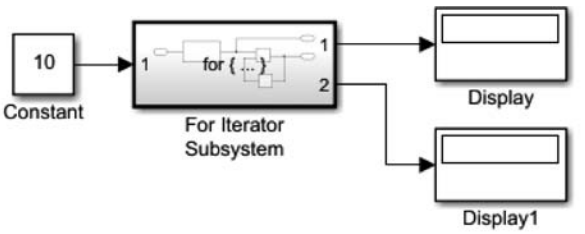


图 5-32　For 迭代器的参数设置

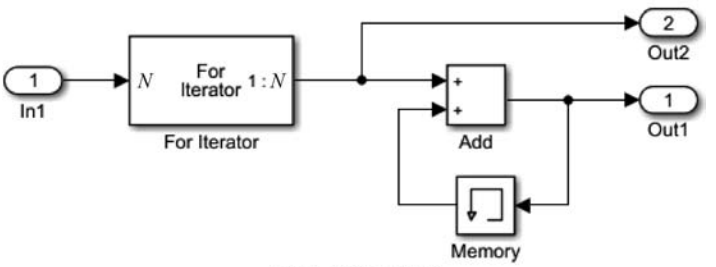
【例 5-7】 搭建如图 5-33 所示的仿真模型。

图 5-33(a)为顶层仿真模型。其中,Constant 模块产生常数 10,通过端子 1 送入 For Iterator Subsystem 模块。子系统的两个输出分别送到两个 Display(数值显示器)模块。

在图 5-33(b)所示的 For 循环迭代子系统中,除了子系统内部默认的一个输出端子 Out1 以外,另外增添了一个输出端子 Out2。



(a) 顶层模型



(b) For迭代子系统

图 5-33　例 5-7 仿真模型

设置迭代器模块 For Iterator 的 States when starting 参数为 reset,则在仿真运行的每个时间步开始,将迭代器的循环计数变量复位为零。设置迭代器的 Iteration limit source 参数为 external(外部),并将模块的输入端子与子系统的 In1 端子相连接,则顶层模型中 Constant 模块输出的常数 10 由该端子送入迭代器,作为循环迭代次数。

迭代器的计数变量值一方面由 Out2 端子送出子系统,并送到顶层模型中的 Display1 模块显示。另一方面,计数变量值还送到 Add(加法器)模块。模型中的 Memory(存储)模

块将其输入保持并延迟一个迭代循环。因此，加法器和存储模块实现循环迭代计数变量值的累加，累加结果由 Out1 端子输出，并送到 Display 模块显示。

根据仿真模型中的设置，该例中 For 循环迭代子系统在每个仿真时间步共循环执行 10 次。因此仿真运行后，计数变量值等于 10。在每个仿真时间步，迭代器的计数变量值都从 1 开始，递增到 10 结束。因此 Display1 模块上显示的计数变量值为 10。每次循环，将计数值进行累加。循环 10 次后，累加结果为 1＋2＋…＋10＝55，因此仿真模型中 Display 模块上显示结果为 55。

3) 仿真时间步与循环迭代步

在例 5-7 中，求解器设置为默认的变步长求解器，仿真时间步取为默认的 auto。在每个仿真时间步(称为主时间步)内，For 循环迭代子系统共循环执行 10 次，每次执行称为一个循环迭代步。由此可见，一个主时间步内会有很多循环迭代步。

为了进一步体会主时间步和循环迭代步的关系，下面举例说明。

【例 5-8】 搭建如图 5-34 所示的仿真模型。

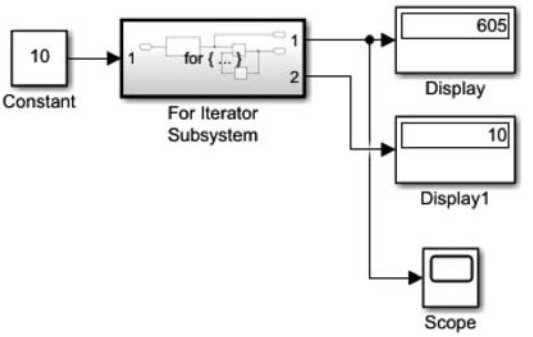


图 5-34　例 5-8 仿真模型

与例 5-7 相比，该例的仿真模型中只是增加了一个示波器模块，用于显示仿真运行的每个主时间步的累加和。此外，For Iterator Subsystem 内部结构与图 5-33(b)相同。

在本例中，将迭代器 For Iterator 模块的 States when starting 参数重新设置为 held，则在仿真运行的每个主时间步开始，For 循环迭代子系统的输出值将保持为上一个主时间步结束时的值。例如，在第一个主时间步结束时，子系统输出累加结果 55。在第二个主时间步开始，子系统输出保持为 55，而计数变量的值从 1 开始重新递增计数。每个循环迭代步，将计数值 1～10 重新累加到该主时间步开始的累加和中。因此，到第二个时间步结束时，计数变量的值变为 55＋(1＋2＋…＋10)＝110。

设置求解器为固定步长求解器，步长为 0.5s，仿真运行 5s。运行后示波器上显示的波形如图 5-35 所示。图中的每个小圈代表对应主时间步内的累加结果。根据步长和仿真运行时间可知，主时间步共执行了 11 次。在每个主时间步内，累加结果增加 55。因此，仿真运行结束后，累加结果达到 55×11＝605，该结果同时显示在仿真模型中的 Display 模块上。

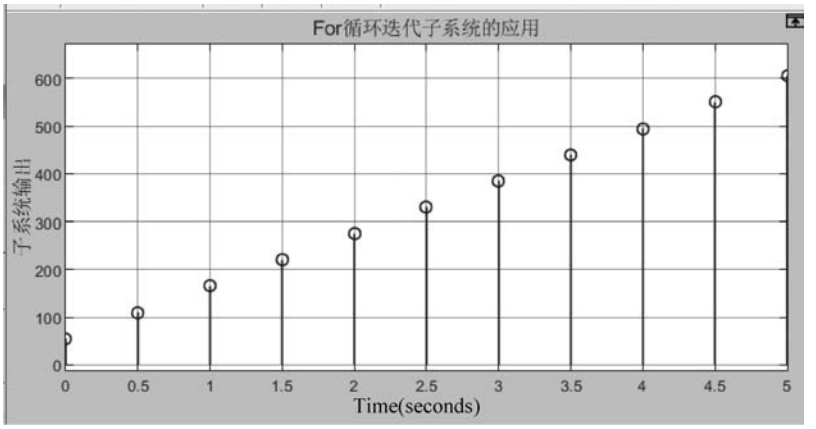


图 5-35　例 5-8 运行结果

注意在运行结束后，Display1 模块上仍然显示 10，代表在每个主时间步内，For 迭代器中的计数变量从 1 递增到 10。但是，在循环迭代步中，计数变量值的递增变化是无法用示波器等模块观察到的。

2. While 循环迭代子系统

在 While 循环迭代子系统中，用 While 迭代器模块控制输入条件的取值为 true 或 1 时，才在当前时间步中重复执行 While 循环迭代子系统。这一控制过程类似于程序中的 while 循环和 do while 循环。

1) While 迭代器

默认情况下，While 迭代器有两个输入端子，没有输出端子。其中，cond(条件)端子用于输入逻辑条件信号，逻辑条件信号的数据类型和取值可以是逻辑值 true(1)和 false(0)，也可以是数值，其中任何非零的整数或者负数表示 true，而 0 表示 false。由于子系统在主时间步内不是通过外部触发的，因此设置逻辑条件的模块必须位于子系统内。

IC(初始条件)端子用于设置 While 循环的初始逻辑条件。在每个时间步的开始，如果 IC 端子输入 false 或者 0，则子系统在时间步内不执行；如果由 IC 输入 true 或者非零值，则子系统开始执行，并且只要 cond 信号为 true 就继续重复执行。

图 5-36 为 While 迭代器的参数对话框，其中设置的参数主要有

(1) Maximum number of iterations：指定在一个时间步内允许的最大迭代次数。该参数可以设为任意的整数，默认设为－1，表示只要 cond 信号为 true，则允许任意迭代次数。

(2) Show iteration number port：设置是否在迭代器图标上显示迭代次数输出端子。如果勾选该选项，则可以通过该输出端子输出循环次数计数值，计数值从 1 开始，每次循环递增 1。

(3) While loop type：可以选择 while 或者 do-while，默认为 while。

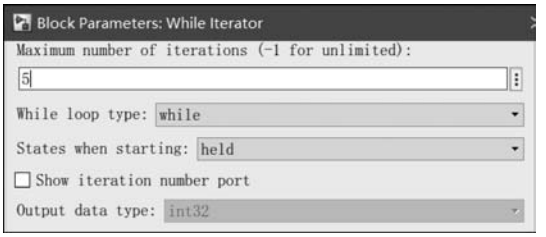


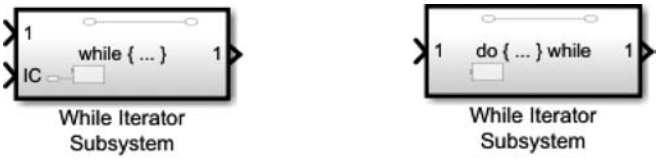
图 5-36　While 迭代器的参数设置

2）While 循环的两种方式

与其他高级语言类似，While 循环迭代子系统可以实现 While 循环和 Do-while 循环，两种循环迭代方式由 While 迭代器的 While loop type 参数指定。

（1）While 循环

在 While 循环方式中，While 迭代器模块有两个输入端，即 cond 和 IC，其中产生 cond 逻辑条件的模块必须位于子系统内部，而产生 IC 信号的信源必须在 While Iterator Subsystem 模块的外部。因此，对应的 While Iterator Subsystem 模块有两个输入端和一个输出端，其图标如图 5-37(a)所示。



(a) While循环方式　　(b) Do-while循环方式

图 5-37　While Iterator Subsystem 模块的图标

在每个主时间步的开始，如果 IC 输入为 true，则在 cond 输入为 true 时重复执行子系统。只要 cond 输入为 true 且迭代次数小于或等于 Maximum number of iterations 参数指定的最大迭代次数，此过程就会在时间步内继续执行；否则停止子系统的执行。如果通过 IC 端子输入为 false，则也将停止执行子系统。

（2）Do-while 循环

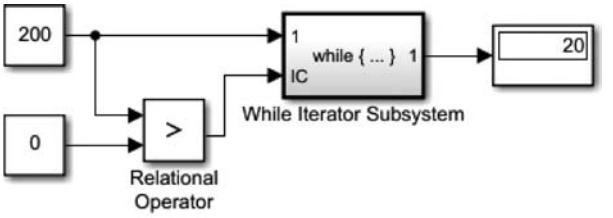
在这种循环方式中，While 迭代器模块只需要一个输入 cond，不需要 IC 端子设置 While 循环的初始逻辑条件。因此，在设置 While 迭代器的 While loop type 为 do-while 时，要将迭代器内部原来的 IC 端子删除。此时，对应的 While Iterator Subsystem 模块只有一个输入端和一个输出端，其图标如图 5-37(b)所示。

在每个主时间步中，当 cond 输入为 true 时，则重复执行子系统。只要 cond 输入为 true，且迭代次数小于或等于 Maximum number of iterations 设置的最大迭代次数，此过程就会一直继续执行，直到 cond 输入变为 false，或者迭代次数超过最大迭代次数。

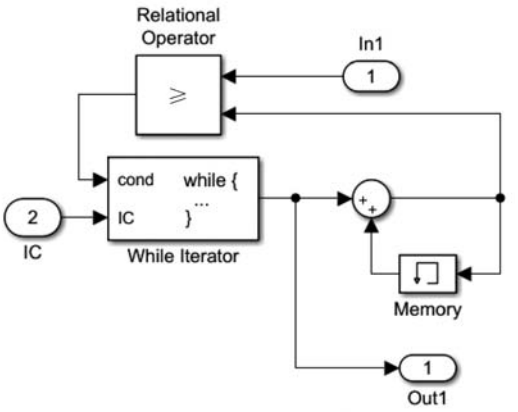
3）While 循环迭代子系统的使用

下面举例说明 While 和 Do-while 循环迭代子系统的创建和使用方法。

【例 5-9】 搭建如图 5-38 所示的仿真模型。



(a) 顶层模型



(b) While迭代子系统

图 5-38 例 5-9 仿真模型

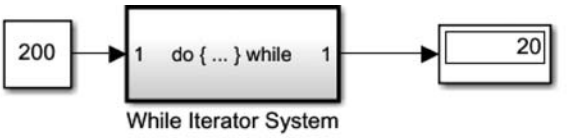
图 5-38(a)为顶层仿真模型，其中 Constant 模块产生常数 200，通过端子 1 送入 While Iterator Subsystem。同时，用 Relational Operator 模块将常数 200 和 0 进行比较，输出逻辑值 true，作为 While 迭代子系统的初始条件，由 IC 端子送入子系统。子系统的输出送到 Display 模块，以显示运行结果。

在图 5-38(b)所示的 While 迭代子系统中，勾选了 While Iterator 模块的 Show iteration number port 参数，因此模块有一个输出端。通过该输出端输出的计数变量值由加法器和存储模块进行累加。累加结果与由 In1 端子输入的常数 200 进行比较。如果累加结果未超过 200，则 Relational Opertor 模块输出 true；否则输出 false。

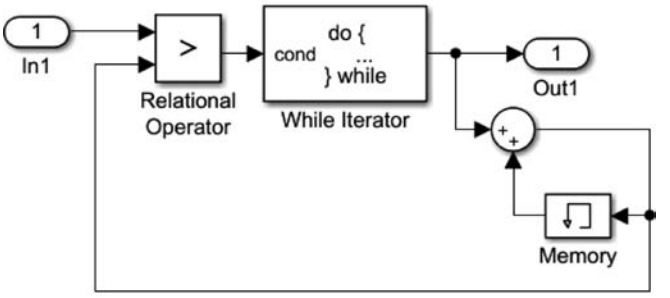
迭代器模块的 IC 端与子系统的 IC 输入端直接相连，因此顶层模型中的 Relational Opertor 模块输出恒定的逻辑值 true 输入迭代器，作为迭代器的初始条件。由于该条件在仿真运行过程中恒为 true，因此迭代器控制子系统重复执行内部的累加运算。当累加结果超过 200 时，迭代器的 cond 端子输入 false，此时停止循环迭代和子系统的执行。

根据上述分析，该例中的仿真模型实现了从 1 开始连续整数的累加运算，直到累加结果超过 200 时，停止累加。在重复执行子系统的过程中，计数变量的值不断从迭代器的输出端子输出，再由子系统的 Out1 端子输入顶层模型中的 Display 模块。当累加结果刚超过 200 时，Display 模块将显示总共执行的循环次数。

【例 5-10】 搭建如图 5-39 所示的仿真模型。



(a) 顶层模型



(b) Do-while迭代子系统

图 5-39 例 5-10 仿真模型

该例实现的功能与例 5-9 相同，只是改为用 Do-while 循环迭代方式来实现。

为此，需要设置 While 迭代器的 While loop type 参数为 do-while，另外注意勾选 Show iteration number port，并确保 Output data type 参数为 double。

由于设置 While loop type 参数为 do-while，因此迭代器及迭代子系统都不再有 IC 端子。在顶层模型中，常数 200 直接输入子系统，与加法器和存储模块构成的累加器累加的结果进行比较。只要累加结果没有超过 200，则送入迭代器的 cond 条件一直为 true，因此累加器就不断将迭代器输出的计数变量值进行累加。

当累加结果超过 200 时，cond 条件变为 false，此时迭代器控制停止执行子系统中的累加运算。累加的次数由子系统的 Out1 端子输出到 Display 模块显示。

5.4 子系统的封装

微课视频

封装(Mask)指为子系统或者自定义模块创建自定义的外观、接口逻辑和隐藏数据等。通过封装可以隐藏子系统的内部结构，使子系统像一个普通的模块一样，具有自己的描述、参数和帮助文档；通过封装，还可以在模块上显示有意义的图标，为封装模块提供自定义对话框，以便获取内部模块的指定参数，提供便于识别的自定义描述，用 MATLAB 代码对参数进行初始化等。

5.4.1 封装编辑器

子系统的封装是通过封装编辑器(Mask Editor)实现的，利用封装编辑器，可以创建和

自定义模块封装。可以用如下方法打开封装编辑器：

(1) 在 Simulink 编辑器窗口的 MODELING 选项卡中，单击 COMPONENT 按钮组中的 Create Model Mask 按钮。

(2) 在仿真模型中选中需要封装的子系统模块，并在 Simulink 编辑器窗口的 SUBSYSTEM BLOCK 选项卡的 MASK 按钮组中，单击 Create Mask 按钮。

(3) 右键单击需要封装的子系统模块，在弹出的快捷菜单中依次选择 Mask 和 Create Mask 菜单命令。

(4) 如果需要对已经封装的子系统的封装进行修改，先选中子系统模块，然后在 SUBSYSTEM BLOCK 选项卡的 MASK 按钮组中，单击 Edit Mask 按钮。

打开的封装编辑器对话框如图 5-40 所示。

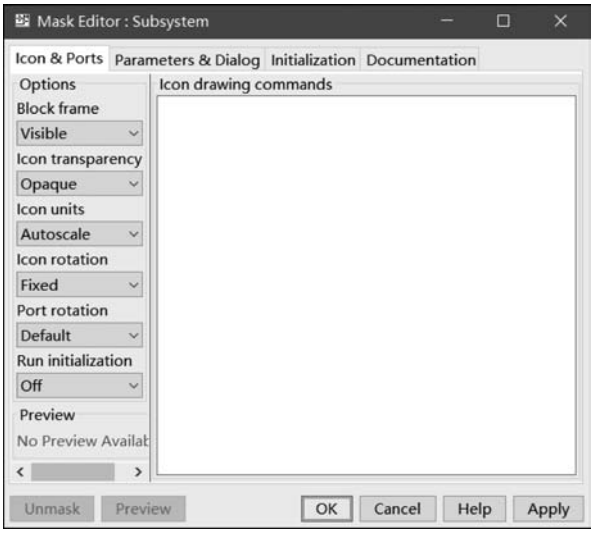


图 5-40　封装编辑器

封装编辑器对话框顶部是 4 个标签，每个标签对应的选项卡分别定义不同的封装功能。其中，Icon & Ports(图标和端子)选项卡用于创建模块封装图标，Parameters & Dialog(参数和对话框)选项卡用于设计封装对话框，Initialization(初始化) 选项卡使用 MATLAB 代码对封装模块进行初始化，Documentation(文档)选项卡用于添加有关模块封装的说明和帮助。

封装编辑器对话框左下角有两个按钮，即 Unmask(解除封装)和 Preview(预览)，右下角是 4 个常规的窗口按钮。

1. Icon & Ports 选项卡

单击封装编辑器对话框顶部的 Icon & Ports 标签，进入 Icon & Ports 选项卡，如图 5-40 所示。该选项卡主要包括 3 个面板，即 Options(选项)、Icon drawing commands(图标绘制命

令)和 Preview(预览)。

1) Options 面板

Options 面板中提供了各种选项,以指定封装图标的属性,包括 Block frame(模块边框)是否可见、Icon transparency(图标的透明度)、Icon units(绘制图表所用的坐标系)、Icon rotation(图标是否旋转)、Port rotation(封装模块的端口是否旋转)、Run Initialization(是否允许运行时初始化图标)。

2) Icon drawing commands 面板

Icon drawing commands 面板中主要是一个编辑框,在其中可以输入各种绘图命令以绘制模块图标。常用的命令有 plot(在封装图标上绘制由一系列点连接而成的图形)、text(在封装图标上的特定位置显示文本)、image(在封装图标上显示 RGB 图像)、color(更改后续封装图标绘制命令的绘图颜色)、disp(在封装图标上显示文本)、dpoly(在封装图标上显示传递函数)等。

3) Preview 面板

该面板用于显示模块封装图标的预览。只有当封装包含绘制的图标时,模块封装预览才可用。当添加图标绘制命令并单击 Apply 按钮时,预览图像将得到刷新,并显示在 Preview 面板中。

2. Parameters & Dialog 选项卡

单击封装编辑器对话框顶部的 Parameters & Dialog 标签,进入 Parameters & Dialog 选项卡,如图 5-41 所示。该选项卡主要包括 3 个面板,即 Controls(控件)、Dialog box(对话框)和 Property editor(属性编辑器)。利用这 3 个面板可以设计封装模块的参数对话框。

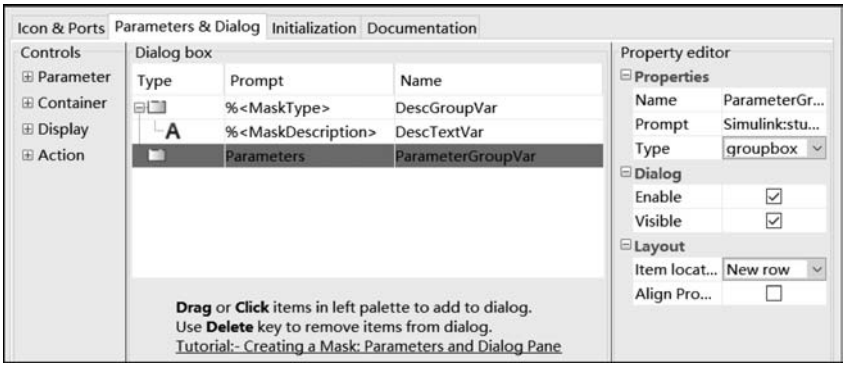


图 5-41 Parameters & Dialog 选项卡

1) Controls 面板

控件(Controls)是在模块封装以后的参数对话框中,用户可与之交互以添加或处理模块参数和数据的元素。在该面板中,提供了四大类控件,即 Parameters(参数)、Container(容器)、Display(显示)和 Action(动作)。

Parameters 控件是参与仿真的用户输入，包括 Edit（编辑框）、Check box（复选框）、Popup（下拉列表框）、Combo box（组合框）、Radio button（单选按钮）、Slider（进度条）等。

Container 控件是用于对上述控件进行布局排列组合的容器，例如 Panel（面板）、Table（表格）等。

Display 控件主要有 Text（文本框）、Image（图像框）、Text area（文本区）、Listbox control（列表）和 Tree control（树形）。

Action 控件主要有 Hyperlink（超链接）和 Button（按钮）两个控件。

2）Dialog box 面板

将上述对话框控件从 Controls 面板拖入 Dialog box 面板中，即可利用层次结构为封装模块创建参数对话框。

在 Dialog box 面板中，共有 3 个字段，即 Type（类型）、Prompt（提示）和 Name（名称）。其中，Type 字段显示对话框控件的类型和编号，Prompt 字段显示对话框控件的提示文本，Name 字段将自动填充，用于唯一地标识对话框控件。

在 Dialog box 面板中，为封装模块参数对话框添加的每个控件将占一行，其中 Parameter 控件以浅蓝色背景显示，而 Display 控件和 Action 控件以白色背景显示。

3）Property editor 面板

利用 Property editor 面板可以查看和设置指定控件的属性。在 Dialog box 面板中单击选中某一个控件，将在 Property editor 面板中显示该控件的所有属性，可以在此对指定属性进行修改和设置。

3. Initialization 选项卡

单击封装编辑器对话框顶部的 Initialization 标签，进入 Initialization 选项卡，如图 5-42 所示。该选项卡有两个面板，即 Dialog variables（对话框变量）和 Initialization commands（初始化命令）。

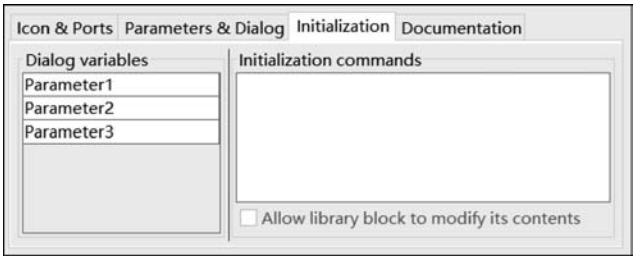


图 5-42　Initialization 选项卡

利用 Initialization 选项卡可以添加用于初始化封装模块的 MATLAB 命令。打开模型时，Simulink 会查找位于模型顶层的可见封装模块。仅当这些可见的封装模块具有图标绘制命令时，Simulink 才对这些模块执行初始化命令。

4. Documentation 选项卡

Documentation 选项卡如图 5-43 所示。在该选项卡中，可以定义或修改封装模块的类型、为封装模块添加说明和帮助文本。

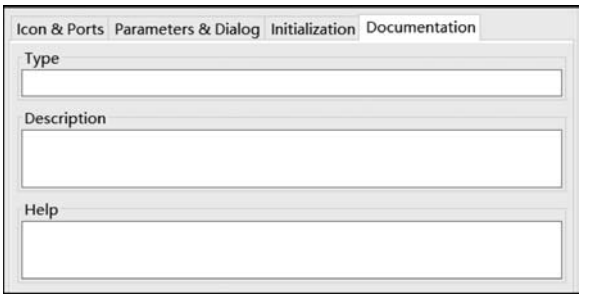


图 5-43　Documentation 选项卡

该面板主要由 3 个编辑框构成，即 Type（封装类型）、Description（描述说明）和 Help（帮助文档）。

1）封装类型 Type

封装类型是显示在封装编辑器中的模块分类。当 Simulink 显示封装编辑器对话框时，它会为封装类型添加后缀 mask。要定义封装类型，可以在 Type 字段中输入类型。文本可以包含任何有效的 MATLAB 字符，但不能包含换行符。

2）描述说明 Description

封装描述说明是描述模块的用途或功能的简要帮助文本。要定义封装说明，可以在 Description 编辑框中输入说明。文本可以包含任何合法的 MATLAB 字符。Simulink 会自动进行换行，也可以使用回车键强制换行。

3）帮助文档 Help

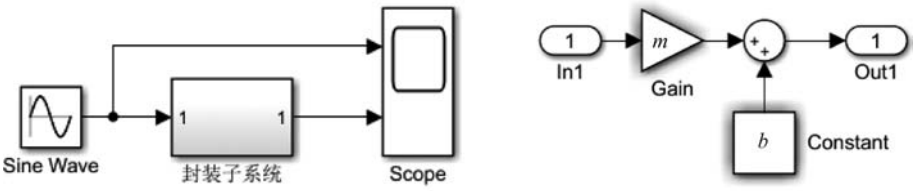
封装模块的 Online Help（联机帮助）提供 Type 和 Description 编辑框所提供信息之外的其他信息。当封装模块用户单击封装对话框上的 Help 按钮时，将会在一个单独的窗口中显示此信息。

5.4.2　子系统封装步骤

下面通过具体仿真模型介绍利用封装编辑器对子系统进行封装的操作步骤。

【例 5-11】　子系统的封装。

首先新建仿真模型，向其中放置一个 Subsystem 模块和一个正弦波信号源模块 Sine Wave、一个示波器模块 Scope，如图 5-44(a)所示。再双击 Subsystem 模块，打开子系统，搭建子系统内部模型，如图 5-44(b)所示。注意修改各模块的名称。



(a) 顶层模型　　(b) 子系统内部模型

图 5-44　例 5-11 仿真模型

设置 Sine Wave 模块的参数为默认值，子系统内部 Gain 和 Constant 模块的 Gain 和 Constant value 参数分别设为变量 m 和 b。由于此时这两个变量尚未赋值，因此两个模块都用有阴影的边框表示，并且 Simulink 编辑器中将给出提示信息。先忽略该提示信息。

1. 设计封装模块的参数对话框

在封装编辑器的 Parameters & Dialog 选项卡左侧面板中，单击 Parameter 下面的 Edit（文本框）两次，从而在中间的 Dialog box 面板中添加两个控件，每个控件占一行。

在刚添加的两个控件中，分别单击 Prompt 列和 Name 列，输入两个控件对应的参数提示和参数名。其中，参数提示可以任意设置为能够反映控件对应的模块参数的意义的字符串，参数名分别设为 m 和 b，必须与仿真模型中 Gain 和 Constant 模块的参数完全相同。设置完成后，选项卡中对话框面板的显示如图 5-45 所示。

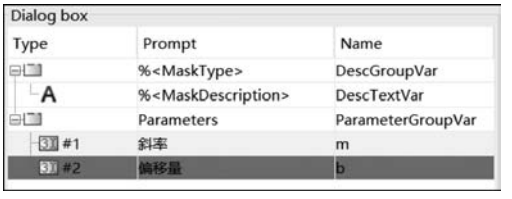


图 5-45　添加控件

在对话框面板选中某一行，在右侧的 Property editor 面板中将列出该行控件的属性。在 Value 文本框中设置参数的默认值。这里假设参数 m 和 b 的默认值分别设为 1 和 0，如图 5-46 所示。

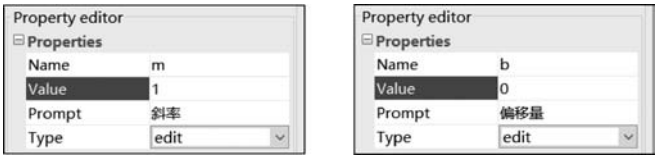


图 5-46　设置控件参数的默认值

2. 添加描述和帮助

在封装编辑器的 Documentation 选项卡中设置封装模块参数对话框中所需要的一些描

述和帮助信息，如图 5-47 所示。

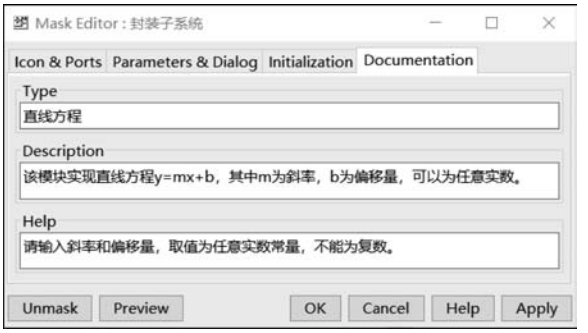


图 5-47　添加帮助和描述

3. 为封装模块添加图标

Simulink 模块库中的所有模块都有专用的图标，模块调入仿真模型后，能够根据其功能在图标上显示形象直观的图案。例如，Sine Wave 模块图标上显示一条正弦波形曲线，Scope 模块图标上显示一个示波器屏幕。

通过封装，也可以在自己创建的子系统模块图标上得到类似的显示效果。以本例中的仿真模型为例，其中的 Subsystem 模块实现直线方程，可以在图标上显示一条具有一定斜率和偏移量的直线。

首先创建一个图片文件，在其中绘制一个二维坐标，在坐标系中绘制一条直线。然后将文件命名为“直线方程.jpg”，放在当前文件夹中。

在封装编辑器中，单击进入 Icon & Ports 选项卡，在 Icon drawing commands 面板中输入如下绘图命令：

```
image('直线方程.jpg')
```

其中参数为已创建的图片文件名。

在 Icon & Ports 选项卡中输入上述命令后，将立即在左侧的 Preview 面板中给出图标图案的预览效果。

4. 效果预览

完成上述设置后，单击封装编辑器左下角的 Preview 按钮，立即在屏幕上弹出封装模块参数对话框，如图 5-48 所示，其中有两个编辑框，用于设置封装模块内部的两个参数。并且，两个编辑框中都有默认值 1 和 0。

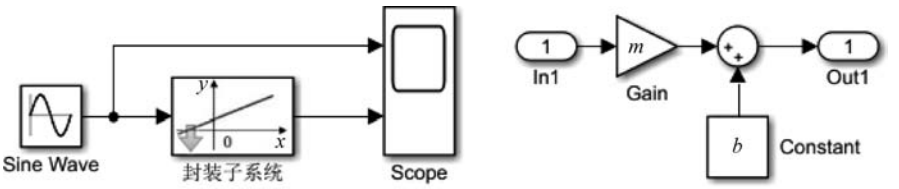
单击图 5-48 所示参数对话框中的 Help 按钮，将打开浏览器，其中显示封装模块的帮助信息，如图 5-49 所示。

预览完成后，单击封装编辑器中的 OK 按钮或者 Apply 按钮，关闭预览效果对话框和

封装编辑器，回到顶层仿真模型。此时，可以发现在 Subsystem 模块的左下角增加了一个灰色向下的粗箭头，如图 5-50(a)所示，表示该模块对应的子系统已经进行了封装。

图 5-48　参数对话框预览效果

图 5-49　模块的帮助信息



(a) 顶层模型　　(b) 子系统内部模型

图 5-50　封装后的仿真模型

右击“封装子系统”模块，在弹出的快捷菜单中依次选择 Mask 和 Look Under Mask 菜单命令，可以打开子系统。可以看到子系统内部的 Gain 和 Constant 模块不再有阴影和错误提示，如图 5-50(b)所示，因为两个模块的参数已经分别具有默认值 1 和 0。

5.4.3　对封装模块的操作

封装后的子系统模块，其图标的左下角将出现一个箭头。与普通的模块一样，双击该模块，将弹出封装模块的参数对话框。对没有封装的子系统模块，双击时将打开该子系统的内部模型，可以对子系统内部结构和内部各模块的参数等进行检查修改。

封装后，如果需要修改子系统的内部模型，可以在封装模块上右击，在弹出的快捷菜单中依次选择 Mask 和 Look Under Mask 菜单命令。

如果在弹出菜单中选择 Mask 和 Edit Mask 菜单命令，可以打开封装编辑器，对封装进行修改。单击封装编辑器对话框左下角的 Unmark 按钮，可以解除封装。

5.5 S 函数


微课视频

S 函数是系统函数(System Function)的简称，是将系统数学方程与 Simulink 可视化模型联系起来并且具有固定格式接口的函数。在 Simulink 中，一切可视化模型(例如 Simulink 库中的所有模块)都是基于 S 函数实现的。

通过编写和使用 S 函数，可以构建出 Simulink 模块难以搭建或搭建过程过于复杂的系统模型，从而极大增强 Simulink 的灵活性，扩充 Simulink 的功能。

S 函数可以用 MATLAB 语言编写，也可以用 C、C++或者 Fortran 等语言编写。通过编译后，可以提高执行的速度。

5.5.1 S 函数的基本概念

S 函数最通常的用法是创建一个自定义的 Simulink 模块，可以在模型中多次使用，只需要在每次使用时改变其参数即可。

Simulink 模型中的每个模块都包括输入变量 u、输出变量 y 和内部状态变量 x。输出变量 y 是输入变量 u、状态变量 x、仿真时间 t 和模块参数的函数，可以表示为

$$\begin{aligned} y &= f_0(t,x,u) && \text{(输出)} \\ x'_c &= f_d(t,x,u) && \text{(导数)} \\ x_{d,k+1} &= f_u(t,x_c,x_{d,k},u) && \text{(更新)} \end{aligned} \tag{5-1}$$

其中，$x_{d,k}$ 和 $x_{d,k+1}$ 分别表示第 k 和 $k+1$ 个时间步内的离散状态。

模块内部的状态可能是连续状态 x_c 或离散状态 x_d。在 Simulink 中，这些状态构成状态向量 $\boldsymbol{x}=[x_c;x_d]$，其中连续状态 x_c 和离散状态 x_d 分别占据状态向量的第一行和第二行。对于没有状态的模块，$\boldsymbol{x}$ 是一个空向量。

在仿真运行的特定阶段，Simulink 反复调用仿真模型中的每个模块，以执行计算输出、更新离散状态或者计算导数等任务。此外，还包括在仿真运行的开始和结束时，执行初始化任务和结束操作。模型完整的仿真运行过程可以参考 3.5 节。

在编写 S 函数的过程中，通常涉及如下基本概念。

1. 直接馈通

直接馈通指模块的输出或者可变采样时间直接受控于输入端子的输入值。正确设置直接馈通，将影响模型中各个模块的执行顺序。

一般情况下，只要S函数满足如下条件，就认为其直接馈通。

(1) 输出变量 y 是输入变量 u 的函数。

(2) S函数是可变采样时间的函数，并且计算下一个采样时刻需要使用输入变量 u。

例如，放大器模块的输出可以表示为

$$y = ku$$

其中，u 为输入，k 为放大倍数。该式表示模块的输出是输入的简单代数关系，即该模块内部具有输入到输出之间的直接馈通。

再例如，积分器模块具有一个内部状态 x，其输出与输入的关系可以表示为

$$y = x$$
$$x' = u$$

执行时，先根据输入变量 u 确定状态变量 x 的导数，再计算 x 的积分而得到输出变量 y。输入/输出之间通过内部状态 x 联系起来，从而使得积分器模块内部没有直接馈通。

2. 动态维矩阵

S函数可以支持任意个数的输入参数，而输入参数的个数同时决定了模块连续状态、离散状态和输出的个数。在这种情况下，当仿真运行开始后，实际的输入参数个数是通过计算驱动S函数的输入向量的长度来动态确定的。

S函数模块只能有一个输入端子。如果需要接收多个输入参数，并且输入参数的具体个数是不确定的，就可以将其设置为宽度是动态可变的。S函数会自动按照合适宽度的输入端子来调用对应的模块。

3. 采样时间

S函数中提供了如下多种采样时间选项，以便在执行时具有高度的灵活性。

(1) 连续采样时间：适用于具有连续状态或者非过零采样的S函数，其输出在每个微步(子时间步)上变化。

(2) 连续微步固定采样时间：适用于需要在每个主时间步上执行，但在微步内不发生变化的S函数。

(3) 离散采样时间：适用于内部只有离散状态的S函数，此时可以定义一个采样时间来控制 Simulink 何时调用该S函数模块，也可以定义一个偏移量以实现采样时间的延迟。

(4) 可变采样时间：采样时间变化的离散采样时间，在每步仿真的开始都需要计算下一次采样时间。

(5) 继承采样时间：没有专门采样时间特性的S函数，可以根据其所连接的输入或输出模块确定采样时间，或者将采样时间设置为仿真模型中所有模块的最短采样时间。

5.5.2 S函数的实现方法和一般结构

S函数可以利用 Level-1 MATLAB 语言、Level-2 MATLAB 语言、C MEX 文件、S-Function Builder 或代码生成工具实现，不同的实现方法得到的 S 函数具有类似的结构。

1. S 函数的实现方法

在各种实现方法中，Level-1 MATLAB 语言为 MATLAB 提供了与 S 函数 API 最小组件之间进行交互的简单接口，Level-2 MATLAB 语言提供了更为丰富的 S 函数 API 接口，并且支持代码生成。对于缺乏 C 语言编程经验的 MATLAB 编程人员，特别是不需要为含有 S 函数模块的模型生成代码时，可以考虑采用 Level-2 MATLAB 语言实现 S 函数。

C MEX 文件 S 函数提供了编程方面最大的灵活性，可以通过 C MEX 手工实现算法，或者编写 S 函数包装器调用现有的 C 语言、C++语言或 Fortran 语言代码。手工编写一个新的 S 函数需要了解 S 函数 API，如果想为 S 函数生成嵌入式代码，还需要了解目标语言编译器(TLC)。

如果需要为含有 S 函数模块的仿真模型生成代码，可以选用 Level-2 MATLAB 语言或者 C MEX 文件实现 S 函数。如果选用 Level-2 MATLAB 实现，则还需要为 S 函数编写 TLC 文件。如果需要加快仿真运行速度，可以用 C MEX 文件实现 S 函数。

S-Function Builder 是实现 S 函数编程的一种图形用户界面，对于不熟悉 C MEX S 函数的编程人员，可以用 S-Function Builder 生成 S 函数，或者向 S 函数导入已有的 C 语言或 C++语言代码，而无须与 S 函数 API 交互。S-Function Builder 也可以产生 TLC 文件，以便于自动生成嵌入式代码。

代码生成工具是一组 MATLAB 命令，用于将普通的 C 语言或 C++语言代码导入 S 函数。与 S-Function Builder 一样，代码生成工具能生成 TLC 文件。

限于篇幅，这里着重介绍用 Level-1 MATLAB 语言实现 S 函数的相关知识。

2. S 函数的一般结构

在 MATLAB 安装文件夹的子文件夹/toolbox/simulink/blocks 中，提供了实现 Level-1 MATLAB S 函数的模板文件 sfuntmpl.m。该模板文件中包括一个顶层函数和一组局部函数框架，这些局部函数称为回调方法，每个回调方法对应一个特定的标志(flag)。顶层函数根据标志调用这些回调方法，由回调方法执行仿真过程中 S 函数所需的实际操作任务。

附录 C 给出了模板文件 sfuntmpl.m 的完整代码。下面对其做一些必要的解释。

1) 顶层函数

模板文件中的顶层函数声明语句为

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

其中，sfuntmpl 是 S 函数的名称，flag 为调用功能标志，t 是仿真时间，x 和 u 分别为模块的内部状态和输入变量。

在模型的仿真运行过程中，Simulink 不断重复调用 S 函数，并自动设置 flag 标志以指示在当前调用 S 函数时需要执行的具体操作，flag 标志的取值与调用的回调方法之间的对应关系如表 5-1 所示。

表 5-1 flag 标志与回调方法

flag	回调方法	执行的任务
0	mdlInitializeSizes()	定义 S 函数模块的基本特性，包括采样时间、连续状态和离散状态的初始值、数组大小等
1	mdlDerivatives()	计算连续状态变量的导数
2	mdlUpdate()	更新离散状态、采样时间和主时间步
3	mdlOutputs()	计算 S 函数的输出
4	mdlGetTimeOfNextVarHit()	以绝对时间计算下一个采样时刻，该方法只用于在 mdlInitializeSizes()方法中指定了一个可变离散采样时间的情况
9	mdlTerminate()	执行所需的仿真结束操作

执行 S 函数后，将返回一个输出向量，其中包括如下元素：

(1) sys：通用返回参数，其返回值取决于 flag 标志。例如，flag＝3，则 sys 返回结果为 S 函数的输出。

(2) x_0：初始状态值。当 flag＝0 时，返回 S 函数模块的初始值。对于 flag 的其他取值，该返回值无效。

(3) str：预留参数，一般设为空矩阵，即写为[]。

(4) t_s：采样时间和偏移量。该参数返回值有两列，分别保存采样时间和偏移量。

(5) simStateCompliance：指定保存和恢复仿真运行时处理模块的方法。

在模板文件的顶层函数中，除了上述声明语句和注释以外，主要有如下代码：

```
switch flag,
  case 0,
    [sys,x0,str,ts,simStateCompliance] = mdlInitializeSizes;
  case 1,
    sys = mdlDerivatives(t,x,u);
  case 2,
    sys = mdlUpdate(t,x,u);
  case 3,
    sys = mdlOutputs(t,x,u);
  case 4,
    sys = mdlGetTimeOfNextVarHit(t,x,u);
  case 9,
```

```
    sys = mdlTerminate(t,x,u);
  otherwise
    DAStudio.error('Simulink:blocks:unhandledFlag',...
                   num2str(flag));
end
```

上述代码利用 switch-case 语句，根据调用 S 函数时入口参数 flag 的取值，分别调用后面的回调方法。

2）回调方法

在模板文件中，除了上述顶层函数内的 switch-case 语句外，还有所有回调方法的框架。这里首先介绍 mdlInitializeSizes()方法。

回调方法 mdlInitializeSizes()实现 S 函数模块的初始化，包括定义 S 函数模块的基本特性，包括采样时间、连续状态和离散状态的初始值、数组大小等。该方法的完整代码如下：

```
function [sys,x0,str,ts,simStateCompliance] = mdlInitializeSizes
  sizes = simsizes;
  sizes.NumContStates  = 0;
  sizes.NumDiscStates  = 0;
  sizes.NumOutputs     = 0;
  sizes.NumInputs      = 0;
  sizes.DirFeedthrough = 1;
  sizes.NumSampleTimes = 1;
  sys = simsizes(sizes);
  x0 = [];
  str = [];
  ts  = [0 0];
  simStateCompliance = 'UnknownSimState';
```

方法中的第一条语句调用 simsizes()函数创建一个空的 sizes 结构，之后需要用相关信息对结构中的各字段赋值，其中主要包括如下字段：

（1）sizes.NumContStates：连续状态变量的个数；

（2）sizes.NumDiscStates：离散状态变量的个数；

（3）sizes.NumOutputs：输出变量的个数；

（4）sizes.NumInputs：输入变量的个数；

（5）sizes.DirFeedthrough：直接馈通标志；

（6）sizes.NumSampleTimes：采样时间的个数。

在上述方法的代码中，对这些字段都设了默认值。在实现 S 函数时，可以根据需要修改和重新设置这些字段的值。在完成 sizes 结构的初始化后，需要重新调用 simsizes()函数，将上述初始化信息传递给向量 sys，以供 Simulink 仿真运行时访问。

除了完成向量 sys 的上述初始化以外，在 mdlInitializeSizes()方法中还需要设置状态变

量的初始值 x_0，设置 str 为空矩阵，设置采样时间向量 t_s 等。大多数情况下，这些初始值取默认值即可。

5.5.3 静态和动态系统的 S 函数实现

静态系统中没有状态变量，只有输入和输出，典型的就是 Simulink 库中的各种信号源模块以及实现各种代数运算的模块。动态系统中含有状态变量，并且一般情况下，状态变量的个数等于系统阶数。对于离散动态系统，其输入、输出和状态变量都是离散信号；对于连续动态系统，其输入、输出和状态变量都是连续信号。

由于这些特点，用 S 函数实现上述各种系统时，实现的方法稍微有所区别。

1. 静态系统的 S 函数实现

由于没有状态变量，因此用 S 函数实现静态系统时，不需要计算和更新状态，也就不需要使用 mdlDerivatives()和 mdlUpdate()回调方法，只需要调用 mdlOutputs()回调方法，计算得到输出。下面举例说明。

【例 5-12】 用 S 函数实现一次函数运算，即 $y=au+b$，其中 a 和 b 是 S 函数的两个参数，u 和 y 分别为 S 函数模块的输入和输出。

实现方法和操作步骤如下：

(1) 打开模板文件 sfuntmpl.m，将文件重新命名为 sfun1.m，并保存到当前文件夹中。同时将其中的函数声明修改为 sfun1，并在顶层函数入口参数列表末尾添加两个入口参数 a 和 b。

(2) 在 mdlInitializeSizes()回调方法中，设置输入和输出变量的个数都为 1，即修改如下两条语句：

```
sizes.NumOutputs      = 1;
sizes.NumInputs       = 1;
```

(3) 将 mdlOutputs()回调方法作如下修改：

```
function sys = mdlOutputs(t,x,u)
    sys = a*u+b;
```

做完上述设置后，S 函数创建完毕。为了使用所创建的 S 函数，继续作如下操作。

(4) 新建仿真模型文件，向其中添加一个 S-Function 模块，该模块位于 Simulink/User-Defined Functions 库中。

(5) 双击 S-Function 模块，打开其参数对话框，如图 5-51 所示。在 S-function name 文本框中输入前面创建的 S 函数名 sfun1，并在 S-function parameters 文本框中输入两个参数 a 和 b。这两个参数也就是在所创建的 S 函数中，除了 t、x 和 u 等参数以外，另外增加的两

个参数。两个参数之间用逗号分隔。

设置完 S-Function 模块的参数后，单击 OK 按钮，在仿真模型中，该模块图标上将显示 S 函数的名称 sfun1。

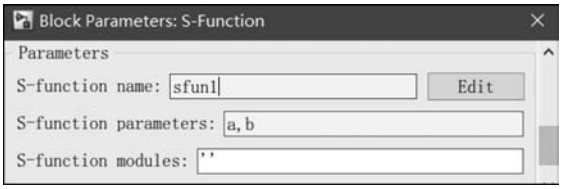


图 5-51　S-Function 模块的参数对话框

（6）向仿真模型中另外添加一个 Sine Wave 模块和一个 Scope 模块，得到完整的仿真模型，如图 5-52 所示。其中，Sine Wave 模块的参数都取默认值。

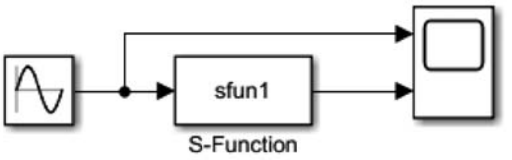


图 5-52　例 5-12 仿真模型

（7）在 MATLAB 命令行窗口输入如下命令：

```
>> a = 2;
>> b = -1;
```

上述命令的作用是为 S-Function 模块及其 S 函数中的入口参数 a 和 b 赋值。

（8）仿真运行，在示波器上观察运行结果，如图 5-53 所示。

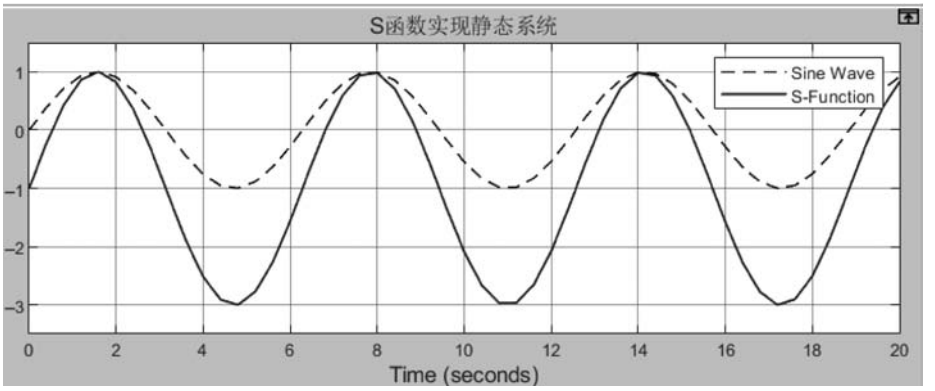


图 5-53　例 5-12 运行结果

2. 离散动态系统的S函数实现

离散动态系统中只有离散状态变量，没有求导运算。在系统的状态空间方程中，状态方程反应的是状态变量的递推关系，输出方程中也只是简单的代数运算。因此用S函数实现时，只需要用到mdlUpdate()和mdlOutputs()方法。

【例5-13】 已知一个离散SISO系统的传递函数为

$$H(z)=\frac{2z+1}{z^2+0.5z+0.8}$$

用S函数实现该系统，并求其单位脉冲响应。

实现方法和操作步骤如下：

(1) 打开模板文件sfuntmpl.m，将文件重新命名为sfun2.m，并保存到当前文件夹中。同时将其中的函数声明修改为sfun2，并在顶层函数入口参数列表末尾添加两个入口参数a和b。

(2) 根据MATLAB中提供的标准模板，用S函数实现动态系统时，需要已知系统的状态空间方程。本例中已知的是系统的传递函数，因此在顶层函数switch-case语句之前添加如下语句：

```
[A,B,C,D] = tf2ss(b,a);
```

该语句调用tf2ss()函数将由矩阵 ***b*** 和 ***a*** 指定的传递函数转换为状态空间方程，得到状态空间方程中的4个矩阵。

(3) 在所有的回调方法中，将上述状态空间方程中的4个矩阵增添为入口参数，例如：

```
case 2,
    sys = mdlUpdate(t,x,u,A,B,C,D);
```

(4) 该离散系统是一个二阶SISO系统，因此分别有一个输入变量、一个输出变量和两个离散状态变量。据此将mdlInitializeSizes()回调方法中的相关语句作如下修改：

```
sizes.NumDiscStates         = 2;
sizes.NumOutputs            = 1;
sizes.NumInputs             = 1;
sizes.DirFeedthrough        = 1;
x0   = zeros(sizes.NumDiscStates,1);        % 状态变量初始化
ts   = [-1 0];                              % 设置采样时间继承于模块的输入
```

(5) 将mdlUpdate()和mdlOutputs()回调方法做如下修改：

```
function sys = mdlUpdate(t,x,u,A,B,C,D)
    sys = A*x+B*u;
function sys = mdlOutputs(t,x,u,A,B,C,D)
    sys = C*x+D*u;
```

(6) 新建仿真模型如图 5-54 所示。在 S-Function 模块的参数对话框中设置 S-function name 为 sfun2,并在 S-function parameters 文本框中输入两个参数 a 和 b。

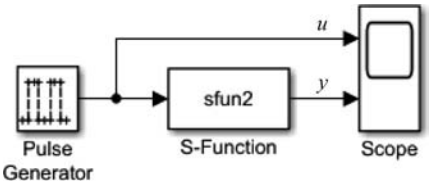


图 5-54　例 5-13 仿真模型

此外,设置模型中 Pulse Generator 模块的参数如图 5-55 所示,则该模块以 0.1s 的采样间隔产生周期为 5s、宽度为 0.1s 的离散周期脉冲序列。

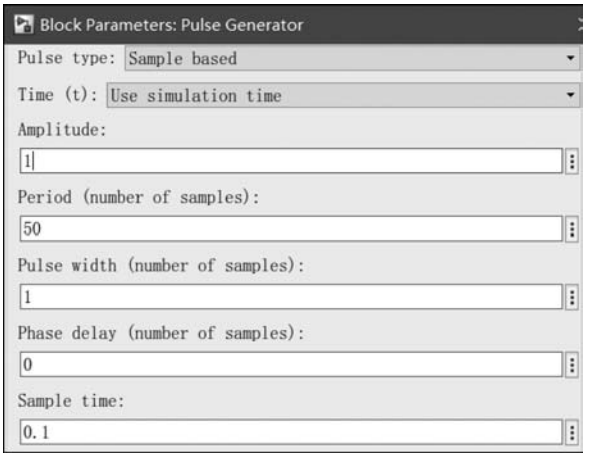


图 5-55　Pulse Generator 模块的参数设置

(7) 在命令行窗口输入如下命令:

```
>> b = [2 1];
>> a = [1 0.5 0.8];
```

执行后,得到矩阵 $\boldsymbol{b}$ 和 $\boldsymbol{a}$,作为上述 S-Function 的参数。

(8) 启动仿真运行,立即在示波器上得到波形,如图 5-56 所示。

3. 连续动态系统的 S 函数实现

连续动态系统中有连续的状态变量,状态空间方程中的状态方程是一组一阶微分方程。分析求解时,需要先用数值微积分算法进行求导运算,得到状态变量。再根据输出方程得到系统的输出变量。因此,在执行过程时,需要在微步中反复调用 S 函数中的 mdlDerivatives() 回调方法执行求导运算。

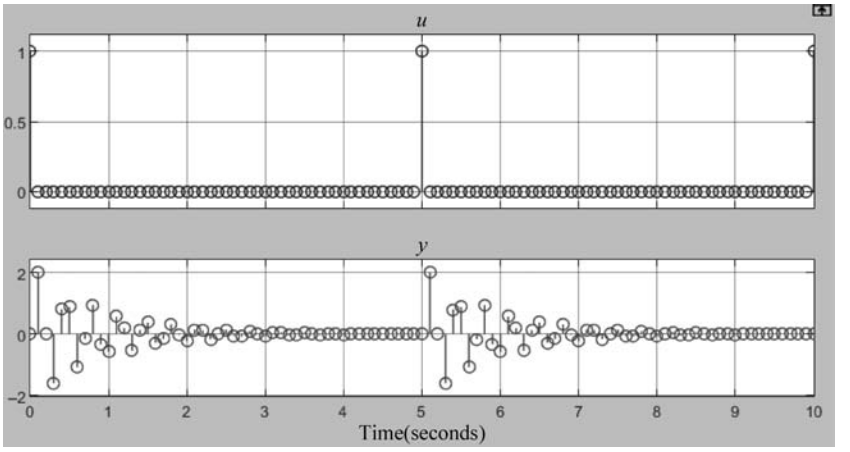


图 5-56　例 5-13 运行结果

【例 5-14】 已知连续系统的状态方程系数矩阵为

$$\boldsymbol{A}=\begin{bmatrix}-0.09 & -0.01\\ 1 & 0\end{bmatrix},\quad \boldsymbol{B}=\begin{bmatrix}1 & -7\\ 0 & -2\end{bmatrix},\quad \boldsymbol{C}=\begin{bmatrix}0 & 2\\ 1 & -5\end{bmatrix},\quad \boldsymbol{D}=\begin{bmatrix}-3 & 0\\ 1 & 0\end{bmatrix}$$

用 S 函数实现该系统。

实现方法和操作步骤如下：

(1) 打开模板文件 sfuntmpl.m，将文件重新命名为 sfun3.m，并保存到当前文件夹中，同时将其中的函数声明修改为 sfun3。

(2) 在顶层函数的 switch 语句之前，输入已知的 4 个系数矩阵。

```
A=[-0.09 -0.01;1 0];
B=[1 -7;0 -2];
C=[0 2;1 -5];
D=[-3 0;1 0];
```

(3) 由已知的状态空间方程分析得知，该连续系统有 2 个输入和 2 个输出，并且有 2 个状态变量。因此，在 mdlInitializeSizes()回调方法中，修改如下 3 条语句，指定该系统的连续状态变量、输出变量和输入变量都为 2。

```
sizes.NumContStates   = 2;
sizes.NumOutputs      = 2;
sizes.NumInputs       = 2;
```

并且，将 x0 语句修改为

```
x0   = zeros(2,1);
```

(4) 在 mdlDerivatives()和 mdlOutputs()两个函数的调用和声明中，将状态空间方程

中的 4 个矩阵作为其入口参数，并将两个函数作如下修改：

```
function sys = mdlDerivatives(t,x,u,A,B,C,D)
    sys  =  A * x + B * u;
function sys = mdlOutputs(t,x,u,A,B,C,D)
    sys  =  C * x + D * u;
```

（5）新建仿真模型如图 5-57 所示，在其中添加一个正弦波信号源 Sine Wave 和一个随机数发生器 Random Number，参数都取默认值。

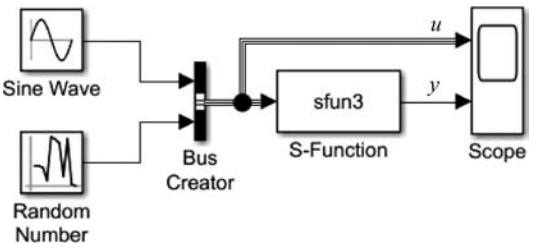


图 5-57　例 5-14 仿真模型

注意，用 Bus Creator 模块将两个输入合并为一路输入 S-Function 模块。S-Function 模块的参数对话框中输入刚创建的 S 函数名，另外两个参数保持默认值。

（6）仿真运行，示波器上的波形显示如图 5-58 所示。

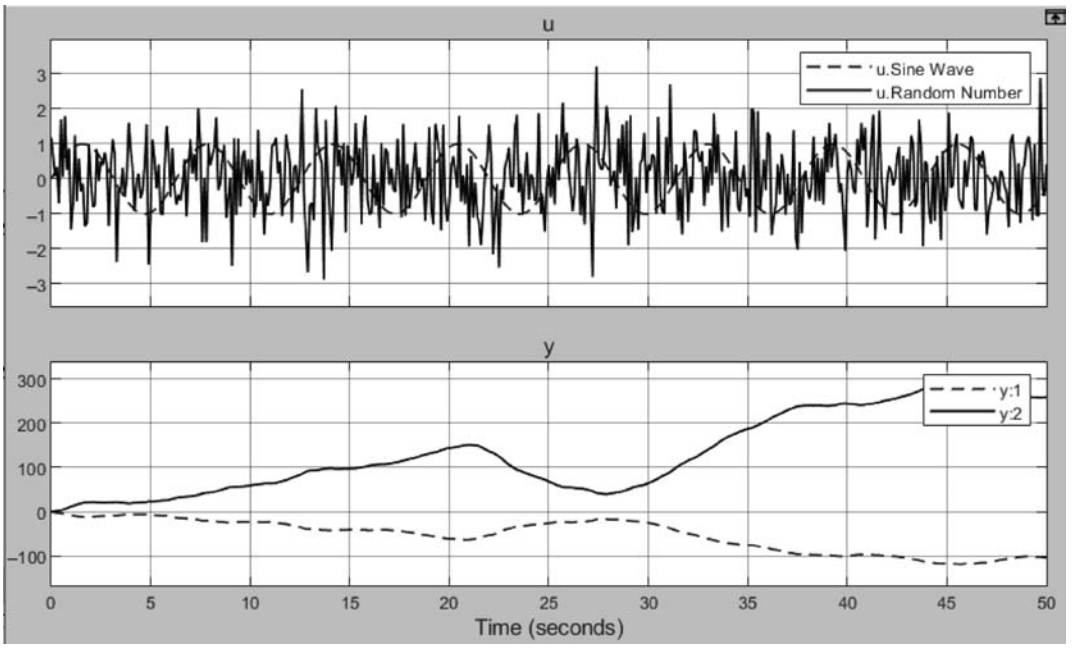


图 5-58　例 5-14 运行结果

5.5.4 S函数模块的封装

在例5-12和例5-13中，实现两个系统的S函数都需要a和b两个参数。在仿真运行之前，必须先为这两个参数赋值，否则运行时将报错。为此，可以将S函数模块进行封装。

与子系统一样，通过封装可以使S函数模块与普通模块一样，在搭建仿真模型时设置所需的参数，然后直接启动运行。S函数模块的封装方法也与前面介绍的子系统封装方法完全一样。下面结合例5-13说明封装步骤。

【例5-15】 仿真模型同例5-13，要求对其中的S-Function模块进行封装。

具体封装步骤如下：

(1) 在仿真模型中右击S-Function模块，在弹出的快捷菜单中依次单击Mask和Create Mask菜单命令，打开封装编辑器对话框。

(2) 进入Parameters & Dialog选项卡，在Controls控件中单击Edit两次，添加两个编辑框控件，将两个控件的Prompt和Name属性按图5-59进行设置。注意，在右侧的Property editor面板中设置这两个参数的默认值都为[1]。

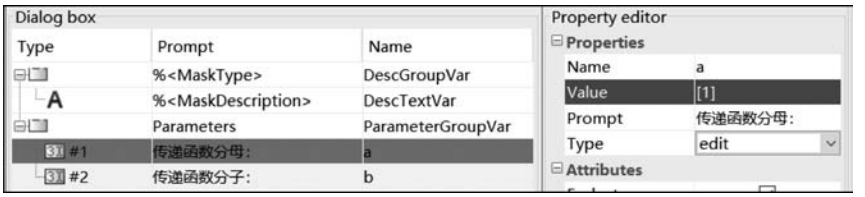


图5-59 添加控件

(3) 在Documentation选项卡中，自行输入相关的描述和帮助信息，如图5-60所示。然后，单击封装编辑器左下角的Preview按钮，预览封装后的参数对话框效果。如果效果不满意，返回封装编辑器修改。

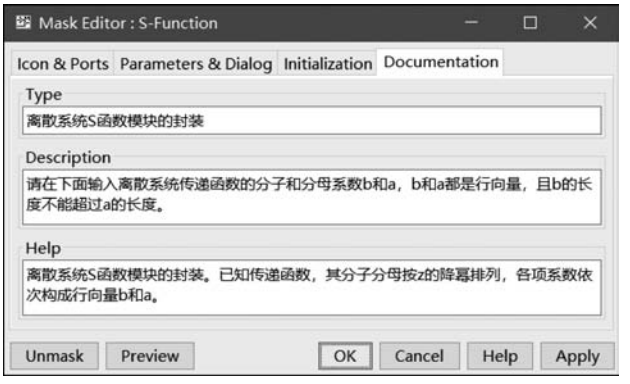


图5-60 添加描述信息和帮助文档

(4) 回到仿真模型,双击 S-Function 模块,在打开的参数对话框中输入传递函数分母多项式和分子多项式系数矩阵,与例 5-13 完全相同,如图 5-61 所示。仿真运行,观察运行结果也与例 5-13 完全一致。

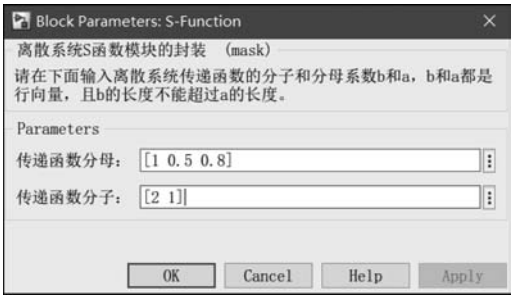


图 5-61 封装后的 S 函数模块参数对话框

本章习题

1. 在 Simulink 中,虚拟子系统主要指用_____模块创建的子系统,而非虚拟子系统又分为_____子系统和_____子系统。

2. 触发子系统与 Subsystem 子系统的主要区别在于,其中含有一个_____模块。

3. 要使触发子系统在触发信号的上升沿和下降沿都能触发执行,应该设置 Trigger 模块的 Trigger type 参数为_____。

4. 一个 If 动作子系统包括一个_____模块和若干个 If Action Subsystem 模块。

5. If Action Subsystem 内部有一个_____模块,用于接收来自外部 If 模块的条件。

6. 用 S 函数实现静态系统时,只需要用到_____和_____两个回调方法。

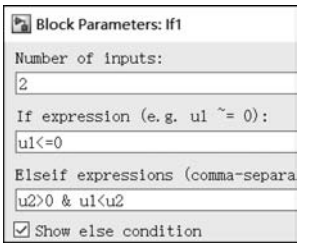
7. 用 S 函数实现离散动态系统时,系统的状态方程放在_____回调方法中,输出方程放在_____回调方法中。

8. 用 S 函数实现连续动态系统时,状态方程放在_____回调方法中。

9. 已知某 3 阶离散动态系统有 2 个输入和 1 个输出,则用 S 函数实现时,在 mdlInitializeSizes()回调方法中,应设置

```
sizes.NumContStates  = _____;
sizes.NumDiscStates = _____;
sizes.NumOutputs = _____;
sizes.NumInputs = _____.
```

10. 已知 If 模块的参数设置如题图 5-1 所示,画出 If 模块的图标,图上要标出所有的输入/输出端子。

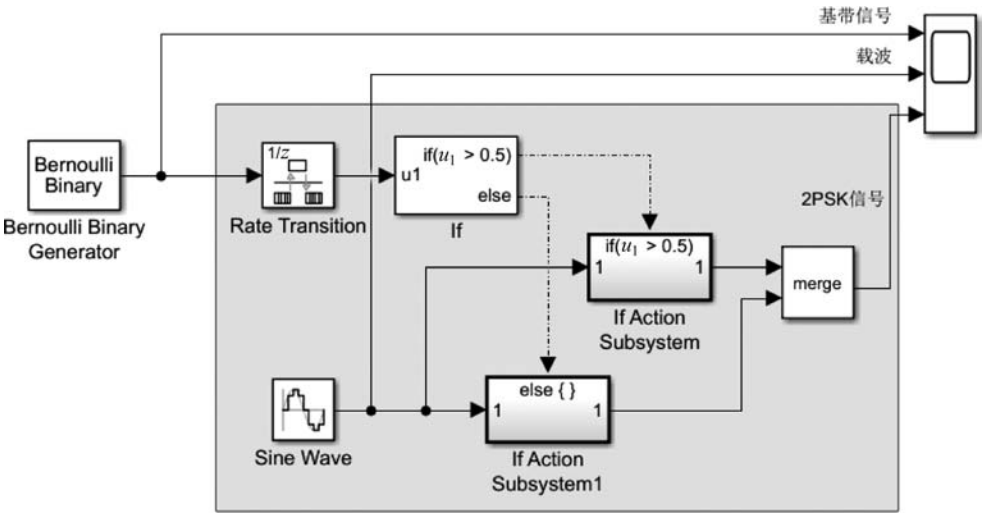


题图 5-1

11. 简述对于封装和未封装的子系统模块,双击后的操作有何区别。

实践练习

1. 搭建如题图 5-2 所示的仿真模型。



题图 5-2

其中，Bernoulli Binary Generator 模块的 Sample time 参数设为 0.1s（即码元速率为 10 baud），Sine Wave 模块的 Frequency 参数设为 40π rad/s，Sample time 和 Rate Transition 模块的 Output port sample time 参数设置相同，为 1ms。其他模块的参数取默认值。

（1）仿真运行，观察基带信号、载波和 2PSK 信号的波形。

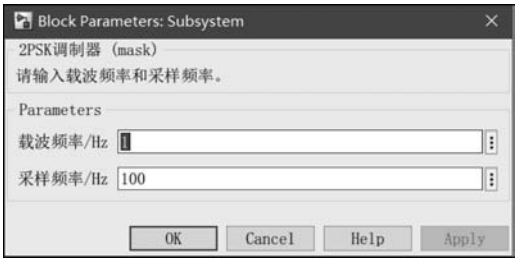
（2）将仿真模型中阴影方框内的部分创建为子系统。

（3）要求将 Sine Wave 模块的 Frequency 参数设为变量 2 * pi * f_c，其中变量 f_c 单位为 Hz；Sine Wave 模块的 Sample time 参数和 Rate Transition 模块的 Output port sample time 参数都设为 $1/f_s$，其中变量 f_s 单位为 Hz。为(2)中的子系统创建如题图 5-3 所示的封装对话框。

2. 用 S 函数实现连续动态系统，并求其单位阶跃响应。已知系统传递函数的零极点增益模型为

$$H(s)=\frac{10(s+100)}{(s+5+\mathrm{j}10\pi)(s+5-\mathrm{j}10\pi)}$$

要求不能用手工进行任何计算，所有功能都用代码实现。



题图 5-3