第十二章　利用状态流 Stateflow 进行控制系统状态转换

Statefolw 是一种图形化的设计开发工具，是有限状态机的图形实现工具，有人称之为状态流。主要用于 simulink 中控制和检测逻辑关系的。用户可以在进行 Simulink 仿真时，使用这种图形化的工具实现各个状态之间的转换，解决复杂的监控逻辑问题。它和 Simulink 同时使用使得 Simulink 更具有事件驱动控制能力。利用状态流可以做以下事情：

1） 基于有限状态机理论的相对复杂系统进行图形化建模和仿真；

2） 设计开发确定的、检测的控制系统；

3） 更容易在设计的不同阶段修改设计、评估结果和验证系统的性能；

4） 自动直接地从设计中产生整数、浮点和定点代码（需要状态流编码器）；

5） 更好地结合利用 Matlab 和 Simulink 的环境对系统进行建模、仿真和分析。

在状态流图中利用状态机原理、流图概念和状态转化图，状态流能够对复杂系统的行为进行清晰、简洁的描述。

Stateflow 生成的监控逻辑可以直接嵌入到 Simulink 模型下，两者之间能够实现无缝连接。仿真初始化时，Simulink 会自动启动编译程序，将 Stateflow 绘制的逻辑框图转换成 C 格式的 S－函数（Mex－文件），产生的代码就是仿真目标，且在状态流内称作 Sfun 目标,这样在仿真过程中直接调用相应的动态连接库文件，将二者组成一个仿真整体。Sfun 目标只能与 Simulink 一起使用。在产生代码前，如果还没有建立名为 sfprj 子目录，状态流会在 Matlab 的当前目录下产生一个 sfprj 子目录。状态流在产生代码的过程中使用 sfprj 子目录存贮产生的文件。

12.1 有限状态机简介

Stateflow 的仿真原理是有限状态机（finite state machine，简称 FSM）理论。为了更快地掌握 Stateflow 的使用方法，用户有必要先了解 FSM 的一些基本知识。

所谓有限状态机是指系统中存在可数的状态，在某些事件发生时，系统从一个状态转换成另一个状态，故有限状态机又称为事件驱动的系统。在有限状态机的描述中，可以设计出由一种状态转换至另一种状态的条件，并将每对可转换的状态均设计出状态迁移的事件，从而构造出状态迁移图。

Simulink/Stateflow 为用户提供了图形界面支持的设计有限状态机的方法。它允许用户建立有限的状态，用图形的形式绘制出状态迁移的条件，并使用其规定的命令设计状态迁移执行的任务，从而构造出整个有限状态机系统。

在 Stateflow 中，状态和状态转换是最基本的元素，有限状态机的示意图如图 12.1 所示。

图 12.1 中有三个（有限个）状态，这几个状态的转换是有条件的，其中有些状态之间是相互转换的，A 状态是自行转换的。在有限状态机系统中，还表明了状态迁移的条件或事件。

Stateflow 模型一般是嵌在 Simulink 模型下运行的，Stateflow 是事件驱动的，这些事件可以来自同一个 Stateflow 图中，也可以来自 Simulink。

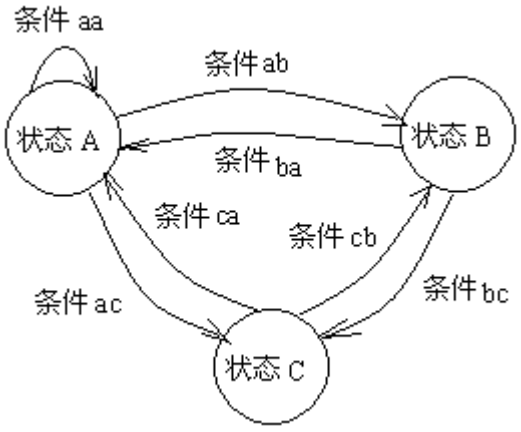


图 12.1 有限状态机示意图

12.2 Stateflow 应用基础

在 MATLAB 命令窗口键入 stateflow 命令，将打开如图 12.2.1 所示的界面。其中，sflib 窗口中有许多 Simulink 给用户提供的仿真算例。

如果用户在 MATLAB 命令窗口键入 sfnew 命令，将打开嵌入 Stateflow 模块 Charts 的 Simulink 窗口 untitled*。如图 12.2(b)所示。其中 Charts 是空白的 Stateflow 模块图标。

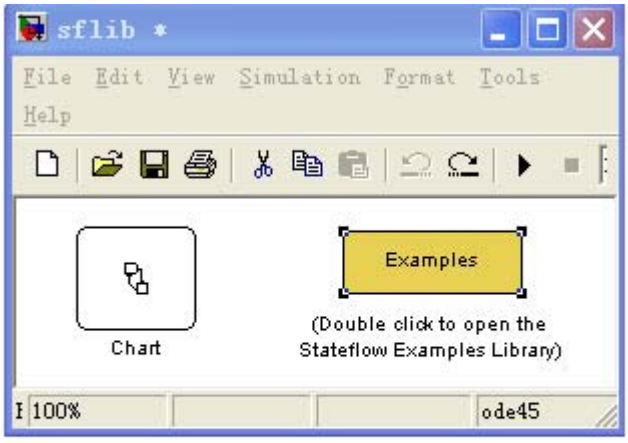


图 12.2（a） sflib 窗口

双击 untitled* 窗口中的 Stateflow 模块打开如图 12.3 所示的 Stateflow 编辑界面，用户可以在此窗口中编辑所需的 Stateflow 模型。Stateflow 提供了强大的图形编辑功能，用户可以使用它描述很复杂的逻辑关系式。

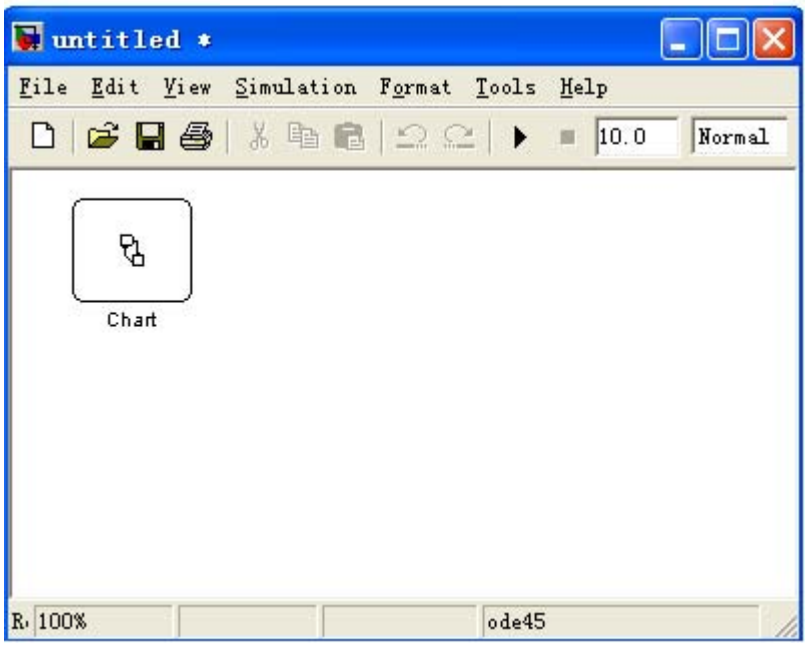


图 12.2（b） 嵌入 Stateflow 模块的 Simulink 窗口

图 12.2 Stateflow 启动窗口

Stateflow 编辑界面中点击鼠标右键，可以看到图 12.4(a)所示的快捷菜单，选择其中的 Properties（属性）菜单，可以打开图 12.4（b）所示的对话框，用户可以在此对话框中设置整个 Stateflow 模型的属性。

用户可以利用 Stateflow 编辑界面左侧的个编辑工具绘制 Stateflow 图形，下面介绍常用的编辑工具。

1、状态工具

系统的状态是指系统运行的模态。在 Stateflow 下，状态有两种行为：活动的（active）和非活动的（inactive）。单击状态工具按钮并拖动到编辑界面的空白处，即可绘制出一个状态的示意模块。用户可以在该模块右上角的问号位置填写状态的名称及动作描述，如标记为 On，本例中状态的动作描述为 entry：speed = 1，将 speed 的值赋为 1。（事实上，状态动作有很多种，不止 entry 一种，为了使读者先掌握状态流的一般知识，复杂的状态流的状态动作类型及其应用我们在随后的部分再来阐述。）激活 On 模块，使用热键 Ctrl＋C 和 Ctrl＋V 或使用 edit 菜单下的复制及粘贴命令，即可再复制一个同样的模块，将复制的模块标记改为 Off，动作描述改为 entry：speed = 0。使用该工具，可以绘制出所有需要的状态，如图 12.5 所示。

如果右击建立的状态图标，并选择快捷菜单中的 Properties 菜单项，可打开图 12.6 所示的设置状态属性的对话框。用户也可以在 Label 栏填写状态的名称和动作描述。

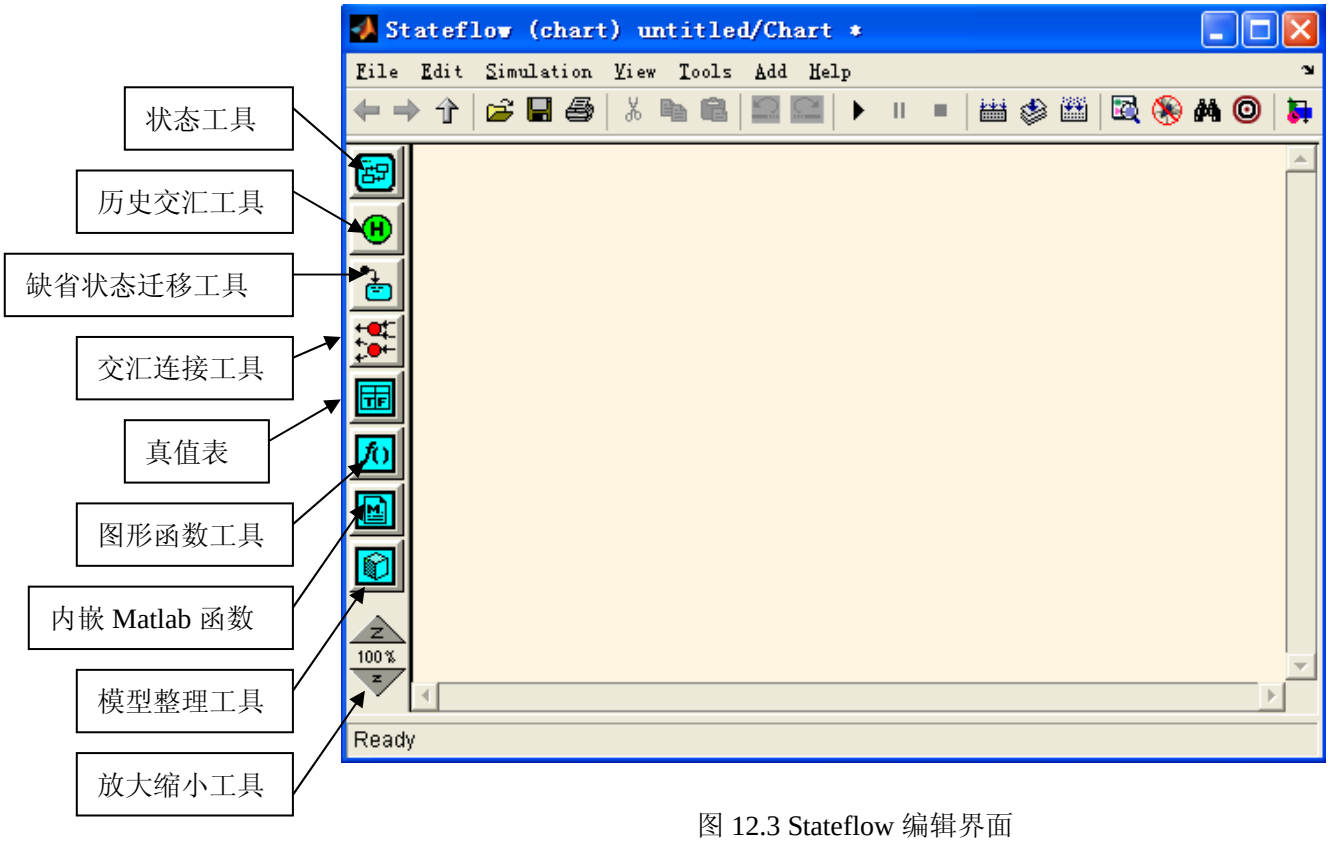
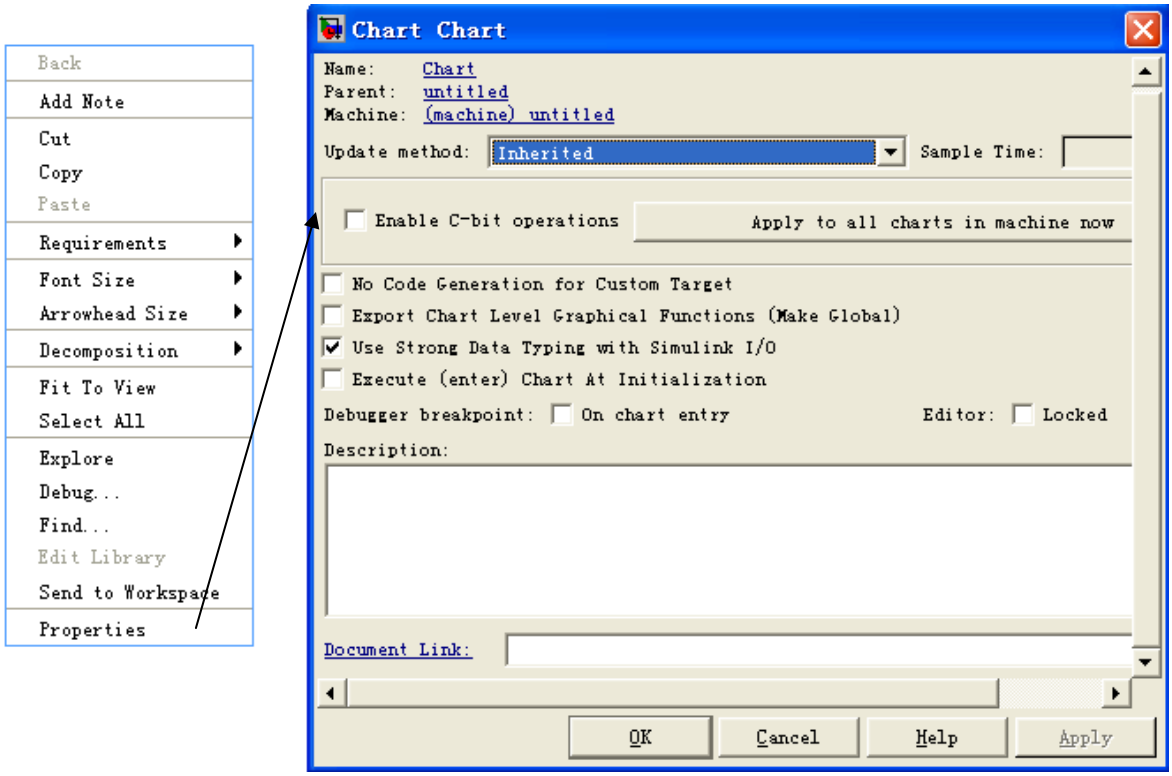


图 12.3 Stateflow 编辑界面



（a）Stateflow 快捷菜单　　（b）Stateflow 属性设置对话框

图 12.4 Stateflow 模型的属性设置

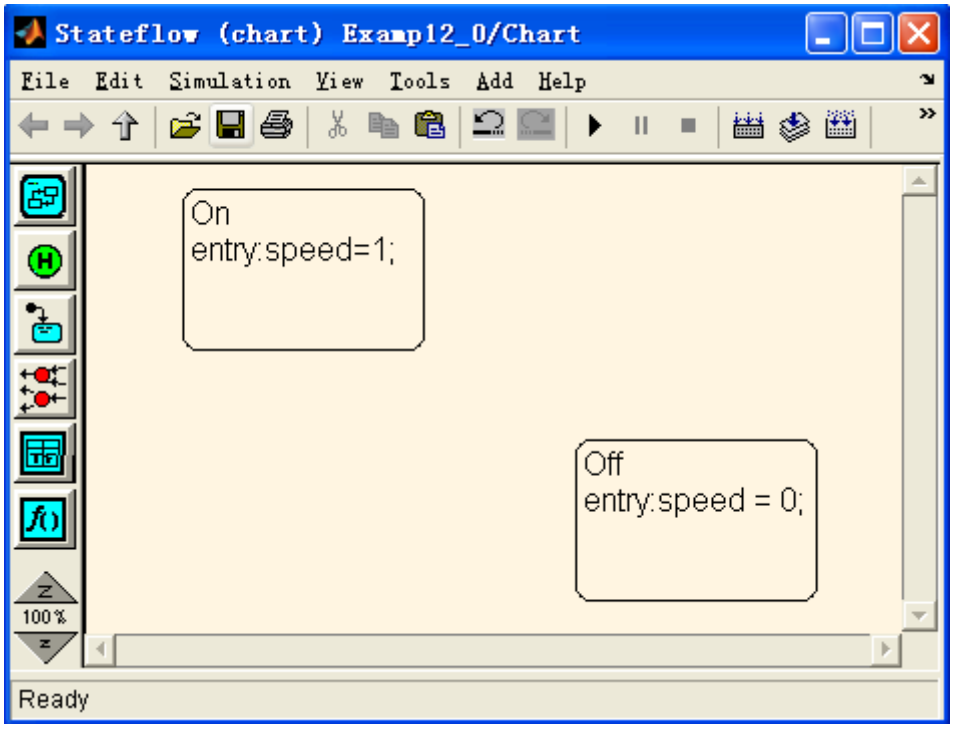


图 12.5　Stateflow 窗口的新建状态及其设置

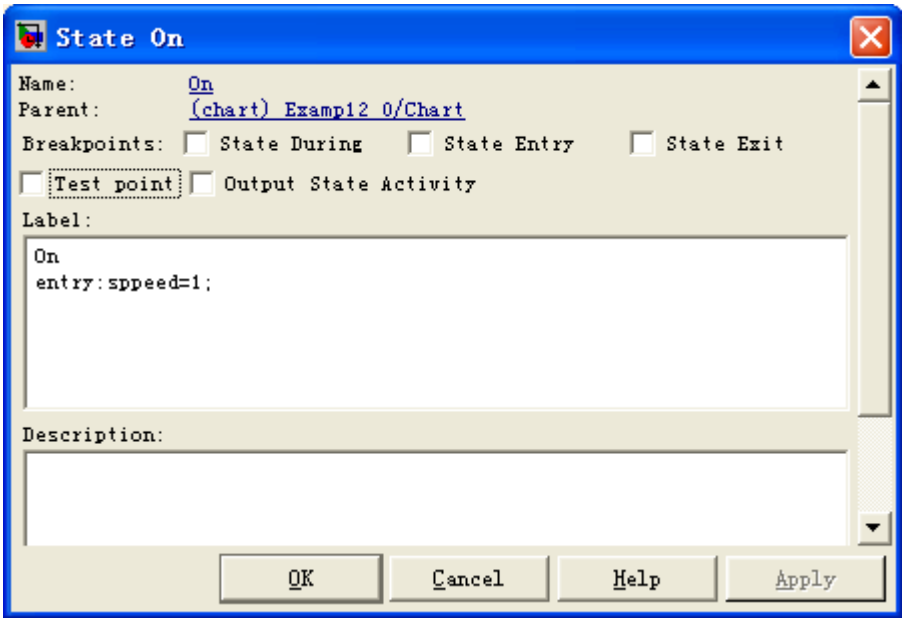


图 12.6 状态属性设置对话框

2、状态迁移关系设置

在一个状态块的边界按下鼠标键并拖动至另一个状态的边界释放，可以绘制出从一个状态到另一个状态的连线。单击此连线，在该连线上会出现一个问号，用户可以在该问号出添加状态迁移标记。状态迁移标记可以含有触发事件、迁移条件、条件动作及迁移动作，或他们中的任意组合。状态迁移

标记的一般形式是

触发事件[迁移条件关系式]{条件动作}/迁移动作

图 12.7 中的状态迁移显示了状态迁移标记的一般形式。

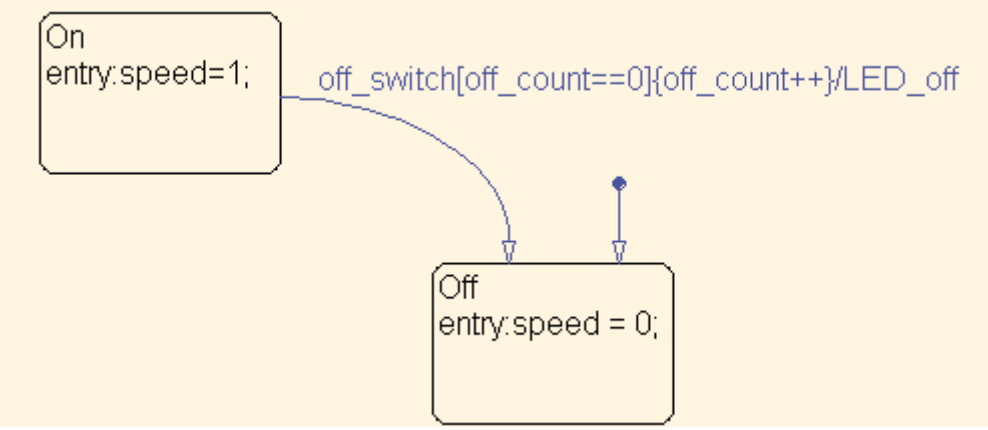


图 12.7 状态迁移标记的一般形式举例

触发事件表示只要迁移关系式是真，该触发事件可以引发状态的迁移。缺省触发事件时，任何事件均可在条件关系式为真的情况下引发状态的迁移。图 12.7 的示例中，只要条件[off_count==0]为真，事件 off_switch 可以引发状态 On 至状态 Off 的迁移。

条件关系式一般是布尔表达式，写在方括号中。该关系式为真时，使得对于特定的触发信号迁移有效。本例中，只要当条件关系式 off_count==0 为真时，发生的事件 off_switch 才可引发状态迁移。

条件动作是指当条件关系式一旦成立（即为真），就执行的动作，通常发生在迁移终点被确定有效之前。如果没有规定条件关系式，则认为条件关系式为真，即刻执行条件动作。条件动作必须写字花括号中。图 12.7 的示例中，只要条件[off_count==0]为真，即可执行条件动作 off_count++。

迁移动作是指当迁移终点已经确定有效，才执行的动作。如果迁移包含很多阶段，迁移动作只有在整个迁移通道到终点确认为有效后方可执行。迁移动作写在斜线'/'之后。图 12.7 的示例中，当条件[off_count==0]为真，发生了 off_switch 事件，迁移终点状态 Off 确认为有效，此时执行迁移动作 LED_off

图 12.8 也给出了一个简单的状态迁移标记示例。在该例中，用户可以使用状态迁移属性设置对话框对状态迁移条件进行设置。右键状态迁移连线即可打开状态迁移属性设置对话框，在 Label 栏中设置即可。请读者自行点击问号，将图中未设置触发事件的迁移事件设置为 on_switch，并保存此例。本例的两个状态迁移均采用的是事件触发，状态迁移也可以采用关系式触发，一旦状态迁移上所描述的关系式成立，则状态迁移开始启动。关系式触发的状态迁移上的关系式格式是 [关系式]，如[temp>=120]等。

3、缺省状态转移设置

缺省状态转移设置的作用是告诉 Stateflow 图形，当它开始工作时，哪个状态先处于激活状态。点击 Stateflow 图形编辑界面中的图标  ，然后将鼠标移动需要设置的

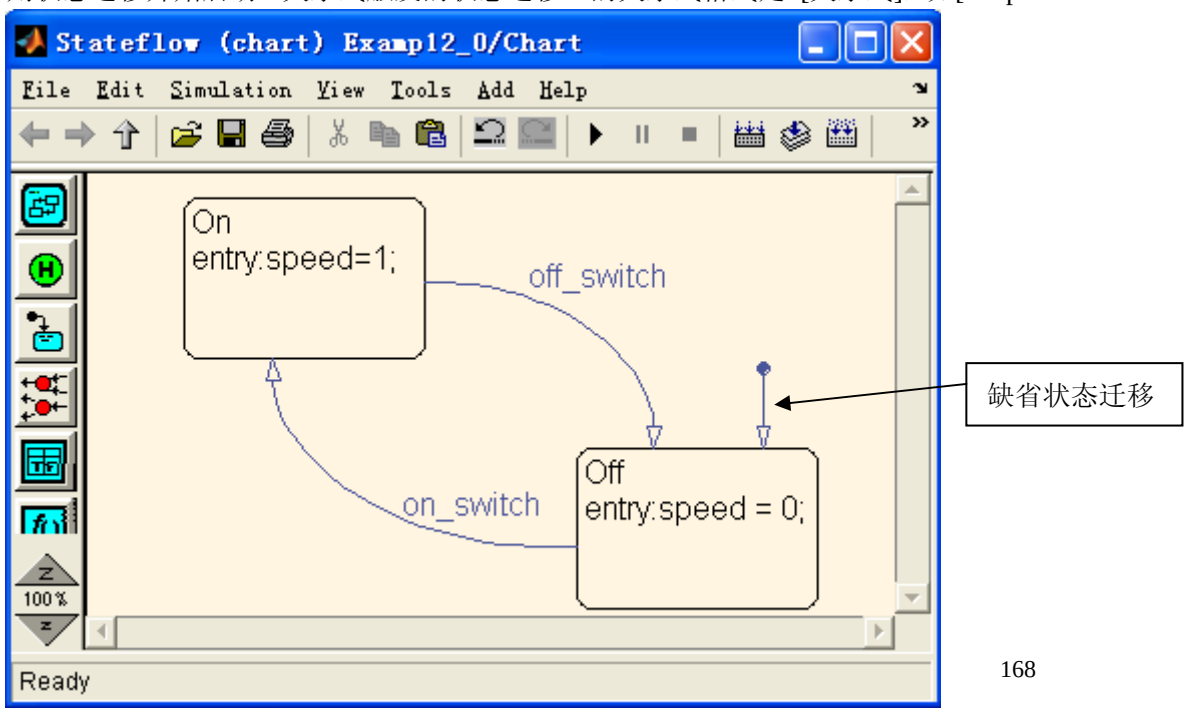


图 12.8 Stateflow 状态迁移设置

状态即可。见图 12.8。

4、 事件与数据设置

前面我们为状态迁移规定了迁移触发事件的名称，也就是说状态的迁移仅在这些事件发生的时候才开始。为了利用这些事件触发，我们必须先定义这些事件。定义 on_ switch 和 off_switch 事件需要一下几步：

a） 从 stateflow 编辑界面的 add 菜单选择 Event，并在随后弹出的下拉菜单下选择 Input from Simulink，打开事件对话框，见图 12.9；

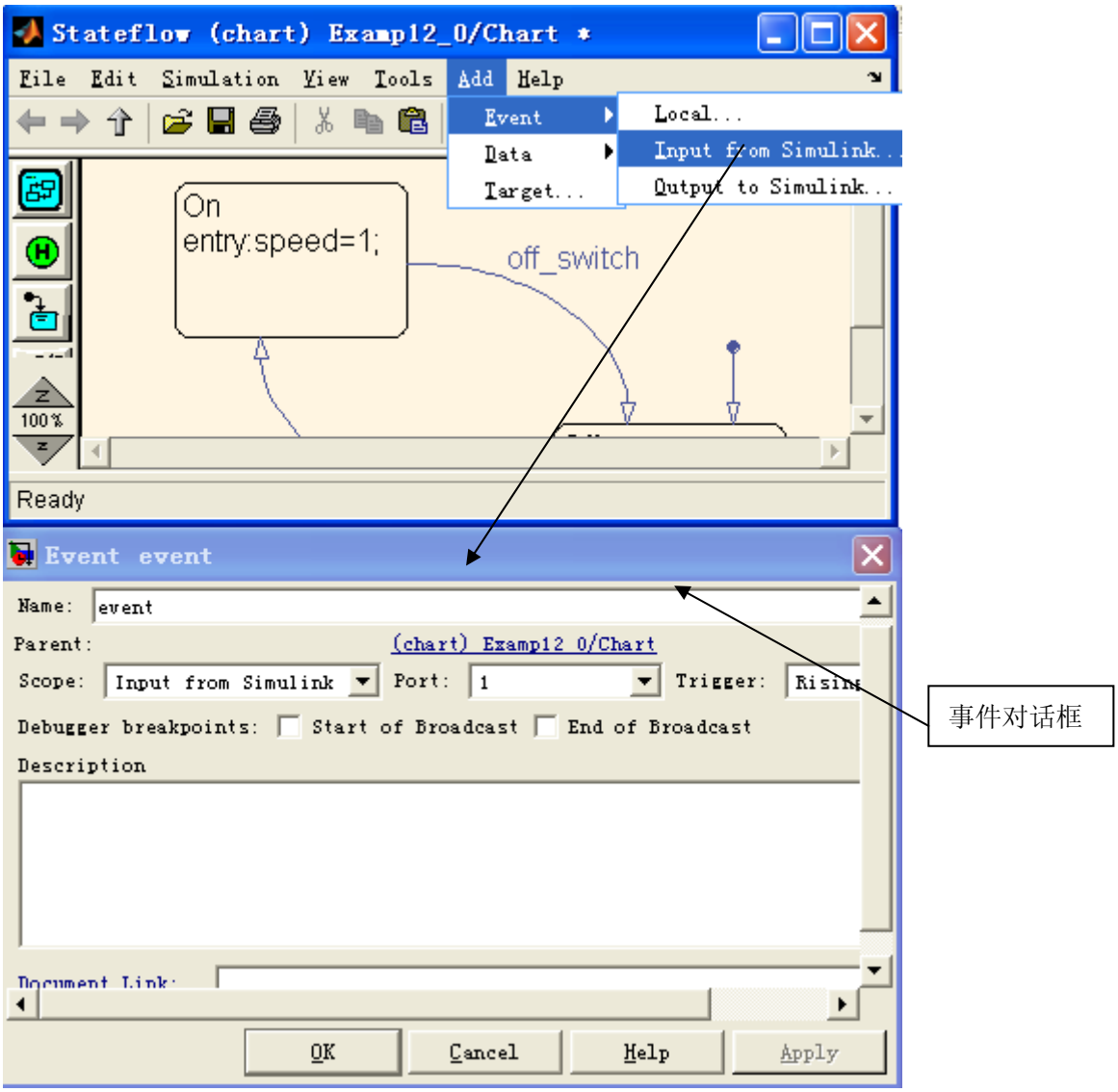


图 12.9 Stateflow 事件对话框

将事件对话框中的 Name 改为 off_switch，trigger 选择为 Falling（即下降沿触发），点击 OK 保存 off_switch 事件的设置。

b） 重复上述步骤设置 on_switch 事件，触发事件仍选择 Input from Simulink，Name 设置为 on_switch，trigger 选择为 Rising（即上升沿触发）。

注：事件的范围（Scope）有三种选项：Local 是指利用本 Stateflow 图形界面产生的触发事件； Input from Simulink 是指从 Simulink 模型引入事件至 Stateflow 图形界面；Output to Simulink 是指将 Stateflow 图形界

面产生的事件输出到 Simulink 模型中。

事件的触发方式亦有多种选择：Either、Rising、Falling 和 Function Call 四种。其中选择 Rising 或 Falling 分别指利用事件的上升沿或下降沿触发，Either 是指不管上升沿还是下降沿事件均可以触发，Function Call 是一种函数调用的触发方式。

前面我们还为状态设置了动作，如状态 On 的动作描述为 entry：speed = 1，是希望在状态 On 激活时将 speed 的值赋为 1，这个数据是要在 Simulink 模型中使用的，所以要将数据传递到 Simulink 模型中。在能够被利用之前，这个数据必须先定义。数据的定义步骤如下

从 stateflow 编辑界面的 add 菜单选择 Data，并在随后弹出的下拉菜单下选择 Output to Simulink，打开数据对话框，将图 12.10；

将数据名 Name 改为 speed，点击 OK 保存设置即可。

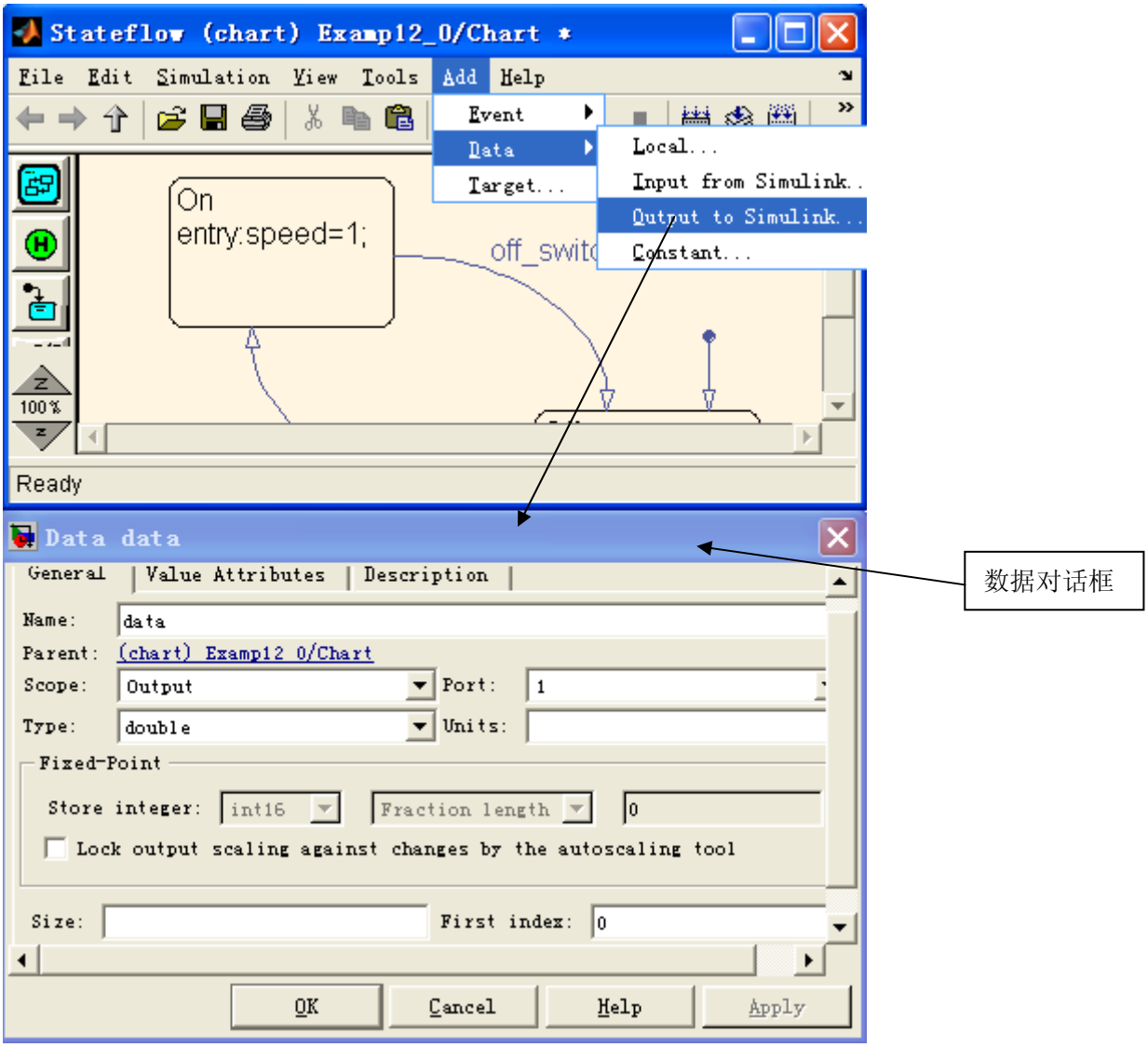


图 12.10　Stateflow 数据对话框

注意：数据范围可以设置为 Local（局部数据）、Input from Simulink（从 Simulink 模型中输入数据，但 Stateflow 图需要利用 Simulink 模型的数据时，需要将相应的数据输入到 Stateflow 图中）、Output to Simulink（向 Simulink 模型输出数据）和 Constant（常数）四种形式。数据的类型可以是 Double（双精度）、Single（单精度）、Int32（整数）及 Boolean（布尔数）等，也可以设置为 Inherited，即继承原来的设置。

这样在 Stateflow 编辑界面中，选择 Tools 菜单中的 Explore，将会打开模型管理器 Model Explorer，如图 12.11 所示。

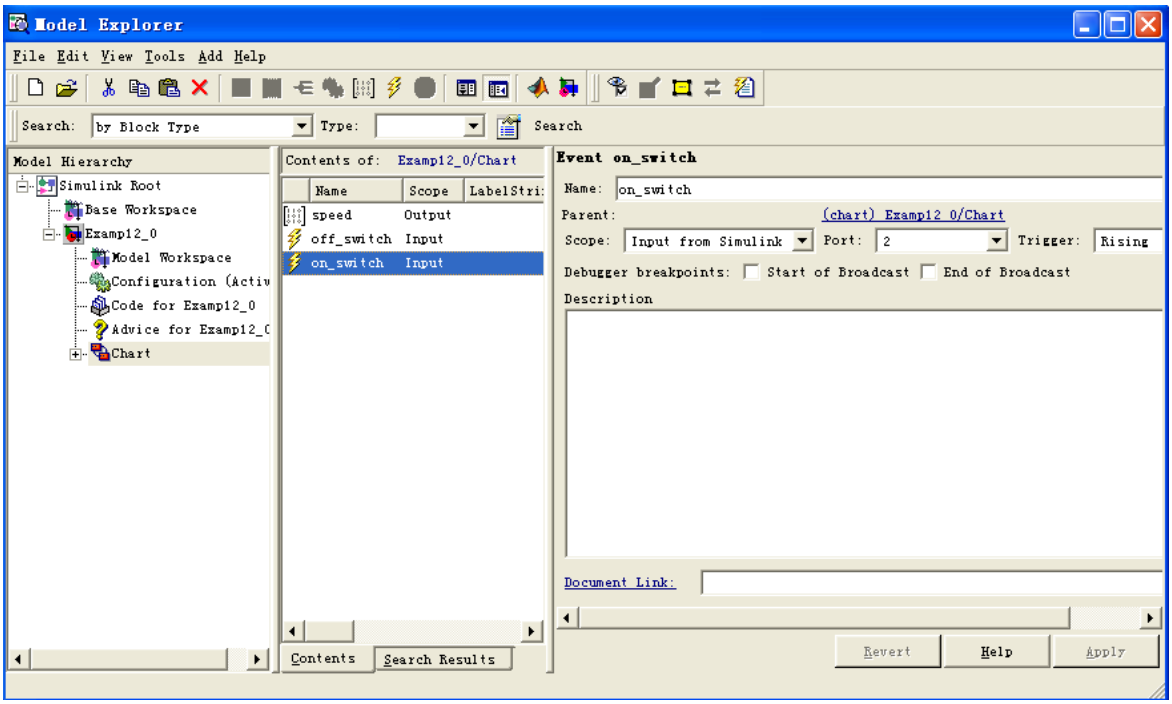


图 12.11　Model Explorer(模型管理器)

事件和数据的定义也均可以在模型管理器中进行。分别点击模型管理器工具栏上的图标 和 即可添加事件和数据，然后点击相应的内容修改事件的范围、触发方式或数据的范围及数据的类型等内容。

设置了事件和数据后，就可以从 Simulink 模型中输入事件和数据，并将 Stateflow 图执行过程中产生的数据或事件输出至 Simulink 模型中。

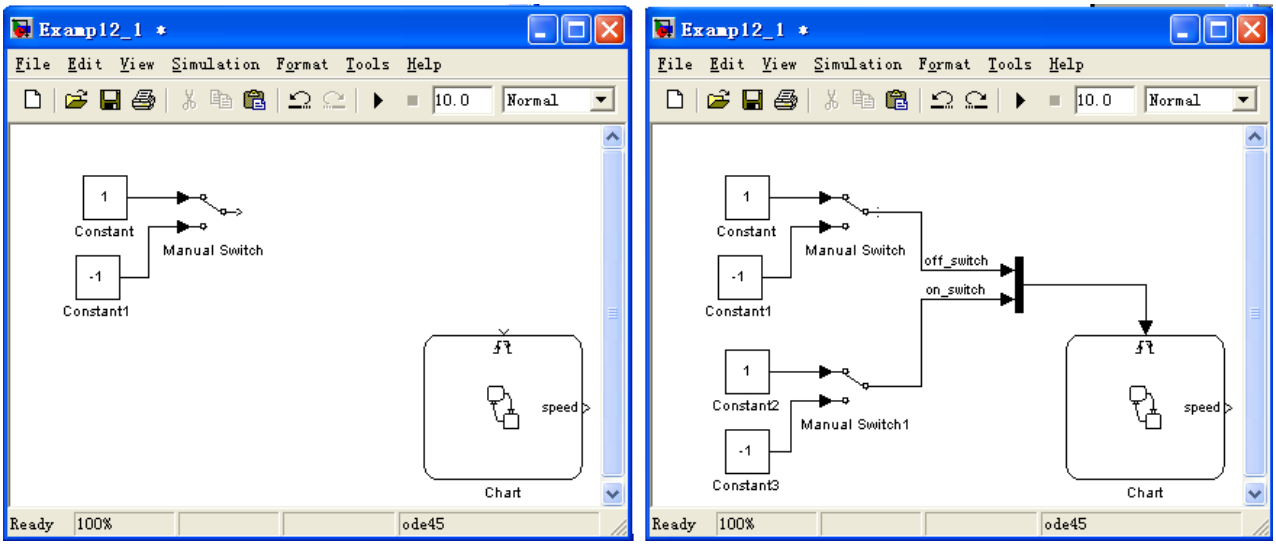
例 12.1　将前面叙述中已设置好数据和事件的 Stateflow 图存为 Examp12.1，从 Simulink 中产生上升沿和下降沿的触发事件，将 Stateflow 图状态动作产生的数据 speed 输出到 Simulink 模型中并加以显示。

解　一、建模　前面的叙述已经讲到了 Stateflow 图的建立方法，包括状态模块的绘制及其设置、状态迁移及缺省状态迁移的设置、事件和数据的设置。下面我们在 Simulink 中产生具有上升沿或下降沿的触发事件 on_switch 和 off_switch，并将数据 speed 的值在 Simulink 模型中显示出来。

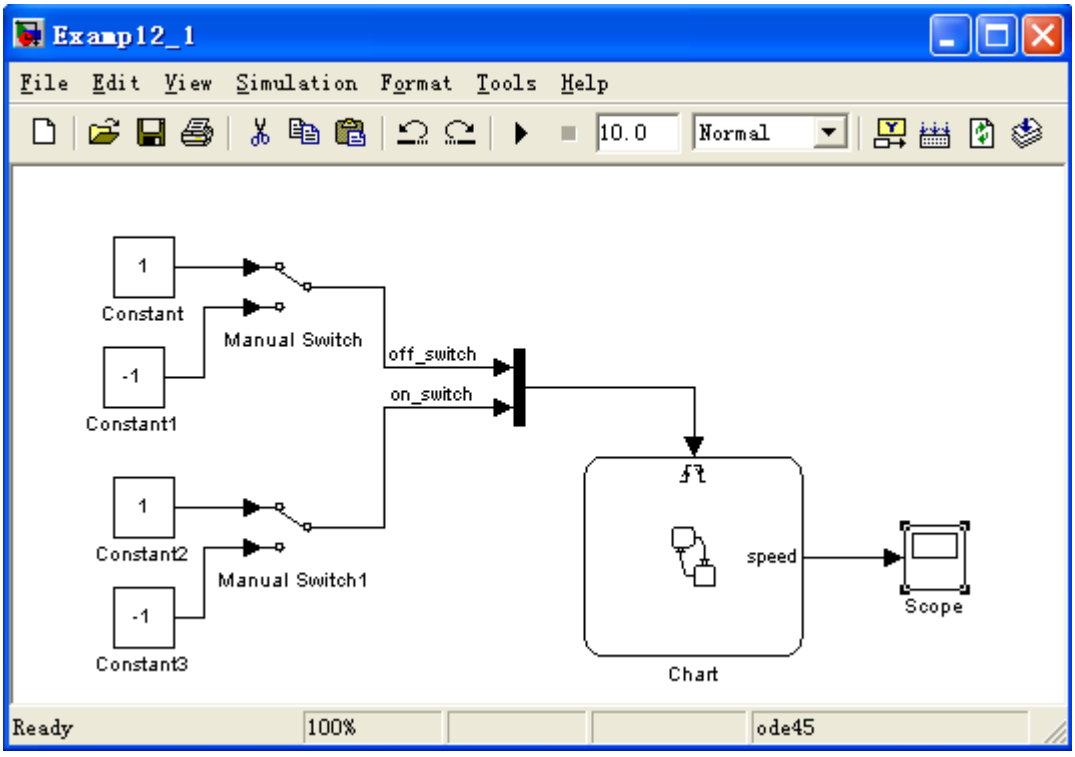
为了产生上升沿或下降沿的触发信号，需要使用 Simulink 模型库中的 Sources 库中的两个 Constant 模块及一个 Signal Routing 库中的 Manual Switch 模块，见图 12.12（a）。因为本例需要两个触发信号，而 Stateflow 模块的触发口仅有一个，所以必须使用 Signal Routing 库中的 Mux 模块组合信号。见图 12.12(b)。将 Sinks 模型库中的 Scope 模块拖进 Simulink 模型中以便显示 speed 的数值。最终的模型见图 12.12(c)。

二、仿真　缺省情况下，可以直接对含有 Stateflow 模块的 Simulink 模型进行仿真。

1）像一般的 Simulink 模型仿真一样，设置仿真参数：Simulink 模型窗口的菜单 Simulation 下选择 Configuration Parameters，打开仿真参数设置对话框，将 Stop time 设置为 inf，见图 12.13。仿真结束时间设置为 inf 表示由用户自行决定仿真结束的时刻。



(a) (b)



(c)

图 12.12　例 12.1 的建模过程

2）选择 simulation 菜单中的 start 或直接点击 Simulink 模型窗口上的图标 启动仿真

注意在启动仿真前，摆好 Simulink 模型窗口、Stateflow 图形编辑界面和 Scope 示波器的位置以便既可以设置手动开关（Manual Switch）的位置以产生所需的上升沿或下降沿触发信号，又可以同时观察 Stateflow 图形窗口中状态迁移情况及 Scope 中数据显示的变化情况。见图 12.14（a），并将手动开关置于图示位置。

启动仿真后，Stateflow 先分析 Stateflow Chart 图是否存在错误，如果有错误，会弹出一个窗口告诉用户错误的类型及位置；如果没有错误，则 Stateflow 为每个 Stateflow Chart 图自动生成 S－函数（Mex 文

件），并将生成的 S－函数放到 MATLAB 当前目录中的 sfprj 子目录下（如果 MATLAB 当前目录中没有 sfprj 子目录，MATLAB 会自动建立此目录）。仿真时，Simulink 只需调用这些 S 函数。

对于本例，在 Stateflow 对 Stateflow Chart 图分析并生产 S 函数后，系统仿真才真正开始，此时 Stateflow

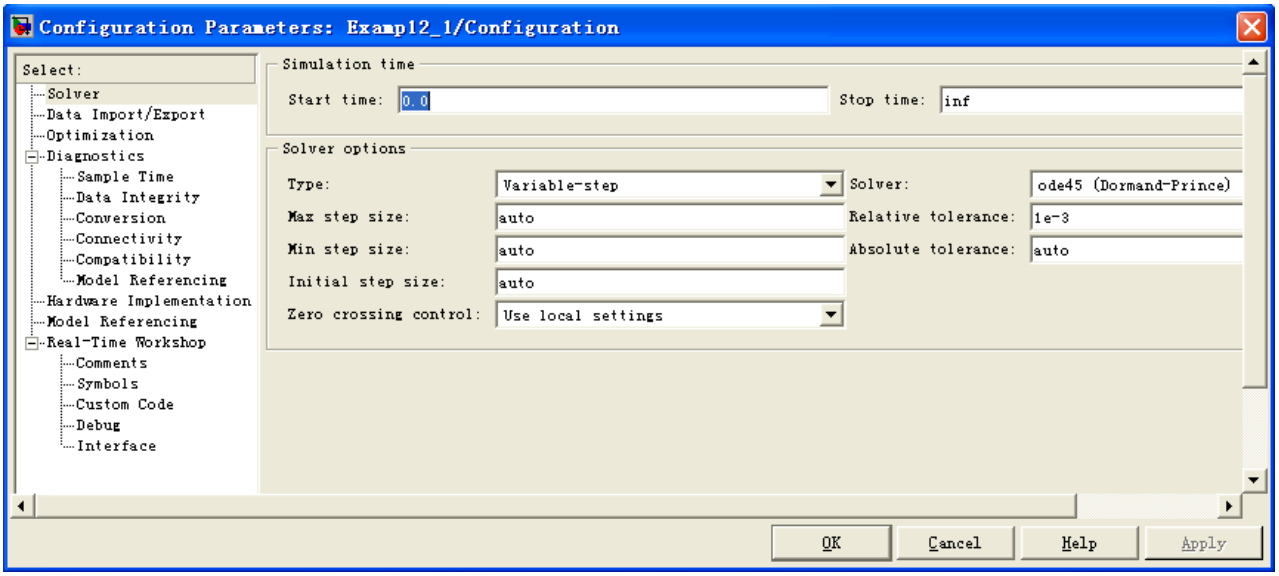


图 12.13　例 12.1 的仿真参数设置

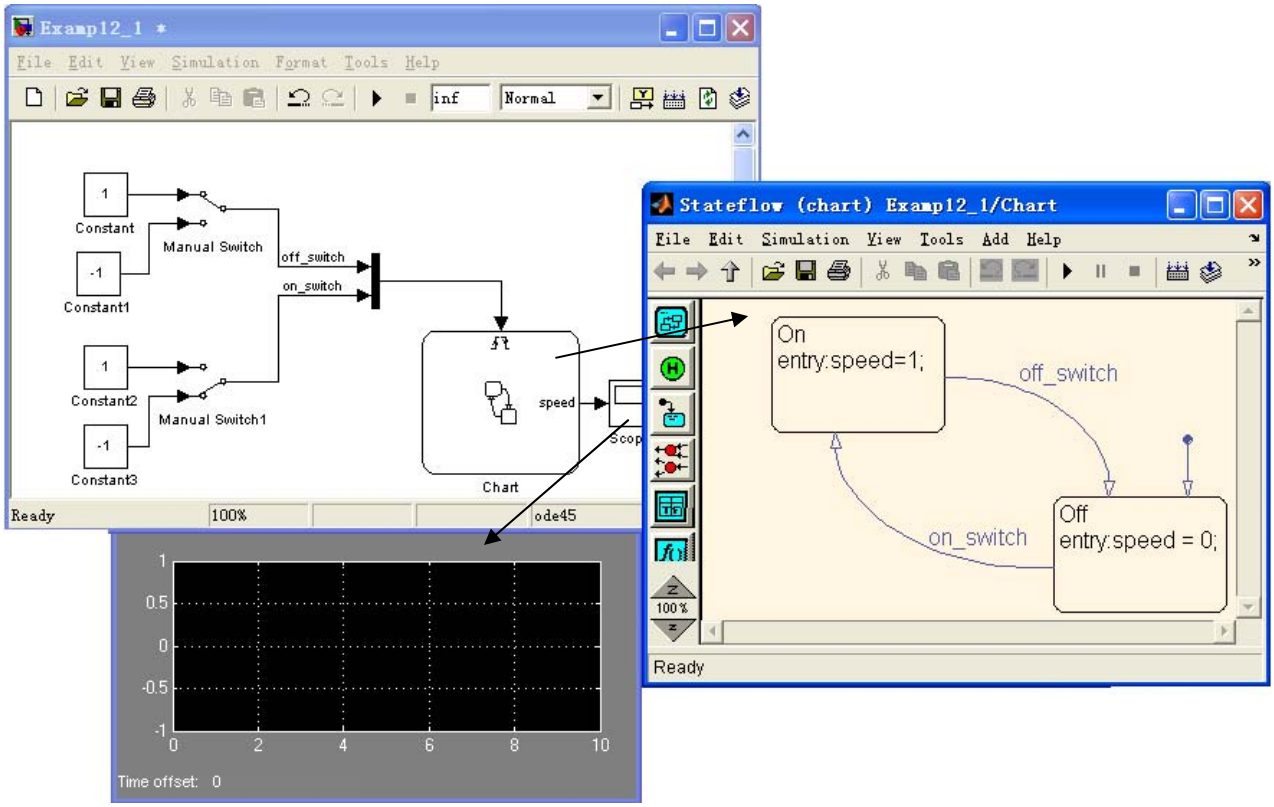
图的背景要变灰。本例中，仿真开始时 Stateflow 图处于休眠状态，没有状态是激活的。点击 Manual Switch1（或 Manual Switch），这时就产生的有效的 on_switch 上升沿触发信号（或有效的 off_switch 下降沿触发信号），这个有效的信号将 stateflow 图唤醒，stateflow 图开始判断应该如何动作。因为有一个缺省状态迁移，所以 Stateflow 图先启动缺省状态迁移，缺省状态迁移信号线变粗，然后激活 Off 状态，Off 状态的边框变粗， sleep 输出数值是 0，见图 12.14(b)。改变 Manual Switch1 开关位置以产生一个具有上升沿的 on_switch 信号，此时 Stateflow 图中的 Off 状态已经处于激活状态，on_switch 也有效，故产生状态迁移，on_switch 状态迁移线先变粗，然后激活状态 On，状态 On 的边框随之变粗。见图 12.14(c)。改变 Manual Switch 开关位置以产生一个下降沿的 off_switch 信号，此时 Stateflow 图中的 On 状态是激活状态，off_switch 触发信号有效，将产生从状态 On 到状态 Off 的迁移，off_switch 状态迁移线先变粗，然后激活状态 Off，状态 Off 的边框随之变粗。见图 12.14(d)。

5、 交汇连接设置

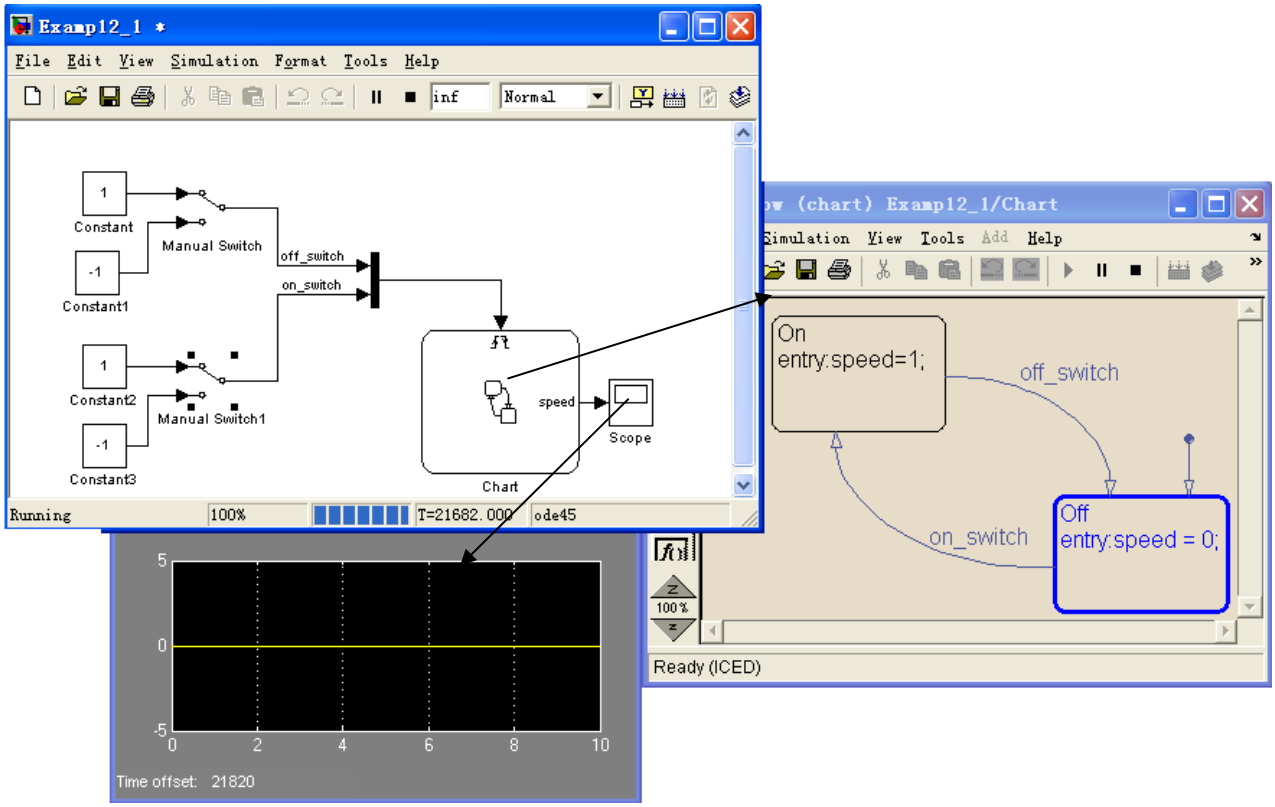
使用 Stateflow 编辑界面中的交汇连接工具  可以产生交汇连接点。交汇连接点主要用来处理状态迁移过程中的迁移信号的分离和汇合的。用户只需激活交汇连接工具  ，然后将鼠标移动适当的位置单击鼠标即可设置一个交汇连接点。我们通过下面的例子讲述交汇连接工具如何进行信号的汇合及分离的。

例 12.2　在例 12.1 的基础上，我们修改系统模型，给系统增加一个温度传感器，使得其每 0.5 秒测量一次温度，系统需要根据温度数值的大小确定输出 speed 值。这种情况下，Simulink 模型需要给 Stateflow 模块输入数据 temp，Stateflow 图中需要根据 temp 值的大小判断需要激活那个状态。

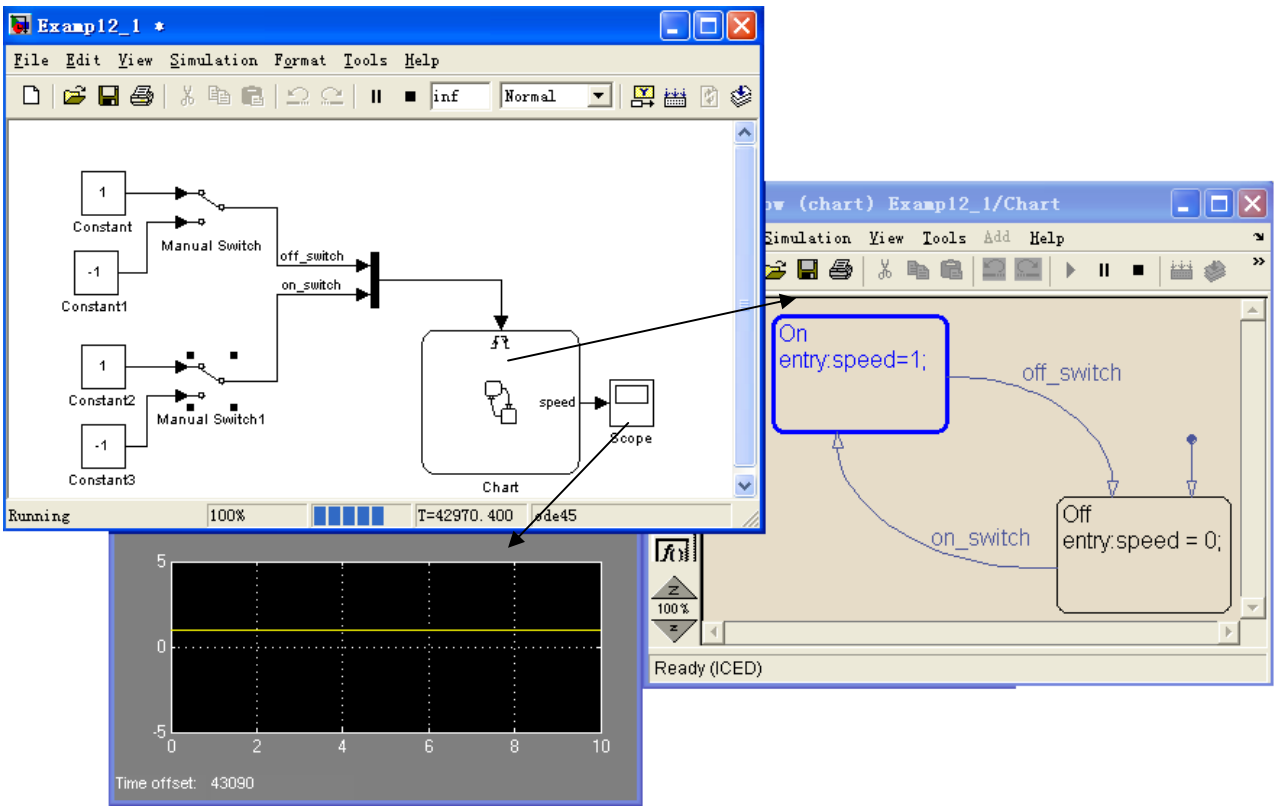
解　一、建模　在例 12.1 的基础上，我们修改系统模型如图 12.15 所示。


Examp12_1 *
File Edit View Simulation Format Tools Help
inf
Normal
Constant
Manual Switch
Constant1
off_switch
on_switch
Constant2
Manual Switch1
Constant3
Chart
speed
Scop
Ready
100%
ode45
Time offset: 0
Stateflow (chart) Examp12_1/Chart
File Edit Simulation View Tools Add Help
On
entry:speed=1;
off_switch
Off
entry:speed = 0;
on_switch
100 %
Ready

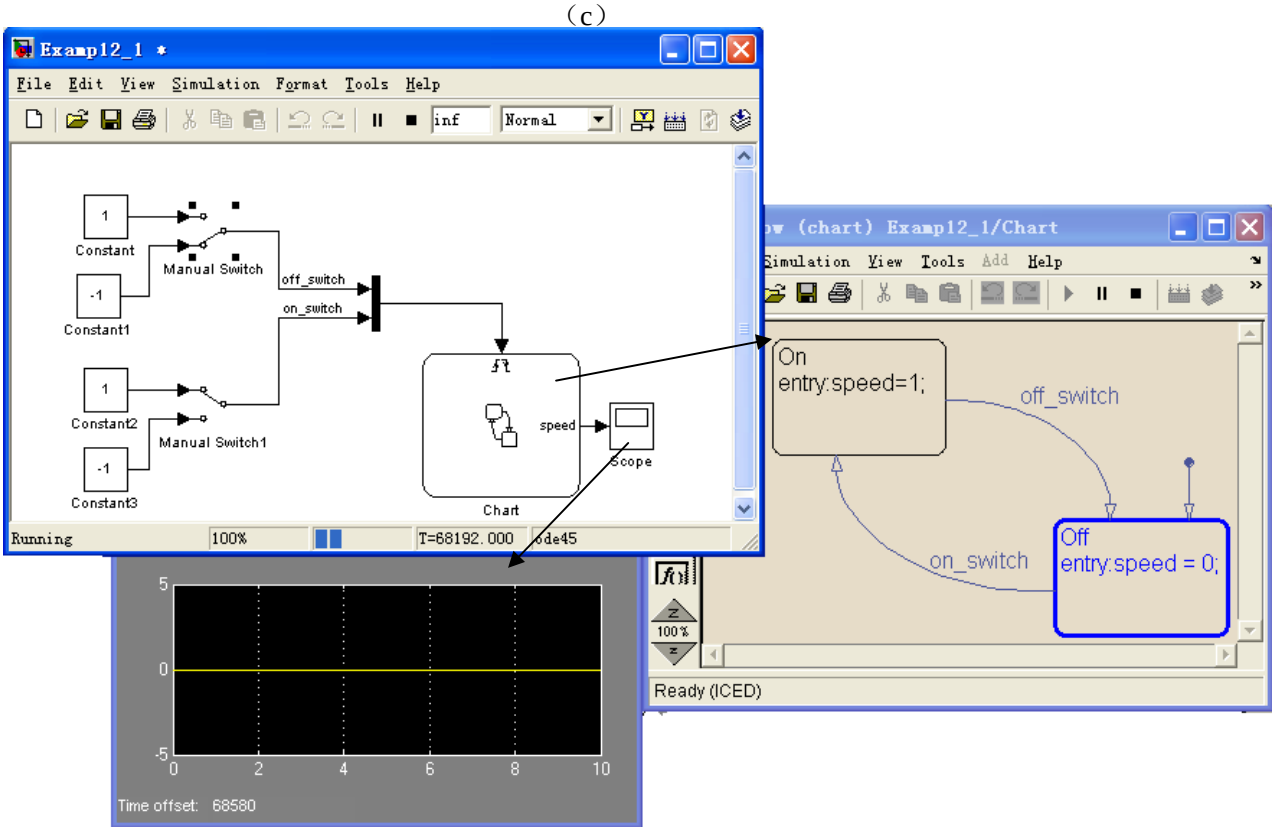
(a)


Examp12_1 *
File Edit View Simulation Format Tools Help
inf
Normal
Constant
Manual Switch
Constant1
off_switch
on_switch
Constant2
Manual Switch1
Constant3
Chart
speed
Scope
Running
100%
T=21682.000
ode45
Time offset: 21820
(chart) Examp12_1/Chart
Simulation View Tools Add Help
On
entry:speed=1;
off_switch
Off
entry:speed = 0;
on_switch
100 %
Ready (ICED)

（b）



（c）



(d)

图 12.14　例 12.1 仿真过程 Stateflow 图及其输出结果

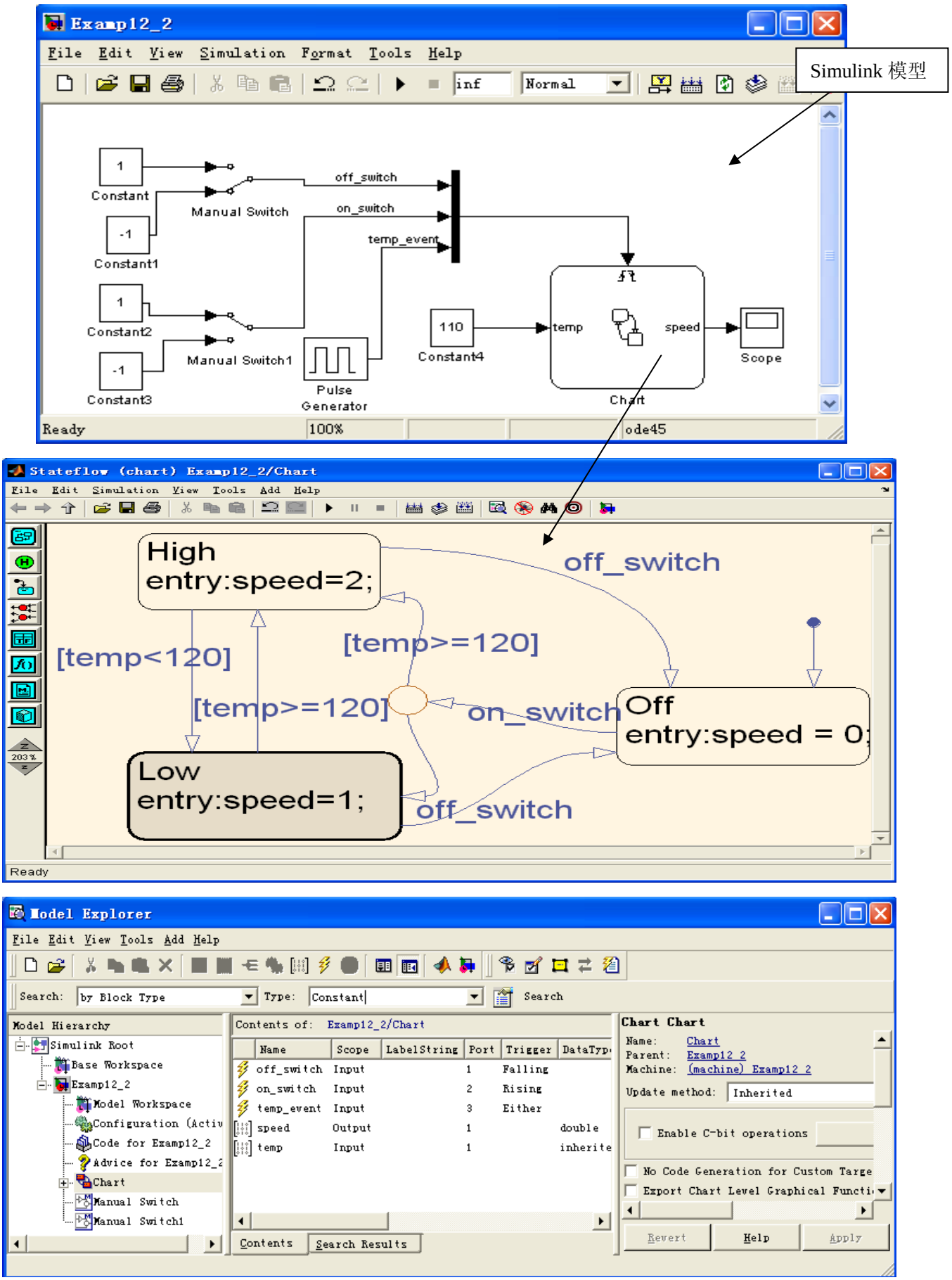


图 12.15 例 12.2 的 Simulink 模型、Stateflow 图及其模型管理器的事件、数值设置

在 Simulink 模型中，用 Constant4 模块模拟测量的温度值，开始时设置为 110。Pulse Generator 产生的周期和幅值均为 1 的方波信号用来触发 Stateflow 图，每 0.5 秒判断一次温度值的大小。实际上，Pulse Generator 产生的方波信号可以模拟温度测量的采样周期。

Stateflow 图中，请读者按照图 12.15 所示建立例 12.2 的 Stateflow 图。此 Stateflow 图含有一个交汇连接工具。此 Stateflow 图的数值 temp 是需要从 Simulink 模型中输入的，除此之外，还需利用来自 Simulink 模型中的 temp_event 事件以触发此 Stareflow 图每 0.5 秒读一次 temp 数值并判断是否应该状态迁移。因而在例 12.1 的基础上，系统中还需添加一个 Input from Simulink 的事件 temp_event 和一个 Input from Simulink 的数值 temp。将事件 temp_event 的触发沿设置成 either。添加后，选择 Stateflow 图中的 Tools 菜单下的 Explorer，打开模型管理器 Model Explore，其中的事件和数值设置应如图 12.15 所示。

二、 仿真

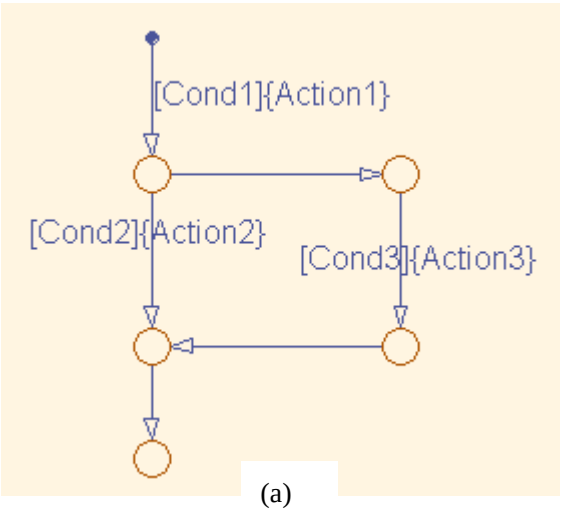
在 Simulink 模型窗口中，选择 Simulation 菜单下的 Configuration Parameters，将仿真终止时间设置为 inf。

选择 Simulation 菜单下的 Start，启动仿真。仿真开始时，Simulink 先分析 Stateflow 图，如没有错误，就按照 Stateflow 图自动生成 S－函数，并存在当前目录下的 sfprj 子目录下。这时真正的仿真才开始。

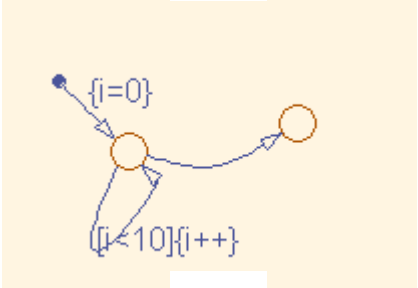
本例开始时，Stateflow 图处于休眠状态，没有状态是激活的。Stateflow 图每 0.5 秒会收到一个事件 temp_event，当接到第一个 temp_event 事件时，Stateflow 图就被唤醒了，此时 Stateflow 图判断应该产生缺省状态迁移，见图 12.27(a)；激活状态 Off，speed 输出为 0，见图 12.27(b)；改变 Manual Switch1 的开关位置，以产生一个上升沿的 on_switch 触发信号，Stateflow 产生状态 Off 到交汇连接点间的状态迁移，到了交汇连接点后，Stateflow 又判断了到 temp 值小于 120，因而又产生交汇连接点到状态 Low 的状态迁移，此后激活状态 Low，执行 Low 的动作，将 speed 值置为 1，见图 12.27(c)（d）。前面讲了，Stateflow 图每 0.5 秒会收到一个 temp_event 事件，如果用户这时将 Simulink 模型中的 Constant4 模块的值改为 130，在改值后 Stateflow 图接到第一个 temp_event 事件时，Stateflow 判断条件转移关系式[temp>=120]成立，即会执行从 Low 状态到 High 状态的状态迁移，激活 High 状态，执行 Hign 状态的动作，将 speed 值置为 2，见图 12.27(e)（f）。如果用户改变 Manual Switch 的开关位置，以产生一个下降沿的 off_switch 触发信号，则再次发生从 High 状态至 Off 状态的状态迁移，激活 Off 状态，将 speed 值置为 0，，见图 12.27(g)（h）。改变 Manual Switch1 的开关位置，以产生一个上升沿的 on_switch 触发信号，Stateflow 产生状态 Off 到交汇连接点间的状态迁移，到了交汇连接点后，Stateflow 又判断了到 temp＝130，大于 120，因而又产生交汇连接点到状态 High 的状态迁移，此后激活状态 High，执行 High 的动作，将 speed 值置为 2，见图 12.27(i)（j）。本例中状态迁移的过程及迁移过程中的信号流向详见图 12.17。读者也可以自行调解参数 temp 值的大小及 Manual Switch、Manual Switch1 的位置，以观察此例的状态迁移情况。

例 12.2 中仅仅是简单的使用了一下交汇连接工具，事实上，交汇连接工具的合理使用可以完成非常复杂的逻辑关系。如图 12.16（a），实现如下 if_then 判断功能

```
if [Cond1] {
Action1
if [Cond2] {
Action2
}elseif [Cond3]{
Action3
```



(a)



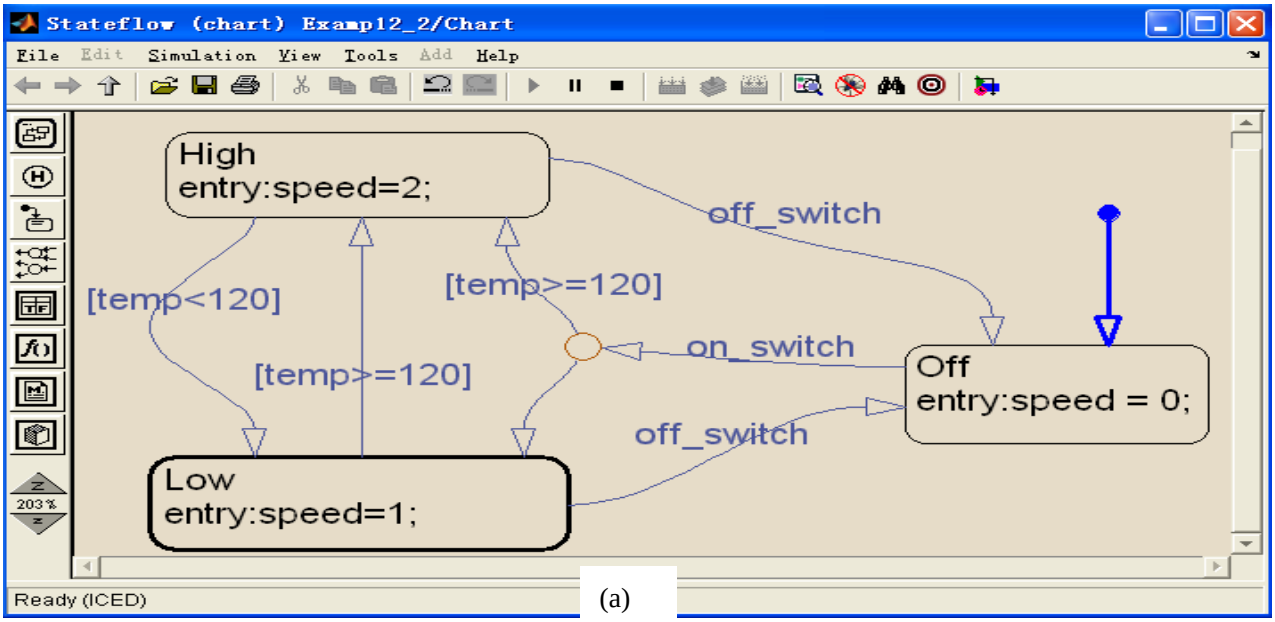
(b)

图 12.16　交汇连接工具完成 if_then 功能

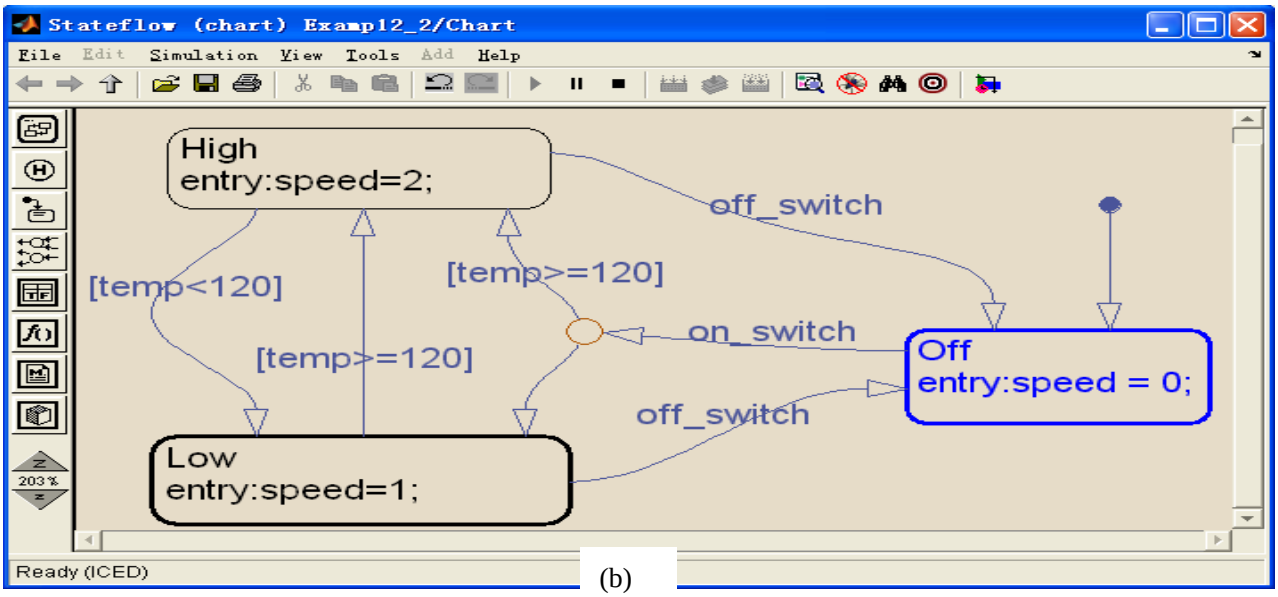
}

}

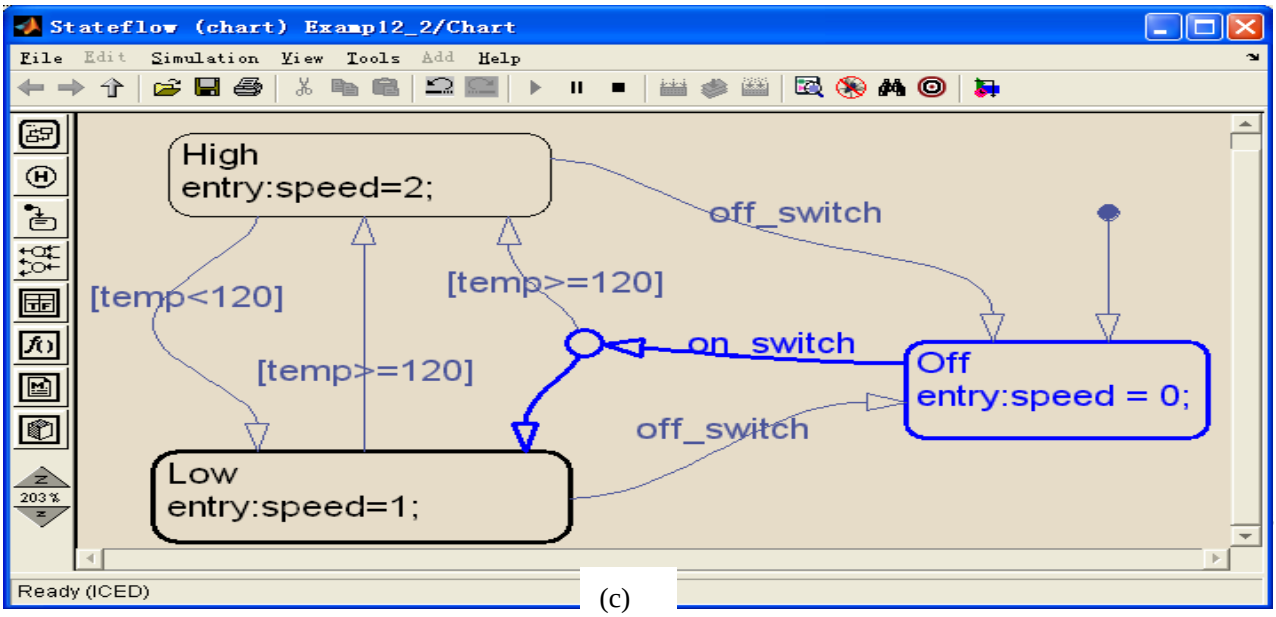
利用交汇连接工具除了可以实现 if_then 判断功能，还可实现 For 循环功能等，见图 12.16(b)。



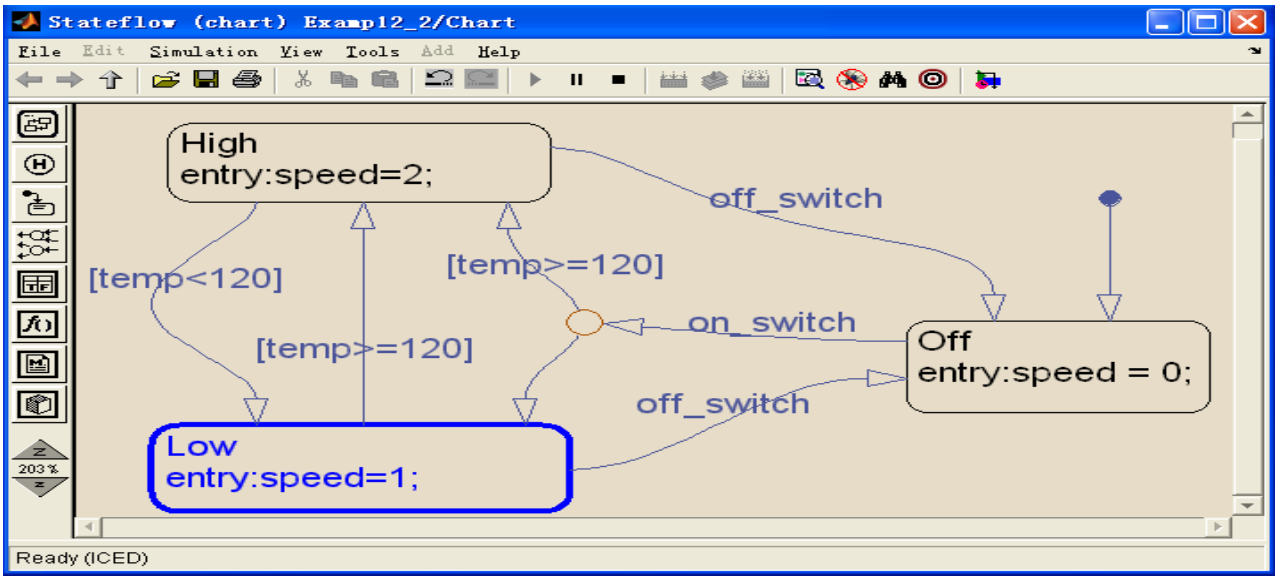
(a)



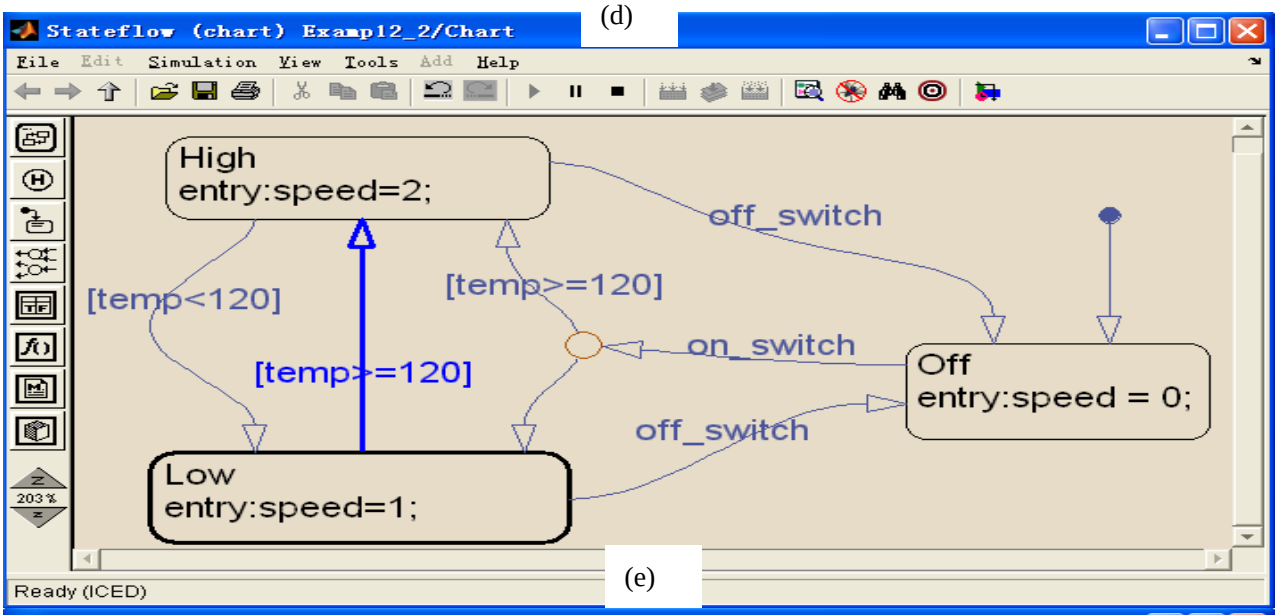
(b)



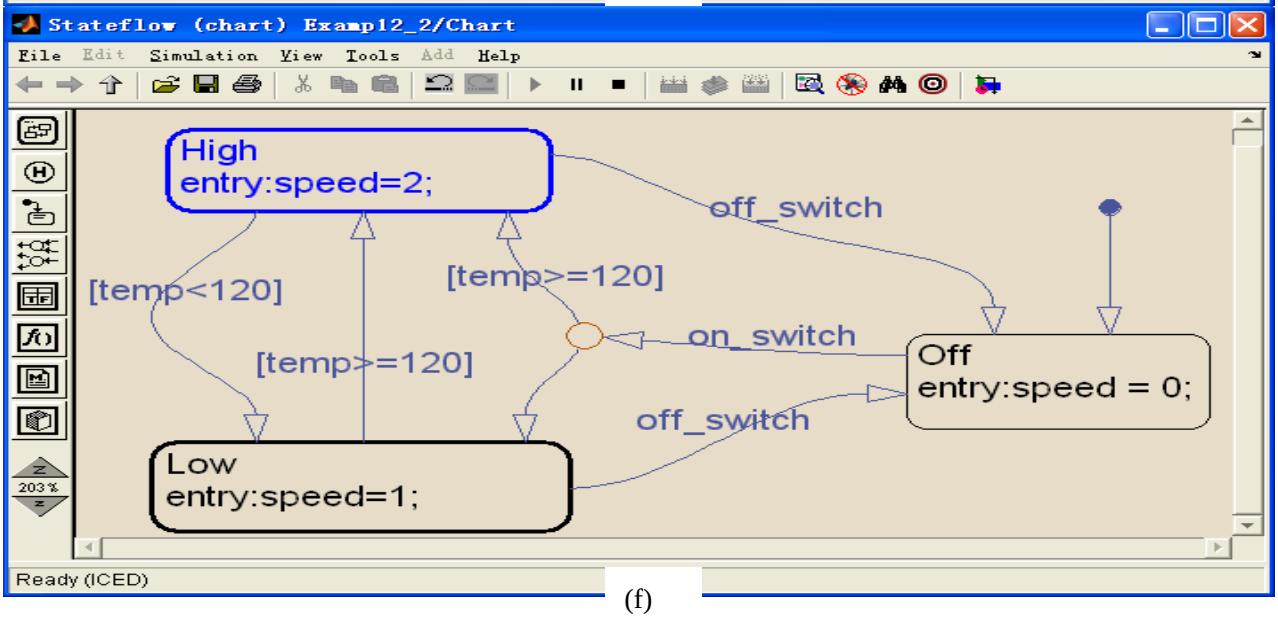
(c)


Stateflow (chart) Examp12_2/Chart
File Edit Simulation View Tools Add Help
High
entry:speed=2;
off_switch
[temp<120]
[temp>=120]
on_switch
[temp>=120]
Off
entry:speed = 0;
off_switch
Low
entry:speed=1;
203%
Ready (ICED)

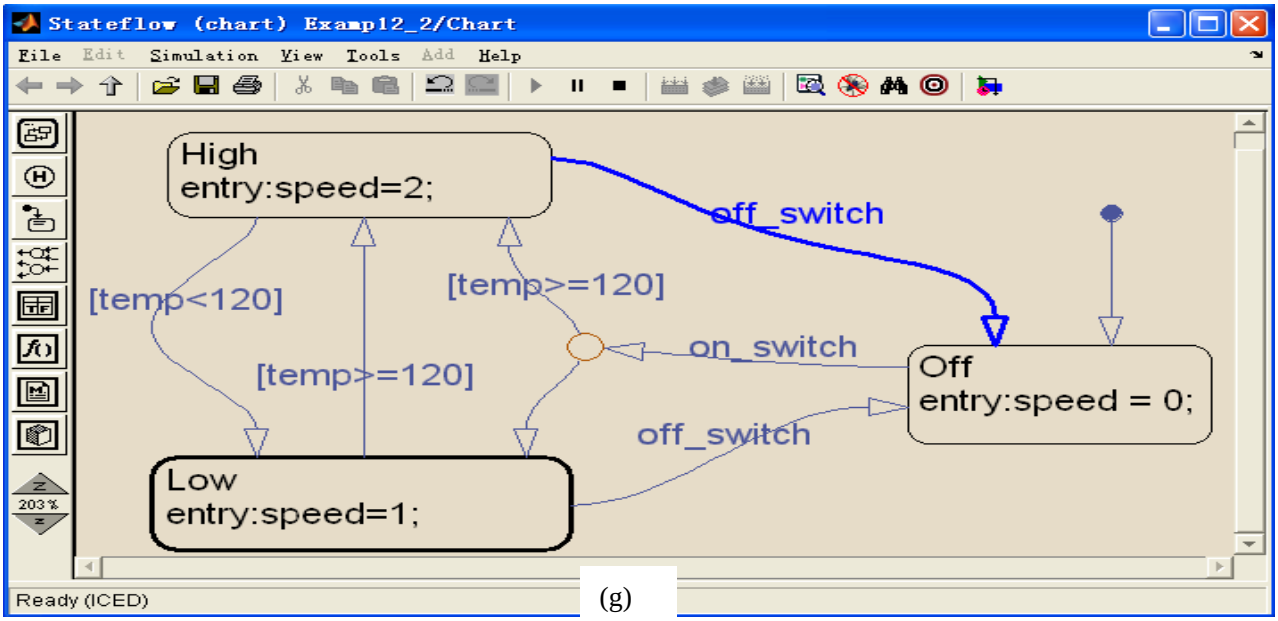
(d)


Stateflow (chart) Examp12_2/Chart
File Edit Simulation View Tools Add Help
High
entry:speed=2;
off_switch
[temp<120]
[temp>=120]
on_switch
[temp>=120]
Off
entry:speed = 0;
off_switch
Low
entry:speed=1;
203%
Ready (ICED)

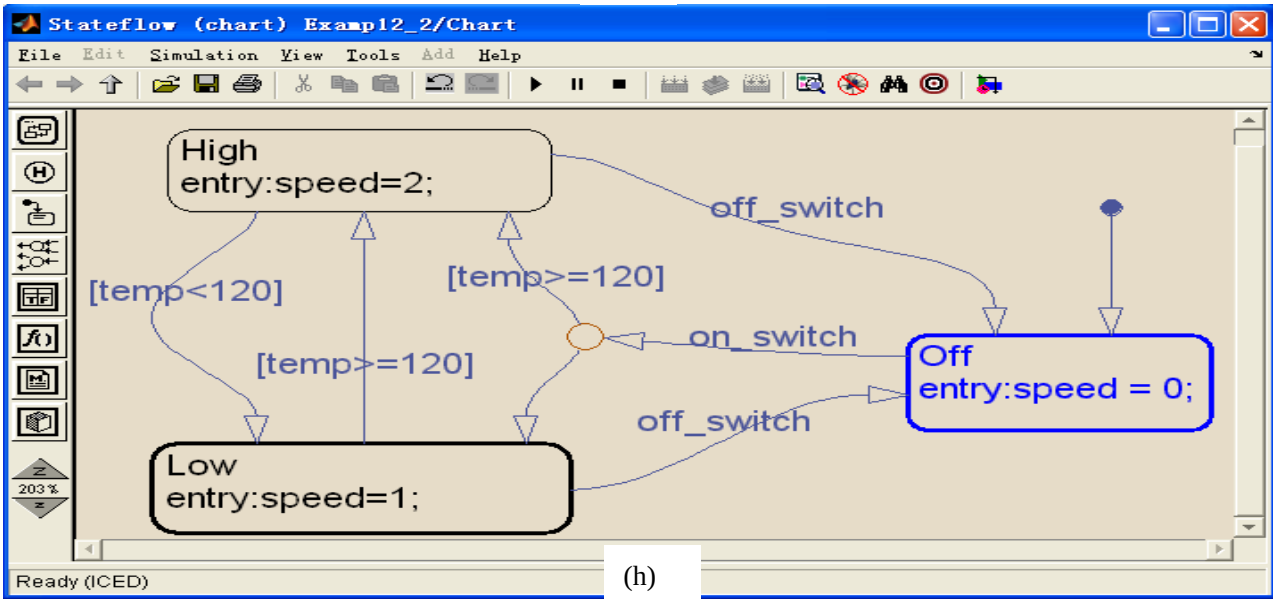
(e)


Stateflow (chart) Examp12_2/Chart
File Edit Simulation View Tools Add Help
High
entry:speed=2;
off_switch
[temp<120]
[temp>=120]
on_switch
[temp>=120]
Off
entry:speed = 0;
off_switch
Low
entry:speed=1;
203%
Ready (ICED)

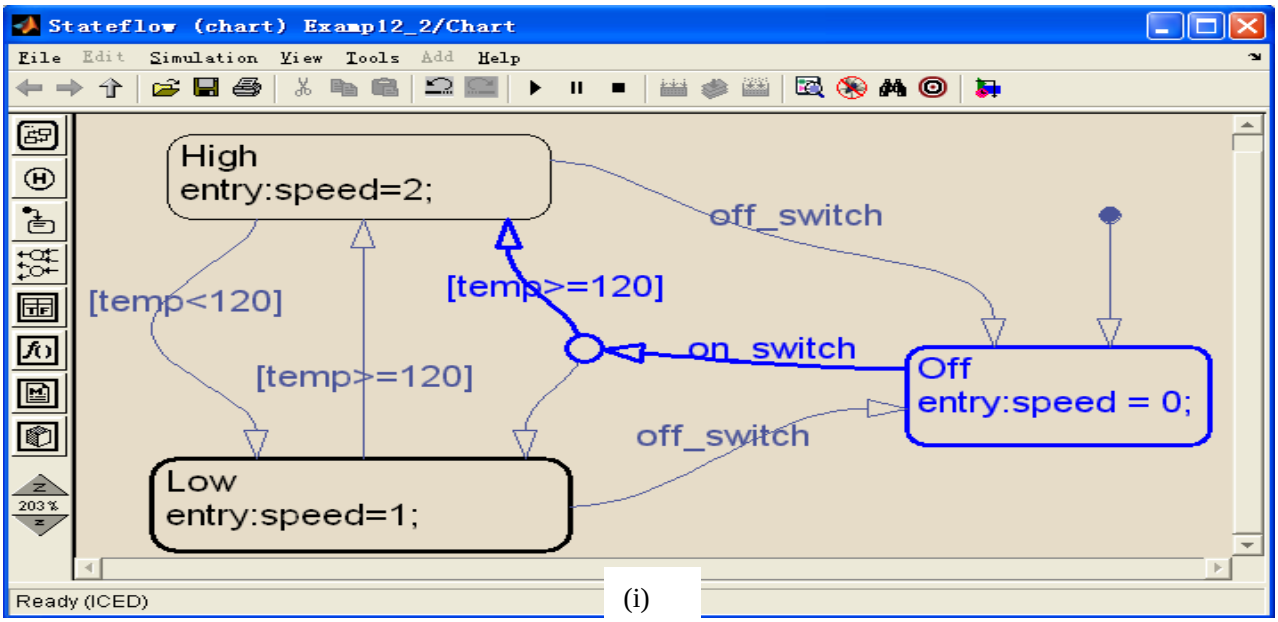
(f)


Stateflow (chart) Examp12_2/Chart
File Edit Simulation View Tools Add Help
High
entry:speed=2;
off_switch
[temp<120]
[temp>=120]
on_switch
[temp>=120]
Off
entry:speed = 0;
off_switch
Low
entry:speed=1;
203%
Ready (ICED)

(g)


Stateflow (chart) Examp12_2/Chart
File Edit Simulation View Tools Add Help
High
entry:speed=2;
off_switch
[temp<120]
[temp>=120]
on_switch
[temp>=120]
Off
entry:speed = 0;
off_switch
Low
entry:speed=1;
203%
Ready (ICED)

(h)


Stateflow (chart) Examp12_2/Chart
File Edit Simulation View Tools Add Help
High
entry:speed=2;
off_switch
[temp<120]
[temp>=120]
on_switch
[temp>=120]
Off
entry:speed = 0;
off_switch
Low
entry:speed=1;
203%
Ready (ICED)

(i)

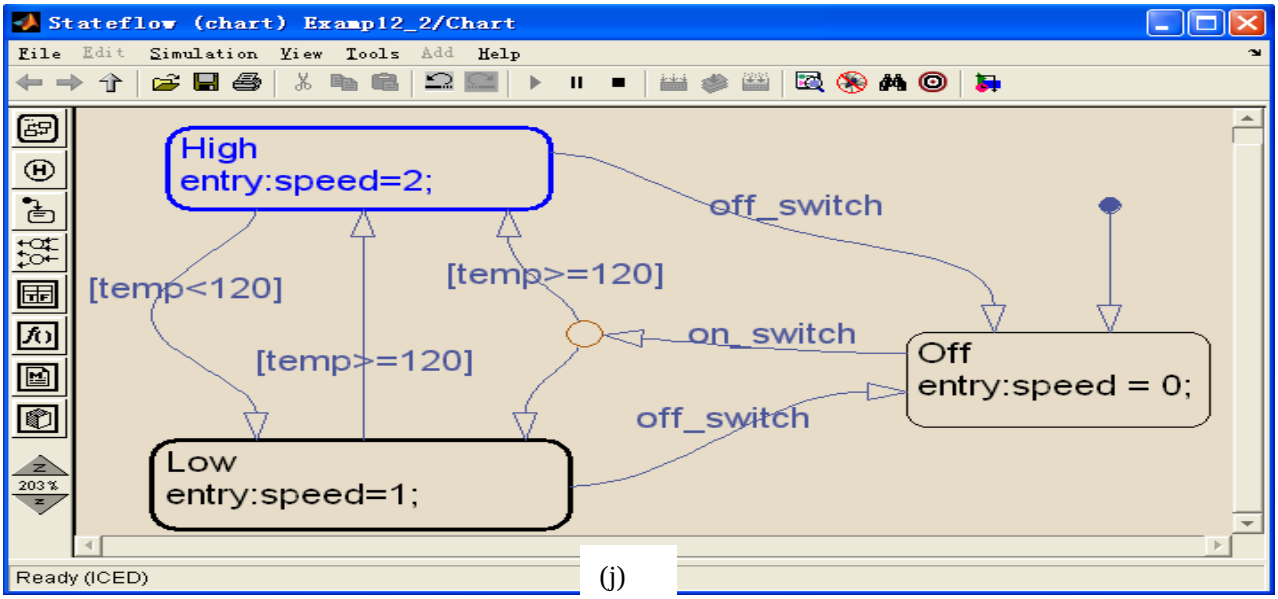


图 12.17　例 12.2 仿真时的状态迁移过程示意图

6、图形函数的设置及其调用

例 12.2 的 Stateflow 图中多次利用了条件关系式[temp>=120]。对于这种多次使用的关系式，我们可以设置一个图形函数 Function，使用时调用这个函数即可。

状态流的图形函数是使用交汇连接工具和状态迁移工具绘制的状态流图形。用户可以建立一个图形函数，在里面加入流程图，然后在状态的动作和迁移过程中反复调用。因为调用函数时，函数必须执行完全，所以图形函数中不能含有状态。一个最小的图形函数至少要包含一个缺省状态迁移和一个终止的交汇连接工具。

要在一个 Stateflow 图添加一个图形函数 Function，只需点击 Stateflow 图中的图形函数工具，移动鼠标至 Stateflow 图中的适当位置，再点击一下鼠标左键即可。在图形函数 function 后写入函数的返回变量及函数名，格式为：返回形参＝函数名（形参）；回车后即可建立该图形函数。一旦建立了图形函数，用户可以在状态流的状态动作和状态迁移中反复调用它，调用的格式与函数的格式完全相同，只是需要将形参换成实际的参数变量。

例 12.3　将例 12.2 中的条件关系式写成图形函数，完成例 12.2 所有的状态迁移功能。

解　系统的 Simulink 模型不会发生变化，同例 12.2 的 Simulink 模型。

Stateflow 图需要引入图形函数。将例 12.2 另存为 examp12_3，打开 Stateflow 图，点击左边的图形函数工具，移动鼠标至合适的位置，点击鼠标左键。在 function 后光标的位置键入 r＝hot（），回车，再点击 Stateflow 图左边的缺省状态迁移工具，将鼠标移入图形函数点击鼠标左键，产生一个缺省状态迁移线，在此迁移线上的问号处输入函数关系式，本例为{r=temp>=120}，意思是将当 temp>=120 为真时置 r 为 1，否则 r 为 0。这样图形函数就设置好了。在调用这个图形函数时，只需在调用的位置写入函数名即可。例 12.3 的 Stateflow 图见图 12.18。

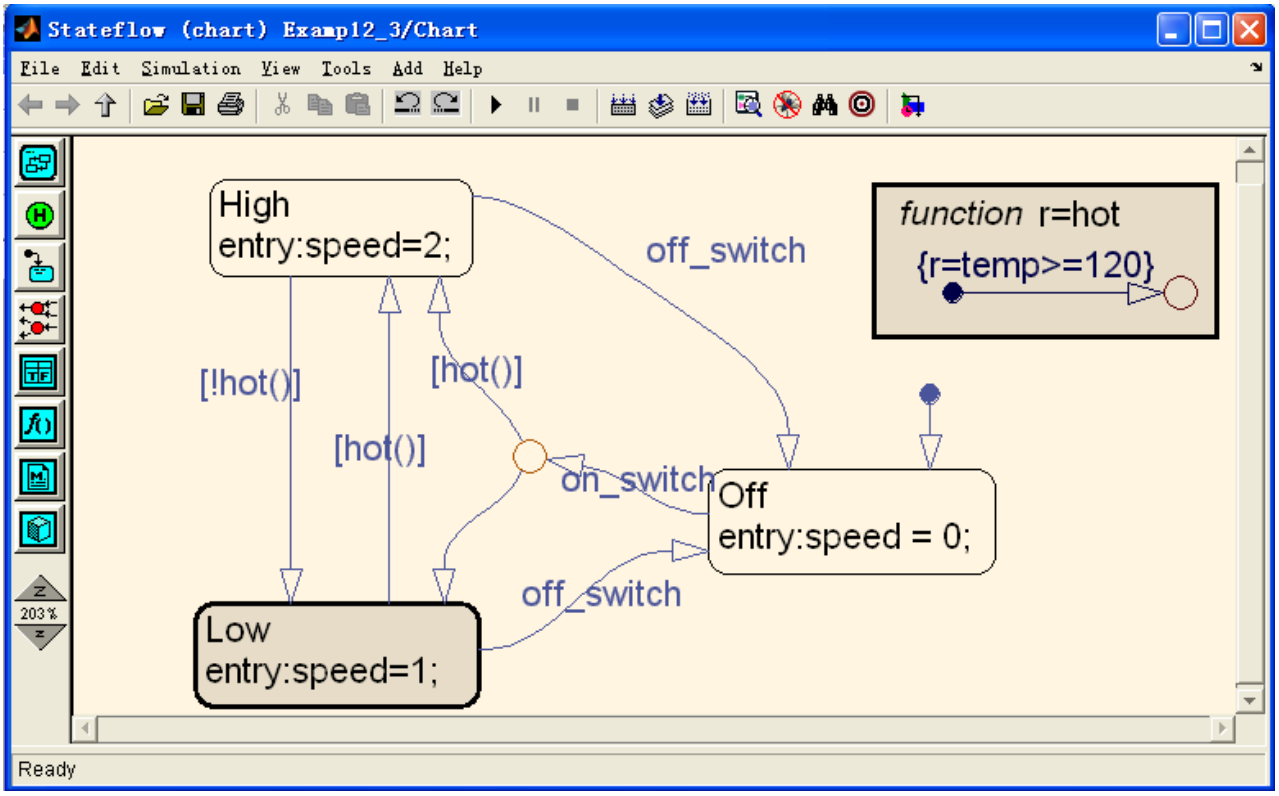


图 12.18　含图形函数的例 12.3 的 Stateflow 图

例 12.3 仿真时状态的迁移情况类似例 12.2，只是例 12.3 在进行仿真计算时，图形函数中的状态迁移线常常处于激活状态（表现为状态迁移线常处于变粗的状态），这是因为 Stateflow 每 0.5 秒时要从 Simulink 处接受一次 temp 值，随后 Stateflow 需要运行此图形函数判断 temp 值是否大于 120，并给 r 赋值。

7、 多层状态的嵌套

在例 12.3 中，当系统开始处于休眠状态时，状态 Off 激活时，状态 High 和 Low 实际上均需要 On_switch 事件激活，这样我们可以利用一个状态来包含这两个状态，这样可以使 Stateflow 图更简单明了。这里我们利用例 12.3 说明如何进行多层状态的设计以及多层状态在仿真时如何进行状态迁移的。

例 12.4　在例 12.3 的 Stateflow 图的基础上，建立上层状态或称作父状态（superstate），使其包含状态 Hign 和状态 Low（状态 High 和 Low 可以称作父状态的子状态），并完成与例 12.3 类似的转换功能。

解　一　建模

将例 12.3 另存为 examp12_4，打开其 Stateflow 图，删去部分状态迁移线，使其如图 12.19(a)所示。点击 Stateflow 图左侧的状态工具，在 Stateflow 图中添加一个状态，注意该状态的位置需要置于需要包含的两个状态的右下部，然后拖动新加的状态的左上角边框，增大该状态使其包含状态 High 和 Low，如图 12.19(b)示。在问号处输入 On，即将此状态称作 On 状态。这样一个上层状态就设置好了。

那么状态 On 和状态 Off 之间是由 off_switch 和 on_switch 事件触发进行迁移的，我们在状态 On 和状态 Off 之间添加状态迁移线，并标记迁移触发事件，如图 12.19(c)。

至此，我们可以用前面的知识分析该 Stateflow 图的状态迁移情况了。这里我们会碰到一个问题：在 Off 状态处于激活状态时，Stateflow 图从 Simulink 模型中收到一个有效的 on_switch 信号，这时发生状态从 Off 到 On 的转移。激活了 On 状态，那么在 On 状态中，具体应该激活哪个状态，执行哪个状态的动作，这时就不好决定了。碰到这种情况，我们必须给这个上层状态包含的状态设置一个缺省的状态或使用历史交汇工具。本例中我们使用缺省状态迁移工具指定进入状态 On 后需先激活的状态。在 On 状态中的 Low 状态上添加一个缺省状态迁移线，见图 12.19(d)。

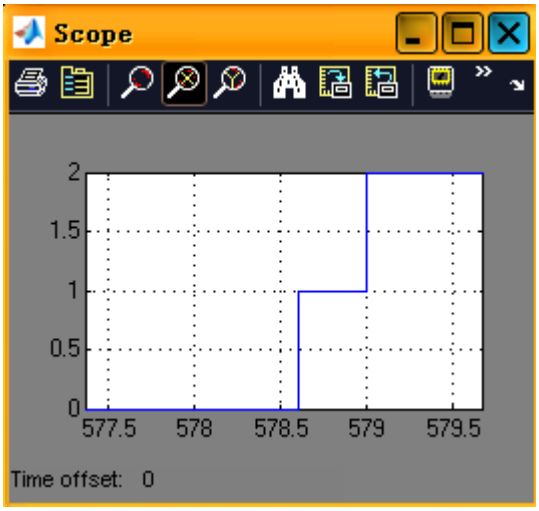
可以将上层状态包含的状态迁移图设置成子图的形式。用户只需在上层状态 On 内的任意点点击鼠标右键，在出现的下拉菜单中选择 Make Contents 中的 Subchart 即可，设置后 Stateflow 图见图 12.19(e)。Make Contents 中的 Subchart 具有复选功能，再选它时，上层状态又变回非子图形式了。

二　仿真

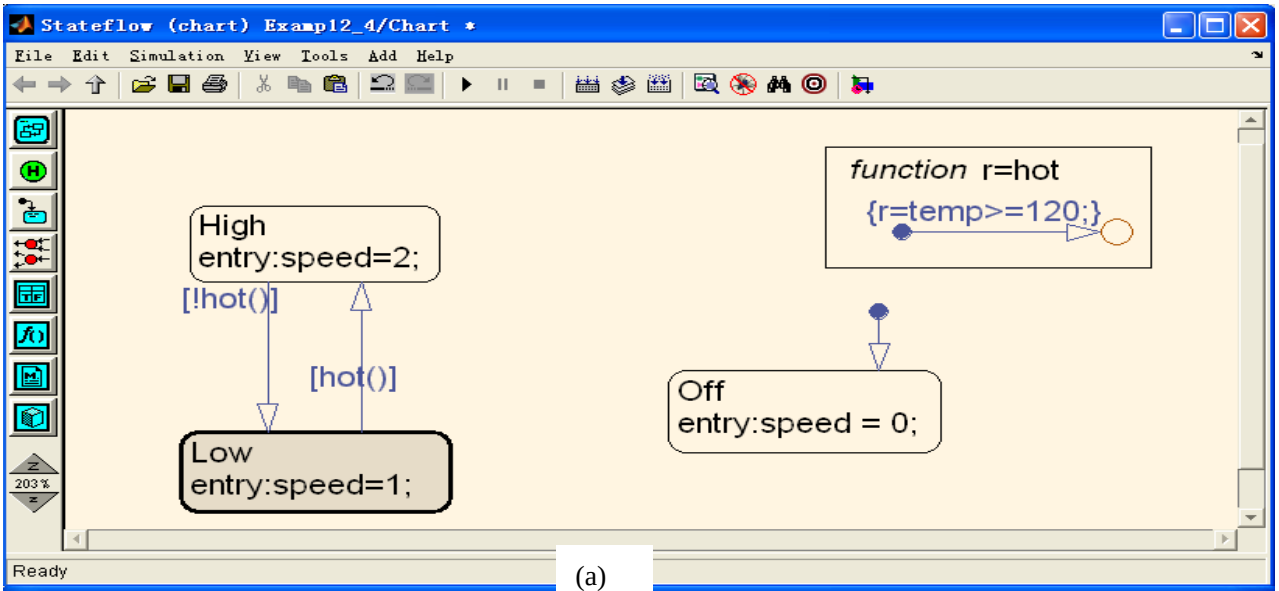
将 temp 值置为 130，点击工具栏图标 启动仿真。Stateflow 先分析本例的 Stateflow 图是否出现错误，若无错，则给此系统的 Stateflow 图自动生成 S－函数。随后正式进行仿真计算，此时 Simulink 模型中的 Pulse Generater 模块每 0.5 秒向 Stateflow 模块发送一个 temp_event 事件，Stateflow 图在接收到第一个 temp_event 事件后即唤醒处于休眠状态的 Stateflow 图，执行缺省的状态迁移，激活状态 Off，执行状态 Off 的动作，将 speed 置为 0，见图 12.20(a)-(b);

点击 Manual Switch1 模块以产生一个具有上升沿的 on_switch 触发信号，这时 Stateflow 图判断到可以进行状态 Off 到状态 On 的迁移，激活迁移信号线（on_switch 触发的信号线），停止状态 Off，激活状态 On，图 12.20(c)-(d);在状态 On 内，要先执行缺省状态迁移，激活其中的 Low 状态，执行 Low 的动作，将 speed 置为 1，见图 12.20(e)-(f);Low 状态与 High 状态之间有个条件迁移关系式，这时 Stateflow 调用图形函数以判断该条件迁移关系式是否成立，对于本例，因为 temp＝130，条件迁移关系式成立，故激活该迁移线，停止状态 Low，激活状态 High，执行状态 High 的动作，将 speed 置为 2，见图 12.20(g)-(h);

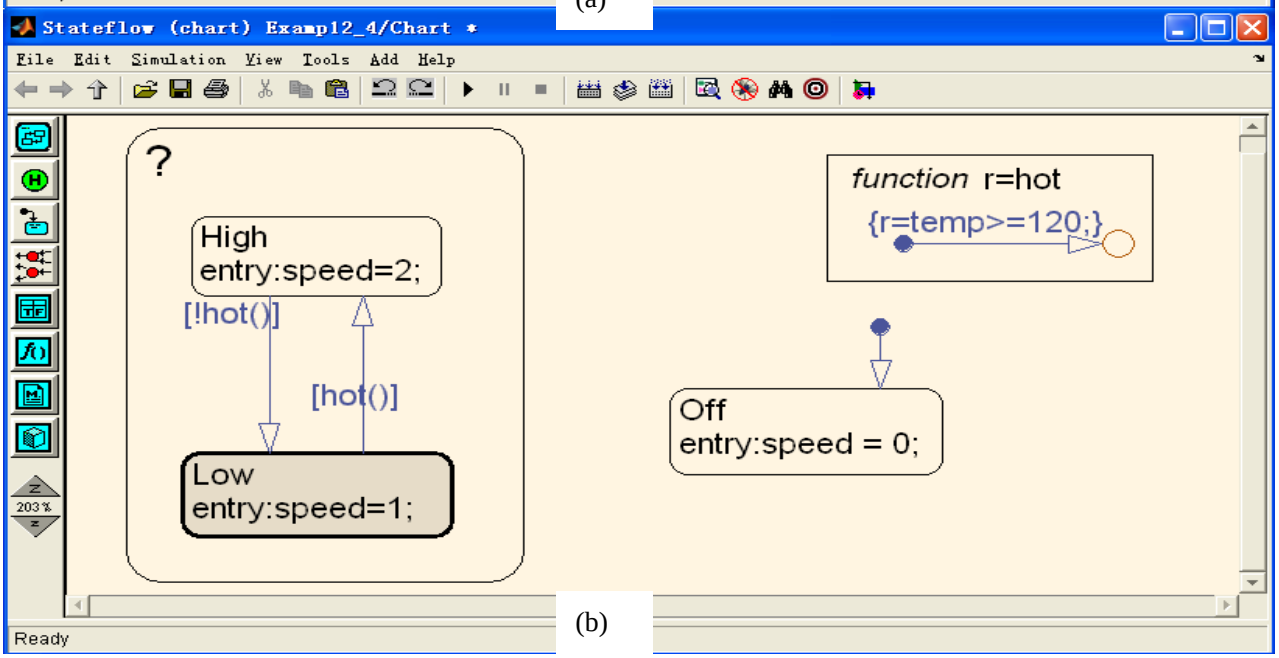
点击 Manual Switch 开关模块，产生一个下降沿的 off_switch 信号，此时 Stateflow 图判断出可以进行状态 On 至状态 Off 的迁移，激活 off_switch 触发的信号线，停止 High 状态，停止 On 状态，激活 Off 状态，执行 Off 状态的动作，将 speed 置为 0，见图 12.20(i)-(j)。



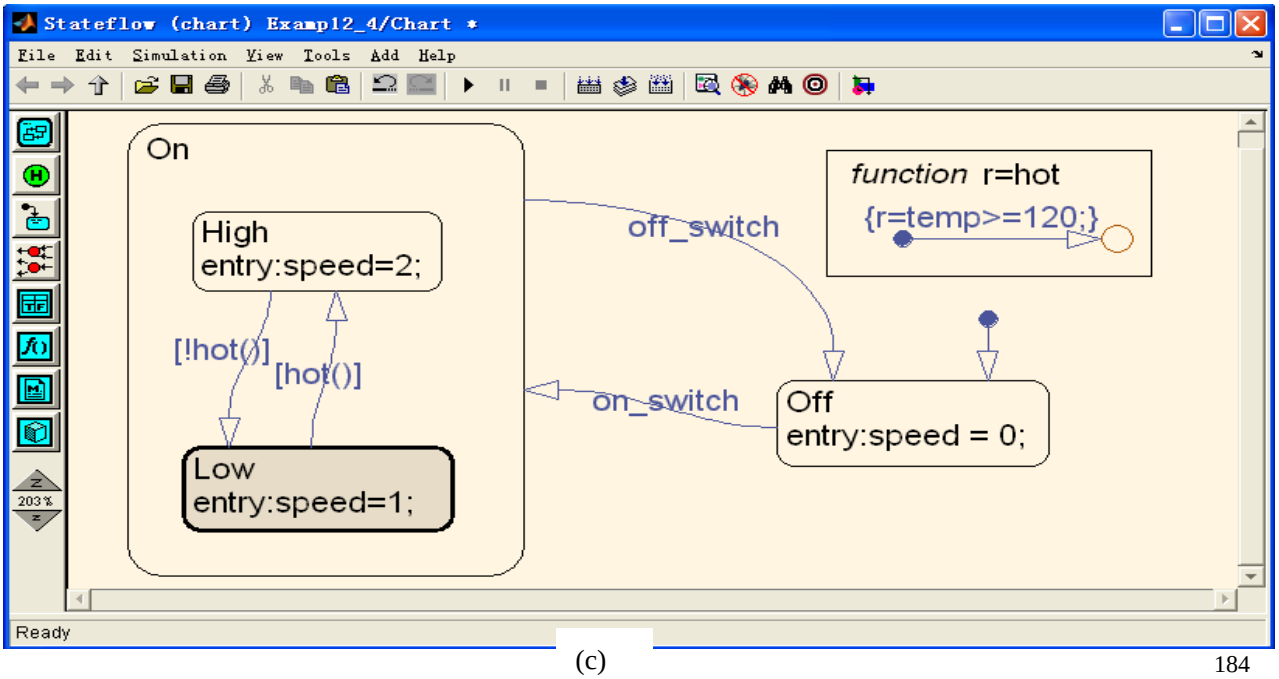
注意：例 12.4 基本上实现了例 12.3 完成的功能。但两者之间还是存在小小的差别的。在例 12.4 完成状态迁移时，要想激活状态 High，一定需要先激活状态 Low，然后才判断出必须执行状态 Low 到状态 High 的迁移，此时才能激活 High 状态。读者如果仔细观察仿真过程就能看到这样的迁移过程，而且对于例 12.4 的 speed 值要从 0 变到 2 中间总是要先将它置为 1，然后才置为 2，这是可以从 Scope 模块中观察到的。见右图。


Stateflow (chart) Examp12_4/Chart *
File Edit Simulation View Tools Add Help
High
entry:speed=2;
[!hot()]
[hot()]
Low
entry:speed=1;
function r=hot
{r=temp>=120;}
Off
entry:speed = 0;
Ready

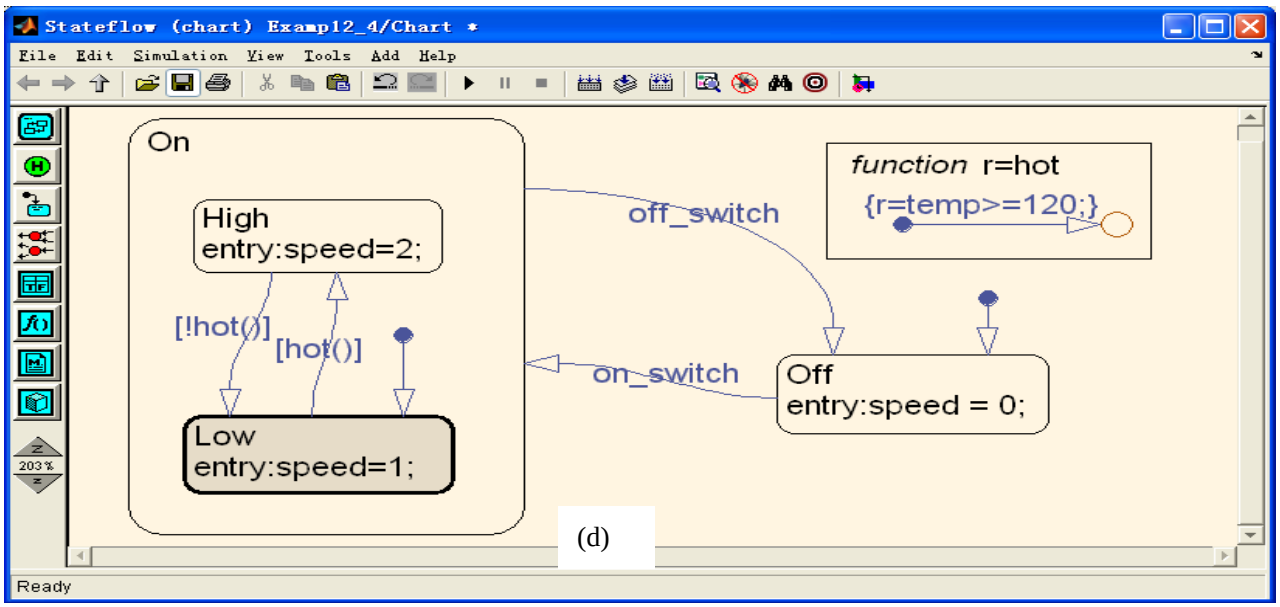
(a)


Stateflow (chart) Examp12_4/Chart *
File Edit Simulation View Tools Add Help
?
High
entry:speed=2;
[!hot()]
[hot()]
Low
entry:speed=1;
function r=hot
{r=temp>=120;}
Off
entry:speed = 0;
Ready

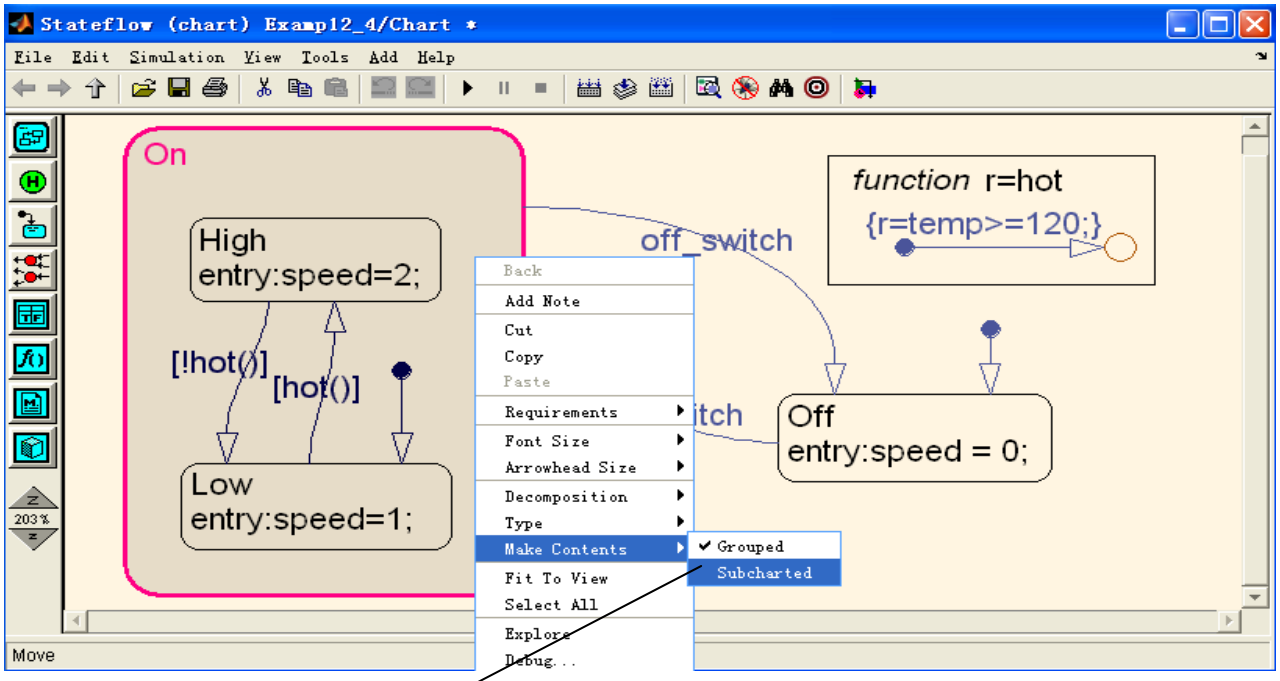
(b)

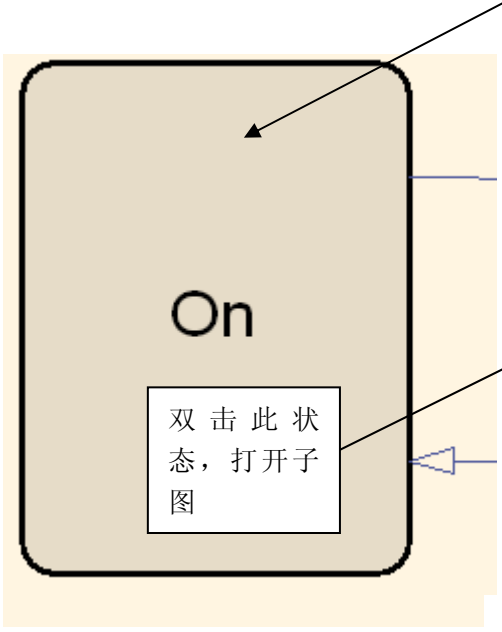

Stateflow (chart) Examp12_4/Chart *
File Edit Simulation View Tools Add Help
On
High
entry:speed=2;
[!hot()]
[hot()]
Low
entry:speed=1;
off_switch
on_switch
function r=hot
{r=temp>=120;}
Off
entry:speed = 0;
Ready

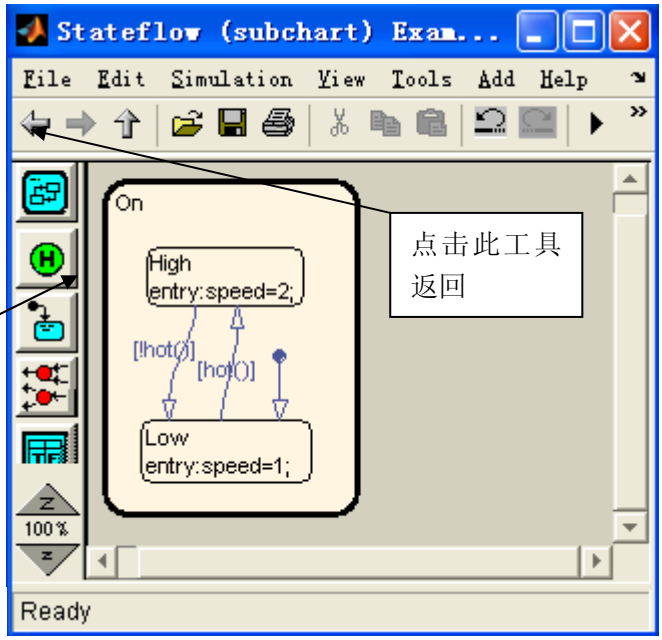
(c)



(d)

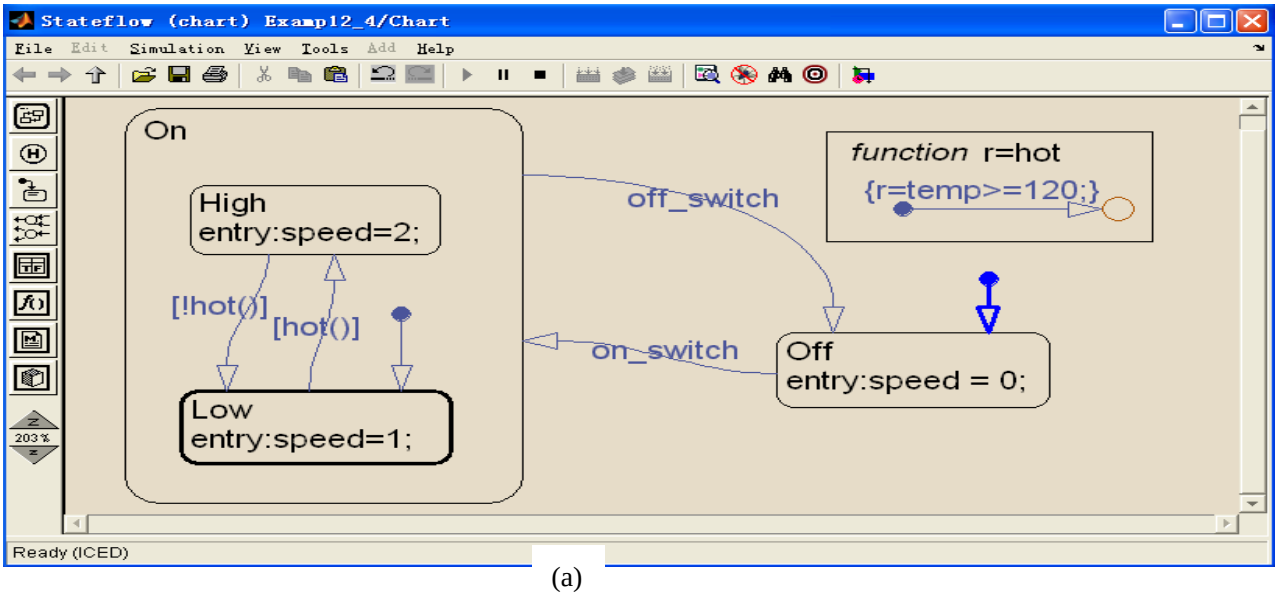




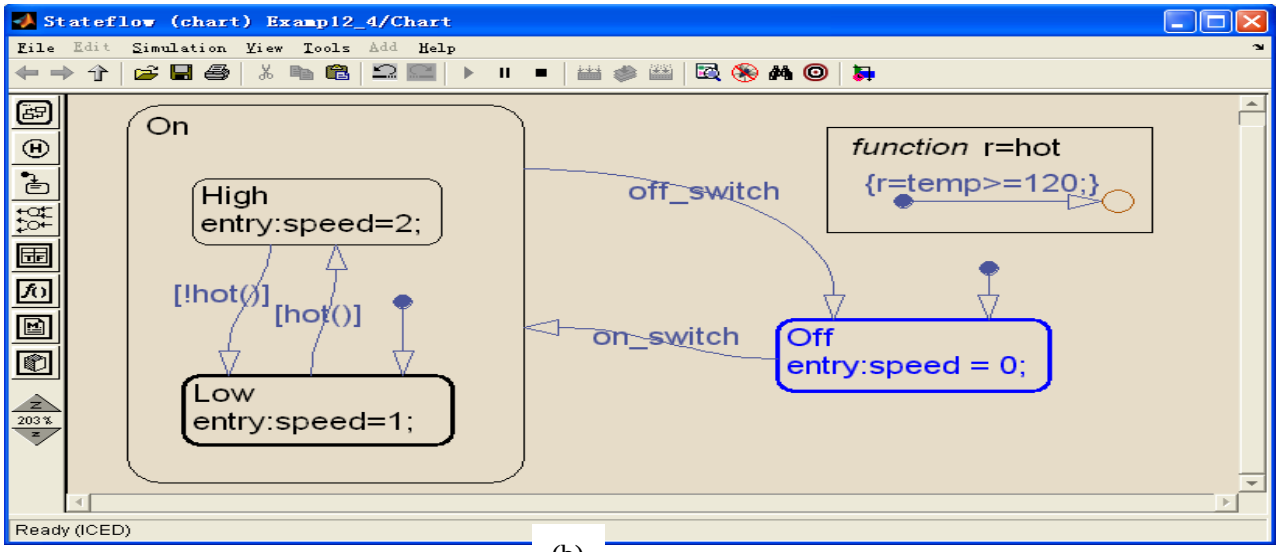


（e）

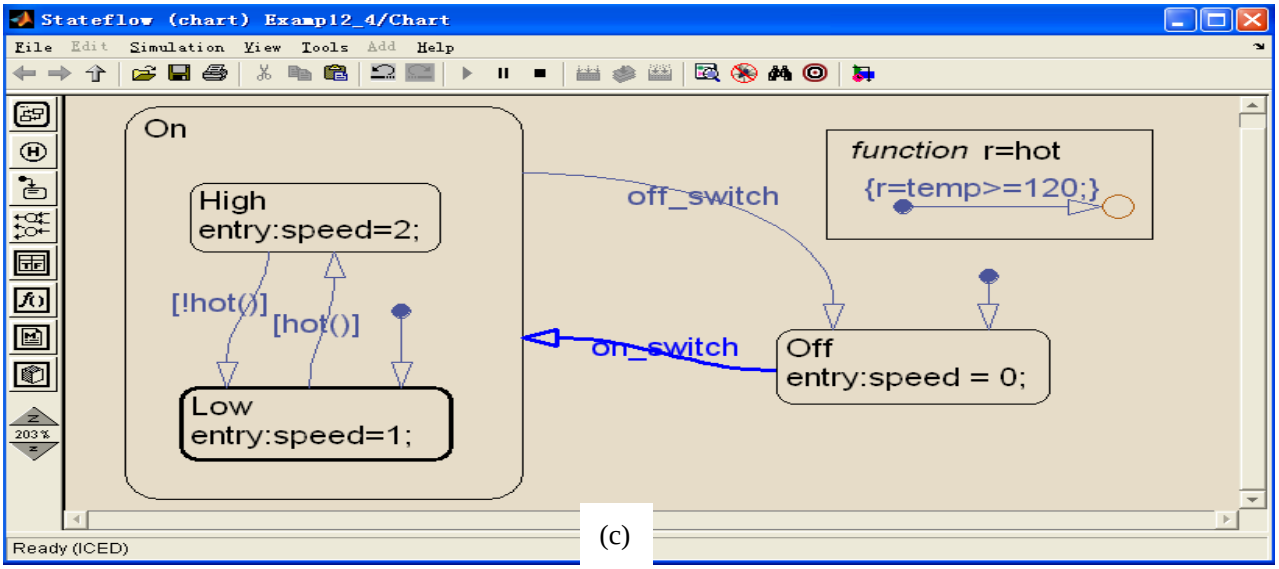
图 12.19　例 12.4 的 Stateflow 图设置过程


Stateflow (chart) Examp12_4/Chart
File Edit Simulation View Tools Add Help
On
High
entry:speed=2;
[!hot()]
[hot()]
Low
entry:speed=1;
off_switch
on_switch
function r=hot
{r=temp>=120;}
Off
entry:speed = 0;
203%
Ready (ICED)

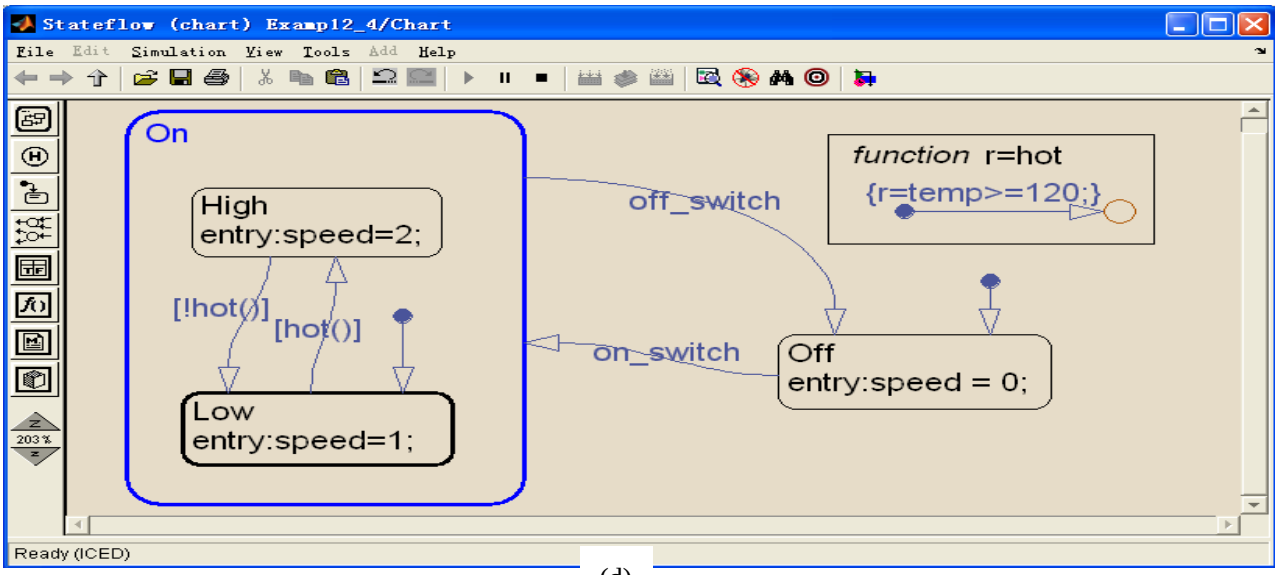
(a)


Stateflow (chart) Examp12_4/Chart
File Edit Simulation View Tools Add Help
On
High
entry:speed=2;
[!hot()]
[hot()]
Low
entry:speed=1;
off_switch
on_switch
function r=hot
{r=temp>=120;}
Off
entry:speed = 0;
203%
Ready (ICED)

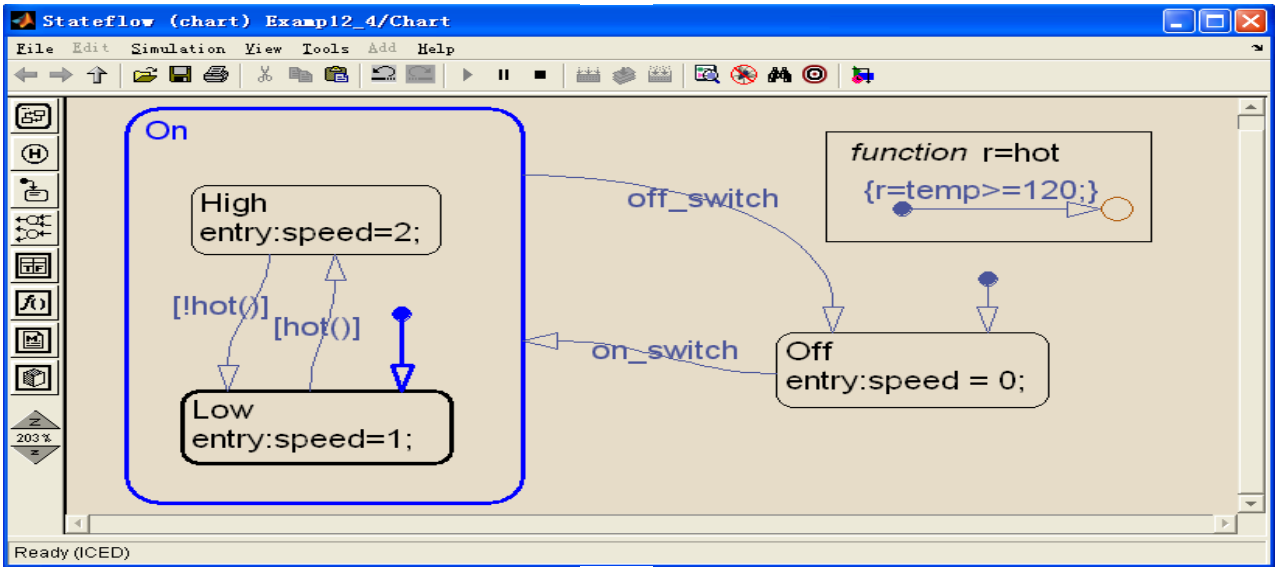
(b)


Stateflow (chart) Examp12_4/Chart
File Edit Simulation View Tools Add Help
On
High
entry:speed=2;
[!hot()]
[hot()]
Low
entry:speed=1;
off_switch
on_switch
function r=hot
{r=temp>=120;}
Off
entry:speed = 0;
Ready (ICED)

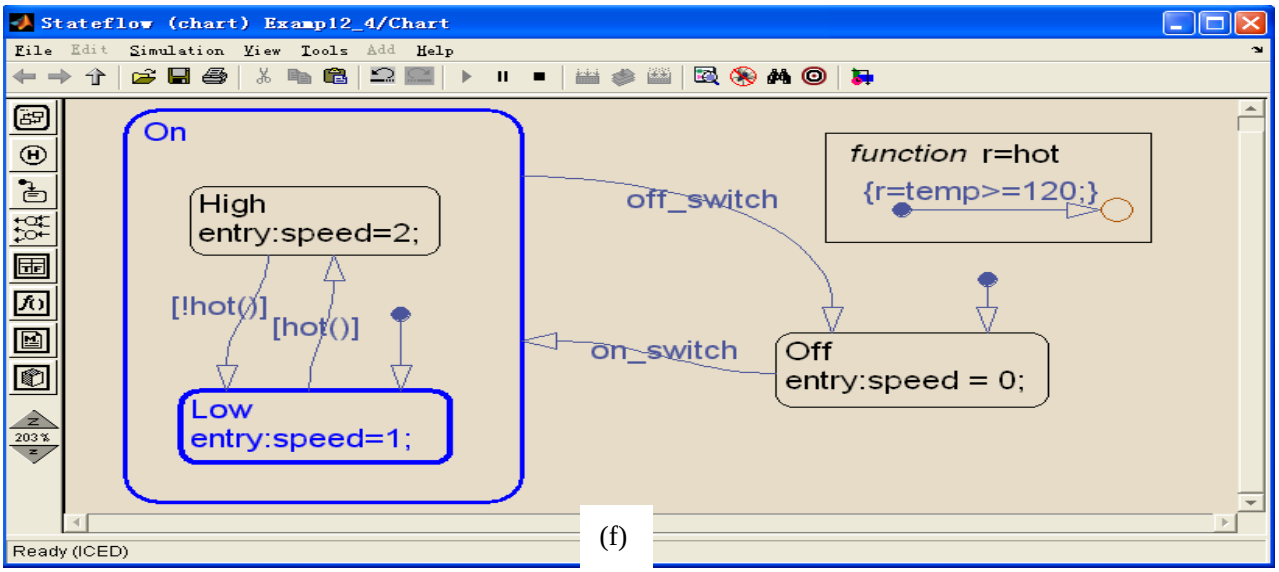
(c)


Stateflow (chart) Examp12_4/Chart
File Edit Simulation View Tools Add Help
On
High
entry:speed=2;
[!hot()]
[hot()]
Low
entry:speed=1;
off_switch
on_switch
function r=hot
{r=temp>=120;}
Off
entry:speed = 0;
Ready (ICED)

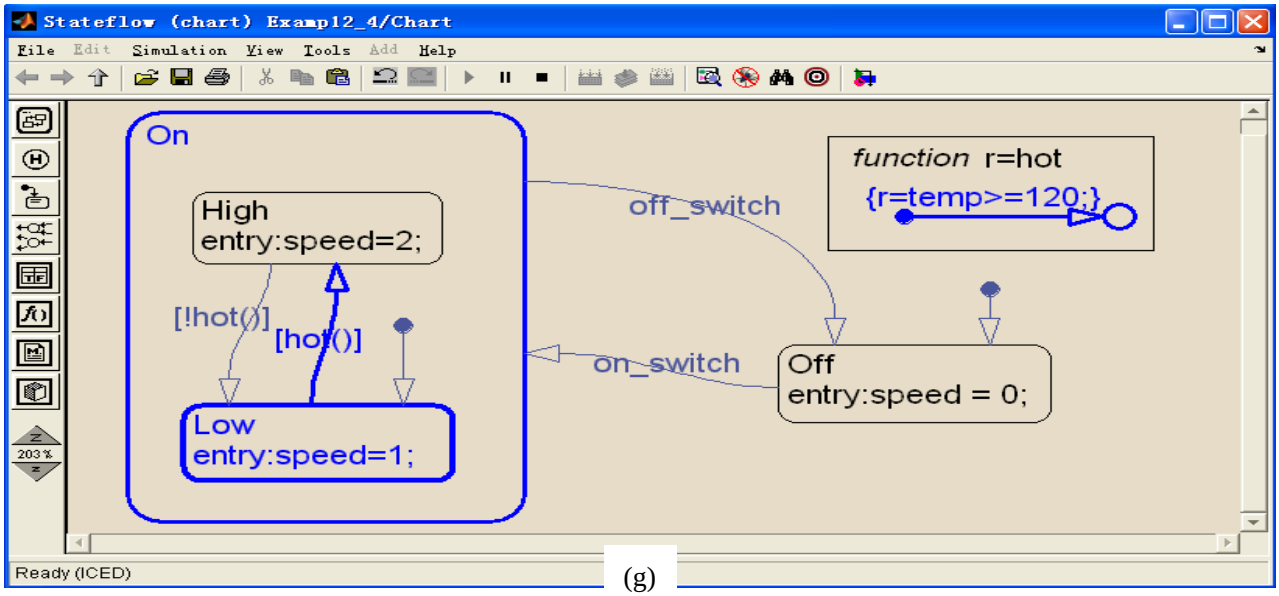
(d)


Stateflow (chart) Examp12_4/Chart
File Edit Simulation View Tools Add Help
On
High
entry:speed=2;
[!hot()]
[hot()]
Low
entry:speed=1;
off_switch
on_switch
function r=hot
{r=temp>=120;}
Off
entry:speed = 0;
Ready (ICED)

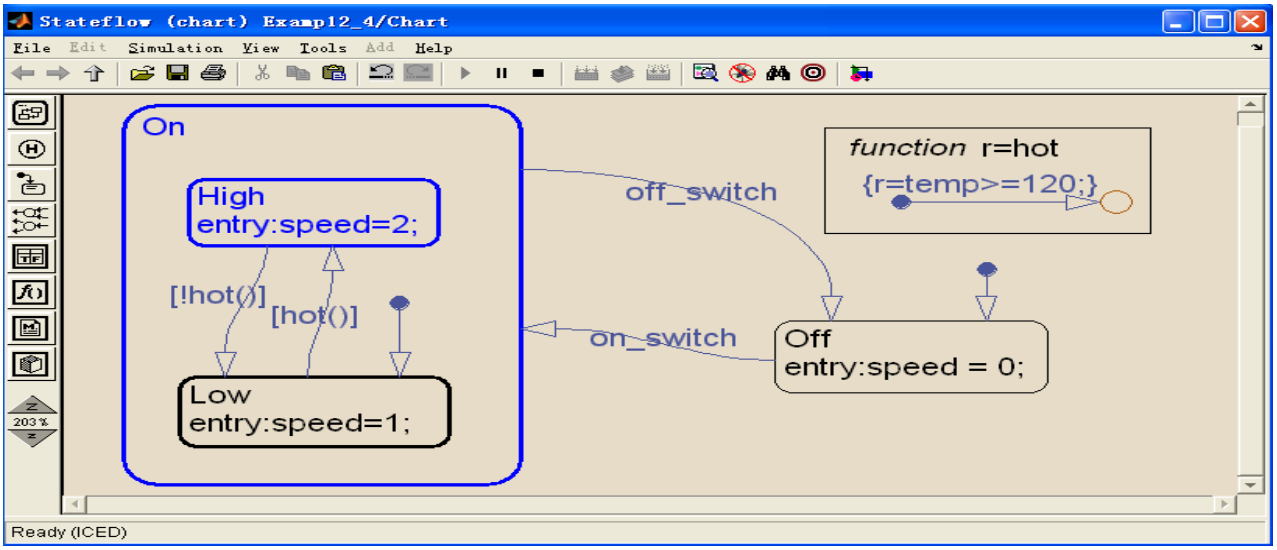
(e)


Stateflow (chart) Examp12_4/Chart
File Edit Simulation View Tools Add Help
On
High
entry:speed=2;
[!hot()]
[hot()]
Low
entry:speed=1;
off_switch
on_switch
function r=hot
{r=temp>=120;}
Off
entry:speed = 0;
Ready (ICED)

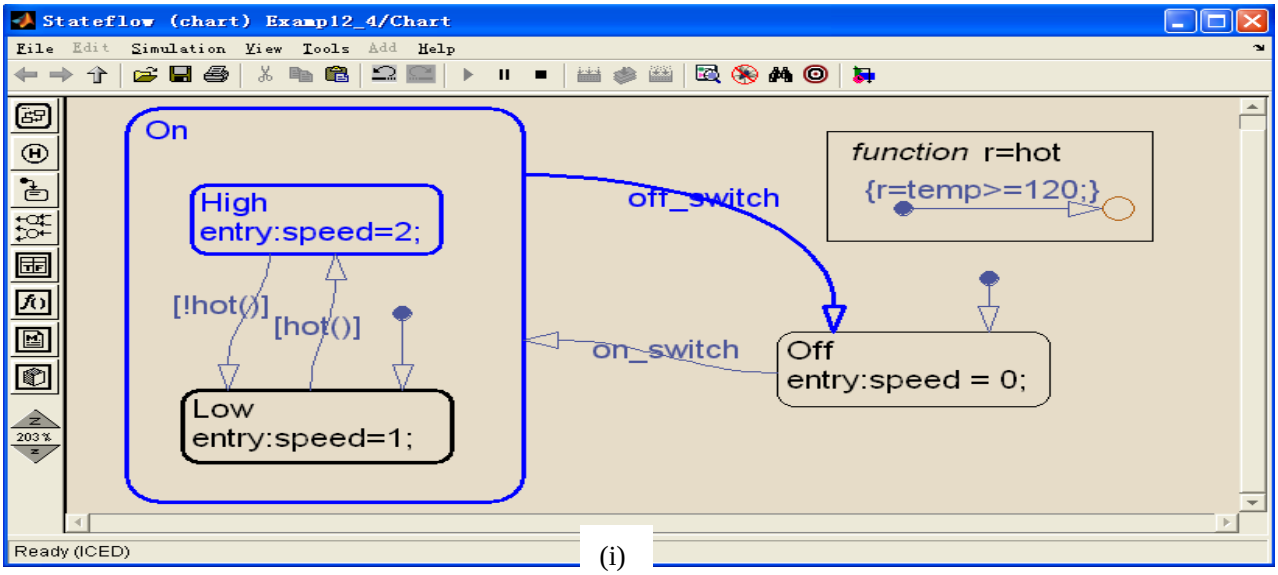
(f)


Stateflow (chart) Examp12_4/Chart
File Edit Simulation View Tools Add Help
On
High
entry:speed=2;
[!hot()]
[hot()]
Low
entry:speed=1;
off_switch
on_switch
function r=hot
{r=temp>=120;}
Off
entry:speed = 0;
Ready (ICED)

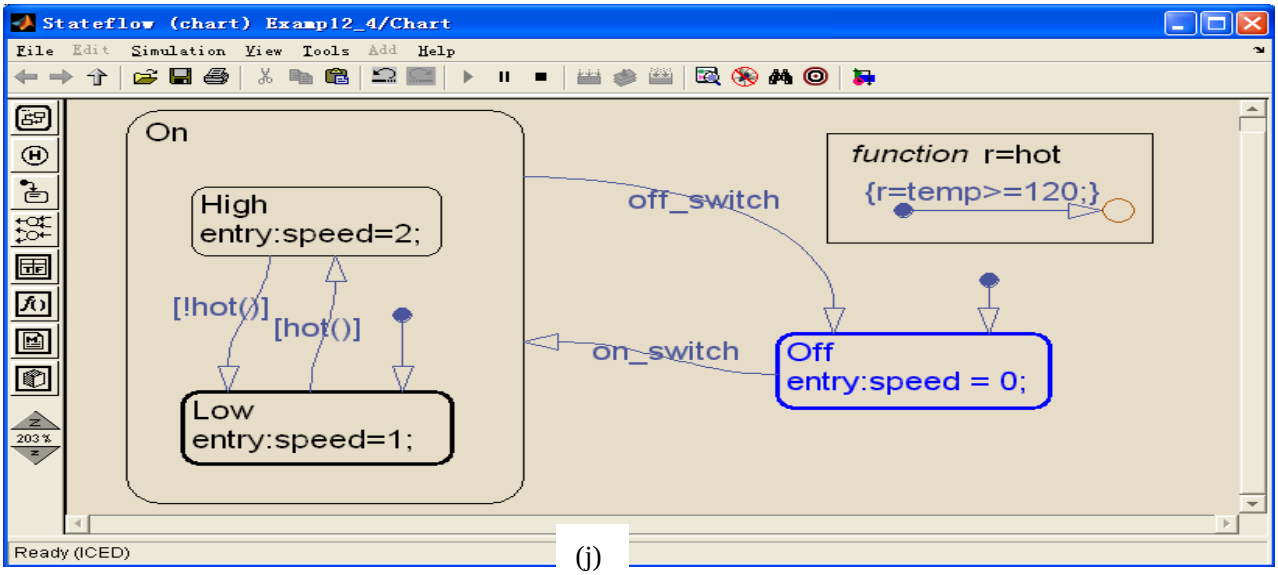
(g)


Stateflow (chart) Examp12_4/Chart
File Edit Simulation View Tools Add Help
On
High
entry:speed=2;
[!hot()]
[hot()]
Low
entry:speed=1;
off_switch
on_switch
function r=hot
{r=temp>=120;}
Off
entry:speed = 0;
Ready (ICED)

(h)



(i)



(j)

图 12.20　例 12.4 仿真的状态迁移示意图

8、 历史交汇工具的功能和使用方法

如例 12.4 那样，Stateflow 图进入上层状态时，缺省状态迁移线连接的子状态首先被激活。但在有些情况下，希望在进入上层状态时首先激活以前最后激活的那个子状态。这样的要求可以通过在上层状态中引入历史交汇工具完成。

例 12.5　在例 12.4 的基础上，在上层状态 On 内添加历史交汇工具，观察系统的状态迁移情况。

解　将例 12.4 的模型 examp12_4 另存为 examp12_5，打开 Stateflow 图，点击 Stateflow 图左侧的历史交汇工具图标，然后将鼠标移至 On 状态中的任意位置，再点击鼠标即添加了一个历史交汇工具，见图 12.21。

打开模型 examp12_4 和 examp12_5。将例 12.5 的仿真条件设置得与例 12.4 的仿真条件完全相同，即 temp＝130。点击工具栏中的图标，启动两个算例的仿真。执行与例 12.4 完全相同的步骤，直至状态 Off 处于激活状态，见图 12.20(j)。这些步骤中，例 12.4 和例 12.5 的状态迁移是完全相同的。此后按下述

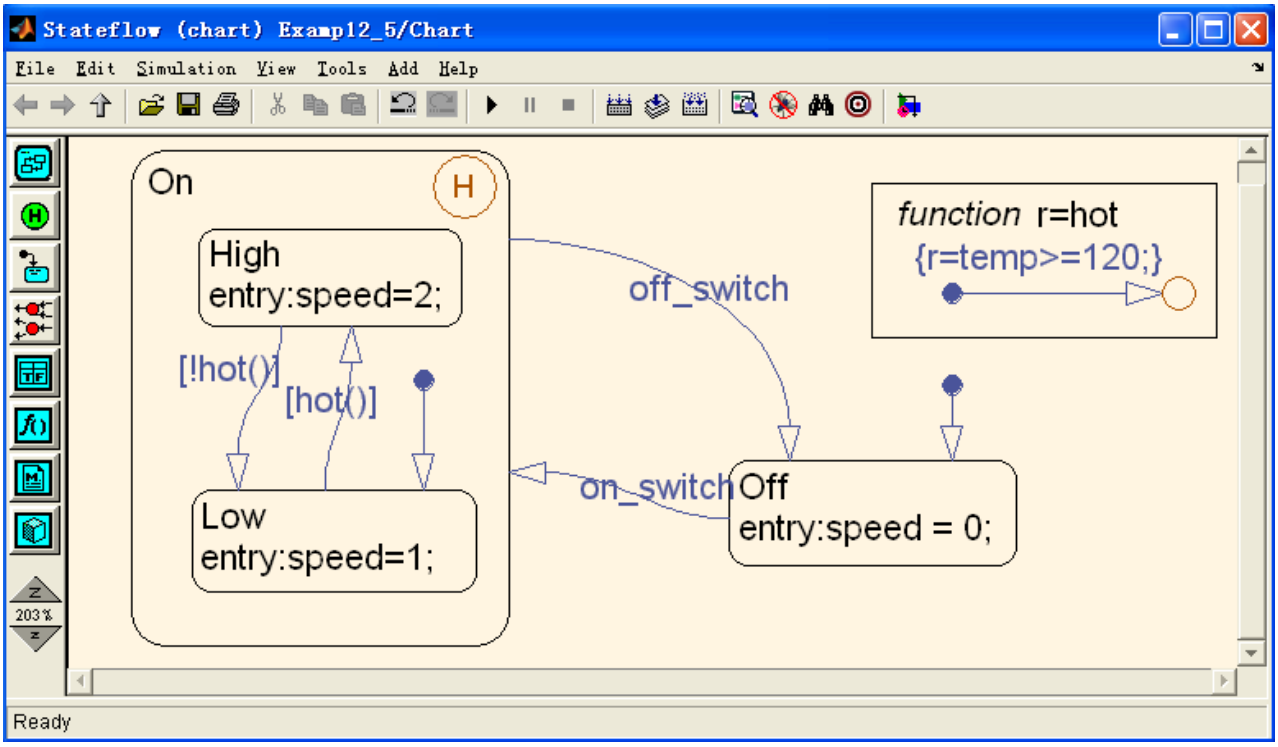


图 12.21　例 12.5 含历史交汇工具的 Stateflow 图

步骤执行，观察例 12.5 和例 12.4 状态迁移过程的区别。

改变 Manual Switch1 的开关位置，产生一个上升沿触发信号，on_switch 触发信号有效，这时 Off 状态停止，激活 On 状态。在例 12.4 中，会激活缺省信号线，随后激活 Low 状态，然后执行图形函数 hot()，判断出 hot()为真，这时激活状态 Low 至状态 High 的迁移线，停止状态 Low，停止两状态之间的迁移线，激活状态 High，系统处于休眠状态，等待下一个触发信号，如图 12.20(e)-(h)；而在例 12.5 中，由于有历史交汇工具，Stateflow 图直接激活 High 状态，然后执行图形函数 hot()，判断出确实应该激活 High 状态，就开始等待下一个触发信号。

问题：若在改变 Manual Switch1 的开关位置之前，先将 temp 值改为 110，请读者分析例 12.4 和例 12.5 的状态迁移情况。

前面我们通过几个例子，介绍了 Stateflow 中最基本对象的使用方法，至此用户就可以自行编写简单的 Stateflow 图对具有相对简单的逻辑判断的系统进行仿真计算了。下面介绍 Stateflow 中也经常用到的其他工具的使用方法。

9、 嵌入式 M 函数的设置及其调用

嵌入式 MATLAB 函数使用户可以利用 MATLAB 强大的功能，在 Stateflow 图中编 MATLAB 语言函数，调用 MATLAB 的各类函数。Simulink 利用嵌入 MATLAB 函数的状态流实现 Simulink 模型中嵌入 MATLAB 模块功能。

要想在 Stateflow 中嵌入 MATLAB 函数，只需在 Stateflow 的编辑器中点击左侧的图标  ，将鼠标移到编辑器中的适当位置，再点击鼠标，并在光标处写入要建立的 MATLAB 内嵌函数名，这时 MATLAB 会自动打开一个编辑 MATLAB 函数的编辑窗口，见图 12.22。内嵌函数的调用类同于图形函数的调用方法，可以在状态的动作和迁移过程中对内嵌函数进行反复的调用。

例 12.6　利用 Stateflow 求一组数组的最大、最小及其均值。

解　一　建模

首先在 MATLAB 命令窗键入 sfnew，打开一个含 Stateflow 模块的 Simulink 模型窗。打开 Simulink

模型窗中的 Stateflow 模块，打开 Stateflow 编辑器，点击 Stateflow 编辑器左端的内嵌 MATLAB 函数图标  ，将鼠标移到 Stateflow 编辑器中适当的位置，再次点击鼠标，即产生一个内嵌的 MATLAB 函数，在光标处键入内嵌 MATLAB 函数名及其形参名 MaxMin（x）即可。这时会自动打开一个含 function MaxMin（x）的内嵌 MATLAB 函数编辑窗，在此编辑窗内键入内嵌函数需要完成的程序。本例的程序见图 12.22。由于本例只是为了求一组数组的最大、最小和均值，这些功能内嵌函数都已实现，因而在 Stateflow 图中，它的调用就非常简单，只需建立一个缺省的状态迁移，在进行状态迁移时执行状态迁移条件动作时调用即可。见图 12.22。本例只需进行 Stateflow 输入输出数据的设置，在 Stateflow 编辑界面点击 add 菜单下的 data－input from Simulink，将变量名设置为 a；输出变量有 3 个，点击 add 菜单下的 data－output to Simulink，将变量名分别设置为 Xmax，Xmin 和 Xmean。由于输入变量 a 是 4 维数组，Stateflow 中内嵌函数 MaxMin 中的形参 x 也应该是 4 维数组。在 Stateflow 编辑界面中，选择 Tools 菜单中的 Explore，将会打开 Model Explorer，如图 12.23 所示。将 Chart 的内嵌函数 MaxMin 中的函数输入 x 的 size 设置为 4（在 size 列下输入 4 即可）。

在 Simulink 模型窗口中创建输入模块 Constant，将其输入的常数设置为 4 维数组[68 39 47 96]。添加 3 个显示 Display 模块，并连接对应的信号线，见图 12.22。将此模型存为 Examp12_6。

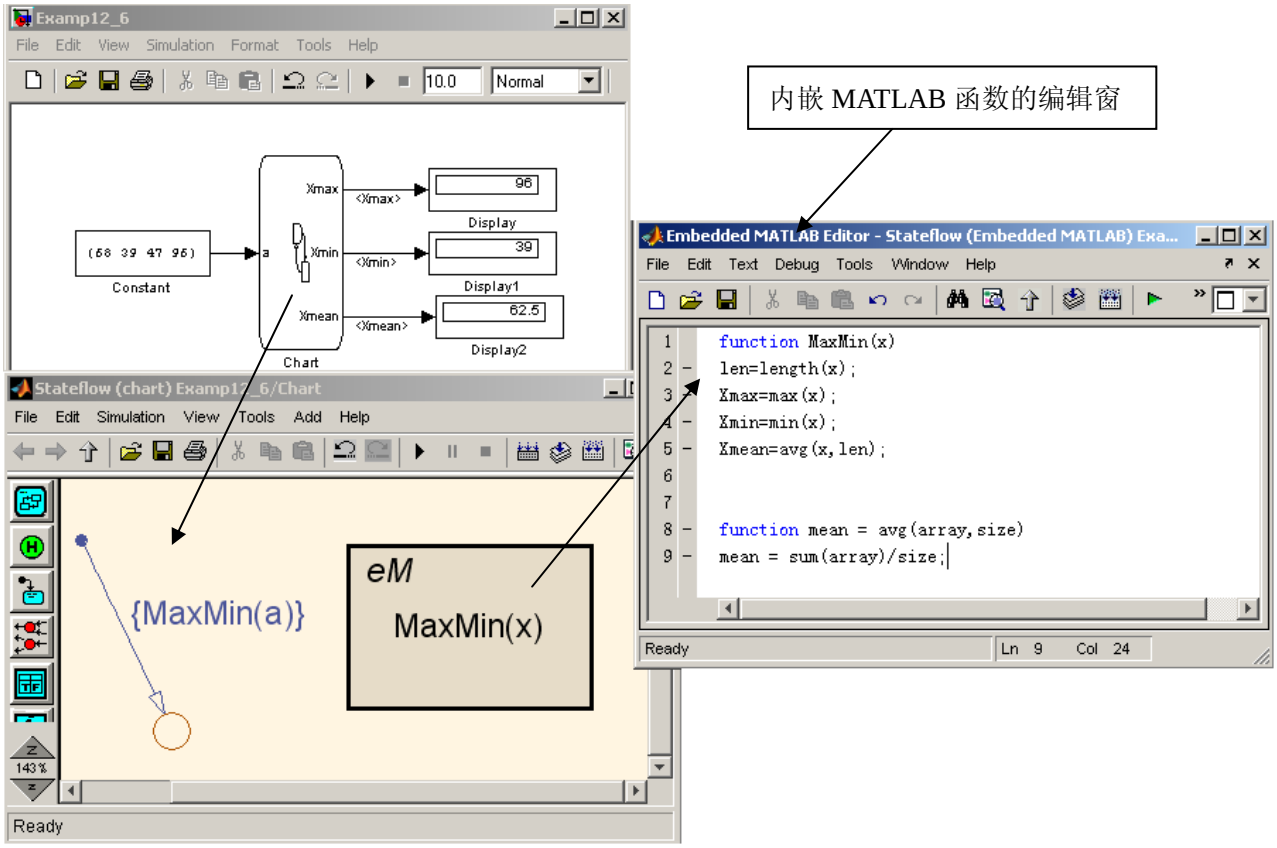


图 12.22 Stateflow 内嵌 M 函数的应用举例

二　仿真

选择 Simulation 菜单下的 Start，启动仿真。仿真开始时，Simulink 先分析 Stateflow 图，如没有错误，就按照 Stateflow 图自动生成 S－函数，并存在当前目录下的 sfprj 子目录下。仿真结束后，3 个 Display 显示模块分别显示出输入数组的最大值、最小值和均值。

上面通过一个很简单的例子说明了 Stateflow 中内嵌 MATLAB 函数的编写、调用及使用方法。需要说明的是这种内嵌 MATLAB 函数不仅可以使用 MATLAB 强大的函数功能，如本例中直接使用 max 和 min 函数，还可以调用用户自行编写的 M 函数，如本例中，函数 MaxMin 还调用了函数 mean。这种情况

下，被调用的 M 函数也只需同时写在内嵌 MATLAB 函数编辑器中就行了。

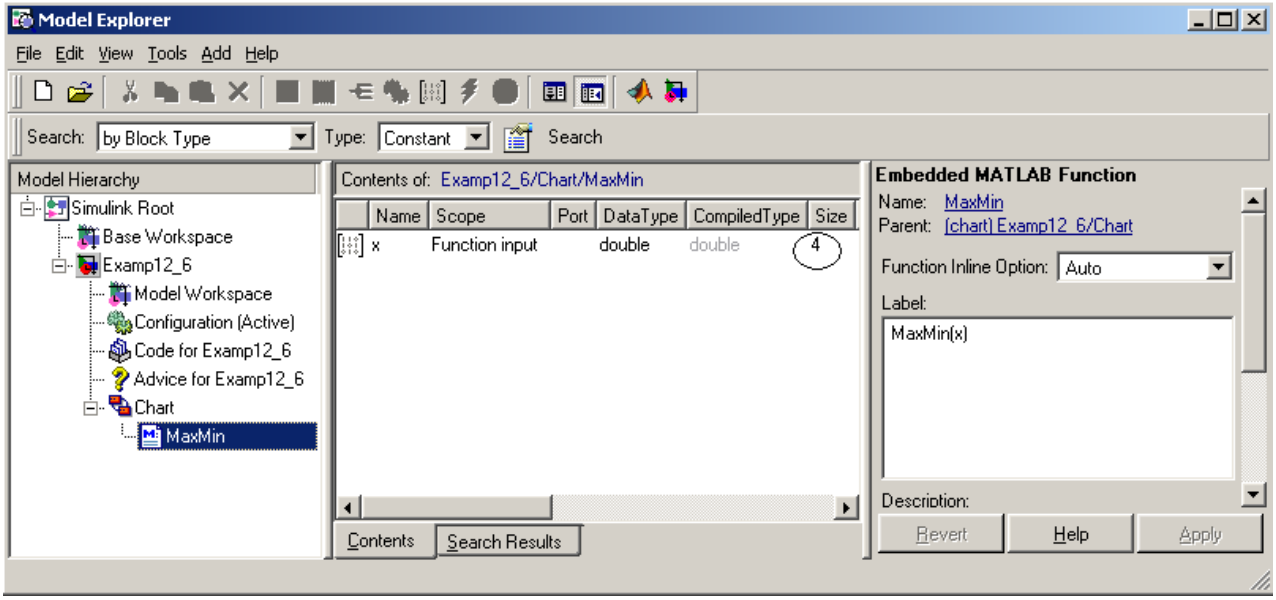


图 12.23 例 12.6 的模型浏览器

10、真值表的设置及其使用

Stateflow 使用函数来处理在 Stateflow 图中需反复处理的动作或判断。在真值表中，用户可以用条件、决策和动作来做逻辑判断，并执行相应的动作。对于纯粹的逻辑来讲，真值表比图形函数更容易编写、维护，也更容易阅读。真值表还可以告诉用户是否对于指定的条件做出了足够的或过多的决策。

Stateflow 使用真值表函数实现逻辑决策及相应动作的执行。Stateflow 真值表含有条件、决策和动作。我们通过算例来说明真值表的设置和工作情况。Stateflow 真值表中条件、决策和动作的排列如表 12.1.

表 12.1 Stateflow 真值表中条件、决策和动作的排列表

Condition（条件）	Decision1（决策 1）	Decision2（决策 2）	Decision3（决策 3）	Default Decision（缺省决策）
x==1	T	F	F	-
y==1	F	T	F	-
z==1	F	F	T	-
Action（动作）	t=1	t=2	t=3	t=4

Condition(条件)列中的每个条件先要判断是真（T）或假（F），对于上表，就是判断 x==1，y==1，z==1 是否成立。每个条件可以如上表标记为 T、F 或-(即不论 T 或 F)。每个 Decision（决策）列隐含着各个条件的“与”操作。表 12.1 中 Decision1 列中，当 x==1 为真，而 y==1 和 z==1 同时为假时，Decision1 为真。执行过程中，Stateflow 会从 Decision1 开始判断真值表中的每个决策，如果哪个 Decision 为真，就执行该 Decision 对应的动作。如当 x==1 为真，而 y==1 和 z==1 同时为假时，Decision1 为真，执行动作将 t 置为 1。上表中的最后一个决策称为缺省决策，它包含着除了前面列举的决策外的所有其他决策。如果 Decision1～3 都是假的，则 Default Decision 自动为真，执行其对应的动作，将 t 值置为 4。

用 if－then 语句实现上述的真值表所完成的功能，需编写 if－then 程序如下，

```
if ((x == 1) & !(y == 1) & !(z == 1))
    t = 1;
elseif (!(x == 1) & (y == 1)& !(z == 1))
    t = 2;
elseif (!(x == 1) & !(y == 1) &(z == 1))
```

```
    t = 3;
else
    t = 4;
end
```

对于简单列举的真值表中仅含 3 个决策和一个缺省决策的情况，见表 12.1，用户就需要很多的 elseif 语句，如果再多些决策那么需要的 elseif 语句就会更多。笔者认为采用真值表做逻辑判断是一种非常简单明了的方法。

下面我们通过一个仿真算例说明真值表的创建方法。

例 12.7　创建 Stateflow 真值表如表 12.1 所示，通过对 3 个输入变量大小的判断，实现对不同条件下输出变量 t 的相应赋值。

解　一　建模

在 MATLAB 命令窗键入 sfnew，打开一个含 Stateflow 模块的 Simulink 模型窗。双击 Simulink 模型窗中的 Stateflow 模块，打开 Stateflow 编辑器，点击 Stateflow 编辑器左端的真值表图标，将鼠标移到 Stateflow 编辑器中适当的位置，再次点击鼠标，即产生一个真值表函数，在光标处键入该真值表函数名及其形参名 t＝truthtable（x，y，z）即可。其中 t 是输出形参，x，y，z 是输入形参，truthtable 是本例的真值表函数名。双击已产生的真值表函数图标，打开真值表编辑器，见图 12.24。点击真值表编辑器中的工具可以增添条件编辑表和动作编辑表中的行数。点击工具可以增添 Decision 决策的列数。在真值表编辑器中键入图 12.24 所示的内容以实现表 12.1 所列举的逻辑判断及动作功能。这样本例的真值表函数就编辑好了。真值表函数的调用类似内嵌 MATLAB 函数和图形函数的调用，在 Stateflow 编辑界面中需调用的地方，键入正确的调用格式。本例中在缺省状态迁移处执行迁移动作时调用，f＝truthtable（a，b，c）表示输入实参是 a、b 和 c，输出实参是 f。

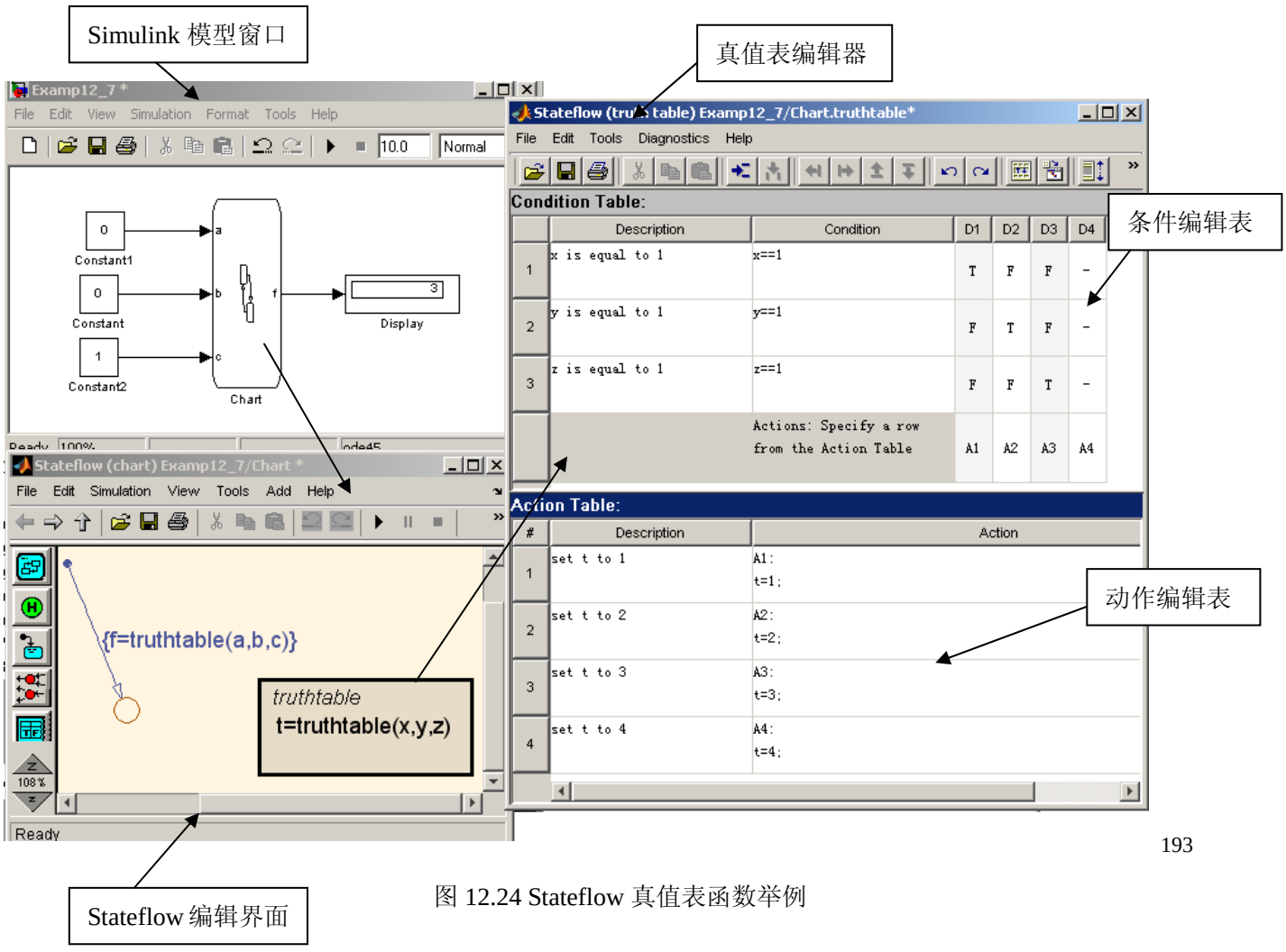


图 12.24 Stateflow 真值表函数举例

这样可知 Stateflow 的输入数据是 a、b 和 c，输出参数是 f，在 Stateflow 界面采用 add 菜单或直接打开其 Model Explorer 添加这些输入输出数据。

在 Simulink 模型窗口中从 Simulink 模型库的 Source 库中添加 3 个 Constant 模块，以便给 Stateflow 输入数据 a、b 和 c。从 Simulink 模型库的 Sinks 库中添加 1 个 Display 模块以便观察仿真结果 f 值。

二　仿真

选择 Simulation 菜单下的 Start，启动仿真。仿真开始时，Simulink 先分析 Stateflow 图，如没有错误，就首先将真值表转换成图形函数，然后按照 Stateflow 图自动生成 S－函数，并存在当前目录下的 sfprj 子目录下。仿真结束后，用户可以右击 Stateflow 编辑器中的真值表函数，从下拉菜单中选择 View Contents 观察由真值表生产的图形函数。

仿真结束后， Display 显示模块显示出 Stateflow 图中的输出 f 值。本例中，将 a 和 b 置为 0，c 置为 1，仿真结果 f＝3。用户可以改变 Simulink 模型窗口中 3 个 Constant 模块的数值，以观察输出结果的变化，并根据真值表的功能，检验显示结果的正确性。

除了决策的动作外，Stateflow 还允许用户给真值表函数添加初始和终止动作。初始动作指定任何决策判断前的动作。终止动作指定真值表函数即将退出最后要执行的动作。在动作编辑表中使用标记 INIT 和 FINAL 来为真值表函数指定初始和终止动作。双击真值表函数图标，打开真值表编辑器，右击动作编辑表的第一行，在产生的下拉菜单中选择 Insert Row 即可在动作编辑表最上端产生一空白行。用户可以在此添加初始动作；点击真值表编辑器中的工具  可以在动作编辑表最下端插入一个空行，可以在此空行处添加终止动作。例如，本例中，初始动作和终止动作都添加为显示信息，添加了初始和终止动作的真值表编辑器如图 12.25 所示。仿真时，在每次调用真值表函数时，MATLAB 命令窗口会显示如下提示

truth table for Examp12.7 entered

truth table for Examp12.7 exited

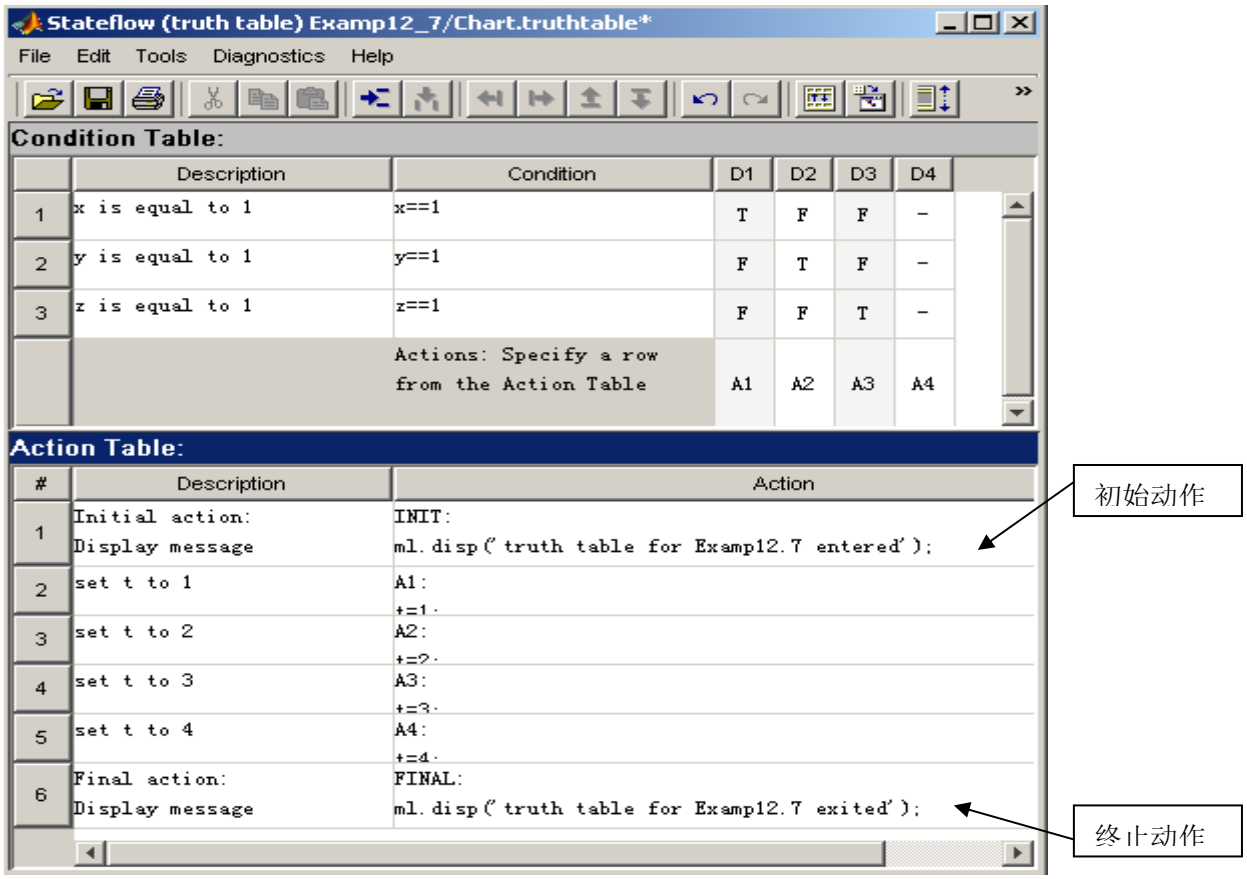


图 12.25　添加了初始动作和终止动作的例 12.7 的真值表编辑器

我们在此就不多述真值表编辑器工具栏上各图标可以完成的功能。

有时用户在编写真值表时会过多地制订一些决策，这些决策有时是条件编辑表前面的决策可能包含的决策；也有的时候用户可能会漏掉一些应用中可能发生的决策。Stateflow 可以在编译时给出用户过多或过少制定决策的错误提示。图 12.26 给出过多决策的示例。在图 12.26 的条件编辑表中，决策 D3（—TT）规定了 FTT 及 TTT，其中 FTT 在 D1 中规定了，而 TTT 也在 D2 中规定了。因而 D3 是属于多余决策。Stateflow 在编译该真值表时，给出了错误提示，见图 12.26。

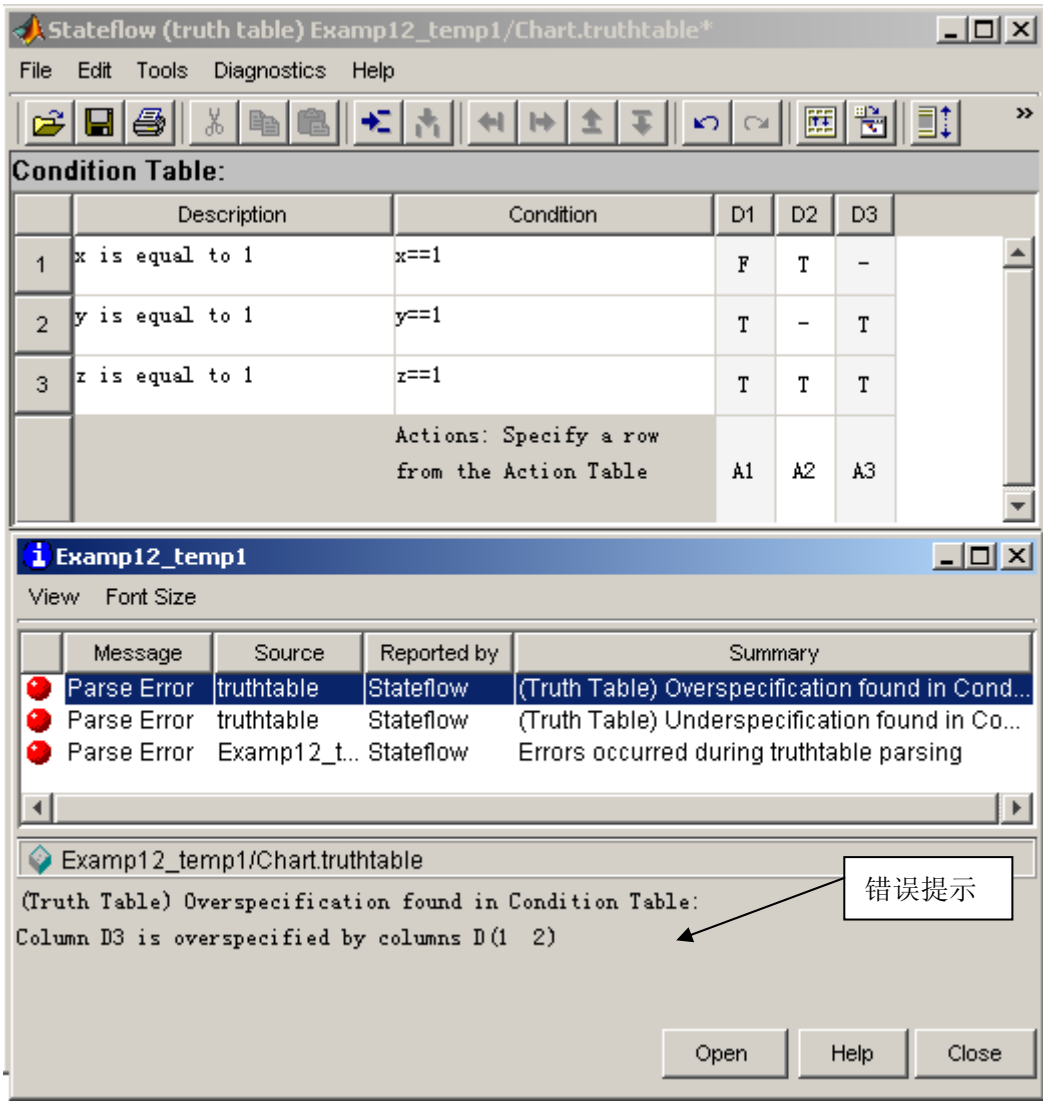


图 12.26 含多余决策的真值表举例

图 12.27 给出了不足决策的示例。条件编辑表中只规定了三种决策，那么可能的决策应该有 8 种，因而 Stateflow 在编辑该真值表时，给出错误信息的同时，指出了其他 5 种可能决策。因而，用户在编写真值表时，可以用象例 12.7 那样，利用真值表的最后缺省决策来表示其他所有前面未规定的决策。

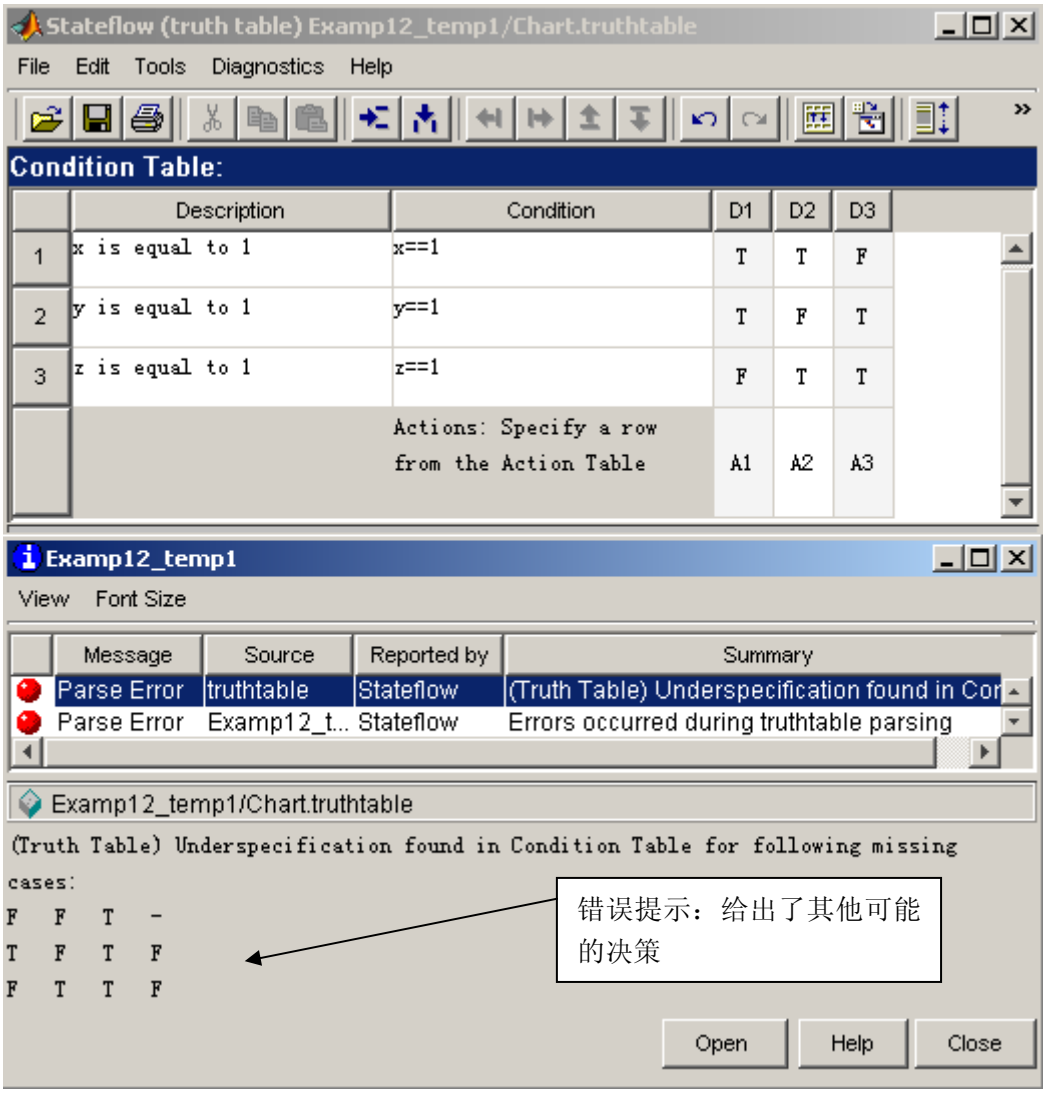


图 12.27 不足决策的真值表举例

11、 用 Box 工具整理状态流图

Boxes 能够很方便地用来整理 Stateflow 图。

Boxes 的创建很简单，有几种方法：

首先，利用 Stateflow 编辑界面左边的 Box 工具  。点击图标  ，将鼠标移至 Stateflow 编辑界面的适当位置，再点击鼠标，即可创建一个 Box 对象，在 Box 的问号处写入该 Box 的名称。

其次，可以先建立一个状态，将该状态转换为 Box。右击创建好的状态，在弹出的下拉菜单中选择 Type－Box 即可创建一个 Box 对象。

创建好 Box 工具后，用户可以在此 Box 工具中创建其他的对象以完成一定的逻辑判断功能。

有的时候，用户可能已经建立了一些对象了，这时，可以利用上述的两种方法创建 Box 对象，然后将 Box 框扩大到能够包含所以需要包入的对象。

将所有的对象放入 Box 中后，用户还可以将整个含对象的 Box 打包成一个图形对象，只要右击 Box，在弹出的下拉菜单中选择 Make Contents－Group 或简单的双击该 Box 对象，即可完成打包过程，打包的对象边框变粗；用户也可以右击 Box，在弹出的下拉菜单中选择 Make Contents－Subcharted 隐含 Box 中的对象，将 Box 中的对象变成子图形式。见图 12.28。

对一个 Box 添加数据，可以使 Box 中的所有元素共享该数据。

大多数情况下，Boxes 不改变 Stateflow 图实现的逻辑判断功能，但是在存在并行状态时，它却影响着 Stateflow 图中的激活顺序。Stateflow 图中的 Box 比它右边任何并行状态和 Box 要先激活。在一个 Box 内，并行状态的激活顺序依然是上下左右的顺序。（关于并行状态的问题，下面即将讲述）

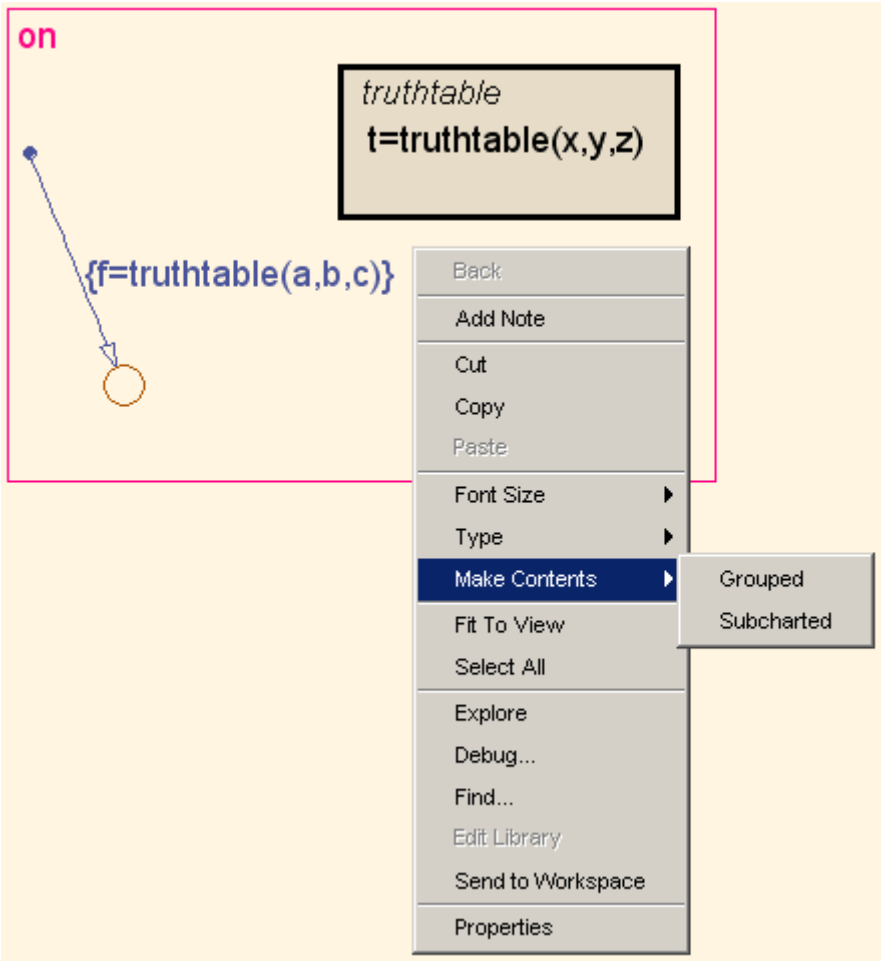


图 12.28 Box 对象的打包和创建分图方法

12、含并行执行状态的 Stateflow 图

到目前为止，我们在前面的举例中所创建的状态都是单一状态（Exclusive State 或称为 Or State），单一状态的边界是实线。单一状态的最大特征是同一时间，父状态中仅有一个子状态可以处于激活状态。

并行状态（Parallel State 或称为 And State）的边界是虚线，并行状态的特点是父状态中的多个子状态可以同时处于激活状态。虽然说并行状态可以同时被激活，但还是存在一个激活顺序的问题。并行状态在状态图中的位置决定着其激活的顺序。位于上部的状态较位于下部的状态较早被激活，同一高度层中，位于左边的状态较位于右边的状态较早被激活。激活的顺序依照上－下，左－右的原则。

下面我们举例来控制两个独立的被控对象，以此说明如何建立并行状态，以及并行状态的激活顺序。

例 12.8　现有一间车间，为了改善工人的工作环境，车间安装了两台排气扇。一般情况下，只需一台排气扇工作，但当室内的温度超过 28℃时，启动第二台排气扇，两台同时工作。这样两台排气扇必须独立控制。我们建立如下模型进行两台排气扇的并行控制。

解　一、建模　Stateflow 图的创建

在 MATLAB 命令窗口键入 sfnew，打开一个含有 Stateflow 模块的 Simulink 模型窗口。点击 Stateflow 模块，打开 Stateflow 编辑界面。建立两个含子状态的上层状态 Fan1 和 Fan2，两状态内含的状态、状态迁移及其迁移事件、条件情况见图 12.29（a）。状态 Fan2 中的迁移条件[in(Fan1.Off) | (temp<=28)]表示若 Fan2 的子状态 On 是激活状态，当状态 Fan1 中的状态 Off 是激活状态或 temp<=28 为真时，Fan2 中发生从状态 On 到状态 Off 的迁移；类似地，状态 Fan2 中的迁移条件[in(Fan1.On) & (temp>28)] 表示若 Fan2 的子状态 Off 是激活状态，当状态 Fan1 中的状态 On 是激活状态且 temp<=28 为真时，Fan2 中发生从状态 Off 到状态 On 的迁移。

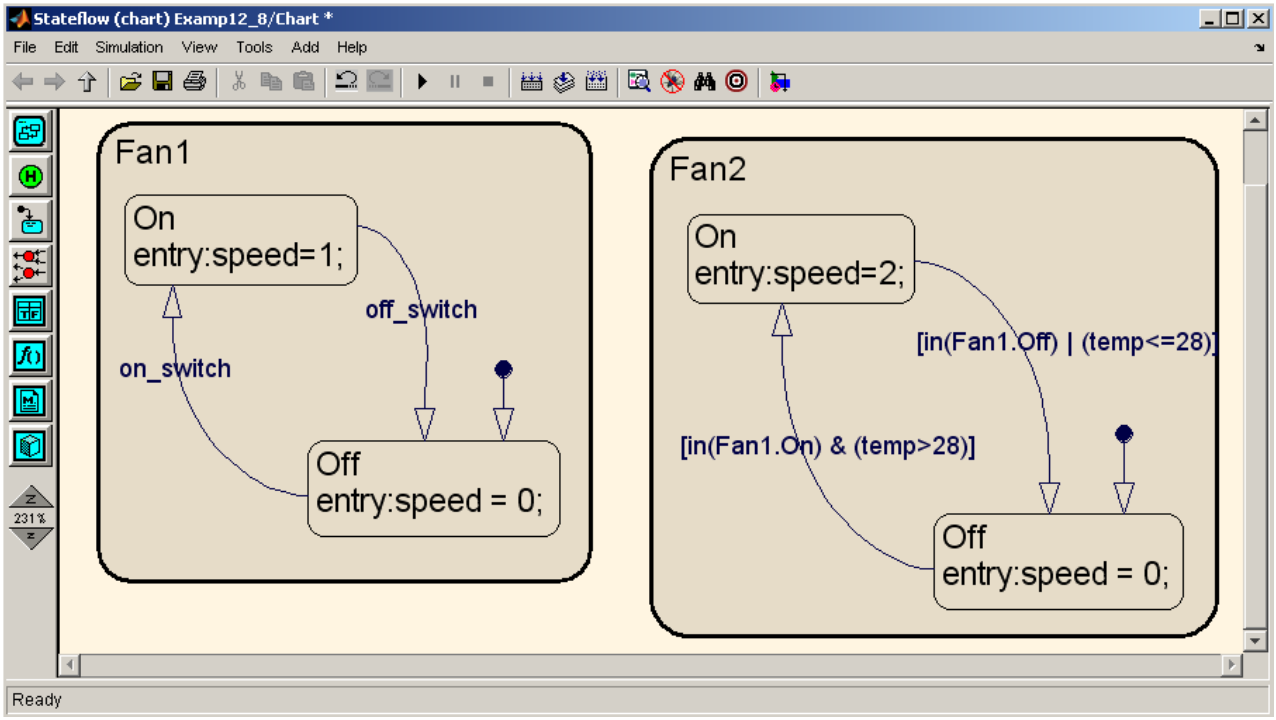
在 Stateflow 编辑界面的空白处，右击鼠标，在弹出的下拉菜单中选择 Decomposition－Parallel（AND）即可将两个状态 Fan1 和 Fan2 设置为并行状态，表现为该两状态的边框变为虚线，且 Stateflow 按照上下左右的原则，自动给出此两个并行状态执行的顺序，并在其右上角标出顺序号。（用户可以通过改变状态 Fan1 和 Fan2 的位置来观察 Stateflow 中并行状态执行顺序的变化）。见图 12.29(b)和 12.29(c)。此 Stateflow 图中的数值 temp 是需要从 Simulink 模型中输入的，除此之外，此 Stateflow 图还需利用 Simulink 模型中的 temp_event 事件以触发此 Stareflow 图每 0.5 秒读一次 temp 数值并判断是否应该状态迁移。Stateflow 图需添加三个 Input from Simulink 的事件 off_switch、on_switch 和 temp_event、一个 Input from Simulink 的数值 temp 和一个 Output to Simulink 的数据 speed。将事件 off_switch、on_switch 和 temp_event 的触发沿分别设置成 Falling、Rising 和 Either。添加后，选择 Stateflow 图中的 Tools 菜单下的 Explorer，打开模型管理器 Model Explore，其中的事件和数值设置应如图 12.29（d）所示。

在 Simulink 模型中，用 Constant4 模块模拟测量的温度值，开始时设置为 25。Pulse Generator 产生的周期和幅值均为 1 的方波信号用来触发 Stateflow 图，每 0.5 秒判断一次温度值的大小。实际上，Pulse Generator 产生的方波信号是来模拟温度测量的采样周期。其他四个 Constant 模块及两个 Manual Switch 模块用来产生触发事件 off_switch 和 on_switch 的。该例的 Simulink 模型见图 12.29(e)。

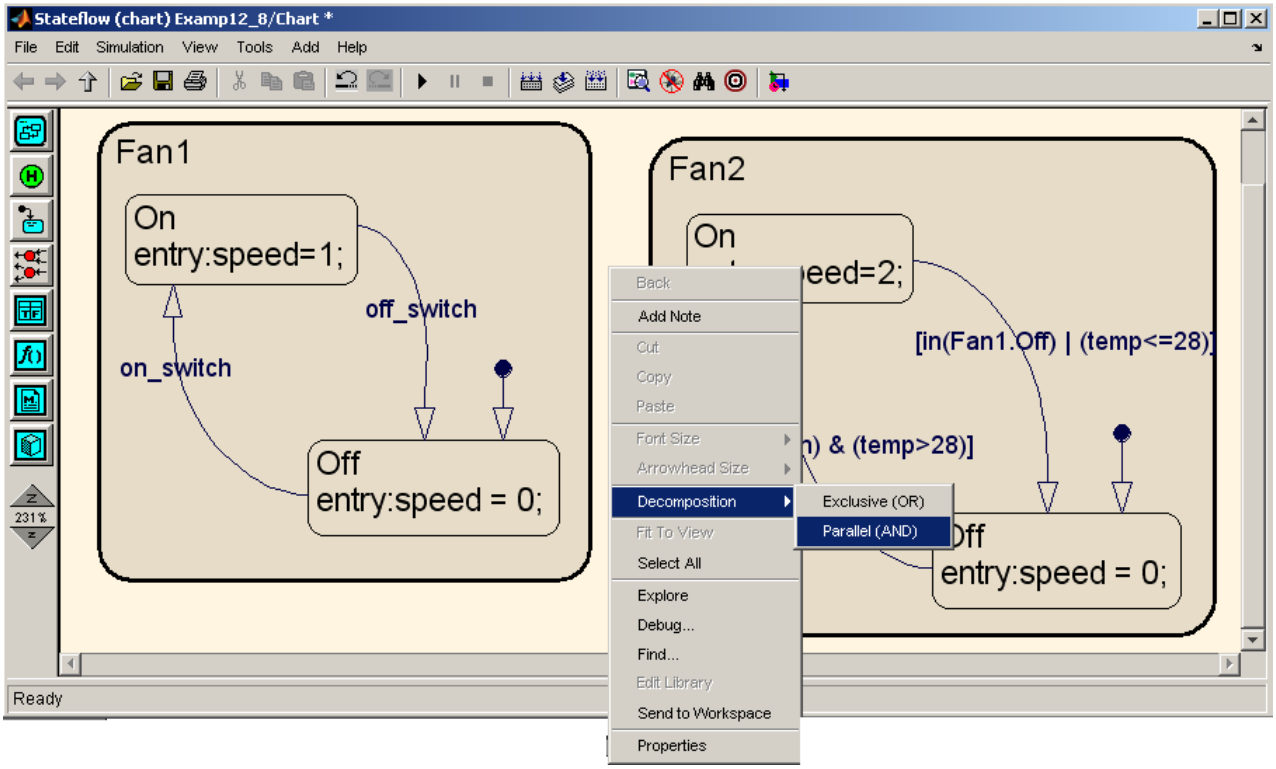
二、 仿真

在 Simulink 模型窗口中，选择 Simulation 菜单下的 Configuration Parameters，将仿真终止时间设置为 inf。

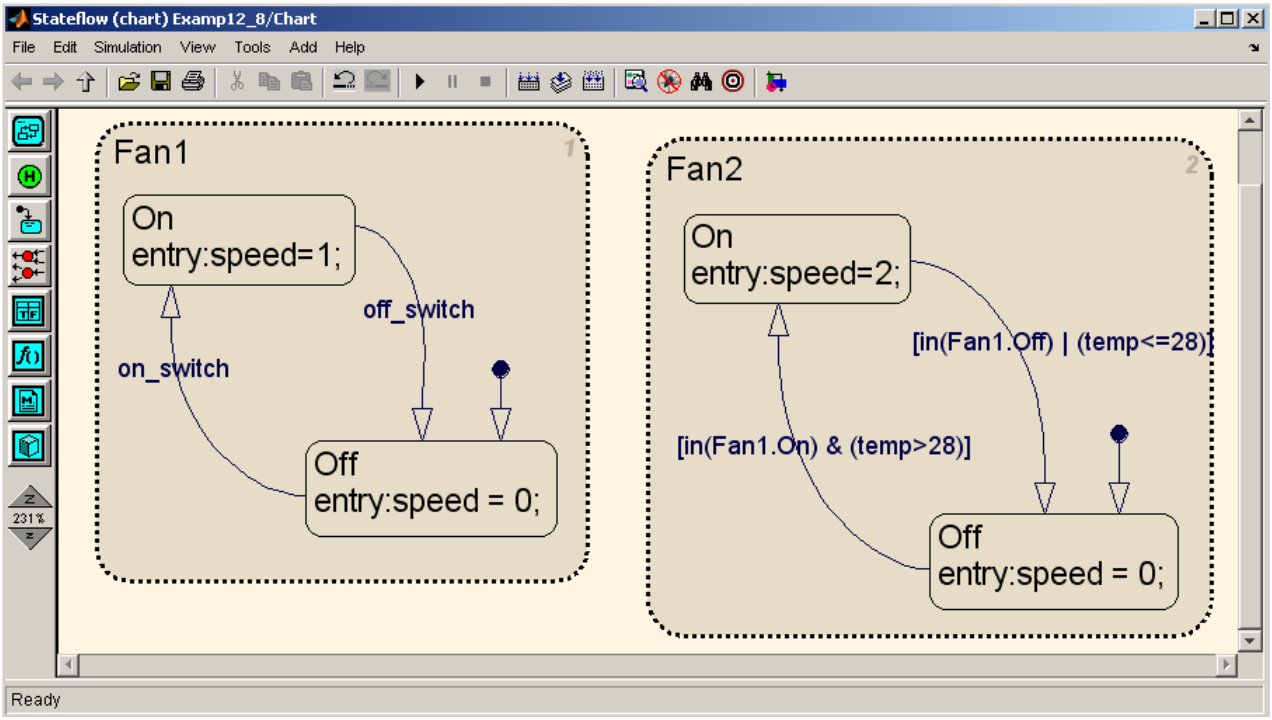
选择 Simulation 菜单下的 Start，启动仿真。仿真开始时，Simulink 先分析 Stateflow 图，如没有错误，就按照 Stateflow 图自动生成 S－函数，并存在当前目录下的 sfprj 子目录下。这时真正的仿真才开始。本例开始时，Stateflow 图处于休眠状态，没有状态是激活的。Stateflow 图每 0.5 秒会收到一个事件 temp_event，当接到第一个 temp_event 事件时，Stateflow 图就被唤醒了，此时 Stateflow 图知道其内含两个平行状态，这时就要先执行编号为 1 的并行状态 Fan1 的缺省状态迁移，激活状态 Fan1.Off，speed 输出为 0，见图 12.30(a)；随后执行并行状态 Fan2 的缺省状态迁移，激活状态 Fan2.Off，speed 输出仍为 0，见图 12.30(b)；改变 Manual Switch1 的开关位置，以产生一个上升沿的 on_switch 触发信号，Stateflow 产生状态 Fan1.Off 到 Fan1.On 的状态迁移，停止状态 Fan1.Off，激活 Fan1.On，speed 输出为 1，此时，因为 Stateflow 判断了到 temp 值小于 28，因而并行状态 Fan2 中不发生状态 Fan2.Off 到状态 Fan2.On 的状态迁移（因为不满足迁移条件[in(Fan1.On) & (temp>28)]），见图 12.30(c)；前面讲了，Stateflow 图每 0.5 秒会收到一个 temp_event 事件，如果用户这时将 Simulink 模型中的 Constant4 模块的值改为 30，在改值后 Stateflow 图接到第一个 temp_event 事件时，Stateflow 判断并行状态 Fan2 中迁移条件[in(Fan1.On) & (temp>28)]满足，即发生状态 Fan2.Off 到状态 Fan2.On 的状态迁移，停止状态 Fan2.Off，激活 Fan2.On，将 speed 值置为 2，见图 12.30(d)；如果用户改变 Manual Switch 的开关位置，以产生一个下降沿的 off_switch 触发信号，则再次发生从 Fan1.On 状态至 Fan1.Off 状态的状态迁移，激活 Fan1.Off 状态，将 speed 值置为 0，同时 Fan2.On 状态至 Fan2.Off 状态的迁移条件也满足，也发生 Fan2.On 状态至 Fan2.Off 状态的状态迁移，激活状态 Fan2.Off，将 speed 值置为 0，见图 12.30(e)。本例中状态迁移的过程及迁移过程中的信号流向详见图 12.30。读者也可以自行调解参数 temp 值的大小及 Manual Switch、Manual Switch1 的位置，以观察此例的状态迁移情况。


Stateflow (chart) Examp12_8/Chart *
File Edit Simulation View Tools Add Help
Fan1
On
entry:speed=1;
off_switch
on_switch
Off
entry:speed = 0;
Fan2
On
entry:speed=2;
[in(Fan1.Off) | (temp<=28)]
[in(Fan1.On) & (temp>28)]
Off
entry:speed = 0;
231 %
Ready

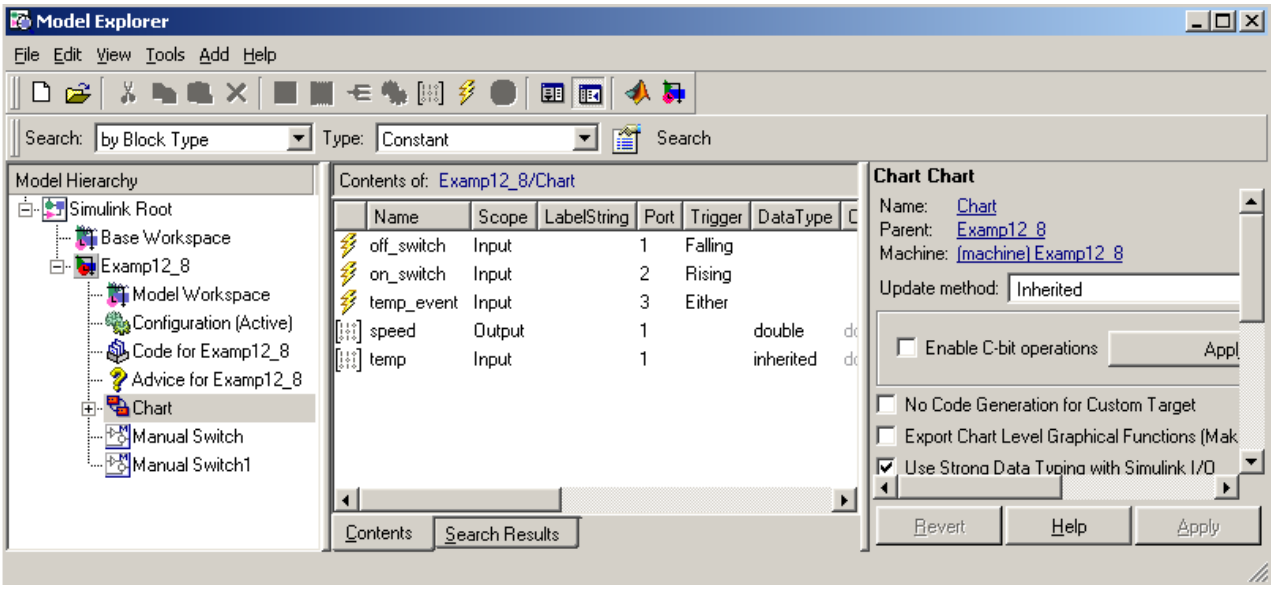
（a）


Stateflow (chart) Examp12_8/Chart *
File Edit Simulation View Tools Add Help
Fan1
On
entry:speed=1;
off_switch
on_switch
Off
entry:speed = 0;
Fan2
[in(Fan1.Off) | (temp<=28)]
entry:speed = 0;
231 %
Ready
Back
Add Note
Cut
Copy
Paste
Font Size
Arrowhead Size
Decomposition
Fit To View
Select All
Explore
Debug...
Find...
Edit Library
Send to Workspace
Properties
Exclusive (OR)
Parallel (AND)

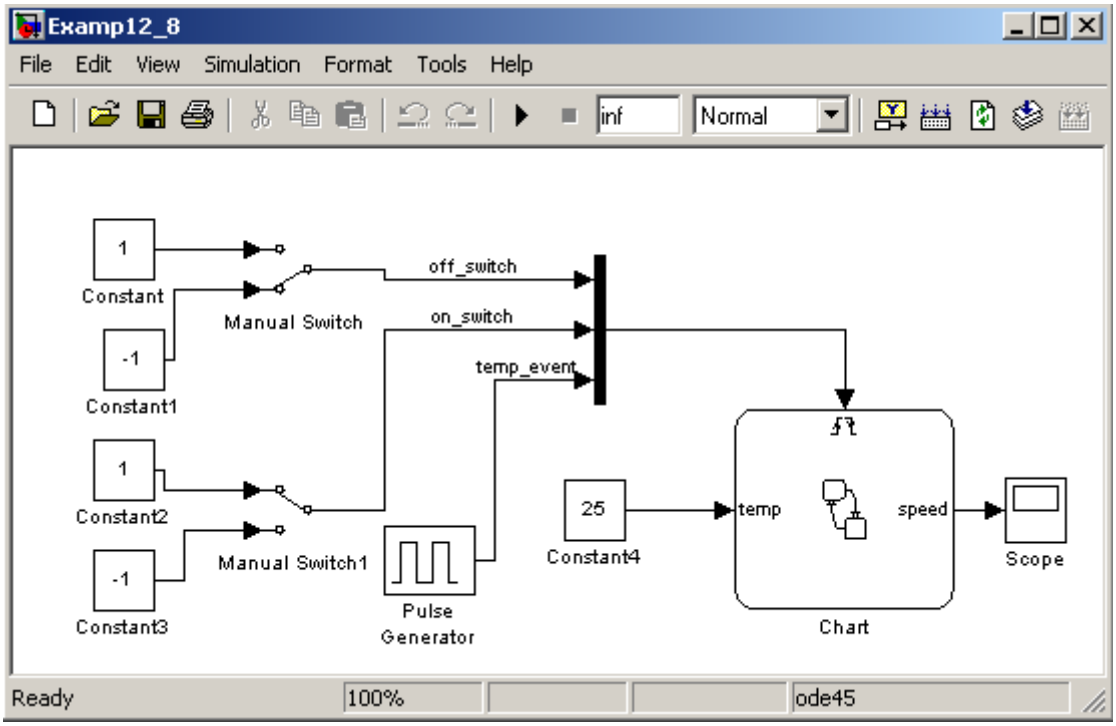
（b）


Stateflow (chart) Examp12_8/Chart
File Edit Simulation View Tools Add Help
Fan1
1
On
entry:speed=1;
off_switch
on_switch
Off
entry:speed = 0;
Fan2
2
On
entry:speed=2;
[in(Fan1.Off) | (temp<=28)]
[in(Fan1.On) & (temp>28)]
Off
entry:speed = 0;
231%
Ready

（c）

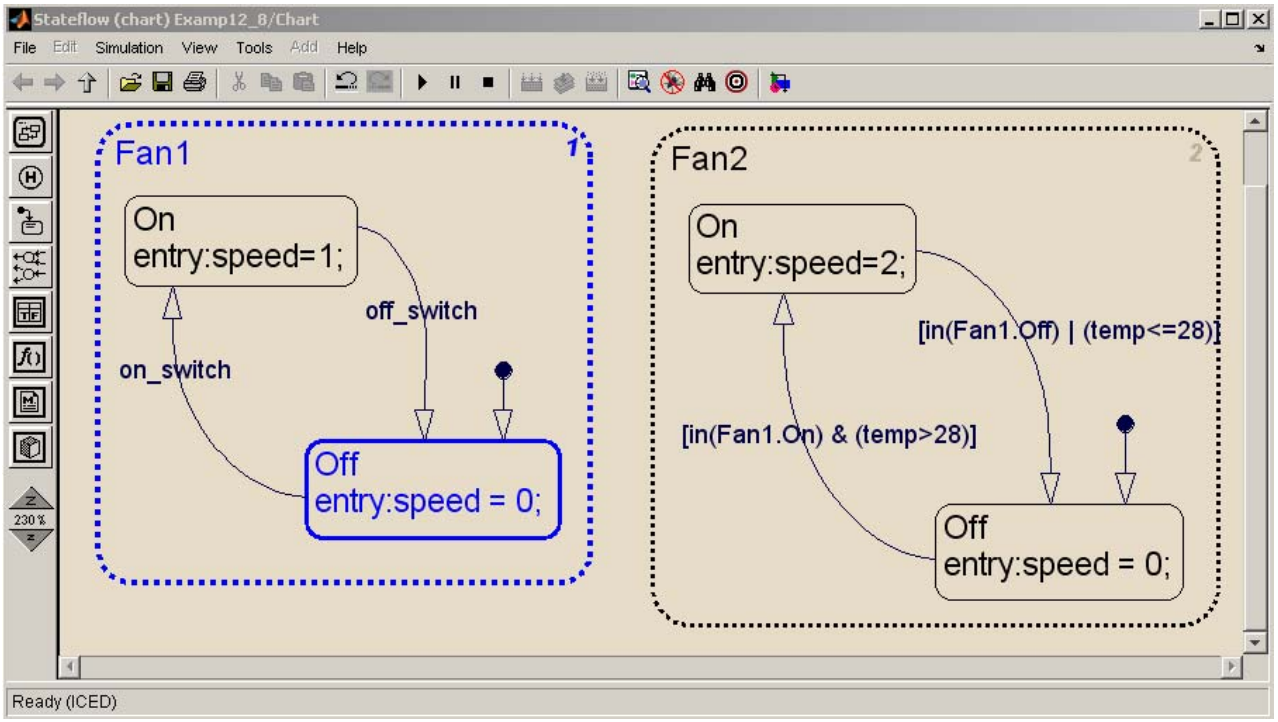

Model Explorer
File Edit View Tools Add Help
Search: by Block Type
Type: Constant
Search
Model Hierarchy
Simulink Root
Base Workspace
Examp12_8
Model Workspace
Configuration (Active)
Code for Examp12_8
Advice for Examp12_8
Chart
Manual Switch
Manual Switch1
Contents of: Examp12_8/Chart
Name Scope LabelString Port Trigger DataType
off_switch Input 1 Falling
on_switch Input 2 Rising
temp_event Input 3 Either
speed Output 1 double
temp Input 1 inherited
Contents
Search Results
Chart Chart
Name: Chart
Parent: Examp12_8
Machine: (machine) Examp12_8
Update method: Inherited
Enable C-bit operations
No Code Generation for Custom Target
Export Chart Level Graphical Functions (Mak
Use Strong Data Typing with Simulink I/O
Revert
Help
Apply

（d）

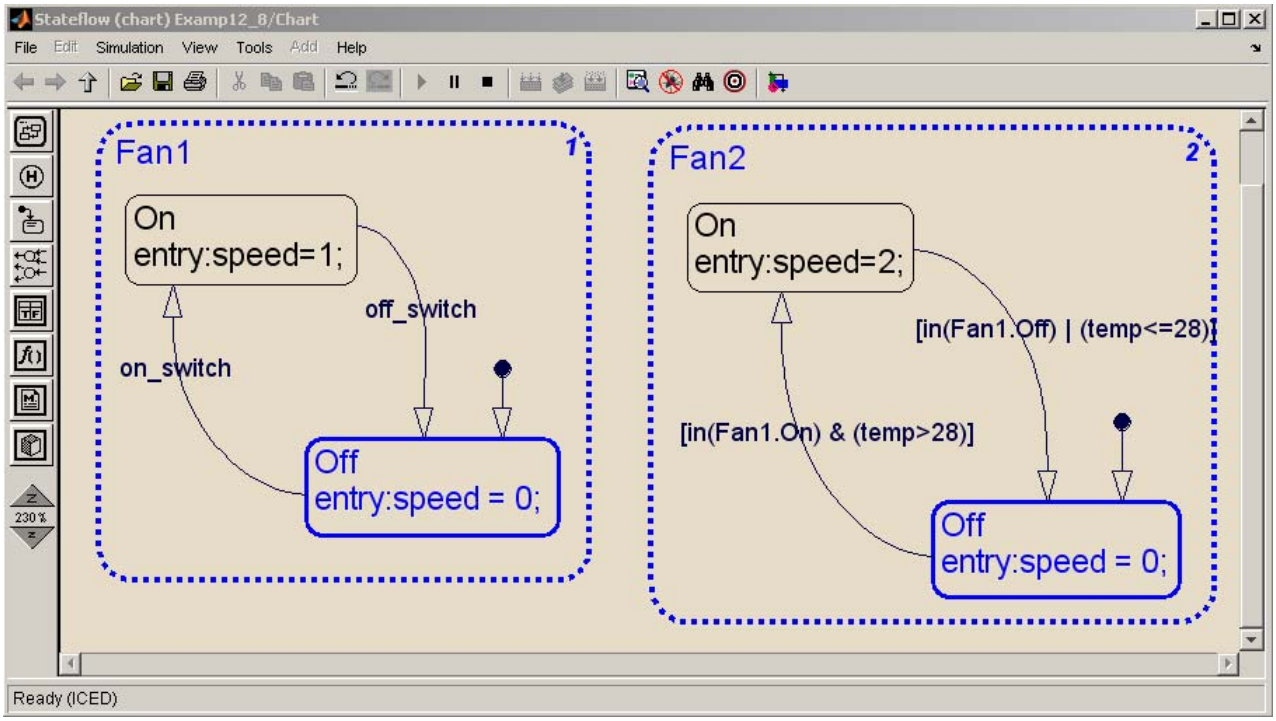


(e)

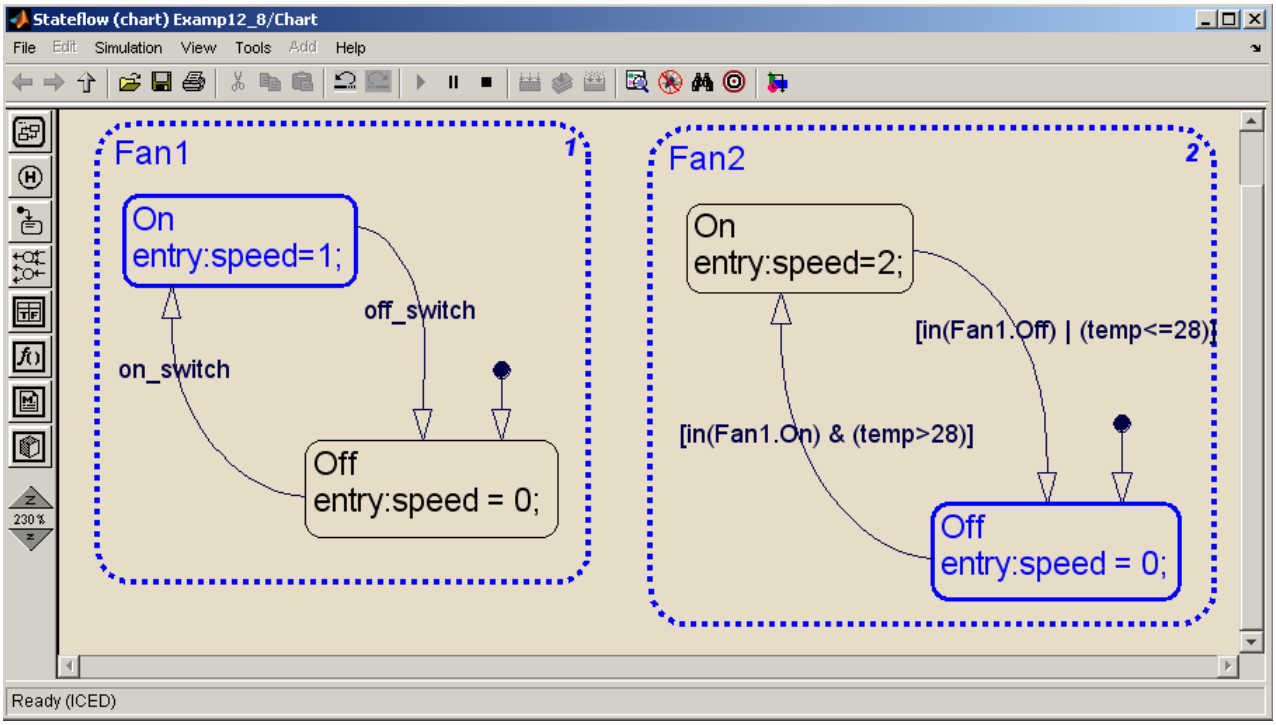
图 12.29　例 12.8 的建模方法



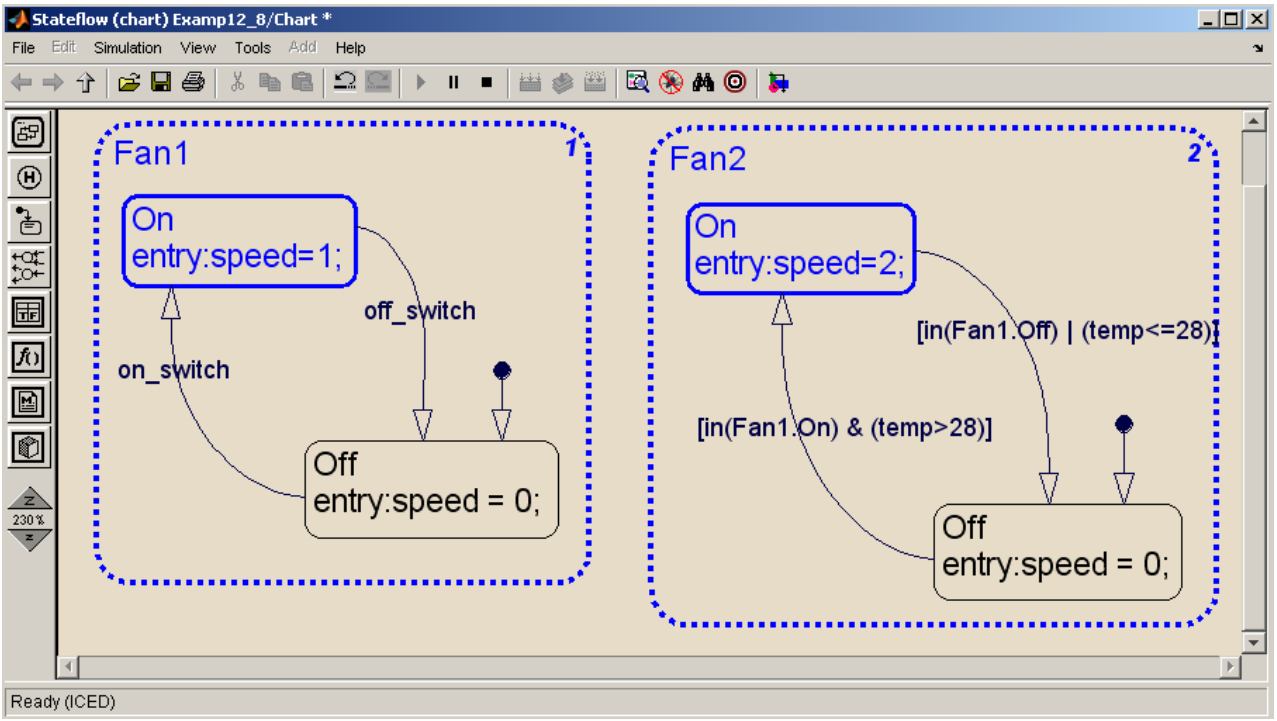
（a）


Stateflow (chart) Examp12_8/Chart
File Edit Simulation View Tools Add Help
Fan1
1
On
entry:speed=1;
off_switch
on_switch
Off
entry:speed = 0;
Fan2
2
On
entry:speed=2;
[in(Fan1.Off) | (temp<=28)]
[in(Fan1.On) & (temp>28)]
Off
entry:speed = 0;
230 %
Ready (ICED)

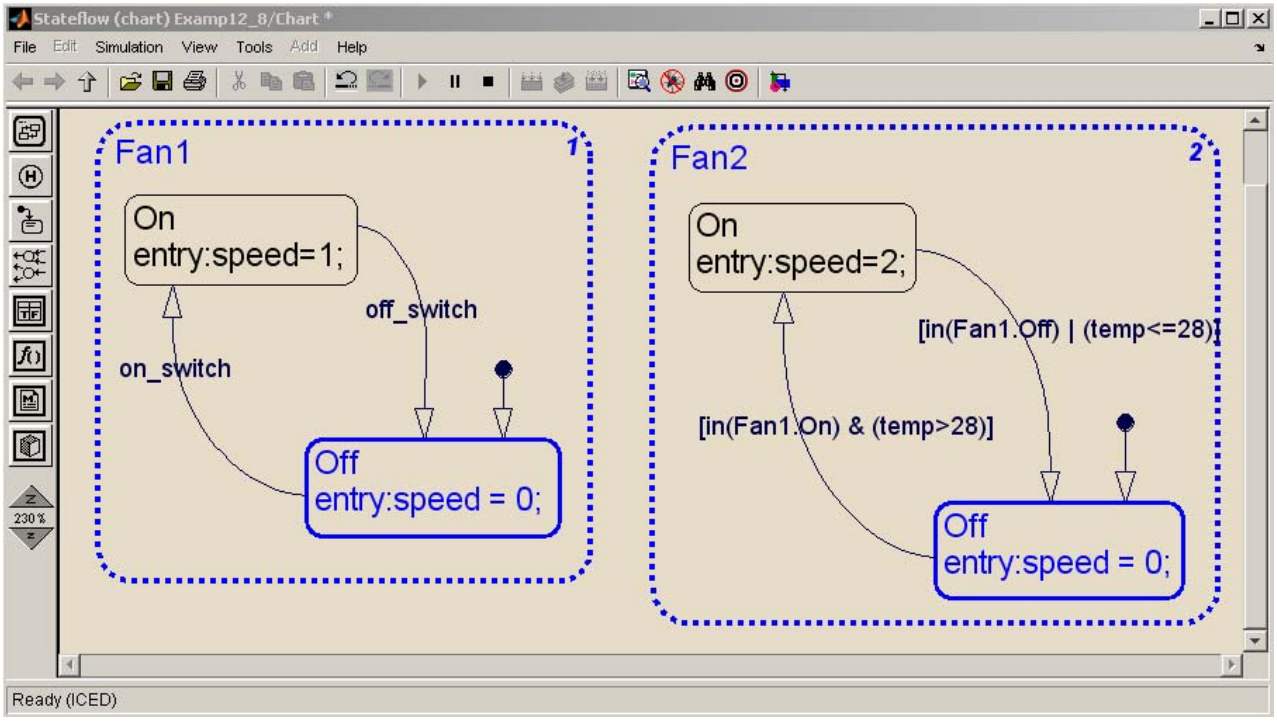
（b）


Stateflow (chart) Examp12_8/Chart
File Edit Simulation View Tools Add Help
Fan1
1
On
entry:speed=1;
off_switch
on_switch
Off
entry:speed = 0;
Fan2
2
On
entry:speed=2;
[in(Fan1.Off) | (temp<=28)]
[in(Fan1.On) & (temp>28)]
Off
entry:speed = 0;
230 %
Ready (ICED)

（c）



（d）



（e）

图 12.30　例 12.8 系统状态迁移过程

12.3　利用 Stateflow 完成复杂的状态逻辑判断及其动作

至此为止，本教材已经对 Stateflow 中的基本工具给予了简单的介绍，用户根据前面讲述可以进行相对简单的逻辑关系的判断。但 Stateflow 中还有很多深层的知识可能在以后的实际项目中用到，我们下面再对这些内容做一些介绍。

1、Stateflow 中的动作

（1）条件动作和迁移动作

前面在讲状态迁移时曾经讲过状态迁移的标记，状态迁移标记的一般形式为

触发事件[迁移条件关系式]{条件动作}/迁移动作

其中包含有条件动作和迁移动作。条件动作是指当条件关系式一旦成立（即为真），就执行的动作，通常发生在迁移终点被确定有效之前。如果没有规定条件关系式，则认为条件关系式为真，即刻执行条件动作。迁移动作是指当迁移终点已经确定有效，才执行的动作。如果迁移包含很多阶段，迁移动作只有在整个迁移通道到最终的终点确认为有效后方可执行。这两种动作在图 12.7 的示例中都给出了简单的介绍，用户可以查询前面的相关内容。

（2）状态动作

除了条件动作和迁移动作外，Stateflow 还有一种很重要的动作——状态动作。Stateflow 中含有多种状态动作，用户可以在不同时间，执行不同的动作。状态动作是在状态中标记的。首先我们来讲述状态标记的方法。

对一个状态进行标记包括给这个状态规定状态名以及这个状态在进入、退出和处于激活状态下又接受到一个事件所应执行的动作。

状态标记的一般格式如下，其中第一行规定状态名称，其余各行规定状态的动作，每个状态的动作必须单独另起一行。

name/

entry:entry actions

during:during actions

exit:exit actions

bind:data and events

on event_name:on event_name actions

表 12.2 给出了 Stateflow 中状态动作的具体解释。

表 12.2　Stateflow 中各种状态动作说明

关键词	输入内容	描述
无	name	在 name 处输入状态名，随后紧跟’/’
entry 或 en	entry actions	entry actions 状态进入动作。表示发生状态迁移，激活了该状态时需要执行的动作
during 或 du	during actions	during actions 状态仍然激活动作。表示原处于激活的状态受到一个事件的触发，不存在从这个状态发出的状态迁移时，此状态仍处于激活状态需要执行的动作
exit or ex	exit actions	exit actions 状态退出动作。表示存在由此状态发出的有效状态迁移时，该状态退出时执行的动作
bind	data and events	数据事件绑定动作。将数据 data 和事件 events 绑定在此状态上。绑定的数据只能在此状态或其子状态内被改写，其他状态只能读取此数据。绑定的事件由此状态或其子状态广播。
on	event_name；on event_name	特定事件发生动作。event_name 规定一个特定的事件；on event_name actions 表示当该状态是激活状态且 event_name 规定的事件发生时需

	actions	要执行的动作。

我们在前面的举例中对状态做了这样的标记

On/

Entry:speed=1;

表示状态名为 On，当进入状态 On 时，执行状态 entry 进入动作，给变量 speed 赋值。

图 12.31 中给出了一个含条件动作、迁移动作和状态动作的 Stateflow 图。

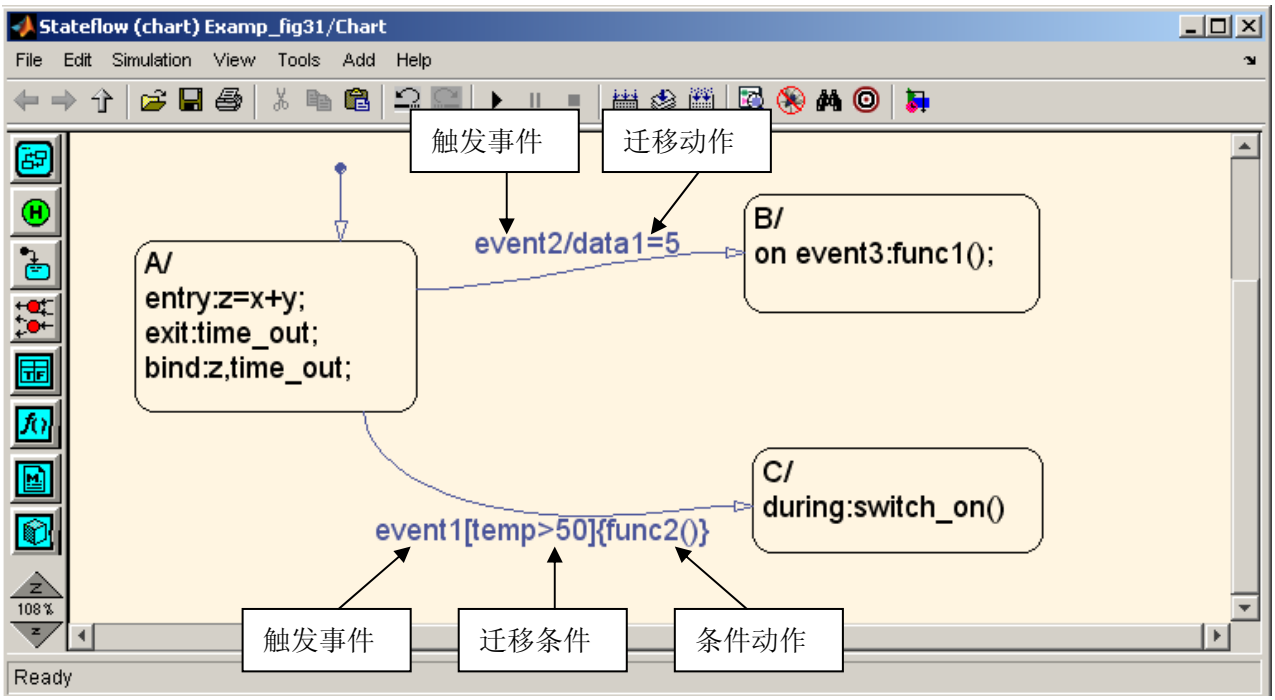


图 12.31　含条件动作、迁移动作和状态动作的 Stateflow 图

当图 12.31 所示的 Stateflow 图被唤醒后，将发生下列迁移或动作：

首先，执行缺省状态迁移，执行状态 A 的 entry 进入动作，即计算 z＝x＋y，事件 time_out 绑定在状态 A 中，激活状态 A。

如果状态 A 是激活状态，Stateflow 图接收到了事件 event2，则状态 A 至 C 的迁移是有效的，执行状态 A 的 exit 退出动作，即广播事件 time_out，停止状态 A，执行状态迁移动作，即将 data1 的值置为 5，激活状态 B。若此时 Stateflow 图接收到了事件 event3，则执行事件 event3 指定的动作，即调用函数 func1（）。

如果状态 A 是激活状态，Stateflow 图接收到了事件 event1，判断迁移条件 temp > 50 是否为真，如果为假，则停止。如果为真，执行条件动作，即调用函数 func2（），状态 A 至 C 的迁移是有效的，执行状态 A 的 exit 退出动作，即广播事件 time_out，停止状态 A，激活状态 C。在状态 C 处于激活状态时，如果此时 Stateflow 图接收到了事件 event3，则不会发生任何状态迁移，状态 C 仍处于激活状态，执行 C 状态的 during 动作，即调用函数 switch_on()。

2、 Stateflow 中的隐含事件

在上节中，我们针对 Stateflow 的事件，介绍的事件添加，事件的触发、事件属性的设置等。事实上除了上节介绍的真实的事件外，Stateflow 中还含有一些隐含的事件。

当 Stateflow 图被唤醒时、进入到某个状态、从某个状态退出或某个内部数据（非输入数据）赋值时，Stateflow 会定义并触发某种事件。这些事件是 Stateflow 自动定义触发的，非用户定义、添加的，故称为隐含事件。

表 12.3 给出了 Stateflow 中的隐含事件的类型和含义

表 12.3　Stateflow 中的隐含事件类型及含义

隐 含 事 件	含　　义
change（data_name） chg(data_name)	当变量 data_name 的数值发生变化时，定义或产生一个局部事件。
enter(state_name) en(state_name)	进入状态 state_name 时，定义或产生一个局部事件。
exit(state_name) ex(state_name)	退出状态 state_name 时，定义或产生一个局部事件。
wakeup	动作图刚刚唤醒时，定义或产生一个局部事件。
tick	同 wakeup

如例 12.8 所述，如果用户要求两个排气扇总是同时工作，则图 12.32 所示的 Stateflow 图中利用两个 enter 隐含事件几乎可以使两个排气扇的状态 On 和状态 Off 同时进入。图 12.32 中，两个排气扇 Fan1 和 Fan2 是并行状态。当 Stateflow 图被某个事件唤醒时，执行缺省状态迁移，状态 Fan1.Off 和 Fan2.Off 激活。当第一次 on_switch 事件发生时，发生从状态 Fan1.Off 到 Fan1.On 的状态迁移。当 Fan1.On 的进入动作执行时，即刻广播一个隐含的局部事件 en(Fan.On) == 1。这个隐含事件触发从状态 Fan2.Off 到 Fan2.On 的状态迁移。类似地，当系统发生从状态 Fan1.On 到 Fan1.Off 的状态迁移时，广播隐含局部事件 en（Fan1.Off），这时触发从 Fan2.On 到 Fan2.Off 的状态迁移。

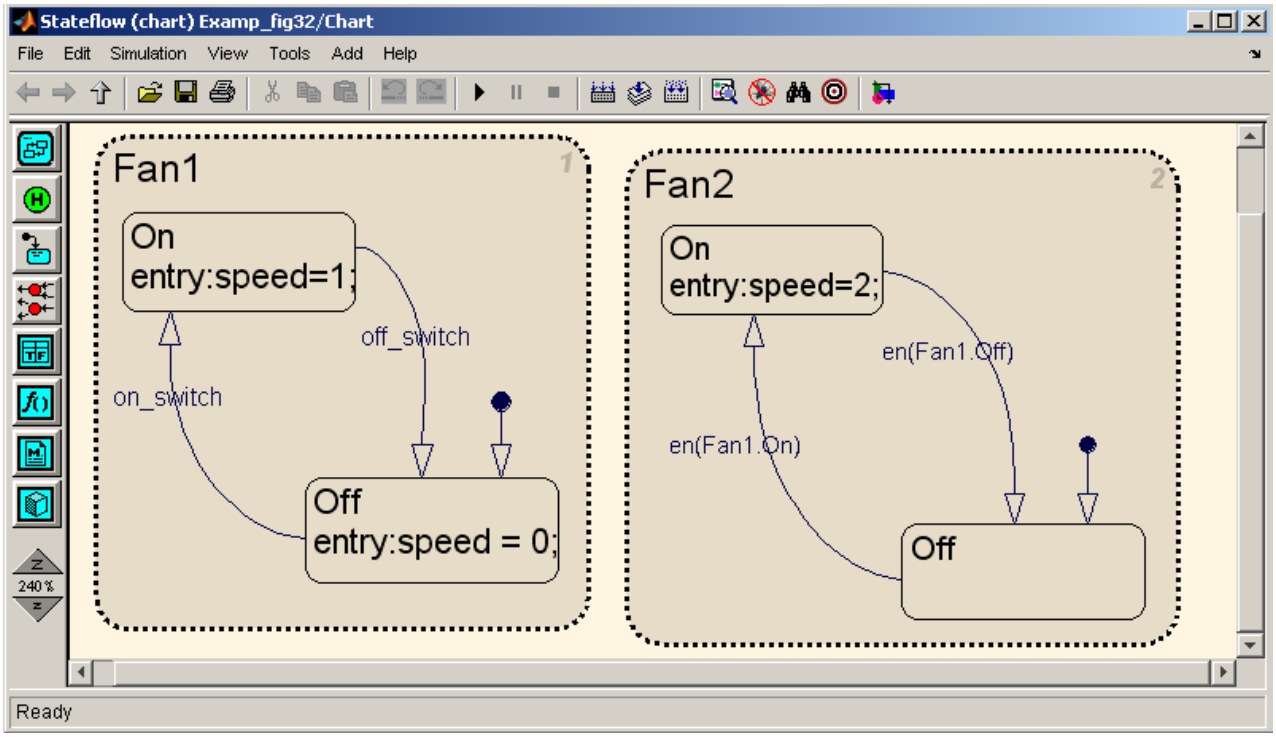


图 12.32　隐含事件应用举例

3、动作中的瞬时逻辑操作（Temporal Logic Operators）

这部分我们介绍如何在动作中使用瞬时逻辑操作。瞬时逻辑操作是布尔操作，对状态流内部事件反复进行计数。图 12.33 是一个含有瞬时逻辑操作的示例。

在瞬时逻辑操作中有几个一般的概念需要先介绍一下。

瞬时操作中被反复计数的 Stateflow 的内部事件称作基事件（base event）。任何的 Stateflow 事件（包括 enter 事件 、exit 事件或 data change 事件等隐含事件）都可以做为瞬时操作的基事件。

对于没有 Simulink 输入事件的 Stateflow 图，用户可以使用隐含事件 wakeup（或 tick）来唤醒 Stateflow

图。瞬时逻辑操作只能发生在从状态发出的状态迁移条件中或发生在状态动作中。这就意味着瞬时逻辑操作不能用在缺省迁移或流程图的迁移中。

含瞬时操作的迁移出发的状态或其动作中出现瞬时操作的状态称为瞬时操作相连状态（temporal operator's associated state）。

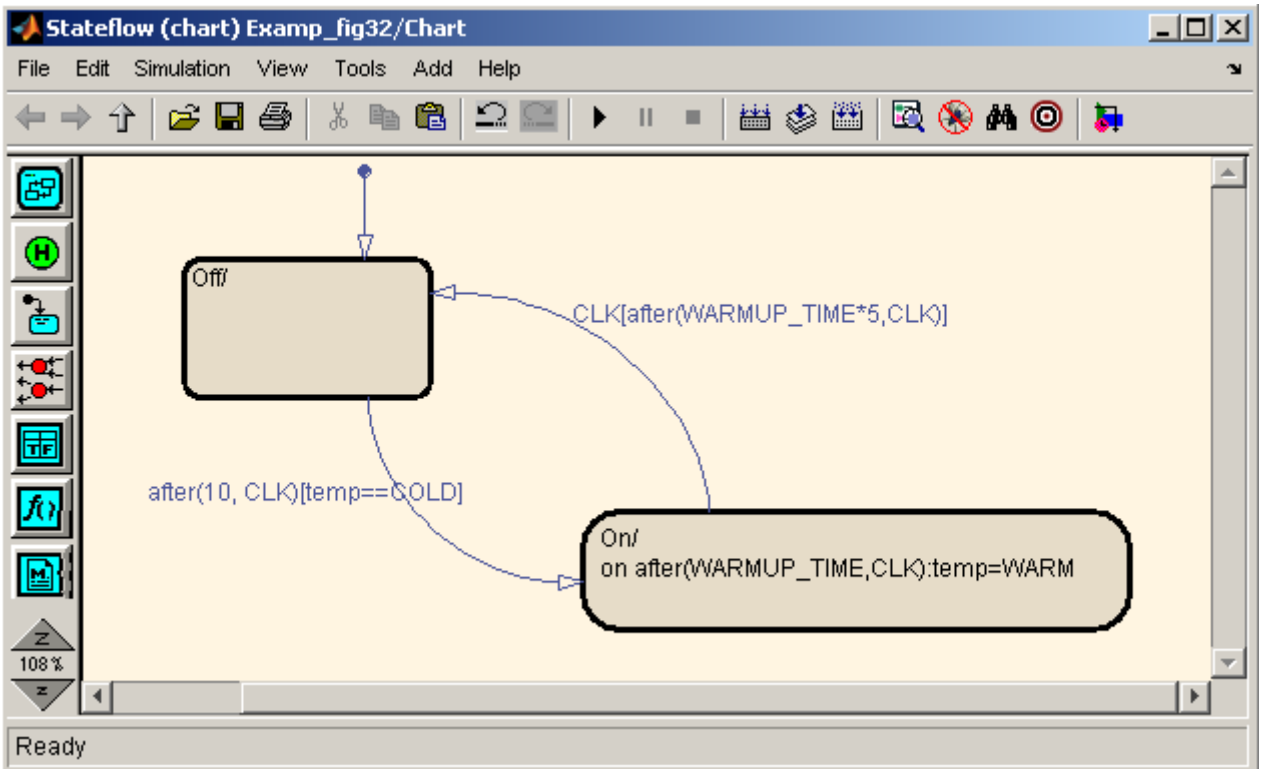


图 12.33　含瞬时逻辑操作的 Stateflow 图示例

常用的瞬时逻辑操作有以下几种：

（1）　after 瞬时逻辑操作（after Temporal Logic Operator）

after 瞬时逻辑操作的格式为 after(n, E)

式中，E 是 after 瞬时逻辑操作的基事件，n 为大于 0 的整数或一个结果是整数且大于 0 的表达式。

相连状态激活后，当基事件 E 发生 n 次后，after 逻辑操作为真，否则为假。对于没有输入事件的 Stateflow 图，after（n，wakeup）（或 after（n，tick））表示当该图被唤醒 n 次后，after（n，wakeup）（或 after（n，tick））瞬时逻辑为真。

需要指出的是 after 瞬时逻辑操作的相连状态每次激活时，基事件 E 的计数器都被复位为 0。

下面给出几个例子说明 after 瞬时逻辑操作的应用。

状态迁移标记中的 after 瞬时逻辑，如 CLK[after(10, CLK) && temp == COLD]表示相连状态激活后，基事件 CLK 发生 10 次，且变量 temp 的值是 COLD 时，发生从相连状态出发的状态迁移。after(10, CLK)[temp == COLD]与 CLK[after(10, CLK) && temp == COLD]的意义相同。

状态迁移标记中的[after(10, CLK)]仅设置了迁移条件，表示相连状态激活后，发生了 10 次基事件 CLK 后，任何其他的触发事件均可触发从相连状态出发的状态迁移。

状态迁移标记中的 CLK[after(10,CLK)] 和 ROTATE[after(10,CLK) 表示不同的含义。CLK[after(10,CLK)] 表示相连状态激活后，发生第 10 次基事件 CLK 时，触发从相连状态出发的状态迁移；而 ROTATE[after(10,CLK)表示相连状态激活后，在不少于 10 次基事件 CLK 发生后，由事件 ROTATION 触发状态迁移，由事件 ROTATION 决定触发状态迁移的时间。

状态动作中的 after 瞬时逻辑，如

on/

on after(5*BASE_DELAY, CLK): status('on');

表示在相连状态激活后，在每个基事件 CLK 周期内显示状态信息，开始时间是 CLK 发生 5×BASE_DELAY 的时间。

（2） before 瞬时逻辑操作（before Temporal Logic Operator）

before 瞬时逻辑操作的格式为 before(n, E)

式中，E 是 before 瞬时逻辑操作的基事件，n 为大于 0 的整数或一个结果是大于 0 的整数的表达式。

相连状态激活后，当基事件 E 发生的次数小于 n，before 逻辑操作为真，否则为假。对于没有输入事件的 Stateflow 图，before（n，wakeup）（或 before（n，tick））表示当该图被唤醒次数少于 n 时，before（n，wakeup）（或 before（n，tick））瞬时逻辑为真。

需要指出的是 before 瞬时逻辑操作的相连事件每次激活时，基事件 E 的计数器都被复位为 0。

下面给出几个例子说明 before 瞬时逻辑操作的应用。

状态迁移标记中的 before 瞬时逻辑，如 ROTATION[before(10, CLK)]表示相连状态激活后，基事件 CLK 发生的次数少于 10 次时，由事件 ROTATION 触发状态迁移，由事件 ROTATION 决定触发状态迁移的时间。

状态动作中的 before 瞬时逻辑，如

on/

on before(MAX_ON_TIME, CLK): temp++;

表示在相连状态激活后，在每个基事件 CLK 周期中，将 temp 值加 1，直到 CLK 事件发生 MAX_ON_TIME 次为止。

（3） at 瞬时逻辑操作（at Temporal Logic Operator）

at 瞬时逻辑操作的格式为 at(n, E)

式中，E 是 at 瞬时逻辑操作的基事件，n 为大于 0 的整数或一个结果是大于 0 的整数的表达式。

相连状态激活后，当第 n 次基事件 E 发生时，at 逻辑操作为真，否则为假。对于没有输入事件的 Stateflow 图，at（n，wakeup）（或 at（n，tick））表示当该图被第 n 次唤醒时，at（n，wakeup）（或 at（n，tick））瞬时逻辑为真。

需要指出的是 at 瞬时逻辑操作的相连事件每次激活时，基事件 E 的计数器都被复位为 0。

下面给出几个例子说明 at 瞬时逻辑操作的应用。

状态迁移标记中的 at 瞬时逻辑，如 ROTATION[at(10, CLK)]表示相连状态激活后，在基事件 CLK 发生第 10 次时，由事件 ROTATION 触发状态迁移；而 at(10, CLK)表示相连状态激活后，第 10 次发生的基事件 CLK 触发状态迁移。

状态动作中的 at 瞬时逻辑，如

on/

on at(10, CLK): status('on');

表示在相连状态激活后，当基事件 CLK 第 10 次发生时，显示状态信息。

（4） every 瞬时逻辑操作（every Temporal Logic Operator）

every 瞬时逻辑操作的格式为 every(n, E)

式中，E 是 every 瞬时逻辑操作的基事件，n 为大于 0 的整数或一个结果是大于 0 的整数的表达式。

相连状态激活后，基事件 E 每发生 n 次时，every 逻辑操作为真，否则为假。对于没有输入事件的 Stateflow 图，every（n，wakeup）（或 every（n，tick））表示当该图每第 n 次被唤醒时，every（n，wakeup）（或 every（n，tick））瞬时逻辑为真。

需要指出的是 every 瞬时逻辑操作的相连事件每次激活时，基事件 E 的计数器都被复位为 0。因而这种操作通常是在状态动作内使用。如

on/

on every(10, CLK): status('on');

表示在相连状态激活后，基事件 CLK 每发生 10 次，显示一次状态信息。

瞬时逻辑操作实际上是产生一个隐含事件，这种方法并不在 Stateflow 模型中创建任何新的事件，可以理解为基事件（base events）积累多时而产生的一个隐含事件。比如，如果用户希望在状态 A 激活后的 10 个时钟周期时产生从状态 A 发出的状态迁移。用我们前面掌握的方法就是添加一个新的事件（如 ALARM），当状态 A 激活后的 10 个时钟周期时由 ALARM 事件触发从状态 A 出发的状态迁移。另一种比较简单的方法就是利用我们这里介绍的瞬时逻辑操作法。在系统时钟事件 CLK 的基础上建立触发从状态 A 发出的状态迁移的瞬时逻辑操作 CLK[after(10,CLK)]或 after（10,CLK）。

4、Stateflow 中的事件广播

在 Stateflow 的动作中广播一个事件是非常有用的同步并行状态的方法。事件的反复广播亦可产生周期的过程。

因为 Stateflow 是单线程工作的，因而当它接收到一个事件时，它必须中断当前的活动来处理被广播事件产生的动作。用户可以通过状态动作或迁移动作来广播事件。

在进行复杂的 Stateflow 过程分析时，一定要清楚下列几点正常 Stateflow 过程的基本公理：

①当一个状态是激活的，他的上层（或父）状态一定也是激活的；

②单一（Exclusive）分解的状态或图，不得有超过一个的激活子状态；

③并行（Parallel）分解的状态或图中，如果某状态是激活状态，那么其处于高优先级的并行状态（Stateflow 图中的位置决定优先级）必定也是激活的状态。

下面通过举例来说明 Stateflow 是如何广播事件的。

（1）通过迁移动作进行直接事件广播。

直接事件广播的格式是

send(事件名,状态名)

利用有效事件名直接进行事件广播的格式是

状态名.事件名

图 12.34 给出了一个直接事件广播的示例。在图 12.34 中，并行状态 A 的子状态 A1 到 A2 的状态迁移上，有一个迁移动作 send（E1，B），该迁移动作完成将事件 E1 发送到并行状态 B 中。send（E1，B）命令等同于使用有效事件名 B.E1 进行事件广播。

图 12.34 中，Stateflow 图开始处于休眠状态，并行子状态 A.A1 和 B.B1 是激活的。根据公理，并行上层状态 A 和 B 都是激活的。这时假设一个事件发生，唤醒了 Stateflow 图。如果条件[data1==1]是真。Stateflow 执行次事件的过程为：

Stateflow 图的根部检查是否此事件是否能产生有效的状态迁移，结果是没有；

这时状态 A 开始检查是否存在该状态内的状态迁移，因为条件[data1==1]是真的，存在状态 A.A1 到 A.A2 的状态迁移；

状态 A.A1 执行 exit 退出动作（exitA1（））,停止状态 A.A1；执行状态迁移动作 send(E1,B)，即将事件 E1 广播给状态 B；

以后开始执行广播事件 E1 引发的动作，因为 B 是激活的，接收到事件 E1 后，状态 B 检查到事件 E1 可以引发 B.B1 到 B.B2 的状态迁移；

状态 B.B1 执行 exit 退出动作 (exitB1())，停止状态 B.B1；

激活状态 B.B2，状态 B.B2 执行 entry 进入动作（entB2（）），至此广播的事件 E1 引发的过程结束，Stateflow 继续事件广播前的过程；

激活状态 A.A2，执行状态 A.A2 的 entry 进入动作(entA2())；

Stateflow 图再次休眠，等待下一个触发事件唤醒。

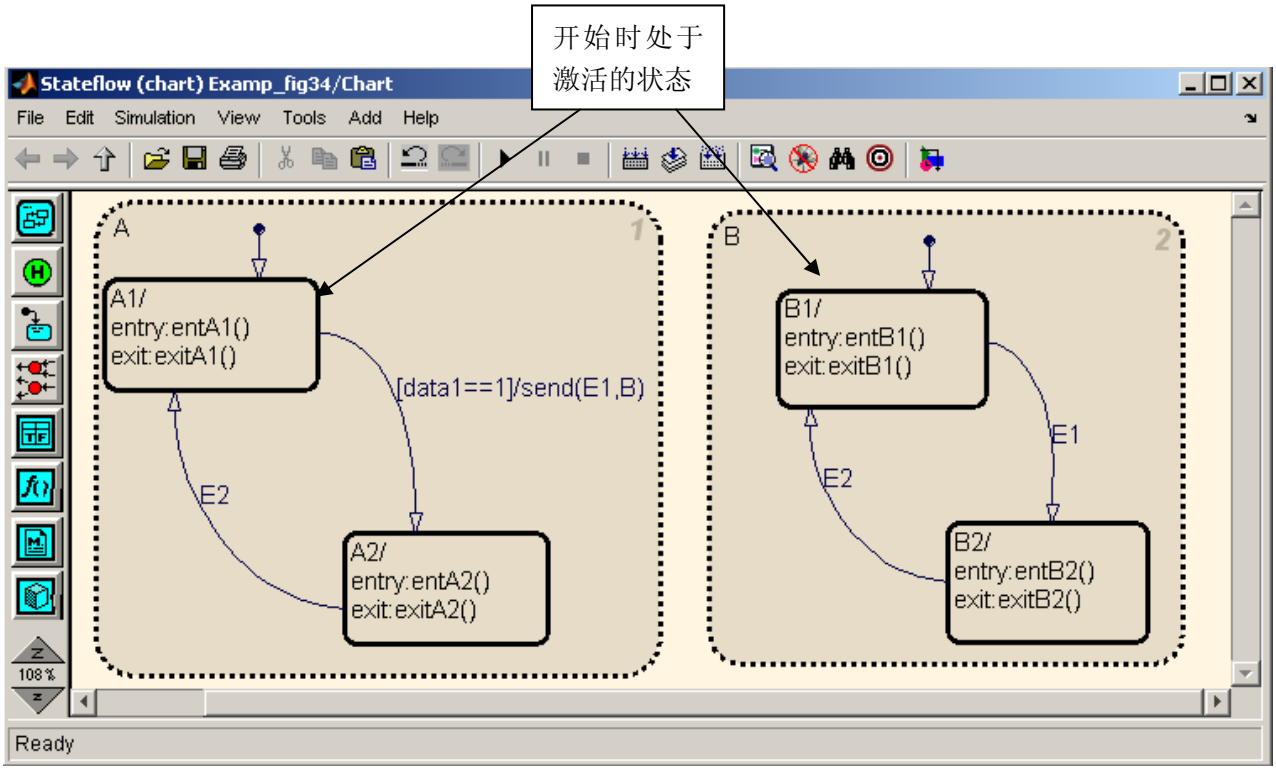


图 12.34　通过状态迁移动作进行事件的直接广播

（2）通过条件动作进行事件广播

图 12.35 给出一个通过条件动作进行事件广播的示例。开始时，图 12.35（a）所示 Stateflow 图处于休眠状态，并行子状态 A.A1 和 B.B1 处于激活状态。事件 E1 发生，唤醒 Stateflow 图。事件 E1 发生后，Stateflow 图的行动过程如下：

Stateflow 图的根部是两个并行状态 A 和 B，依据从上到下、从左到右的优先级原则，先判断状态 A。因为 A 的子状态 A.A1 是激活的，那么 A 一定也是激活的，这时执行状态 A 的 during 动作（durA（））；

状态 A 检查是否有由 E1 能触发的状态迁移，检查结果是存在这样有效的状态迁移，从状态 A.A1 到 A.A2，同时还存在条件动作；

条件动作是在条件一旦满足时就执行的，一般早于状态迁移的发生。此例先执行条件动作，广播事件 E2；此时状态 A.A1 还是处于激活状态；

由于广播的事件 E2，Stateflow 要先中断事件 E1 引发的过程，先执行事件 E2 引发的过程；

事件 E2 再次激活 Stateflow 图，先判断状态 A，状态 A 执行 during 动作（durA（））；

状态 A 检查不存在由事件 E2 引发的有效状态迁移；

判断状态 B，状态 B 执行 during 动作（durB（））；状态 B 检查到一个由 E2 触发的有效状态迁移，从状态 B.B1 到 B.B2，状态 B.B1 执行 exit 动作（exitB1（）），停止状态 B.B1，激活状态 B.B2，执行状态 B.B2 的 entry 动作（entryB2（））；

至此，由事件 E2 触发的 Stateflow 行为执行完毕，此时 Stateflow 图中状态 A.A1 和 B.B2 是处于激活的状态，见图 12.35（b）。下面继续事件 E1 触发的行为。

状态 A.A1 执行 exit 退出动作（exitA1（））；

停止状态 A.A1；

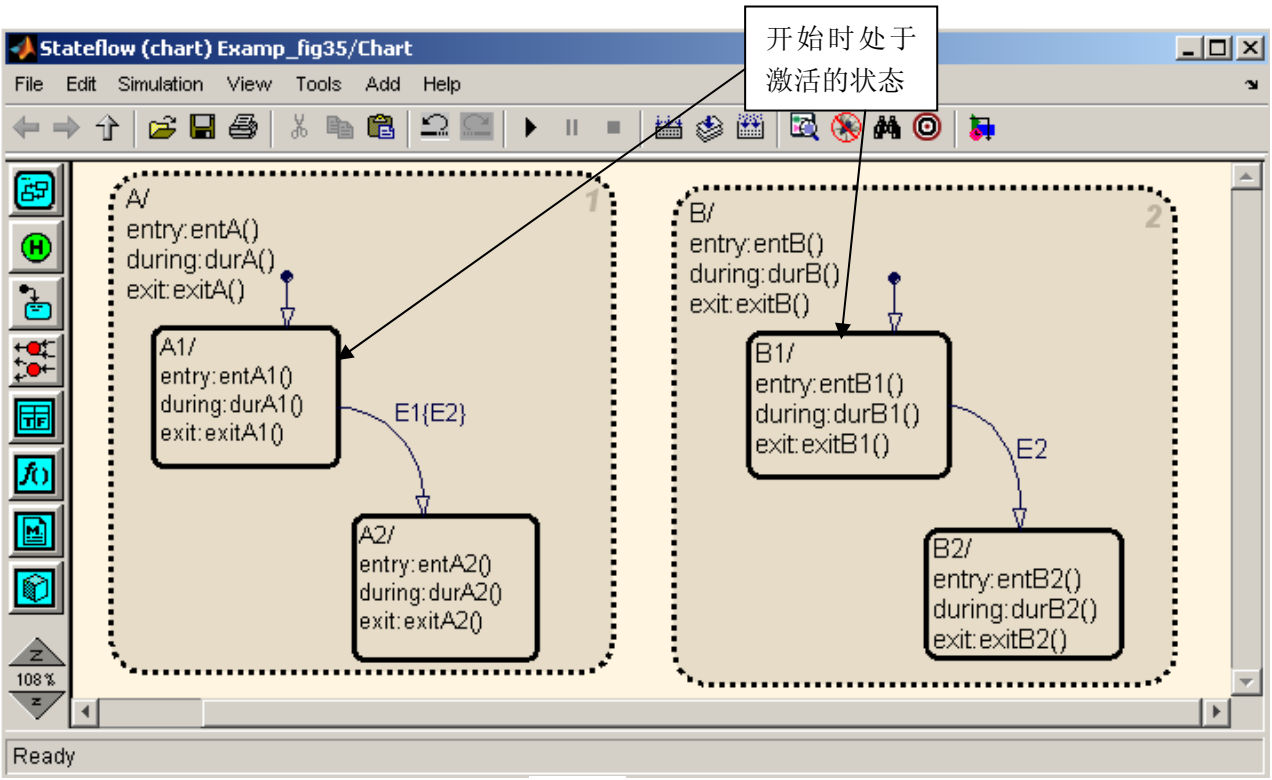
激活状态 A.A2；

执行 A.A2 的 entry 进入动作（entA2（））；

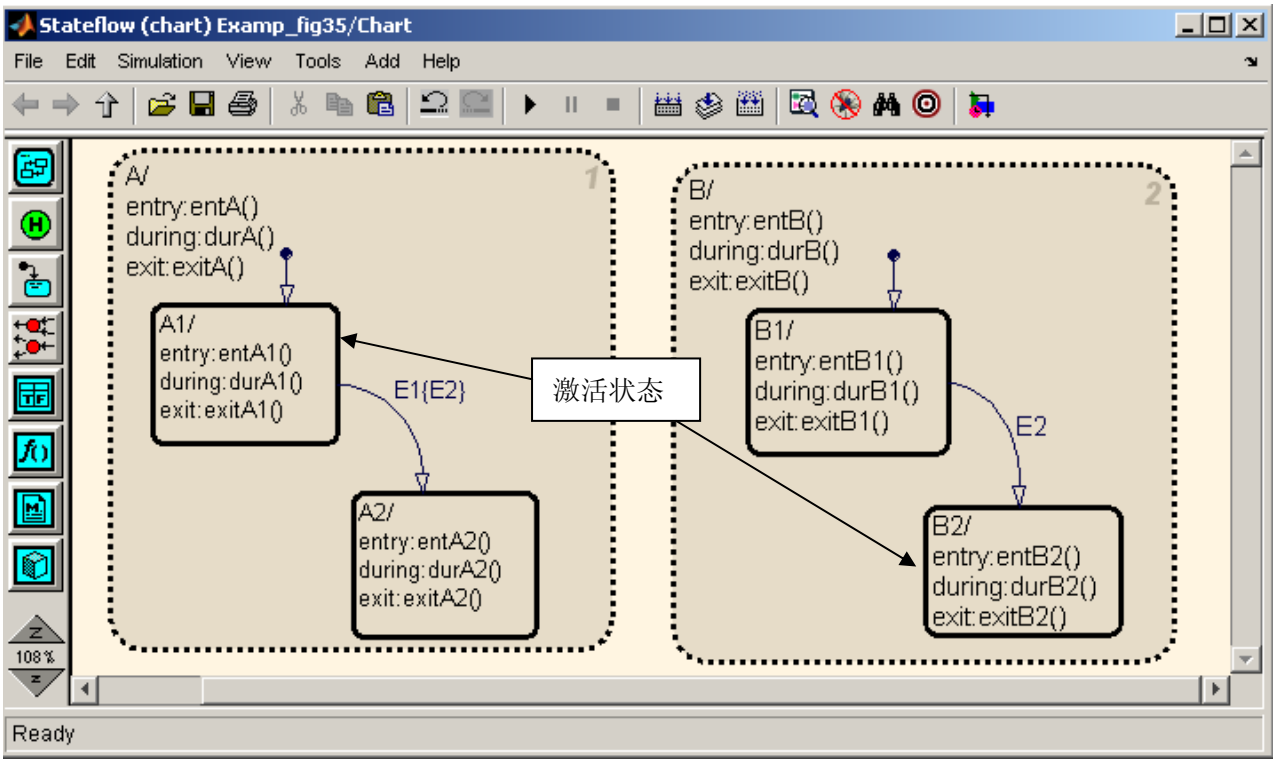
判断状态 B，状态 B 执行 during 动作（durB（））；

状态 B 判断其内部不存在由事件 E1 引发的状态迁移，状态 B.B2 执行 during 动作（durB2（））；Stateflow 图再次处于休眠状态，等待下一个事件来唤醒。

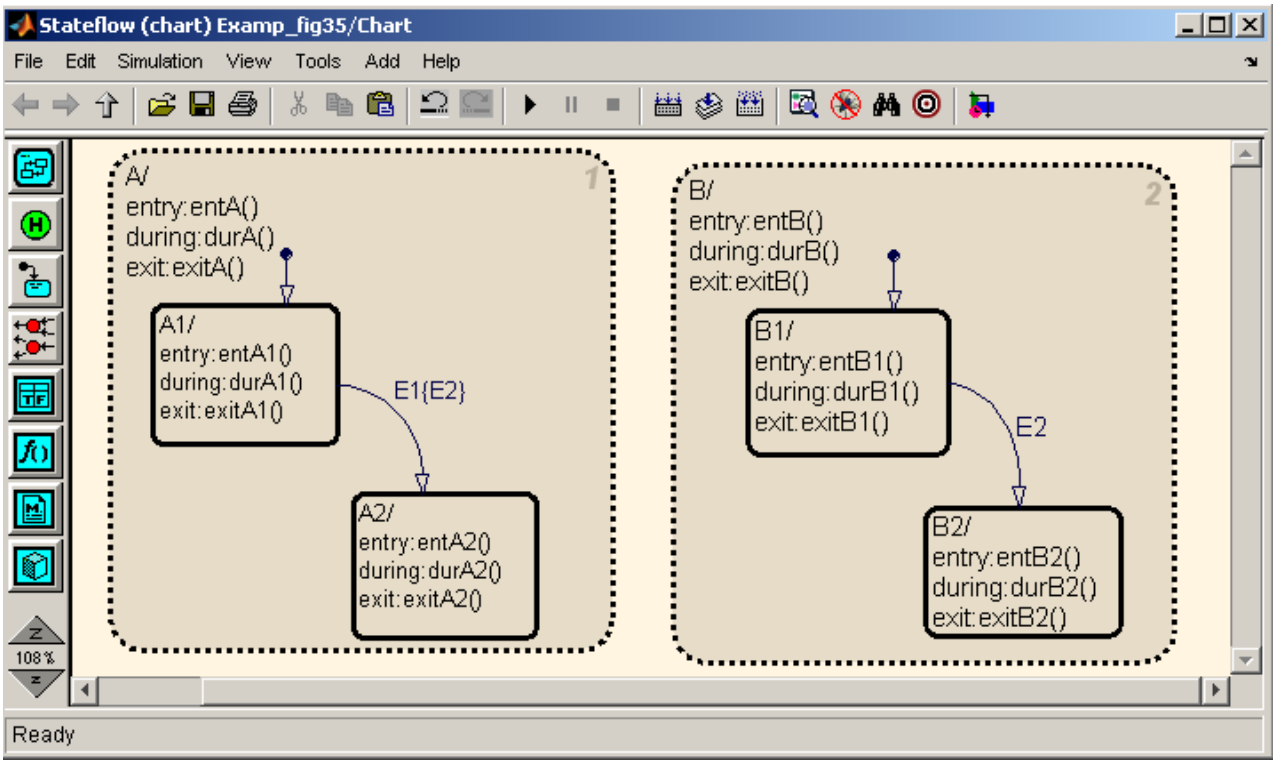
本例经过事件 E1 触发，通过条件动作广播事件 E2，Stateflow 图中最后激活的状态是 A.A2 和 B.B2，见图 12.35（c）。



（a）



（b）



（c）

图 12.35 通过条件动作进行事件广播

（3）通过状态动作进行事件广播

图 12.36 给出了一个通过状态动作进行事件广播的示例。图 12.36 (a)所示的 Stateflow 图开始时处于休眠状态，并行子状态 A.A1 和 B.B1 处于激活状态。事件 E1 发生，唤醒 Stateflow 图。事件 E1 发生后，Stateflow 图的行动过程如下：

Stateflow 图的根部是两个并行状态 A 和 B，依据从上到下、从左到右的优先级原则，先判断状态 A。因为 A 的子状态 A.A1 是激活的，那么 A 一定也是激活的，这时执行状态 A 的 during 动作（durA（））；

执行状态 A 的 on E1 动作，广播事件 E2；（during 和 on 事件动作的执行顺序由他们在状态中标记中的位置决定，先写的先执行）

由于广播了事件 E2，Stateflow 要先中断事件 E1 引发的过程，先执行事件 E2 引发的过程；

事件 E2 再次唤醒 Stateflow 图，先判断状态 A，状态 A 执行 during 动作（durA（））；

状态 A 检查不存在由事件 E2 引发的有效状态迁移；

判断状态 B，状态 B 执行 during 动作（durB（））；

状态 B 检查到一个由 E2 触发的有效状态迁移，从状态 B.B1 到 B.B2，状态 B.B1 执行 exit 动作（exitB1（））；

停止状态 B.B1；

激活状态 B.B2；

执行状态 B.B2 的 entry 进入动作（entryB2（））；

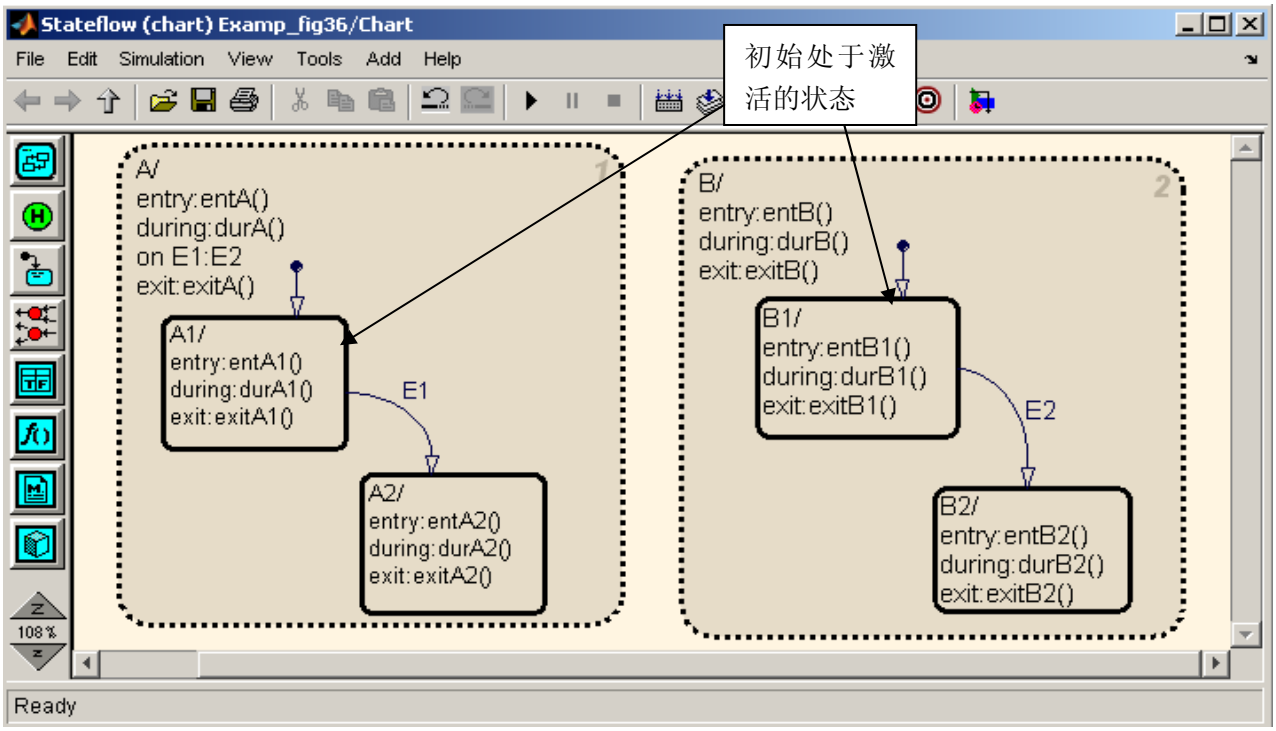
至此，由事件 E2 触发的行为执行完毕，此时 Stateflow 图中状态 A.A1 和 B.B2 是处于激活的状态，见图 12.36(b)。下面继续事件 E1 触发的行为。

状态 A 检查是否有由 E1 能触发的状态迁移，检查结果是存在这样有效的状态迁移，从状态 A.A1 到 A.A2，状态 A.A1 执行 exit 动作（exitA1（））；

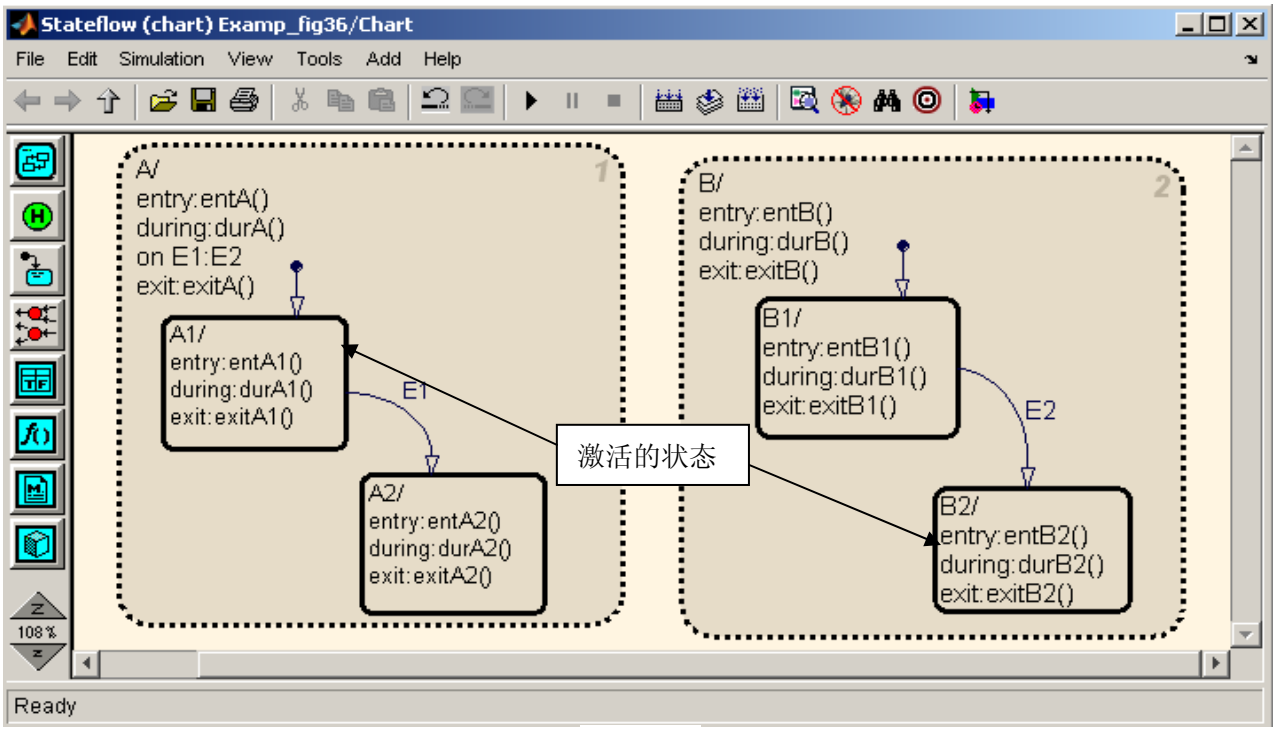
停止状态 A.A1；

激活状态 A.A2；
执行 A.A2 的 entry 进入动作（entA2（））；
判断状态 B，状态 B 执行 during 动作（durB（））；
状态 B 判断其内部不存在由事件 E1 引发的状态迁移，状态 B.B2 执行 during 动作（durB2（））；
Stateflow 图再次处于休眠状态，等待下一个事件来唤醒。

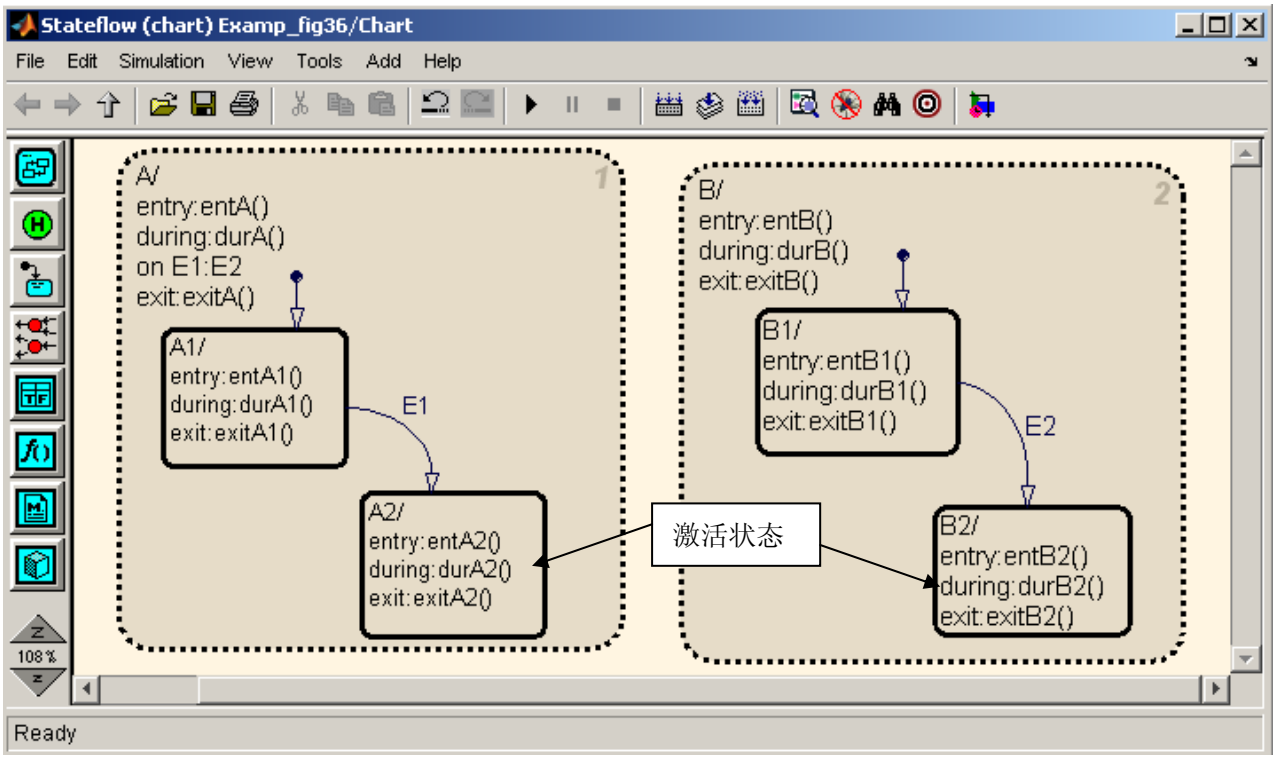
本例由事件 E1 的触发，通过状态动作广播事件 E2，Stateflow 图中最后激活的状态是 A.A2 和 B.B2，见图 12.36(c)。



（a）



（b）



（c）

图 12.36　通过状态动作进行事件广播

习　题

12.1 现有一间车间，为了改善工人的工作环境，安装了 3 台排气扇。要求

（1） 3 台排气扇同时通电；

（2） 一般情况下，只需一台排气扇工作，但当室内的温度超过 28℃时，启动第二台排气扇，两台同时工作。

（3） 当车间从事特种加工（作为一种事件输入）时，打开第三台排气扇，由 3 台排气扇同时工作。因为这样两台排气扇必须独立控制。

试建立 Stateflow 模型模拟该车间 3 台排气扇的工作情况。

12.2 在 MATLAB 命令窗口键入 fuelsys，打开一个对燃油系统进行容错控制的系统。系统中模拟了各种传感器的故障模式下，容错系统如何进行工作的，并可计算相应故障模式下系统输出的结果。感兴趣的读者可以仔细分析该系统。