

Stateflow®

快速入门指南



MATLAB® & SIMULINK®

R2020b



如何联系 MathWorks



最新动态: www.mathworks.com
销售和服务: www.mathworks.com/sales_and_services
用户社区: www.mathworks.com/matlabcentral
技术支持: www.mathworks.com/support/contact_us



电话: 010-59827000



迈斯沃克软件 (北京) 有限公司
北京市朝阳区望京东园四区 6 号楼
北望金辉大厦 16 层 1604

Stateflow® 快速入门指南

© COPYRIGHT 2004–2020 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

商标

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

专利

MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

修订历史记录

2004 年 6 月	第一次印刷	6.0 版 (版本 14) 中的新增内容
2004 年 10 月	仅限在线版本	6.1 版 (版本 14 SP1) 中的修订内容
2005 年 3 月	仅限在线版本	6.2 版 (版本 14 SP2) 中的修订内容
2005 年 9 月	仅限在线版本	6.3 版 (版本 14 SP3) 中的修订内容
2005 年 10 月	重印	版本 6.0
2006 年 3 月	第二次印刷	6.4 版 (版本 2006a) 中的修订内容
2006 年 9 月	重印	6.5 版 (版本 2006b)
2007 年 3 月	仅限在线版本	6.6 版 (版本 2007a) 中的再发布内容
2007 年 9 月	第三次印刷	7.0 版 (版本 2007b) 中的再发布内容
2008 年 3 月	第四次印刷	7.1 版 (版本 2008a) 中的修订内容
2008 年 10 月	第五次印刷	7.2 版 (版本 2008b) 中的修订内容
2009 年 3 月	第六次印刷	7.3 版 (版本 2009a) 中的修订内容
2009 年 9 月	仅限在线版本	7.4 版 (版本 2009b) 中的修订内容
2010 年 3 月	仅限在线版本	7.5 版 (版本 2010a) 中的修订内容
2010 年 9 月	仅限在线版本	7.6 版 (版本 2010b) 中的修订内容
2011 年 4 月	第七次印刷	7.7 版 (版本 2011a) 中的修订内容
2011 年 9 月	仅限在线版本	7.8 版 (版本 2011b) 中的修订内容
2012 年 3 月	仅限在线版本	7.9 版 (版本 2012a) 中的修订内容
2012 年 9 月	仅限在线版本	8.0 版 (版本 2012b) 中的修订内容
2013 年 3 月	仅限在线版本	8.1 版 (版本 2013a) 中的修订内容
2013 年 9 月	仅限在线版本	8.2 版 (版本 2013b) 中的修订内容
2014 年 3 月	仅限在线版本	8.3 版 (版本 2014a) 中的修订内容
2014 年 10 月	仅限在线版本	8.4 版 (版本 2014b) 中的修订内容
2015 年 3 月	仅限在线版本	8.5 版 (版本 2015a) 中的修订内容
2015 年 9 月	仅限在线版本	8.6 版 (版本 2015b) 中的修订内容
2015 年 10 月	仅限在线版本	8.5.1 版 (版本 2015a SP1) 中的再发布内容
2016 年 3 月	仅限在线版本	8.7 版 (版本 2016a) 中的修订内容
2016 年 9 月	仅限在线版本	8.8 版 (版本 2016b) 中的修订内容
2017 年 3 月	仅限在线版本	8.9 版 (版本 2017a) 中的修订内容
2017 年 9 月	仅限在线版本	9.0 版 (版本 2017b) 中的修订内容
2018 年 3 月	仅限在线版本	9.1 版 (版本 2018a) 中的修订内容
2018 年 9 月	仅限在线版本	9.2 版 (版本 2018b) 中的修订内容
2019 年 3 月	仅限在线版本	10.0 版 (版本 2019a) 中的修订内容
2019 年 9 月	仅限在线版本	10.1 版 (版本 2019b) 中的修订内容
2020 年 3 月	仅限在线版本	10.2 版 (版本 2020a) 中的修订内容
2020 年 9 月	仅限在线版本	10.3 版 (版本 2020b) 中的修订内容

1	Stateflow 产品简介	
	Stateflow 产品说明	1-2
2	Stateflow 基础知识	
	对有限状态机建模	2-2
	Stateflow 图的示例	2-2
	将图作为 MATLAB 对象执行	2-2
	将图作为具有本地事件的 Simulink 模块进行仿真	2-3
	将图作为带时序条件的 Simulink 模块进行仿真	2-6
	构造并运行 Stateflow 图	2-9
	构造 Stateflow 图	2-9
	将图作为 Simulink 模块进行仿真	2-12
	将图作为 MATLAB 对象执行	2-13
	使用动作定义图行为	2-17
	状态和转移动作示例	2-17
	状态动作类型	2-17
	转移动作的类型	2-18
	检查图行为	2-19
	创建层次结构来管理复杂系统	2-22
	状态层次结构	2-22
	层次结构的示例	2-22
	通过使用状态层次结构实现行为	2-23
	使用子图简化图的外观	2-24
	探索示例	2-25
	使用并行机制对同步子系统建模	2-27
	状态分解	2-27
	并行分解的示例	2-27
	组合使用互斥 (OR) 状态和并行 (AND) 状态	2-28
	并行状态的执行顺序	2-29
	探索示例	2-29
	通过广播事件同步并行状态	2-31
	广播本地事件	2-31
	事件广播的示例	2-31
	将子系统建模为并行状态	2-32
	与其他 Simulink 模块协调	2-33

探索示例	2-35
使用激活状态数据监视图的活动	2-38
激活状态数据	2-38
激活状态数据的示例	2-39
交通控制器子图的行为	2-39
探索示例	2-41
使用时序逻辑调度图动作	2-45
时序逻辑运算符	2-45
时序逻辑的示例	2-45
Bang-Bang 循环的时序	2-46
状态 LED 的时序	2-47
探索示例	2-49

其他学习工具

3

Stateflow 产品简介

Stateflow 产品说明

使用状态机与流程图对决策逻辑进行建模和仿真

Stateflow 提供了一种图形语言，包括状态转移图、流程图、状态转移表和真值表。您可以使用 Stateflow 来说明 MATLAB® 算法和 Simulink® 模型如何响应输入信号、事件和基于时间的条件。

Stateflow 使您能够设计和开发监控、任务调度、故障管理、通信协议、用户界面和混合系统。

使用 Stateflow，您可以对组合和时序决策逻辑进行建模，使其可作为 Simulink 模型中的模块进行仿真或作为 MATLAB 中的对象执行。图形动画使您能够在执行逻辑时对其进行分析和调试。编辑时和运行时检查可确保在实现前具有设计一致性和完整性。

Stateflow 基础知识

本章介绍 Stateflow 中基于事件的建模的基本概念。

- “对有限状态机建模”（第 2-2 页）
- “构造并运行 Stateflow 图”（第 2-9 页）
- “使用动作定义图行为”（第 2-17 页）
- “创建层次结构来管理复杂系统”（第 2-22 页）
- “使用并行机制对同步子系统建模”（第 2-27 页）
- “通过广播事件同步并行状态”（第 2-31 页）
- “使用激活状态数据监视图的活动”（第 2-38 页）
- “使用时序逻辑调度图动作”（第 2-45 页）

对有限状态机建模

Stateflow 是基于有限状态机的图形化编程环境。使用 Stateflow，您可以测试和调试您的设计，考虑不同仿真场景，并从状态机生成代码。

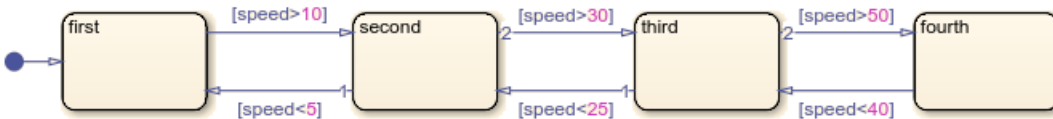
有限状态机是从一种工作模式（状态）转移到另一种工作模式的动态系统的表示。状态机可以：

- 作为复杂软件设计过程的高级起点。
- 让您能够专注于工作模式和从一种模式转至下一种模式所需的条件。
- 帮助您设计即使模型复杂度增加也能保持简洁清晰的模型。

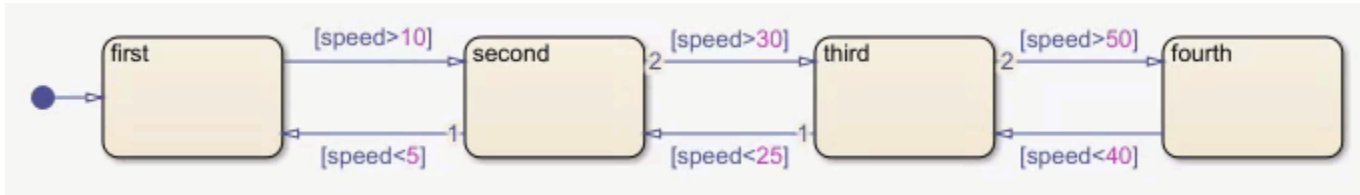
控制系统设计在很大程度上依赖于状态机来管理复杂的逻辑。应用包括设计飞机、汽车和机器人控制系统。

Stateflow 图的示例

Stateflow 图将状态、转移和数据组合起来实现有限状态机。以下 Stateflow 图提供了汽车四速自动传动系统换挡逻辑的简化模型。该图通过状态表示每个挡位，分别显示为一个带有 **first**、**second**、**third** 或 **fourth** 标签的矩形。像它们代表的挡位一样，这些状态是互斥的，因此在某个时刻只有一个状态被激活。



图左侧的箭头表示默认转移，并指示第一个状态被激活。当您执行该图时，此状态在画布上突出显示。其他箭头表示各状态之间可能的转移。要定义状态机的动态，请将每个转移与一个布尔条件或触发事件相关联。例如，下图监控车速，并在速度超过固定阈值时切换到另一个挡位。在仿真过程中，图中的突出显示随着不同状态被激活而变化。

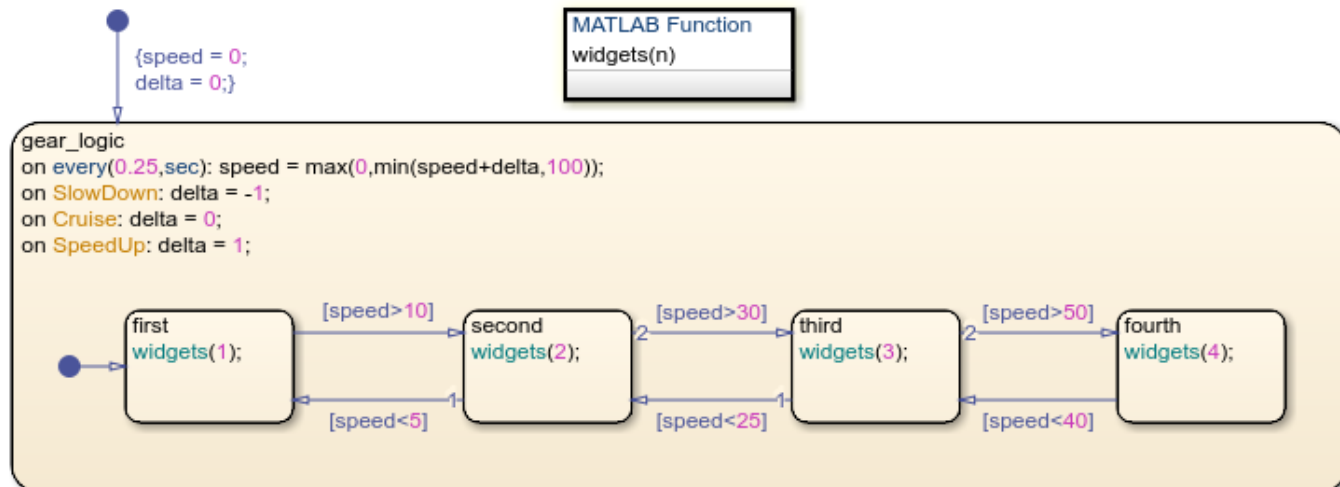


此图仅提供了简单设计，未考虑发动机转速和扭矩等重要因素。您可以通过将此 Stateflow 图与 MATLAB 或 Simulink 中的其他组件链接起来，构建一个更全面和更真实的模型。可以采用以下三种方法。

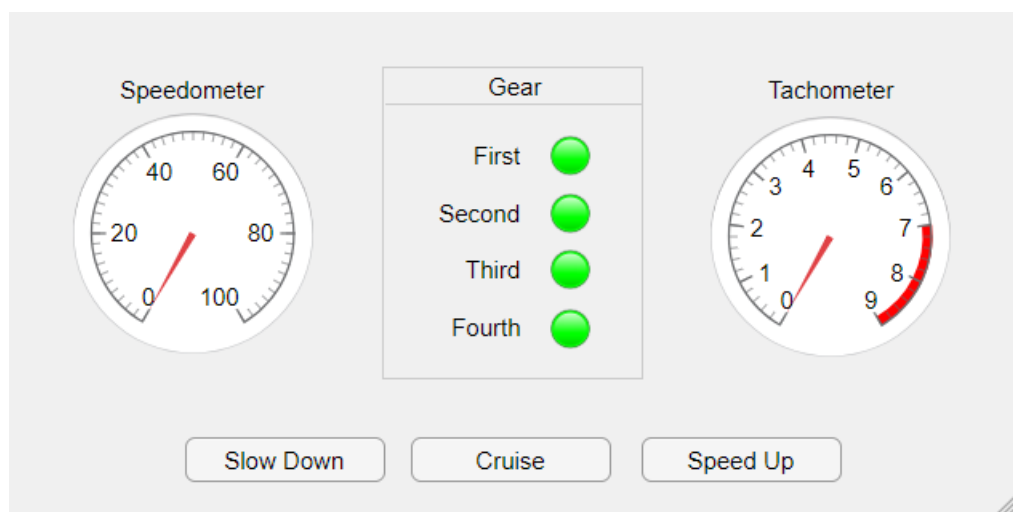
将图作为 MATLAB 对象执行

此示例展示了自动传输系统的一个修改版本，其中包含了状态层次结构、时序逻辑和输入事件。

- **层次结构：**该图包含一个父状态 **gear_logic**，其中包含了上例中的四速自动变速器图。此父状态控制车辆的速度和加速度。在执行期间，**gear_logic** 始终处于激活状态。
- **时序逻辑：**在 **gear_logic** 状态中，动作 **on every(0.25,sec)** 决定车速。运算符 **every** 创建一个 MATLAB 计时器，它执行图并每隔 0.25 秒更新一次图数据 **speed**。
- **输入事件：**输入事件 **SpeedUp**、**Cruise** 和 **SlowDown** 重置图数据 **delta** 的值。此数据决定汽车在每个执行步中是加速还是保持其速度。



您可以通过命令行窗口或使用脚本直接在 MATLAB 中将该图作为对象来执行。您还可以编写一个 MATLAB App 以通过图形用户界面控制图状态。例如，当您点击某按钮时，该用户界面向图发送输入事件。在图中，MATLAB 函数 `widgets` 控制界面上仪表和信号灯的值。



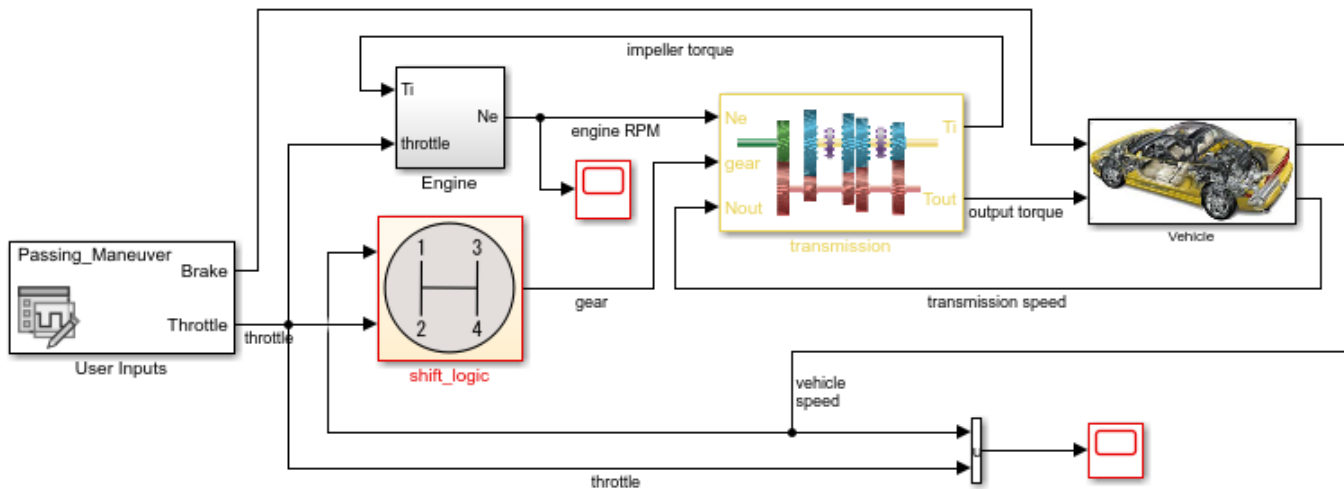
要启动示例，请在 App 设计工具的工具条中，点击 **Run**。示例将继续运行，直到您关闭用户界面窗口。

或者，在 Stateflow Editor 的 **State Chart** 选项卡中，点击 **Run**。要控制车速，请使用 Symbols 窗格中的 **SpeedUp**、**SlowDown** 和 **Cruise** 按钮。要停止示例，请点击 **Stop**。

有关将 Stateflow 图作为 MATLAB 对象执行的详细信息，请参阅“在 MATLAB 中执行”。

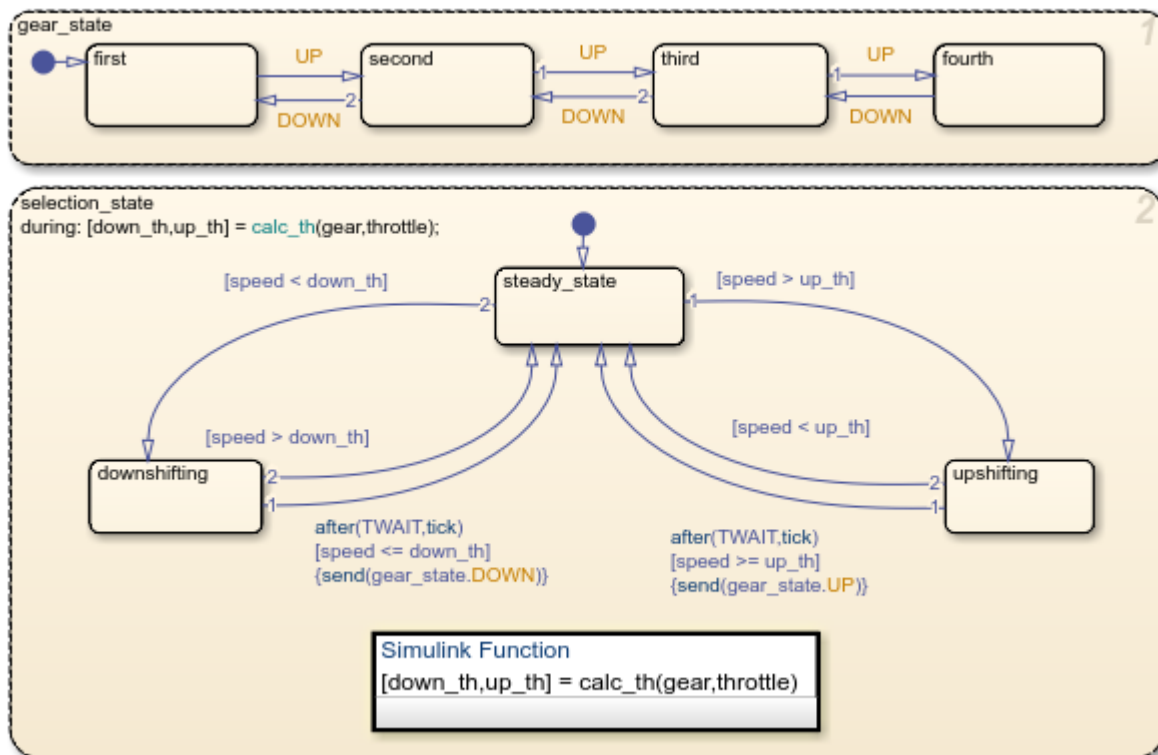
将图作为具有本地事件的 Simulink 模块进行仿真

此示例提供了一种更复杂的自动变速器系统设计。Stateflow 图在 Simulink 模型中显示为模块。模型中的其他模块代表相关汽车组件。该图通过输入和输出连接来共享数据，进而与其他模块进行对接。要打开该图，请点击 `shift_logic` 模块左下角的箭头。



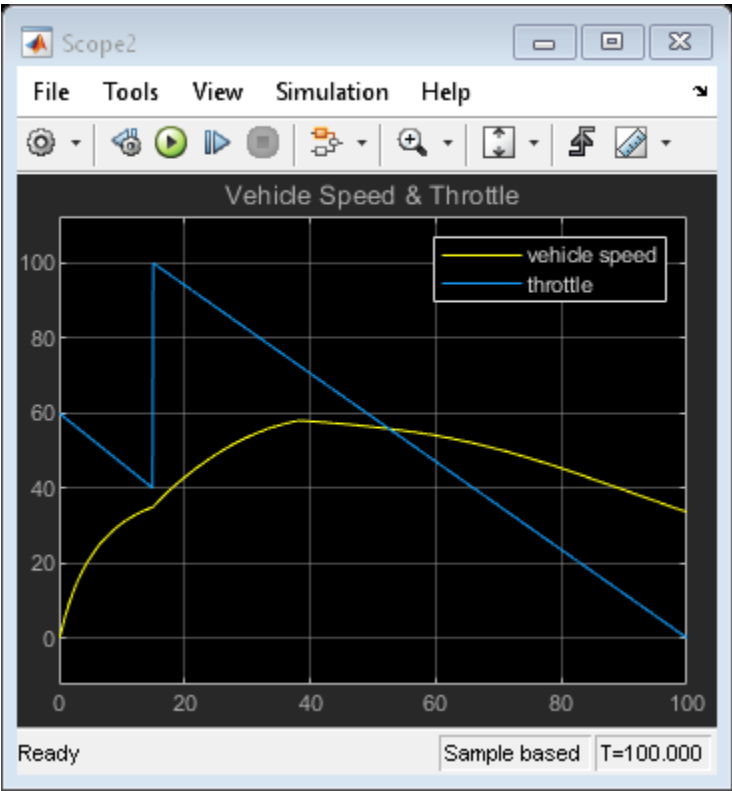
该图合并了状态层次结构、并行机制、激活状态数据、本地事件和时序逻辑。

- **层次结构**：状态 `gear_state` 包含四速自动变速器图的一个修改版本。状态 `selection_state` 包含代表稳定状态、升挡和降挡工作模式的子状态。当需要升挡或降挡时，这些子状态将被激活。
- **并行机制**：并行状态 `gear_state` 和 `selection_state` 显示为带有虚线边框的矩形。这些状态同时工作，即使其内部的子状态存在互斥也是如此。
- **激活状态数据**：输出值 `gear` 反映仿真过程中挡位的选择。图会根据 `gear_state` 中的激活子状态生成此值。
- **本地事件**：此图不使用布尔条件，而是使用本地事件 `UP` 和 `DOWN` 触发挡位之间的转移。这些事件由 `selection_state` 中的 `send` 命令触发，当车速超出所选挡位的工作范围时会发出这些命令。Simulink 函数 `calc_th` 根据选择的挡位和发动机转速确定工作范围的边界值。
- **时序逻辑**：为了防止连续快速换挡，`selection_state` 使用时序逻辑运算符 `after` 来延迟 `UP` 和 `DOWN` 事件的广播。仅当所需的换挡时间超过某个预定时间 `TWAIT` 时，状态才会广播这些事件之一。



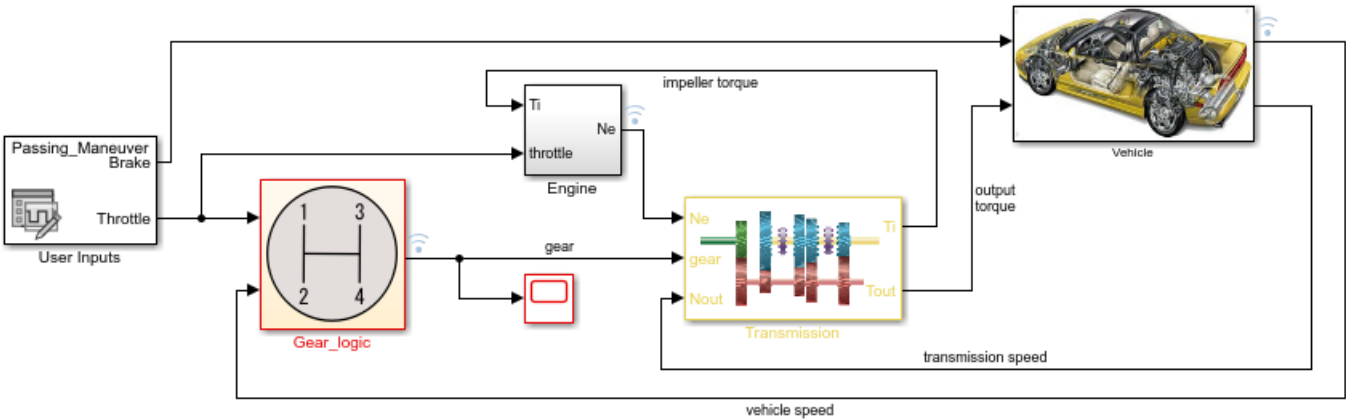
要运行模型仿真，请执行以下操作：

- 1 双击 **User Inputs** 模块。在 Signal Editor 对话框中，从 **Active Scenario** 列表选择一个预定义的刹车到油门的配置文件。默认配置文件是 **Passing Maneuver**。
- 2 点击 **Run**。在 Stateflow Editor 中，图动画会突出显示仿真过程中的激活状态。要减慢动画速度，请在 **Debug** 选项卡中，从 **Animation Speed** 下拉列表中选择 **Slow**。
- 3 在 Scope 模块中，检查仿真结果。每个示波器都会在仿真过程中显示其输入信号的图形。



将图作为带时序条件的 Simulink 模块进行仿真

此示例提供了汽车传动系统建模的另一种方法。Stateflow 图在 Simulink 模型中显示为模块。模型中的其他模块代表相关汽车组件。该图通过输入和输出连接来共享数据，进而与其他模块进行对接。要打开该图，请点击 Gear_logic 模块左下角的箭头。

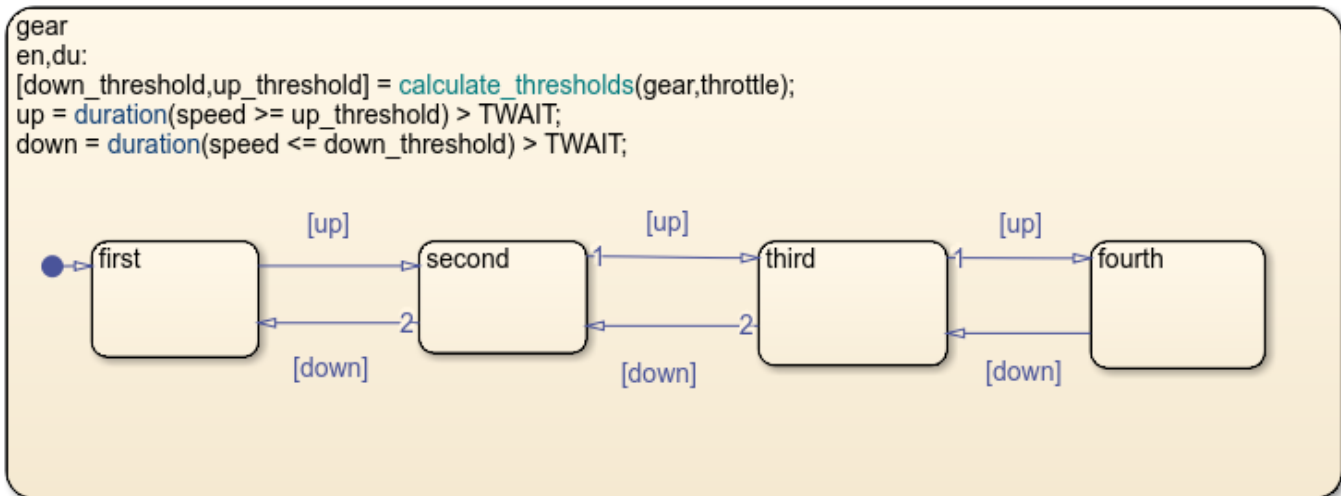


该图合并了状态层次结构、激活状态数据和时序逻辑。

- **层次结构**：此模型将四速自动变速器图置于父状态 **gear** 中。该父状态监控车辆速度和发动机转速，并触发换挡。状态 **gear** 左上角列出的动作确定了所选挡位的运行阈值以及布尔条件 **up** 和 **down** 的值。

标签 **en,du** 指示在状态第一次被激活 (**en** = **entry**) 和在状态已激活时的每个后续时间步 (**du** = **during**) 执行状态动作。

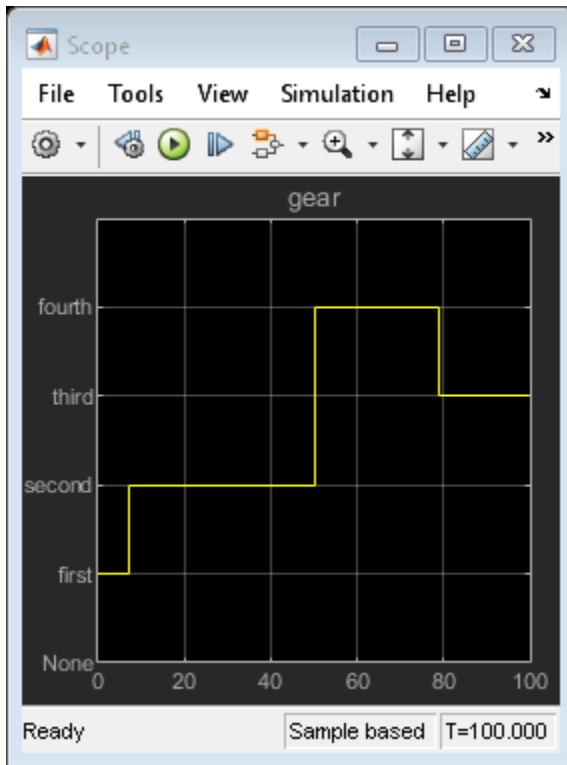
- **激活状态数据**：输出值 **gear** 反映仿真过程中挡位的选择。图会根据 **gear** 中的激活子状态生成此值。
- **时序逻辑**：为了防止连续快速换挡，布尔条件 **up** 和 **down** 使用时序逻辑运算符 **duration** 来控制挡位之间的转移。当车速保持在所选挡位工作范围之外超过某个预定时间 **TWAIT**（以秒为单位测量）时，条件有效。



Simulink Function
`[down_th,up_th] = calculate_thresholds(gear,throttle)`

要运行模型仿真，请执行以下操作：

- 1 双击 **User Inputs** 模块。在 Signal Editor 对话框中，从 **Active Scenario** 列表选择一个预定义的刹车到油门的配置文件。默认配置文件是 **Passing Maneuver**。
- 2 点击 **Run**。在 Stateflow Editor 中，图动画会突出显示仿真过程中的激活状态。要减慢动画速度，请在 **Debug** 选项卡中，从 **Animation Speed** 下拉列表中选择 **Slow**。
- 3 在 Scope 模块中，检查仿真结果。示波器显示仿真期间的挡位选择变化图。



另请参阅

after | duration | every | send

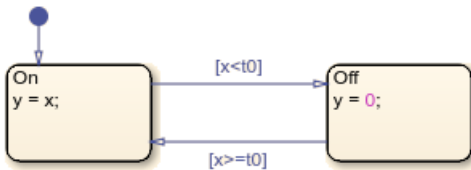
详细信息

- “构造并运行 Stateflow 图” (第 2-9 页)
- “使用动作定义图行为” (第 2-17 页)
- “创建层次结构来管理复杂系统” (第 2-22 页)
- “使用并行机制对同步子系统建模” (第 2-27 页)
- “通过广播事件同步并行状态” (第 2-31 页)
- “使用激活状态数据监视图的活动” (第 2-38 页)
- “使用时序逻辑调度图动作” (第 2-45 页)

构造并运行 Stateflow 图

Stateflow 图是有限状态机的图形表示，由状态、转移和数据组成。您可以创建 Stateflow 图来定义 MATLAB 算法或 Simulink 模型如何响应外部输入信号、事件和基于时间的条件。有关详细信息，请参阅“对有限状态机建模”（第 2-2 页）。

例如，下面的 Stateflow 图展示半波整流器的基础逻辑。该图包含两个标签为 **On** 和 **Off** 的状态。在 **On** 状态下，图输出信号 y 等于输入 x 。在 **Off** 状态下，输出信号设置为零。当输入信号跨越某个阈值 $t0$ 时，图在这些状态之间转移。各个状态下的动作在仿真的每一时间步都会更新 y 的值。



此示例说明如何创建这样的 Stateflow 图，以在 Simulink 中进行仿真和在 MATLAB 中执行。

构造 Stateflow 图

打开 Stateflow Editor

Stateflow Editor 是一个图形环境，用于设计状态转移图、流程图、状态转移表和真值表。在打开 Stateflow Editor 之前，需要先确定最能满足您需求的图执行模式。

- 要建立周期性或连续时间 Simulink 算法的条件、基于事件和基于时间的逻辑模型，请创建一个可在 Simulink 模型中作为模块进行仿真的 Stateflow 图。在 MATLAB 命令提示符处，输入：

```
sfnew rectify % create chart for simulation in a Simulink model
```

Simulink 创建一个名为 `rectify` 的模型，其中包含一个空的 Stateflow Chart 模块。要打开 Stateflow Editor，请双击图模块。

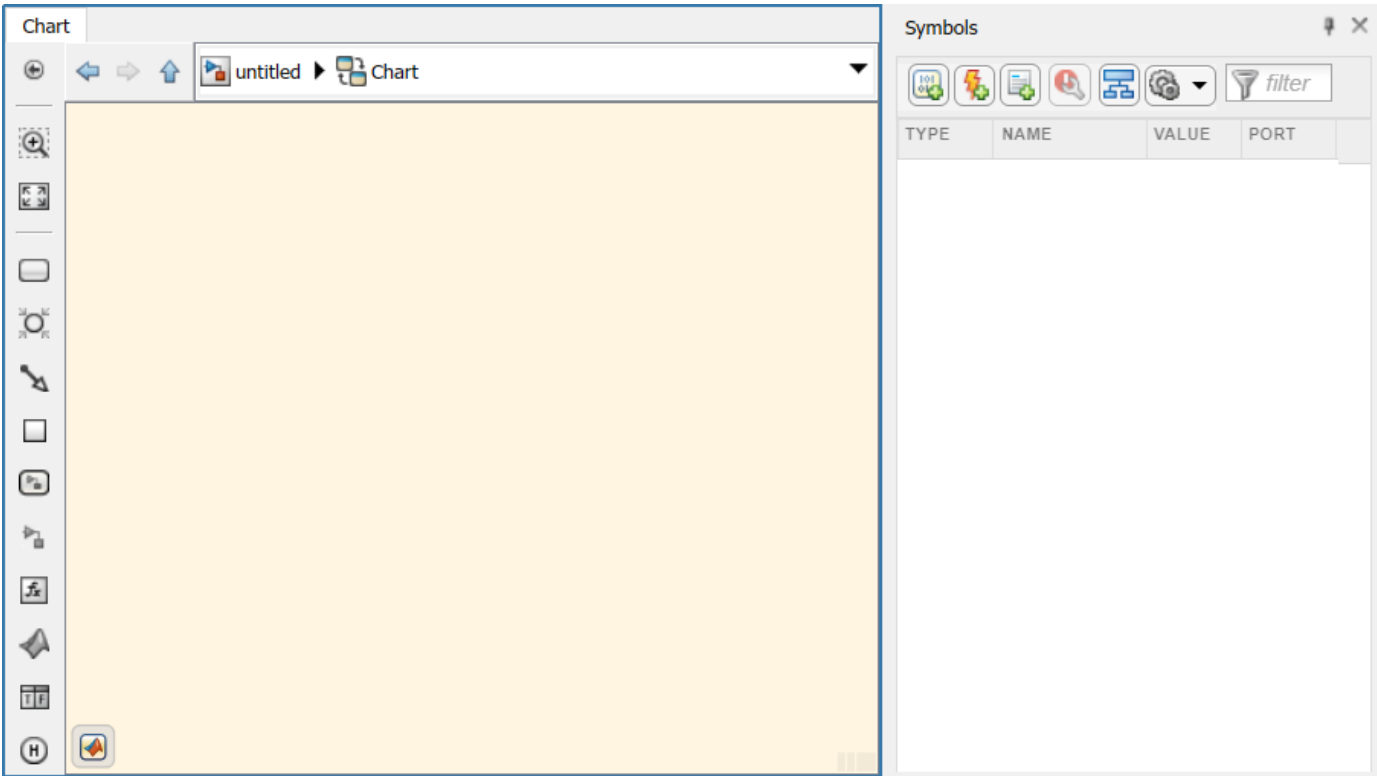
- 要为 MATLAB 应用程序设计可重用的状态机和时序逻辑，请创建可作为 MATLAB 对象执行的独立 Stateflow 图。在 MATLAB 命令提示符处，输入：

```
edit rectify.sfx % create chart for execution as a MATLAB object
```

如果文件 `rectify.sfx` 不存在，Stateflow Editor 将创建名为 `rectify` 的空图。


Stateflow Editor 的主要组件是对象选项板、图画布和 Symbols 窗格。

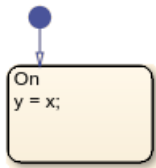
- 图画布是一个绘图区域，您可以在其中通过组合状态、转移和其他图形元素来创建图。
- 在画布的左侧有一个对象选项板，其中显示了一组可向图中添加图形元素的工具。
- 在画布的右侧有一个 Symbols 窗格，您可以用它向图添加新数据并解析任何未定义或未使用的符号。



提示 在构造 Stateflow 图后，您可以将其内容复制到另一个具有不同执行模式的图中。例如，您可以构造在 MATLAB 中执行的图，并将其内容复制到在 Simulink 中进行仿真的图中。

添加状态和转移

- 1
- 在对象选项板中，点击 **State** 图标  并将指针移至图画布。将出现具有默认转移的状态。要放置该状态，请点击画布上的某个位置。在文本提示中，输入状态名称 **On** 和状态动作 $y = x$ 。

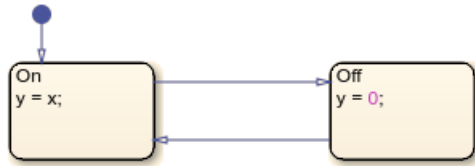


- 2
- 添加另一个状态。右键点击并拖动 **On** 状态。蓝色图形提示可以帮助您水平或垂直对齐状态。新状态的名称变为 **Off**。双击该状态并将状态动作修改为 $y = 0$ 。

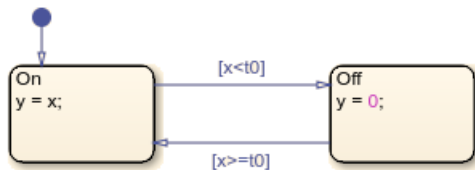


- 3 重新对齐两个状态并在两个状态之间的空白处停留片刻。蓝色转移提示指示您可以连接状态的几种方式。要添加转移，请点击适当的提示。


或者，要绘制转移，请点击并从一个状态的边拖动到另一个状态的边。




- 4 双击每个转移并输入适当的转移条件 $x < t0$ 或 $x \geq t0$ 。条件出现在方括号内。



- 5 清理图：

- 为使图更加清晰，将每个转移标签移到其对应转移上方或下方的方便位置。
- 要对齐图的图形元素并调整其大小，请在 **Format** 选项卡中，点击 **Auto Arrange** 或按 **Ctrl + Shift + A**。
- 要调整图的大小以适合画布，请按空格键或点击 **Fit To View** 图标 .




解析未定义的符号


在执行图之前，必须定义图中使用的每个符号并指定其作用域（例如，输入数据、输出数据或本地数据）。在 Symbols 窗格中，未定义的符号用红色错误标记  进行标记。**Type** 列根据每个未定义符号在图中的使用情况显示其建议作用域。

- 1 打开 Symbols 窗格。

- 如果您构建的是 Simulink 模型中的图，请在 **Modeling** 选项卡中，在 **Design Data** 下，选择 **Symbols Pane**。
- 如果您构建的是要在 MATLAB 中执行的独立图，请在 **State Chart** 选项卡中，选择 **Add Data > Symbols Pane**。


- 2 在 Symbols 窗格中，点击 **Resolve Undefined Symbols** .

- 如果构建的是在 Simulink 模型中的图，Stateflow Editor 会将符号 x 和 $t0$ 解析为输入数据 ，将 y 解析为输出数据 .
- 如果您构建的是要在 MATLAB 中执行的独立图，Stateflow Editor 则将 $t0$ 、 x 和 y 解析为本地数据 .



TYPE	NAME	VALUE	PORT
!	t0		
!	x		
!	y		



TYPE	NAME	VALUE	PORT
101 010	t0		1
101 010	x		2
101 010	y		1

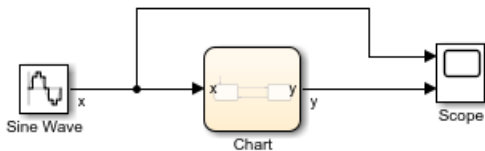
- 3 由于阈值 **t0** 在仿真过程中不会更改，因此将其作用域更改为常量数据。在 **Type** 列中，点击 **t0** 旁边的数据类型图标，然后选择  “Constant Data”。
- 4 设置阈值 **t0** 的值。在 **Value** 列中，点击 **t0** 旁边的空白输入框，并输入值 0。
- 5 保存您的 Stateflow 图。


您的图现在即可在 Simulink 中进行仿真，或者在 MATLAB 中执行。

将图作为 Simulink 模块进行仿真

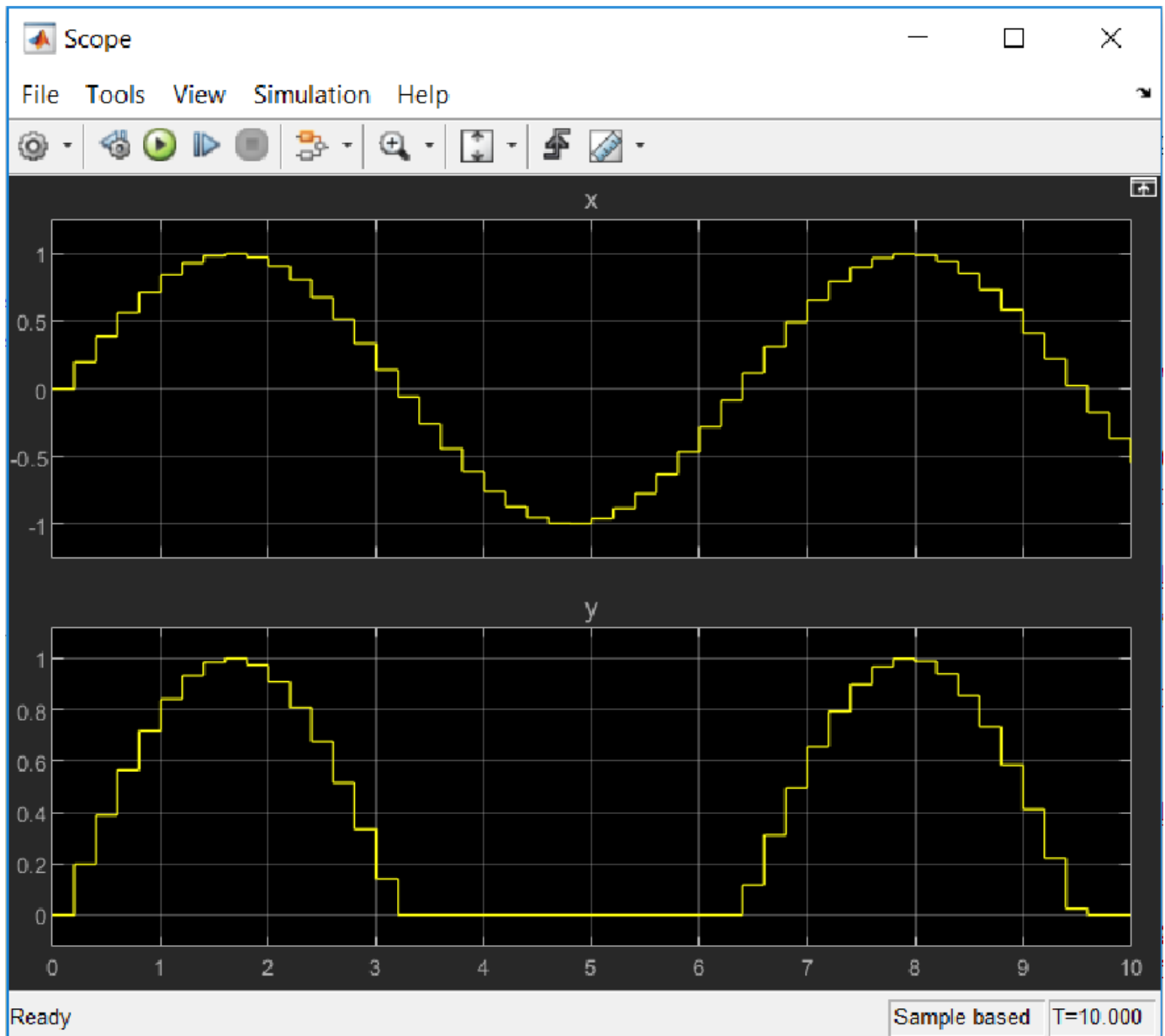
要在 Simulink 模型中对图进行仿真，请通过输入和输出端口将图模块连接到模型中的其他模块。如果您要从 MATLAB 命令行窗口中执行图，请参阅“将图作为 MATLAB 对象执行”（第 2-13 页）。

- 1 要返回到 Simulink Editor，请在画布顶部的浏览器栏中点击 Simulink 模型的名称：
 “rectify”。如果浏览器栏不可见，请点击对象选项板顶部的 **Hide/Show Explorer Bar** 图标 。
- 2 执行下列操作以将信源添加到模型中：
 - 从 Simulink Sources 库中，添加一个 Sine Wave 模块。
 - 双击 Sine Wave 模块并将 **Sample time** 设置为 0.2。
 - 将 Sine Wave 模块的输出连接到 Stateflow 图的输入。
 - 将信号标记为 **x**。
- 3 向模型中添加一个信宿：
 - 从 Simulink Sinks 库中，添加一个具有两个输入端口的 Scope 模块。
 - 将 Sine Wave 模块的输出连接到 Scope 模块的第一个输入。
 - 将 Stateflow 图的输出连接到 Scope 模块的第二个输入。
 - 将信号标记为 **y**。
- 4 保存 Simulink 模型。



- 5 要仿真模型，请点击 **Run** 。在仿真过程中，Stateflow Editor 通过图动画突出显示激活状态和转移。

6 对模型进行仿真后，双击 Scope 模块。示波器显示 Stateflow 图的输入信号和输出信号图。



仿真结果显示整流器滤除了负输入值。

将图作为 MATLAB 对象执行

要在 MATLAB 命令行窗口中执行图，请创建一个图对象，并调用其 `step` 函数。如果您要在 Simulink 模型中对图进行仿真，请参阅“将图作为 Simulink 模块进行仿真”（第 2-12 页）。

- 1 通过使用包含图定义作为函数的 `sfx` 文件的名称，创建一个图对象 `r`。将图数据 `x` 的初始值指定为名称-值对组。

```
r = rectify('x',0);
```

- 2 初始化图执行的输入和输出数据。向量 **X** 包含来自正弦波的输入值。向量 **Y** 是一个空的累加器。

```
T = 0:0.2:10;  
X = sin(T);  
Y = [];
```

- 3 通过多次调用 **step** 函数来执行图对象。将来自向量 **X** 的单个值作为图数据 **x** 传递。在向量 **Y** 中收集 **y** 的结果值。在执行过程中，Stateflow Editor 通过图动画突出显示激活状态和转移。

```
for i = 1:51  
    step(r,'x',X(i));  
    Y(i) = r.y;  
end
```

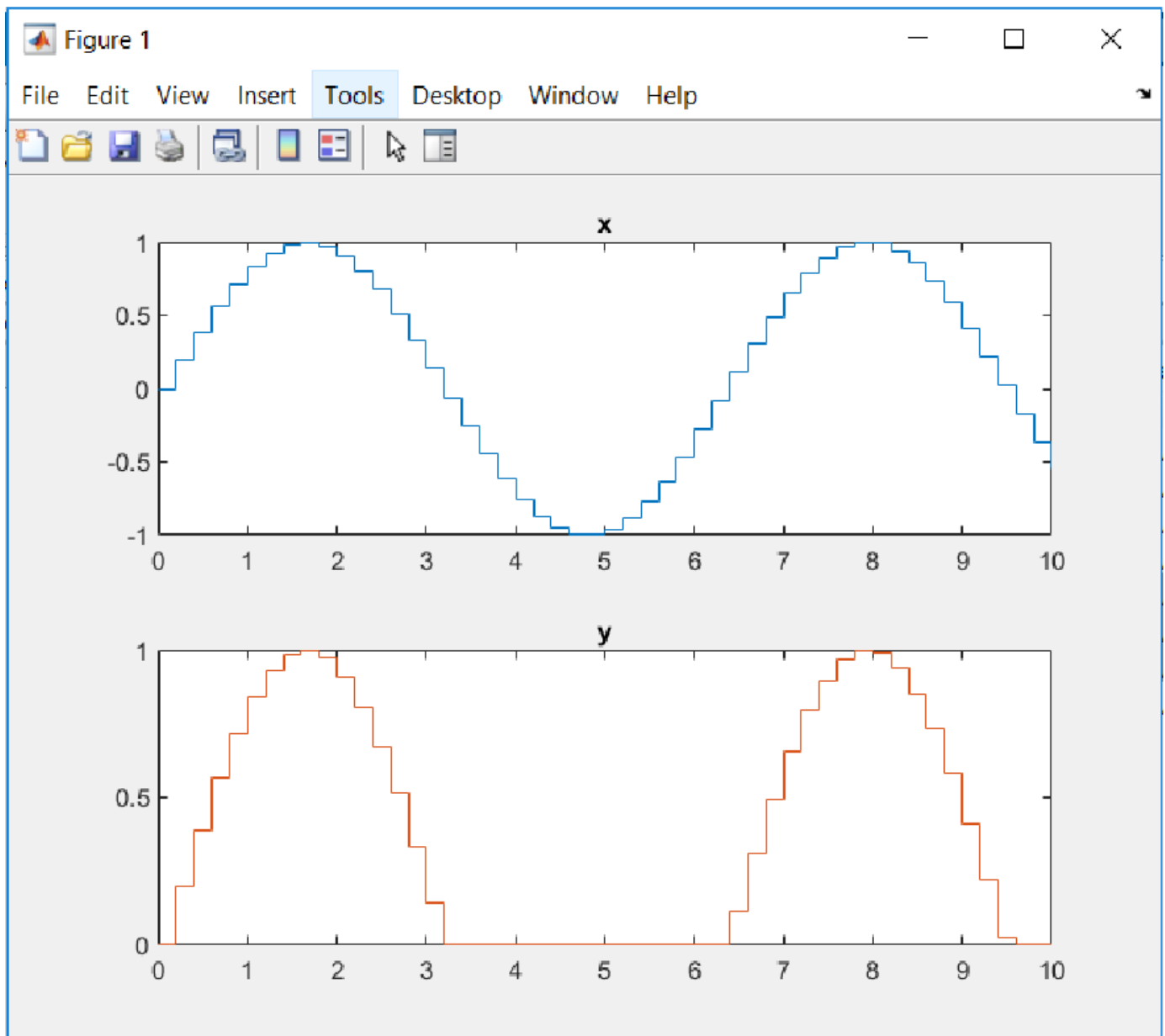
- 4 从 MATLAB 工作区中删除图对象 **r**。

```
delete(r)
```

- 5 检查图执行的结果。例如，您可以调用 **stairs** 函数来创建一个阶梯图，用于比较 **X** 和 **Y** 的值。

```
ax1 = subplot(2,1,1);  
stairs(ax1,T,X,'color','#0072BD')  
title(ax1,'x')
```

```
ax2 = subplot(2,1,2);  
stairs(ax2,T,Y,'color','#D95319')  
title(ax2,'y')
```



执行结果显示整流器滤除了负输入值。

另请参阅

Chart | Scope | Sine Wave | `sfnew` | `stairs`

详细信息

- “对有限状态机建模” (第 2-2 页)
- “使用动作定义图行为” (第 2-17 页)
- “Stateflow Editor 操作”

- “解析图中未定义的符号”
- “Create Stateflow Charts for Execution as MATLAB Objects”

使用动作定义图行为

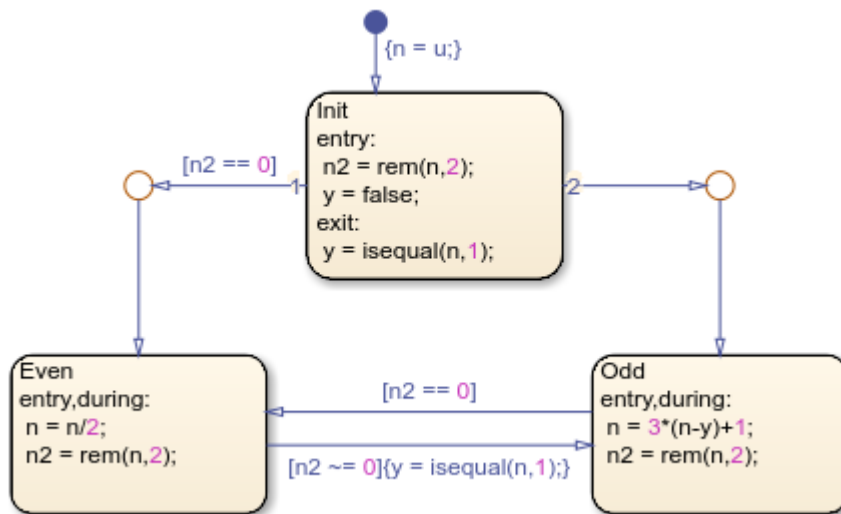
状态和转移动作是您在状态或转移旁编写的指令，用于定义 Stateflow 图在仿真过程中的行为。有关详细信息，请参阅“对有限状态机建模”（第 2-2 页）。

状态和转移动作示例

下图中的动作定义了一个以实验方式验证 Collatz 猜想实例的状态机。对于给定的数值输入 u ，该图通过迭代以下规则来计算冰雹序列 $n_0 = u, n_1, n_2, n_3, \dots$ ：

- 如果 n_i 为偶数，则 $n_{i+1} = n_i/2$ 。
- 如果 n_i 为奇数，则 $n_{i+1} = 3n_i + 1$ 。

Collatz 猜想指出，每个正整数都有一个最终达到 1 的冰雹序列。



该图由三个状态组成。在仿真开始时，Init 状态初始化图数据：

- 本地数据 n 设置为输入 u 的值。
- 当 n 除以 2 时，本地数据 $n2$ 设置为余数。
- 输出数据 y 设置为 `false`。

根据输入的奇偶性，图转移到 **Even** 或 **Odd** 状态。当状态活动在 **Even** 和 **Odd** 状态之间切换时，图会计算冰雹序列中的数字。当序列达到 1 值时，输出数据 y 变为 `true`，并触发 Simulink® 模型中的 Stop Simulation (Simulink) 模块。

状态动作类型

状态动作定义当状态被激活时 Stateflow 图的动作。最常见的状态动作类型是 **entry**、**during** 和 **exit** 动作。

状态动作的类型	缩写	说明
entry	en	当状态被激活时，动作在时间步上发生。
during	du	当状态已激活并且图未转移出该状态时，动作在时间步上发生。
exit	ex	当图转移出状态时，动作在时间步上发生。

您可以通过状态动作的完整关键字（**entry**、**during**、**exit**）或其缩写（**en**、**du**、**ex**）指定状态动作的类型。您还可以使用逗号组合各状态动作类型。例如，组合类型 **entry,during** 当状态被激活时在时间步上发生，并且在状态保持激活时在每个后续时间步上发生。

下表列出了冰雹图中每个状态动作的结果。

状态	动作	结果
Init	entry: n2 = rem(n,2); y = false;	在仿真开始，Init 被激活时，确定 n 的奇偶性并将 y 设置为 false。
	exit: y = isequal(n,1);	当经过一个时间步后转移出 Init 时，确定 n 是否等于 1。
Even	entry,during: n = n/2; n2 = rem(n,2);	计算冰雹序列的下一个数字 ($n / 2$) 并在以下时间步上更新其奇偶性： <ul style="list-style-type: none">Even 首次被激活时的时间步。Even 保持激活的每个随后的时间步。
Odd	entry,during: n = 3*(n-y)+1; n2 = rem(n,2);	计算冰雹序列的下一个数字 ($3n+1$) 并在以下时间步上更新其奇偶性： <ul style="list-style-type: none">Odd 首次被激活时的时间步。Odd 保持激活的每个随后的时间步。 <p>在整个仿真过程的大部分阶段，y 的计算结果为零。在最后一个时间步上，当 $n = 1$ 时，y 的计算结果为 1，因此该动作在仿真停止之前不修改 n 或 n2。</p>

转移动作的类型

转移动作定义当转移出激活状态时 Stateflow 图执行的动作。最常见的转移动作类型是条件和条件动作。要指定转移动作，请使用采用以下语法的标签：

[condition]{conditional_action}

condition 是布尔表达式，用于确定是否发生转移。如果不指定条件，则假定采用一个计算结果为 true 的隐含条件。

conditional_action 是在判断转移的条件为 true 时执行的指令。条件动作发生在条件后，但在任何 **exit** 或 **entry** 状态动作之前。


下表列出了冰雹图中每个转移动作的结果。

转移	动作	动作类型	结果
到 Init 的默认转移	n = u	条件动作	在仿真开始时，将输入值 u 赋给本地数据 n。

转移	动作	动作类型	结果
从 Init 到 Even 的转移	$n2 == 0$	条件	当 n 为偶数时，发生转移。此转移起始处的数字 1 表示此转移将先于指向 Odd 的转移进行计算。
从 Init 到 Odd 的转移		无	当 n 为奇数时，发生转移。此转移起始处的数字 2 表示此转移将在指向 Even 的转移之后进行计算。
从 Odd 到 Even 的转移	$n2 == 0$	条件	当 n 为偶数时，发生转移。
从 Even 到 Odd 的转移	$n2 \sim= 0$	条件	当 n 为奇数时，发生转移。
	$y = \text{isequal}(n,1)$	条件动作	在发生转移时，确定 n 是否等于 1。

检查图行为

假设您要计算从 9 开始的冰雹序列。

- 在 Model Configuration Parameters 对话框的 **Solver** 下，选择以下选项：
 - **Start time:** 0.0
 - **Stop time:** inf
 - **Type:** "Fixed-step"
 - **Fixed-step size:** 1
- 在 Symbols 窗格中，选择本地数据 n 。在 Property Inspector 的 **Logging** 下，选择 **Log signal data**。
- 在 Constant 模块中，输入 $u = 9$ 。
- 在 **Simulation** 选项卡中，点击 **Run** 。

图用下列动作予以响应：

- 在时间 $t = 0$ 处，发生到 **Init** 的默认转移。
 - 转移动作将 n 的值设置为 9。
 - **Init** 状态被激活。
 - **Init** 中的 **entry** 动作将 $n2$ 设置为 1 且将 y 设置为 **false**。
- 在时间 $t = 1$ 处，条件 $n2 == 0$ 为 **false**，因此图准备转移到 **Odd**。
 - **Init** 中的 **exit** 动作将 y 设置为 **false**。
 - **Init** 状态变为非激活。
 - **Odd** 状态被激活。
 - **Odd** 中的 **entry** 动作将 n 设置为 28，将 $n2$ 设置为 0。
- 在时间 $t = 2$ 处，条件 $n2 == 0$ 为 **true**，因此图准备转移到 **Even**。
 - **Odd** 状态变为非激活。
 - **Even** 状态被激活。
 - **Even** 中的 **entry** 动作将 n 设置为 14，将 $n2$ 设置为 0。
- 在时间 $t = 3$ 处，条件 $n2 \sim= 0$ 为 **false**，因此图不会发生转移。

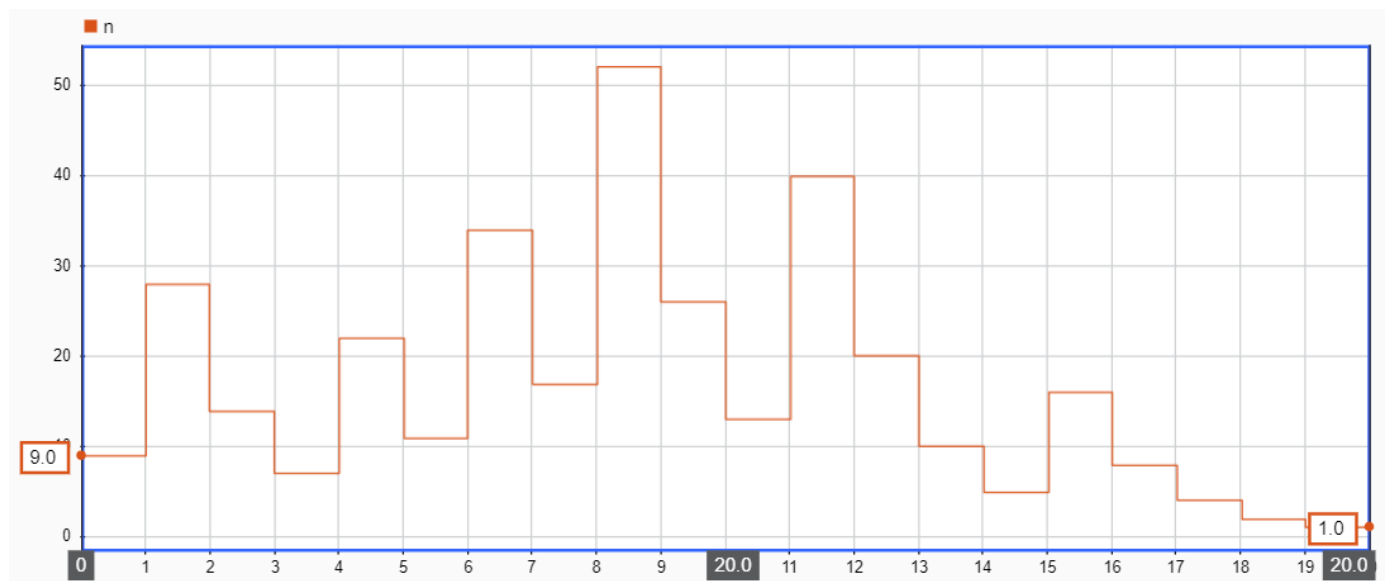
- **Even** 状态保持激活。
- **Even** 中的 **during** 动作将 **n** 设置为 7，将 **n2** 设置为 1。
- 在时间 $t = 4$ 处，条件 $n2 \sim= 0$ 为 true，因此图准备转移到 **Odd**。
 - 转移动作将 **y** 设置为 false。
 - **Even** 状态变为非激活。
 - **Odd** 状态被激活。
 - **Odd** 中的 **entry** 动作将 **n** 设置为 22，将 **n2** 设置为 0。
- 该图继续计算冰雹序列，直到它在时间 $t = 19$ 处达到 $n = 1$ 的值。
- 在时间 $t = 20$ 处，图准备从 **Even** 转移到 **Odd**。
 - 在 **Even** 状态变为未激活之前，转移动作将 **y** 设置为 true。
 - **Odd** 状态被激活。
 - **Odd** 中的 **entry** 动作不会修改 **n** 或 **n2**。
 - 连接到输出信号 **y** 的 Stop Simulation 模块停止仿真。

5

在 **Simulation** 选项卡中的 **Review Results** 下，点击 **Data Inspector** .

6

要查看冰雹序列的值，请在 Simulation Data Inspector 中，选择记录的信号 **n**。



另请参阅

Stop Simulation

另请参阅

详细信息

- “对有限状态机建模” (第 2-2 页)
- “状态动作类型”

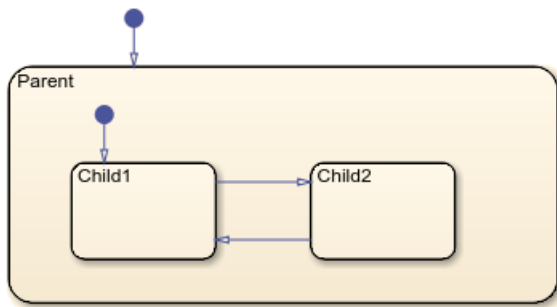
- “转移动作的类型”
- “通过广播事件同步并行状态” (第 2-31 页)
- “使用时序逻辑调度图动作” (第 2-45 页)

创建层次结构来管理复杂系统

构建模型时，以一次添加一个子组件的方式创建一个包含嵌套状态的层次结构。这样，您便可以控制 Stateflow 图中复杂系统的各个层级。有关详细信息，请参阅“对有限状态机建模”（第 2-2 页）。

状态层次结构

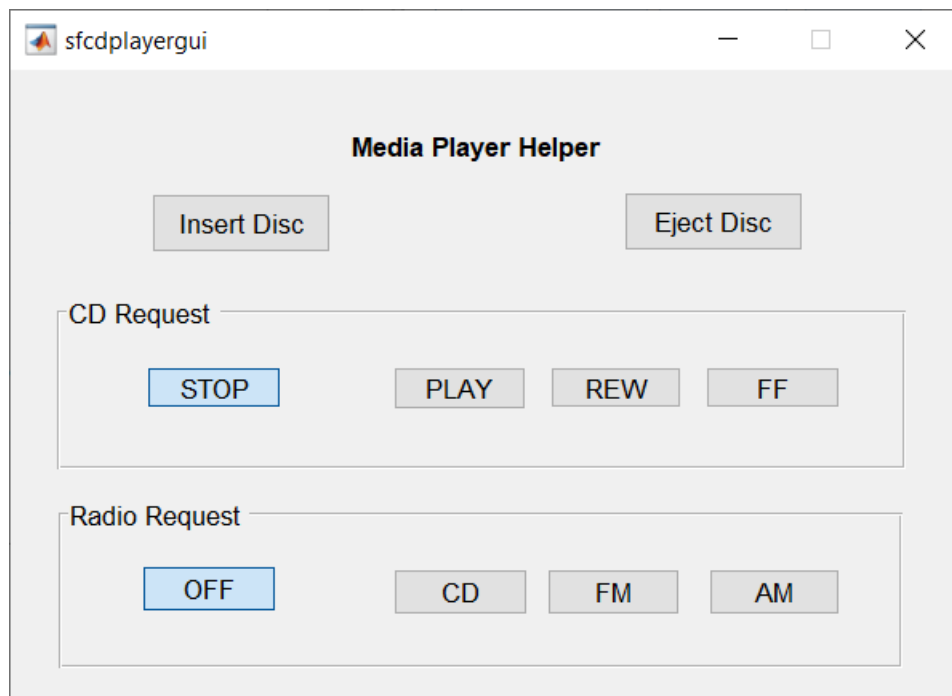
要创建状态层次结构，请将一个或多个状态置于另一个状态的边界内。内部状态是外部状态的子级状态（或子状态）。外部状态是内部状态的父级（或父状态）。



父状态的内容在行为上与较小的图类似。当父状态被激活时，其子状态之一也被激活。当父状态变为非激活时，其所有子状态都变为非激活。

层次结构的示例

此示例对一个由 AM 无线电、FM 无线电和 CD 播放器组成的立体声音响系统进行建模。在仿真过程中，您可以通过点击 Media Player Helper 用户界面上的按钮来控制立体声音响系统。



立体声音响最初处于待机模式 (OFF)。当您选择一个 Radio Request 按钮时，立体声音响将打开对应的子组件。如果您选择 CD 播放器，则可以点击其中一个 CD Request 按钮以选择 Play、Rewind、Fast-Forward 或 Stop。您可以在仿真过程中的任何时刻插入或弹出光盘。

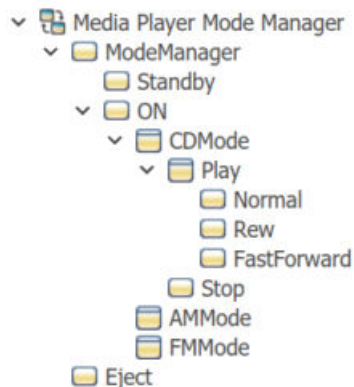
通过使用状态层次结构实现行为

最初，此立体声音响系统的完整实现看起来相当复杂。但是，通过一次只关注一个层面的活动，您可以逐步完成整个系统设计。例如，下列是 CD 播放器进入 Fast-Forward 播放模式是必要条件：

- 1 打开了立体声。
- 2 打开了 CD 播放器。
- 3 播放了光盘。
- 4 您在用户界面中点击 FF 按钮。

您可以每次考虑其中一个条件，逐步构建一个层次结构模型。例如，最外层可以定义立体声音响打开和关闭之间的转移。中间各层定义不同立体声音响子组件之间的转移，以及 CD 播放器的停止和播放模式之间的转移。最底层定义当满足播放光盘的所有其他条件时对 CD Request 按钮的响应。

为实现立体声音响系统的行为，`sf_cdplayer` 使用 Model Explorer 在 Media Player Mode Manager 图下方列出的嵌套状态层次结构。要打开 Model Explorer，请在 **Modeling** 选项卡中，选择 **Model Explorer**。

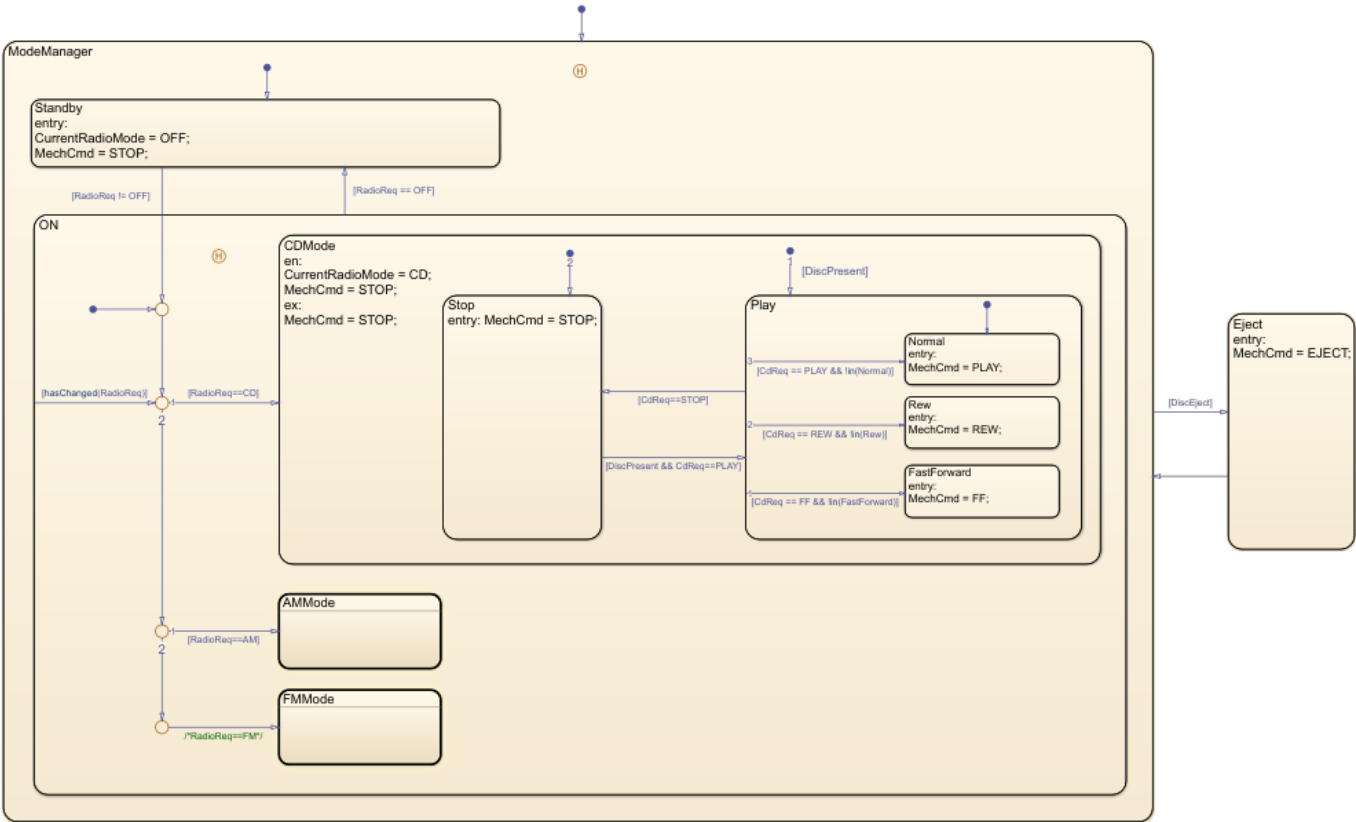


下表列出了层次结构中每个状态的角色。

层级	状态	说明
顶层 (Media Player Mode Manager 图)	ModeManager	立体声音响系统的正常工作模式
	Eject	光盘弹出模式 (中断所有其他立体声音响功能)
立体声音响系统活动 (ModeManager 的子状态)	Standby	立体声音响系统处于待机模式 (OFF)
	ON	立体声音响系统被激活 (ON)
立体声音响子组件 (On 的子状态)	AMMode	AM 无线电子组件被激活
	FMMODE	FM 无线电子组件被激活
	CDMode	CD 播放器子组件被激活
CD 播放器活动 (CDMode 的子状态)	Stop	CD 播放器停止

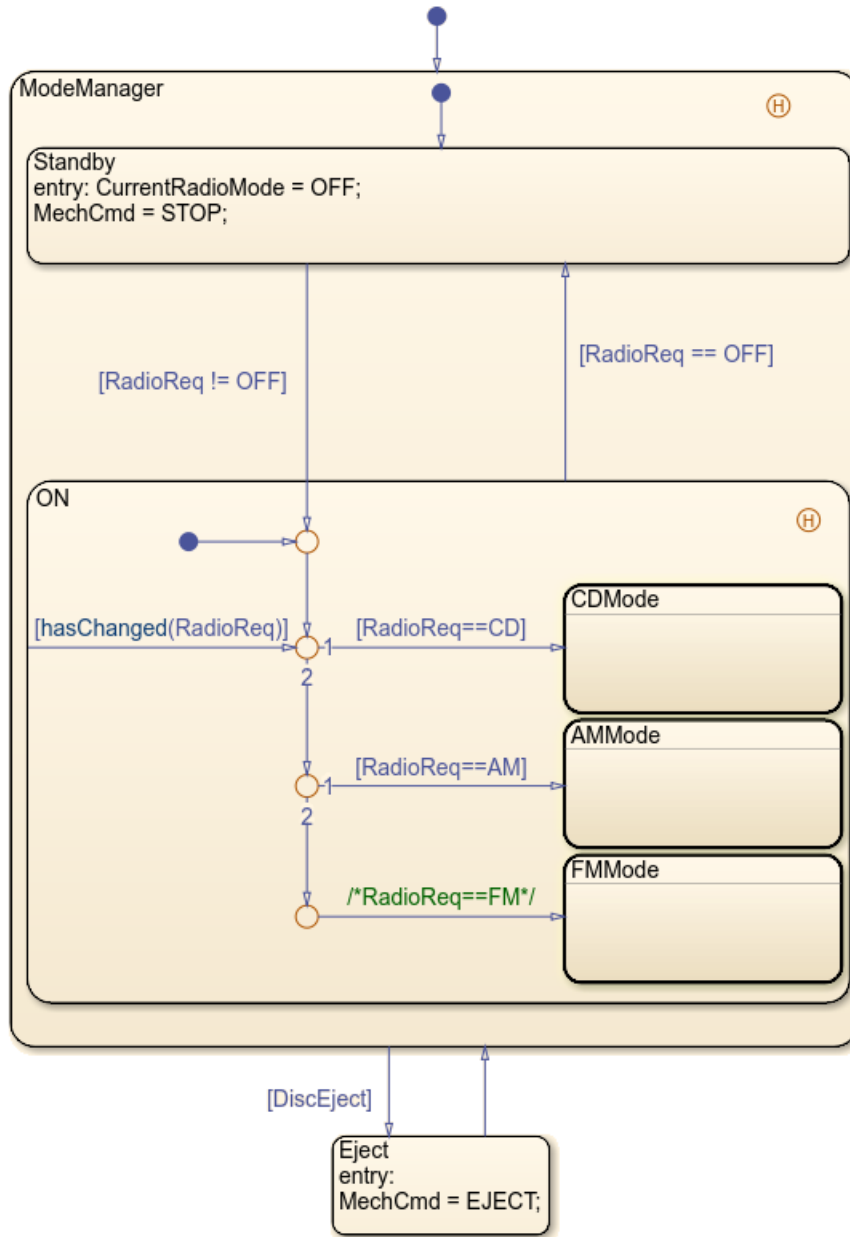
层级	状态	说明
光盘播放模式（Play 的子状态）	Play	CD 播放器正在播放光盘
	Normal	正常播放模式
	Rew	反向播放模式
	FastForward	Fast-Forward 播放模式

下图显示了图中状态的完整布局。



使用子图简化图的外观

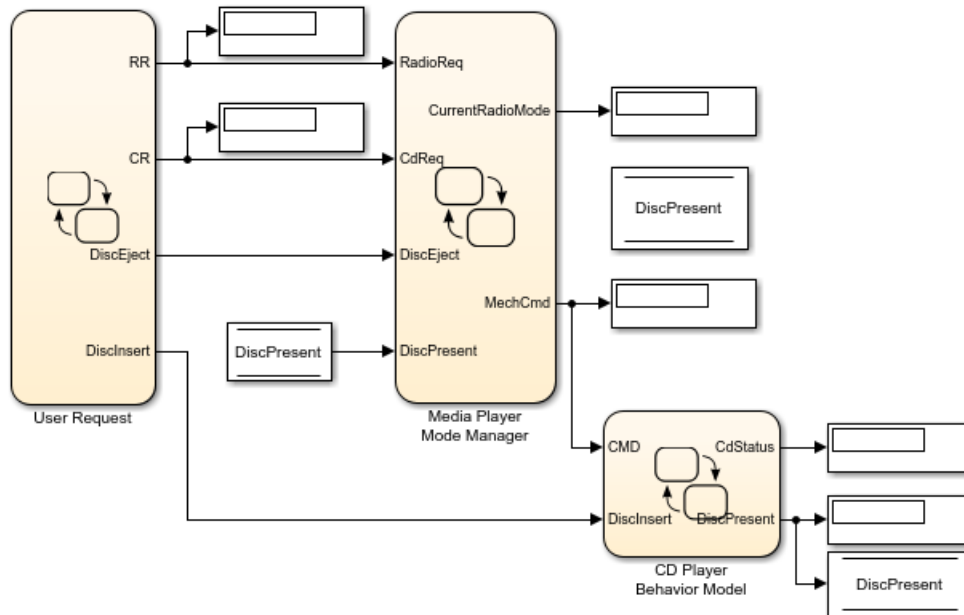
通过隐藏子图（显示为不透明的框）内的低层级细节，可以简化具有复杂层次结构的图的整体外观。子图的使用不会更改图的行为。例如，在 `sf_cdplayer` 中，立体声音响子组件 `AMMode`、`FMMode` 和 `CDMode` 作为子图实现。当您打开 Media Player Mode Manager 图时，只会看到三个状态层级。要查看其中一个子图内的详细信息，请双击该子图。



探索示例

示例 `sf_cdplayer` 包含另外两个 Stateflow 图：

- User Request 使用 UI 来管理界面，并将输入传递给 Media Player Mode Manager 和 CD Player Behavior Model 图。
- CD Player Behavior Model 接收来自 User Request 和 Media Player Mode Manager 图的输出，并模拟 CD 播放器的机械行为。



在仿真过程中，您可以研究每个图如何响应与 Media Player Helper 的交互。要在各图之间快速切换，请使用 Stateflow Editor 顶部的选项卡。

另请参阅

相关示例

- “Model Media Player by Using Enumerated Data”
- “Modeling a CD Player/Radio Using State Transition Tables”
- “Simulate a Media Player”

详细信息

- “对有限状态机建模”（第 2-2 页）
- “State Hierarchy”
- “Encapsulate Modal Logic by Using Subcharts”

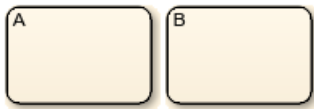
使用并行机制对同步子系统建模

要实现同时运行的多个工作模式，请在 Stateflow 图中使用并行机制。例如，作为复杂系统设计的一部分，您可以使用并行状态对同时被激活的独立组件或子系统建模。有关详细信息，请参阅“对有限状态机建模”（第 2-2 页）。

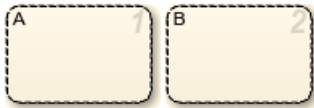
状态分解

Stateflow 图可以组合互斥 (OR) 状态和并行 (AND) 状态：

- **互斥 (OR) 状态**表示互斥的工作模式。在同一层级上不能有两个互斥状态同时被激活或执行。Stateflow 以实心矩形表示每个互斥状态。



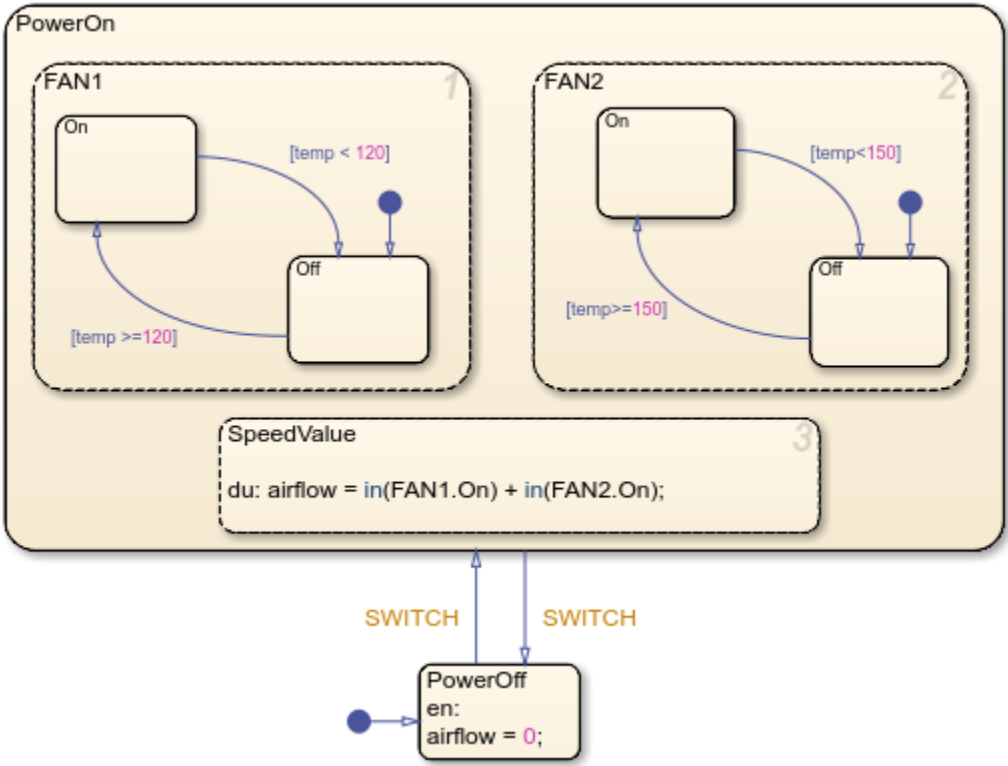
- **并行 (AND) 状态**表示独立的工作模式。虽然并行状态会依次执行，但可以有两个或多个并行状态同时处于激活状态。Stateflow 用虚线矩形表示每个并行状态，其中的数字表示其执行顺序。



一个给定层级上的所有状态必须具有相同的类型。父状态（顶层状态的父状态即为图本身）具有 OR（互斥）或 AND（并行）分解。默认的状态分解类型是 OR（互斥）。要更改分解类型，请右键单击父状态并选择 **Decomposition > AND (Parallel)**。

并行分解的示例

此示例采用并行机制来实现一个空调控制器，该控制器将实际被控对象中的气温保持在 120 度。



控制器使用两个风扇来工作。当气温升至 120 度以上时，启动第一个风扇。当气温升至 150 度以上时，启动第二个风扇以提供额外的冷却。图将这两个风扇建模为并行状态 FAN1 和 FAN2，在控制器打开时这两个状态都会被激活。除工作阈值不同以外，这两个风扇都具有表示两种风扇工作模式（On 和 Off）的相同状态和转移配置。

第三个并行状态 SpeedValue 根据每个时间步内循环启动的风扇数来计算输出数据 airflow 的值。当 FAN1 的 On 状态被激活时，布尔表达式 in(FAN1.On) 的值为 1。否则，in(FAN1.On) 等于 0。同样，in(FAN2.On) 的值指示 FAN2 是处于循环打开还是关闭状态。这些表达式的总和表示在每个时间步中打开的风扇的数量。

组合使用互斥 (OR) 状态和并行 (AND) 状态

下表列出了在空调控制器图中使用互斥 (OR) 和并行 (AND) 状态的基本原理。

状态	分解	有理数
PowerOff、PowerOn	互斥 (OR) 状态	控制器不能同时处于开启和关闭状态。
FAN1、FAN2	并行 (AND) 状态	风扇作为独立的组件运行，根据所需冷却量的多少打开或关闭。
FAN1.On、FAN1.Off	互斥 (OR) 状态	风扇 1 不能同时处于打开和关闭状态。
FAN2.On、FAN2.Off	互斥 (OR) 状态	风扇 2 不能同时处于打开和关闭状态。
SpeedValue	并行 (AND) 状态	SpeedValue 表示独立的子系统，用于监视每个时间步的风扇状态。

注意 要为在图层次结构的不同部分具有相同名称的对象赋予唯一标识符，请使用圆点表示法，例如 **Fan1.On** 和 **Fan2.On**。有关详细信息，请参阅“使用圆点表示法标识数据”。

并行状态的执行顺序

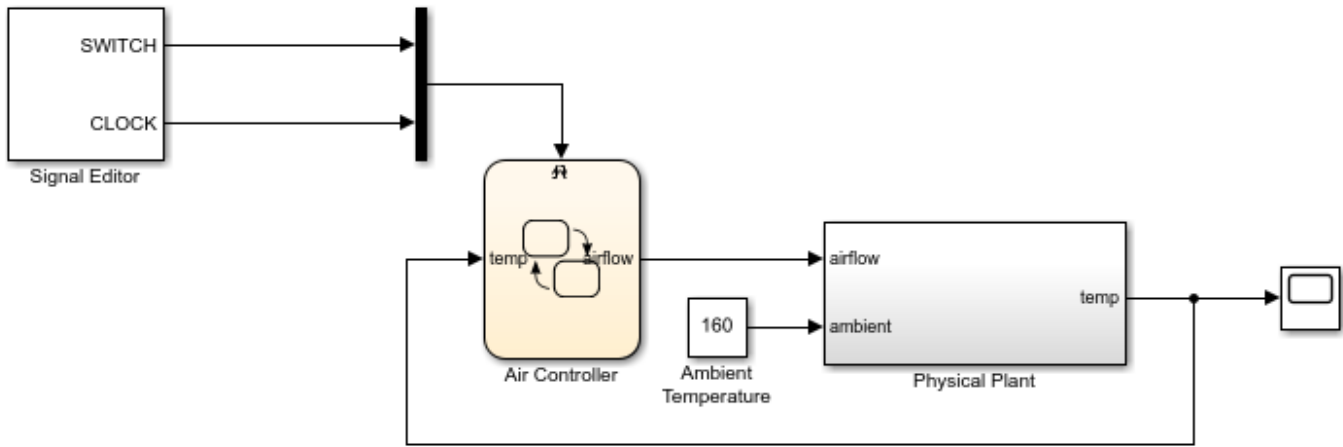
即使 **FAN1**、**FAN2** 和 **SpeedValue** 同时处于激活状态，这些状态在仿真过程中也是依次执行的。状态右上角的数字指定执行顺序。以下是此执行顺序的基本原理：

- **FAN1** 应首先执行，因为它循环打开的温度低于 **FAN2** 打开的温度。无论 **FAN2** 是打开还是关闭，它都可以打开。
- **FAN2** 以第二顺位执行，因为它循环打开的温度高于 **FAN1** 打开的温度。它仅在 **FAN1** 已打开时才能打开。
- **SpeedValue** 最后执行，因而可以观测 **FAN1** 和 **FAN2** 的最新状态。

默认情况下，Stateflow 根据图中并行状态的创建顺序来分配其执行顺序。要更改某并行状态的执行顺序，请右键点击该状态并从 **Execution Order** 下拉列表中选择值。

探索示例

Stateflow 示例包含 Stateflow 图和 Simulink 子系统。



根据气温 **temp**，Air Controller 图打开风扇并将 **airflow** 的值传递给 Physical Plant 子系统。此输出值确定冷却活动的量，如下表所示。

airflow 的值	说明	冷却活动因子 $k_{\text{冷却}}$
0	没有风扇在运行。 temp 的值不会降低。	0
1	一个风扇在运行。 temp 的值根据冷却活动因子降低。	0.05
2	两个风扇在运行。 temp 的值根据冷却活动因子降低。	0.1

Physical Plant 模块根据公式

$$\begin{aligned} \text{temp}(0) &= T_{\text{Initial}} \\ \text{temp}'(t) &= (T_{\text{Ambient}} - \text{temp}(t)) \cdot (k_{\text{Heat}} - k_{\text{Cool}}), \end{aligned}$$

更新工厂内的气温，其中：

- T_{Initial} 是初始温度（默认值 = 70 °）
- T_{Ambient} 是环境温度（默认值 = 160 °）
- k_{Heat} 是工厂的热传递因子（默认值 = 0.01）
- k_{Cool} 是对应于 **airflow** 的冷却活动因子

temp 的新值决定仿真的下一个时间步的冷却量。

另请参阅

in

详细信息

- “对有限状态机建模”（第 2-2 页）
- “State Decomposition”
- “并行状态的执行顺序”
- “Check State Activity by Using the in Operator”
- “通过广播事件同步并行状态”（第 2-31 页）

通过广播事件同步并行状态

事件帮助并行状态相互协调，从而使一个状态可触发另一个状态中的动作。要在同一个 Stateflow 图中同步并行状态，请直接从一个状态向另一个状态广播事件。有关并行状态的详细信息，请参阅“使用并行机制对同步子系统建模”（第 2-27 页）。

广播本地事件

本地事件是非图形对象，它可以触发 Stateflow 图的并行状态中的转移或动作。当您将事件广播到某个状态时，该事件将在接收状态以及该状态层次结构中的任何子状态中生效。要广播事件，请使用 **send** 运算符：

```
send(event_name,state_name)
```

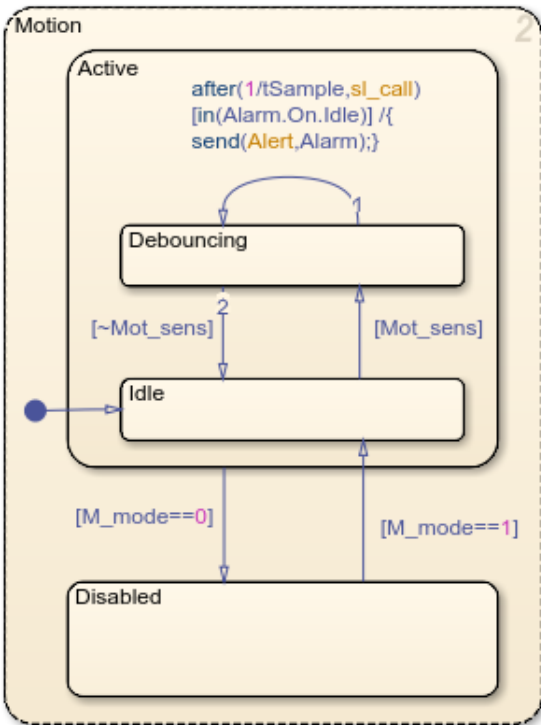
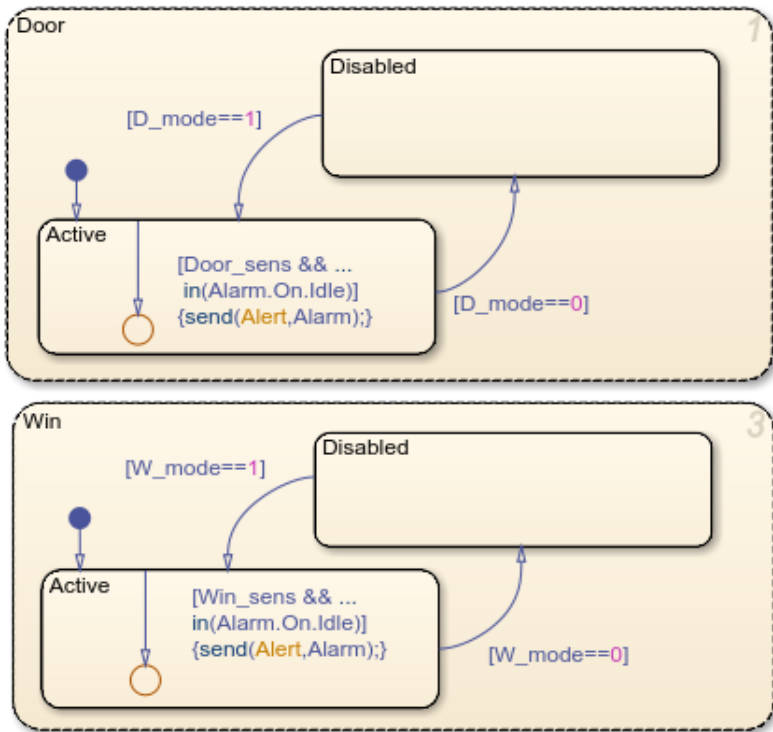
event_name 是要广播的事件的名称。**state_name** 在广播期间处于激活状态。

事件广播的示例

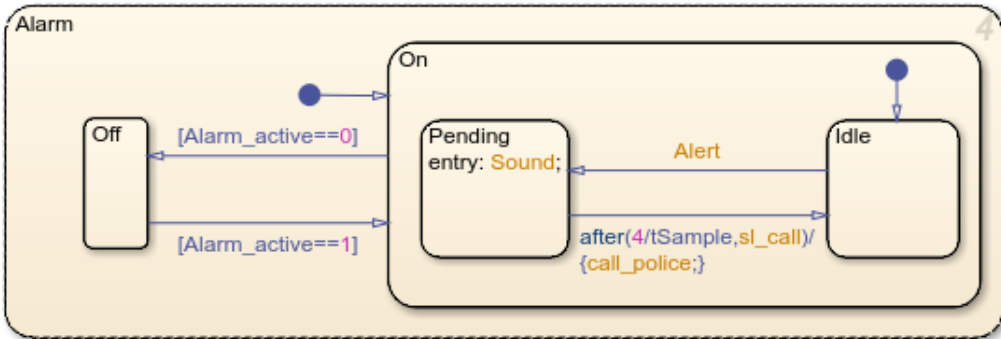
此示例使用本地事件作为家庭安全系统设计的一部分。

The Door and Win states model anti-intrusion sensors at a window and a door. When these sensors are active, the corresponding states check the value of D_mode and W_mode. If one of these variables is true, the states broadcast a local event to the Alarm state.

Input from the motion sensor can be sporadic, so a debouncing state is used. The debouncing state remains active while Mot_sens is true. After a delay period, the debouncing state broadcasts a local event to the Alert state.



When an Alert event is broadcast to the Alarm state, the Pending state becomes activate for a delay period. If the alarm is not disabled during the delay period, the police are contacted by the output event call_police.



该安全系统由一个警报器和三个防入侵传感器（窗口传感器、门传感器和运动检测器）组成。在系统检测到入侵后，您可以在较短的一段时间内禁用警报。否则，系统会向警方报告。

将子系统建模为并行状态

Security System 图用单独的并行状态为每个子系统建模。使能输入信号在 On 和 Off 模式（对于警报）之间或在 Active 和 Disabled 模式（对于传感器）之间进行选择。启用之后，每个传感器都会监控指示可能有入侵的触发输入信号。

子系统	状态	使能信号	触发信号
门传感器	Door	D_mode	Door_sens
窗口传感器	Win	W_mode	Win_sens
运动检测器	Motion	M_mode	Mot_sens
警报	Alarm	Alarm_active	

如果传感器在警报子系统打开时检测到入侵，则使用以下命令广播 **Alert** 事件：

```
send(Alert,Alarm)
```

为减轻偶发误报的影响，运动检测器采用了一种去抖设计，因此只有持续的正触发信号才会产生警报。相反，门传感器和窗口传感器将单个正触发信号解释为入侵并立即发出警报。

在警报子系统中，**Alert** 事件引起从 **Idle** 子状态到 **Pending** 子状态的转移。当此状态被激活时，会发出警告声，提醒住户可能发生入侵。如果发生误报，则住户可以在较短的一段时间内关闭安全系统。如果在这段时间内没有禁用，系统会向警方报警。

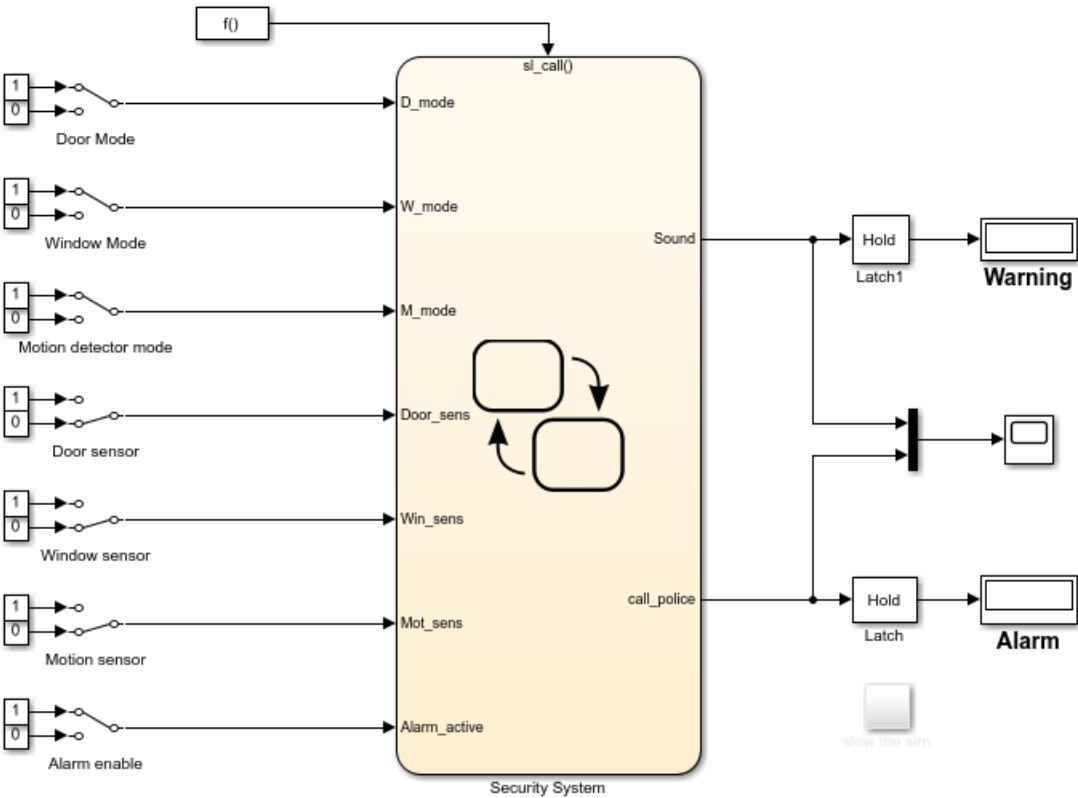
与其他 Simulink 模块协调

Stateflow 图可以使用事件与 Simulink 模型中的其他模块进行通信。例如，在 **sf_security** 示例中：

- 输出事件 **Sound** 和 **call_police** 驱动处理警告声音和向警方报警的外部模块。用于广播这些事件的命令在 **Alarm.On** 状态中发生：
 - **Sound** 命令作为 **Pending** 子状态中的 entry 动作执行。
 - **call_police** 命令作为 **Pending** 到 **Idle** 子状态转移中的动作执行。

在每种情况下，发出输出事件的命令都是事件的名称。

- 输入事件 **sl_call** 控制运动检测器去抖器的时序以及向警方报警之前的短时间延迟。在每个实例中，对时序运算符 **after** 的调用内会对该事件计数，并在图接收事件达一定次数后引发转移。



输出事件

输出事件是在 Stateflow 图中发生但在 Stateflow 图之外的 Simulink 模块中可见的事件。这种类型的事件支持 Stateflow 图将该图中发生的事件通知给其他模块。

每个输出事件映射到图右侧的一个输出端口。根据其配置，对应的信号可以控制触发子系统或函数调用子系统。要配置输出事件，请在 Property Inspector 中将 **Trigger** 字段设置为下列选项之一。

触发器的类型	说明
Either Edge	输出事件广播导致传出信号在 0 和 1 之间切换。
Function call	输出事件广播导致 Simulink 函数调用事件。

在 sf_security 示例中，输出事件 Sound 和 call_police 使用边沿触发器来激活 Simulink 模型中的一对锁存子系统。当每个锁存子系统检测到其输入信号中的值发生变化时，它会短暂输出值 1，然后再返回到 0 输出。

输入事件

输入事件发生在 Simulink 模块中，但在 Stateflow 图中可见。这种类型的事件支持其他 Simulink 模块（包括其他 Stateflow 图）将在特定 Stateflow 图之外发生的事件通知该图。

外部 Simulink 模块通过连接到 Stateflow 图顶部的触发端口的信号发送输入事件。根据其具体配置，输入事件是由信号值的变化或通过 Simulink 模块中的函数调用产生的。要配置输入事件，请在 Property Inspector 中将 **Trigger** 字段设置为下列选项之一。

触发器的类型	说明
Rising	当输入信号从零或负值变为正值时，图被激活。
Falling	当输入信号从正值变为零或负值时，图被激活。
Either	当输入信号在任一方向变化且过零时，图被激活。
Function call	通过从 Simulink 模块的函数调用来激活图。

在 `sf_security` 示例中，Simulink Function-Call Generator 模块通过周期性函数调用触发输入事件 `sl_call` 来控制安全系统的时序。

探索示例

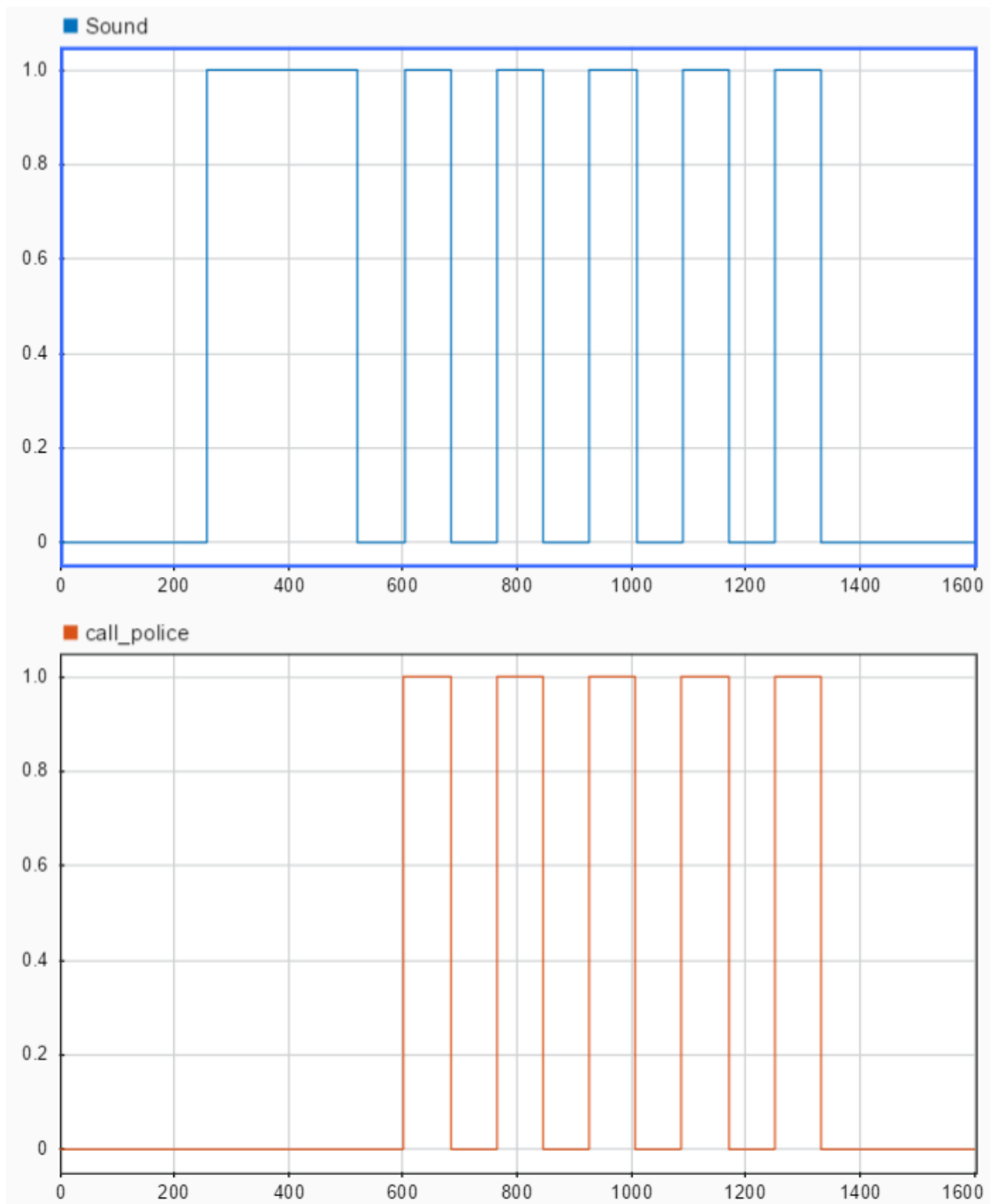
Security System 图从几个 Manual Switch 模块接收输入，并输出到一对连接到 Display 模块的锁存子系统。在仿真过程中，您可以：

- 通过点击 Switch 模块启用警报和传感器子系统并触发入侵检测。
- 观看图动画，其中突出显示了图中的各种激活状态。
- 查看 Scope 模块和 Simulation Data Inspector 中的输出信号。

要调整仿真的时序，请双击 Function-Call Generator 模块，然后在对话框中修改 **Sample time** 字段。例如，假设您将采样时间设置为 1，并在所有子系统都打开且所有传感器触发器都关闭的情况下开始仿真。在仿真过程中，您执行以下动作：

- 1 在 `t = 250` 秒的时刻，您触发门传感器。警报开始响起 (`Sound = 1`)，因此您立即禁用警报系统。您关闭触发器并重新打开警报。
- 2 在 `t = 520` 秒的时刻，您触发窗口传感器并且警报开始响起 (`Sound = 0`)。这次，您不禁用警报。在大约 `t = 600` 秒的时刻，安全系统向警方报警 (`call_police = 1`)。`Sound` 和 `call_police` 信号继续每隔 80 秒在 0 和 1 之间切换一次。
- 3 在 `t = 1400` 秒的时刻，您禁用警报。`Sound` 和 `call_police` 信号停止切换。

Simulation Data Inspector 显示 `Sound` 和 `call_police` 信号对您的动作的响应。



另请参阅

相关示例

- “Model a Security System”

详细信息

- “使用并行机制对同步子系统建模” (第 2-27 页)
- “通过发送输出事件激活 Simulink 模块”
- “通过发送输入事件激活 Stateflow 图”
- “使用触发子系统” (Simulink)
- “使用函数调用子系统” (Simulink)
- “使用时序逻辑调度图动作” (第 2-45 页)

使用激活状态数据监视图的活动

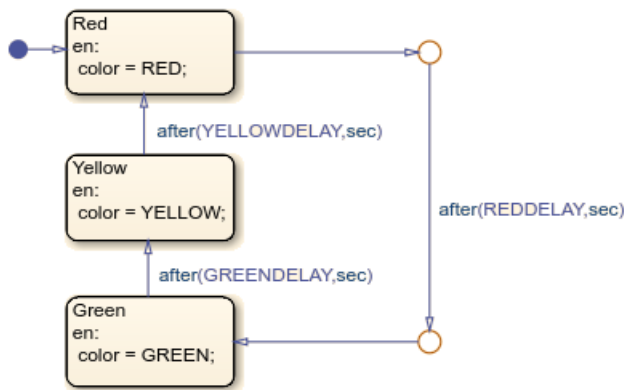
如果 Stateflow 图包含与图层次结构高度相关的数据，则可以使用激活状态数据来简化设计。通过启用激活状态数据，您可以：

- 避免手动更新反映图活动的数据。
- 在 Simulation Data Inspector 中记录并监视图活动。
- 使用图活动数据来控制其他子系统。
- 将图活动数据导出到其他 Simulink 模块。

有关详细信息，请参阅“创建层次结构来管理复杂系统”（第 2-22 页）。

激活状态数据

使用激活状态数据输出可以简化一些 Stateflow 图的设计。例如，在以下交通信号模型中，被激活的状态决定符号 **color** 的值。启用激活状态数据时，Stateflow 可以通过跟踪状态活动来提供交通信号的颜色。不再需要显式更新 **color**，因此您可以删除此符号并简化图的设计。



Stateflow 通过输出端口将激活状态数据提供给 Simulink 或将其作为本地数据提供给图。下表列出了可用的激活状态数据的不同模式。

活动模式	数据类型	说明
Self activity	布尔	该状态是否为激活？
Child activity	枚举	哪个子状态被激活？
Leaf state activity	枚举	哪个叶状态被激活？

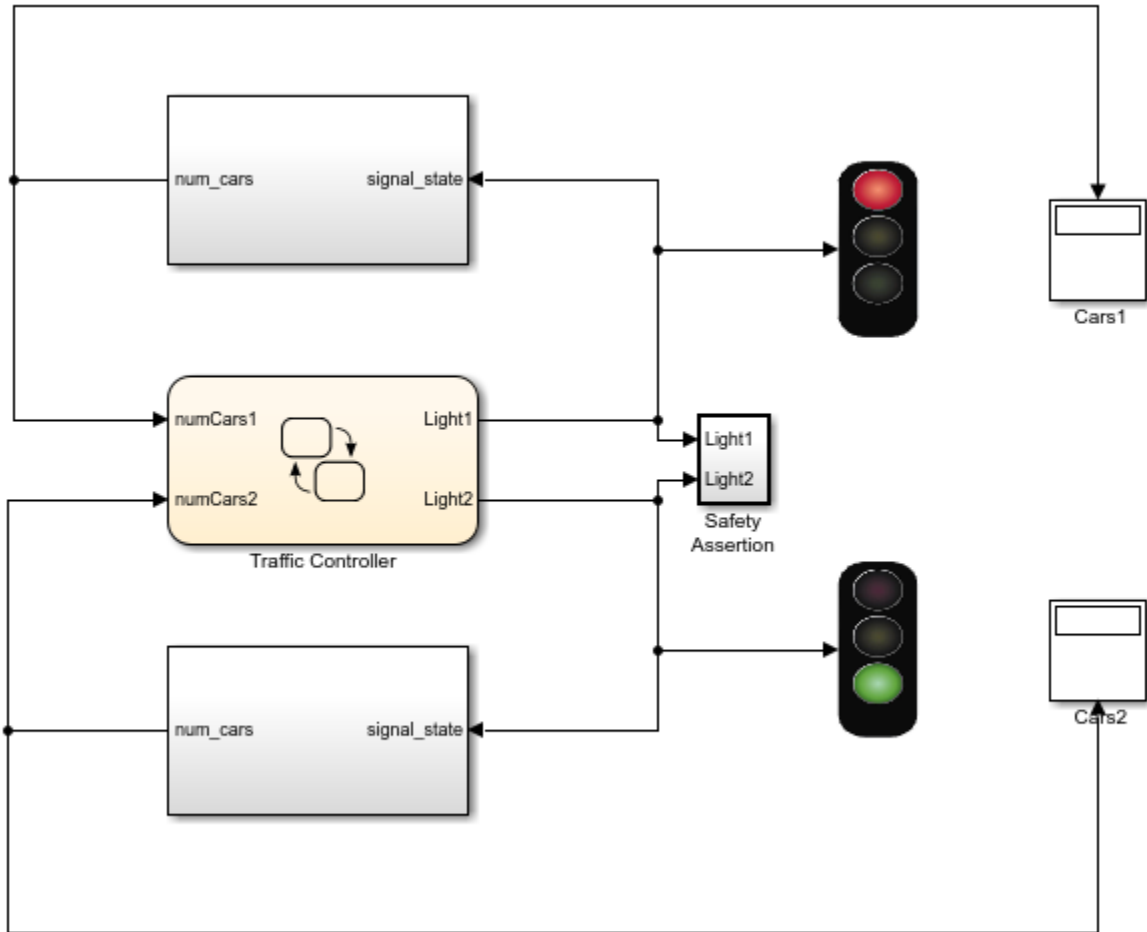
要启用激活状态数据，请使用 Property Inspector。

- 1 选中 **Create output for monitoring** 复选框。
- 2 从下拉列表中选择一种活动模式。
- 3 输入激活状态数据符号的 **Data name**。
- 4 （可选）对于子状态或叶状态活动，输入激活状态数据类型的 **Enum name**。

默认情况下，Stateflow 以输出数据形式报告状态活动。要将激活状态数据符号的作用域更改为本地数据，请使用 Symbols 窗格。

激活状态数据的示例

此示例使用激活状态数据为一对交通信号灯的控制器系统建模。



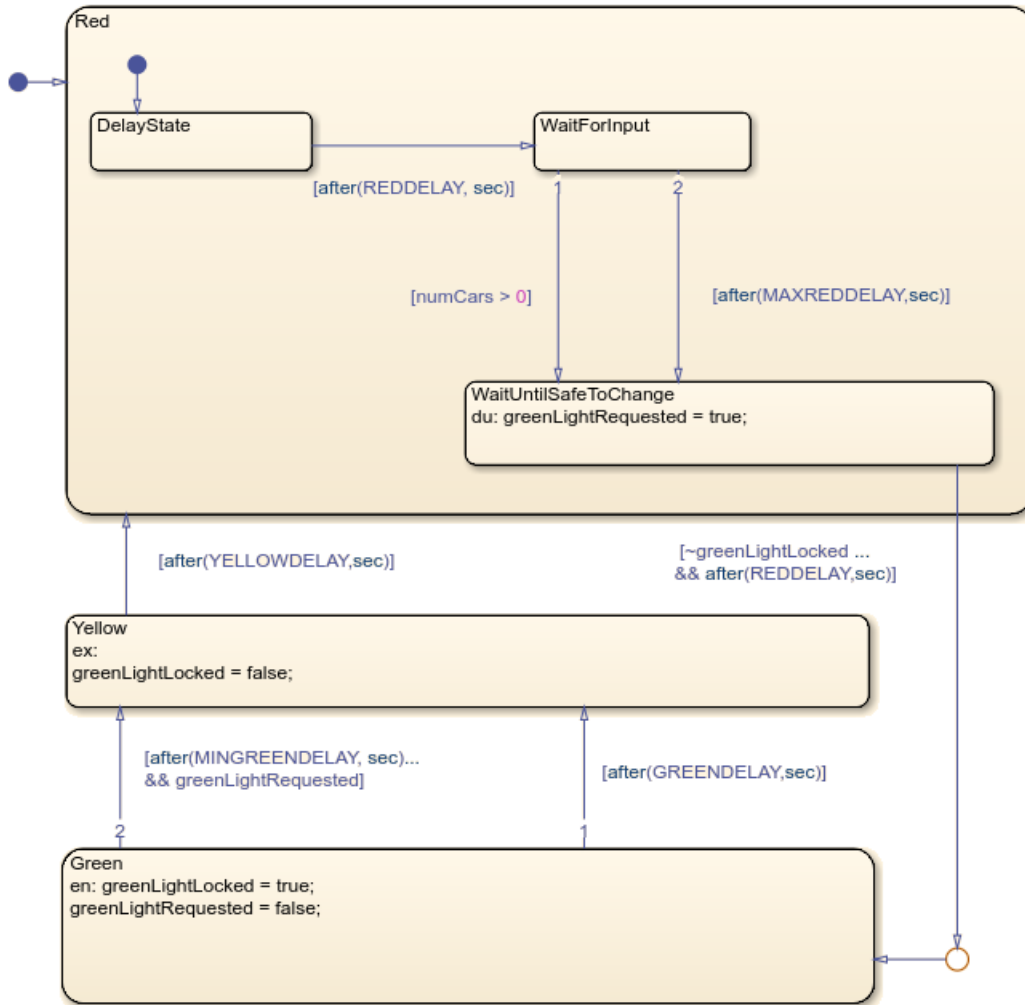
在 Traffic Controller 图中，两个并行的子图管理着控制交通信号灯的逻辑。这两个子图具有相同的层次结构，即包含三个子状态：Red、Yellow 和 Green。输出数据 Light1 和 Light2 对应于子图中的激活子状态。这些信号：

- 确定动画交通信号灯的阶段。
- 帮助统计在每个信号灯下等待的汽车数量。
- 驱动 Safety Assertion 子系统，确认两个交通信号灯永远不会同时呈绿色。

要查看 Traffic Controller 图中的子图，请点击图左下角的箭头。

交通控制器子图的行为

每个交通控制器循环遍历其子状态，从 Red 到 Green 到 Yellow，然后再返回到 Red。每个状态对应于交通信号灯循环中的一个阶段。输出信号 Light1 和 Light2 指示在任意给定时刻哪个状态被激活。



红灯

当 Red 状态被激活时，交通信号灯循环开始。经过短暂的延迟后，控制器检查在十字路口等待的汽车。如果检测到有至少一辆汽车或者经过了一定的时间，则控制器会将 `greenLightRequest` 设置为 `true`，以此来请求绿灯。在发出请求后，控制器继续保持 Red 状态较短的一段时间，直到检测到另一个交通信号为红色，控制器才会将状态转移到 Green。

绿灯

当 Green 状态被激活时，控制器会将 `greenLightRequest` 设置为 `false`，以此来取消其绿灯请求。控制器将 `greenLightLocked` 设置为 `true`，以防止另一个交通信号变绿。一段时间后，控制器会检查是否有来自另一个控制器的绿灯请求。如果它收到请求或经过了一定的时间，则控制器会转移到 Yellow 状态。

黄灯

在转移到 Red 状态之前，控制器将保持 Yellow 状态一定的时间。当 Yellow 状态变为非激活时，控制器会将 `greenLightLocked` 设置为 `false`，指示另一个交通信号灯可以安全地变绿。然后开始下一个交通信号灯循环。

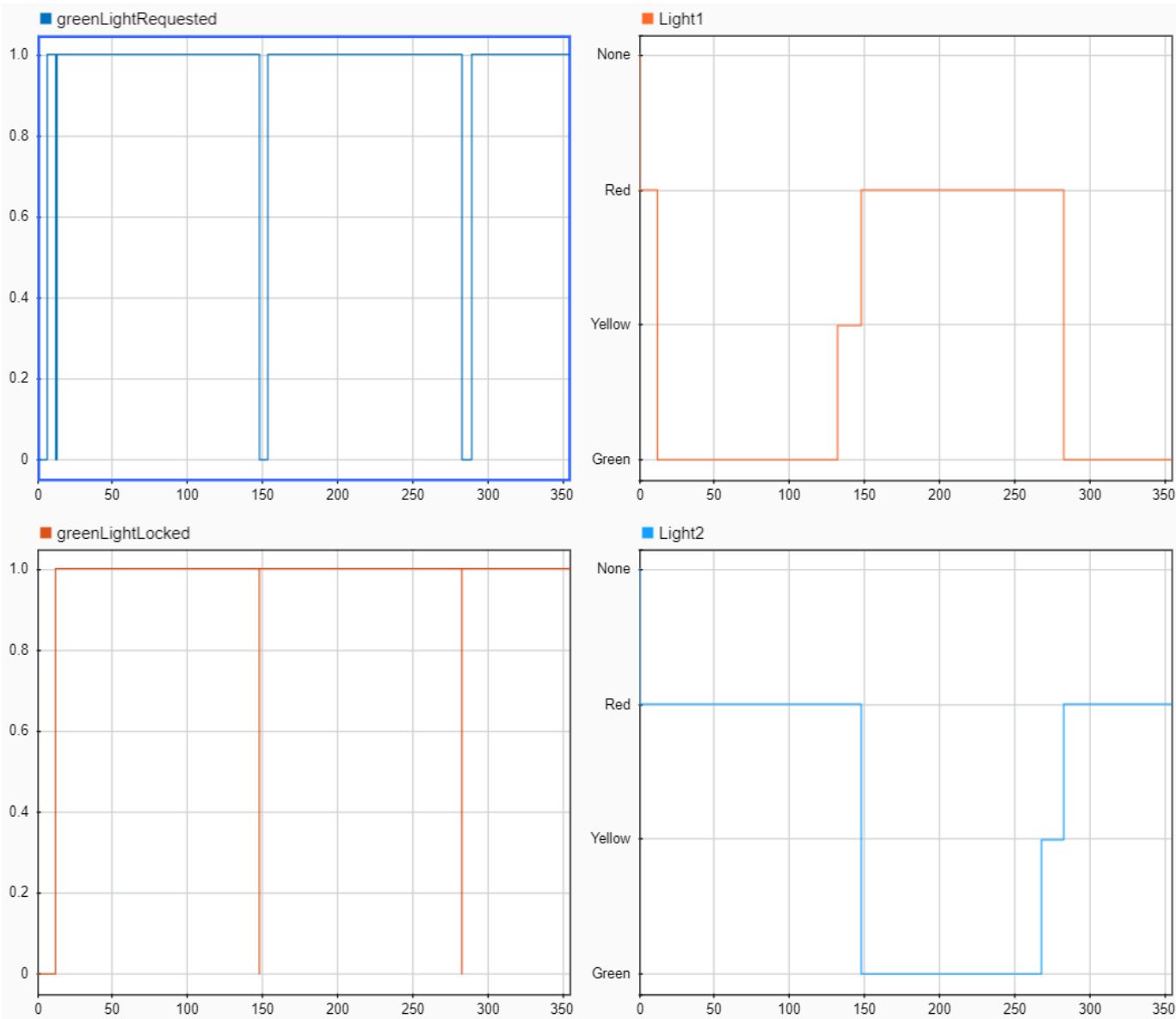
交通信号灯的时序

交通信号灯循环的时序通过几个参数来定义。要更改这些参数的值，请双击 Traffic Controller 图并在 Block Parameters 对话框中输入新值。

参数	预设值	说明
REDDELAY	6 秒	从红灯激活到控制器开始检查十字路口等待车辆之间的时间长度。这也是控制器发出绿灯请求后，交通信号灯可以变绿的最短时间。
MAXREDDelay	360 秒	控制器在发出绿灯请求前检查车辆的最长时间。
GREENDELAY	180 秒	交通信号灯保持绿色的最长时间。
MINGREENDELAY	120 秒	交通信号灯保持绿色的最短时间。
YELLOWDELAY	15 秒	交通信号灯保持黄色的时间长度。

探索示例

- 为这些符号中的每个符号启用日志记录。在 Symbols 窗格中，选择每个符号。在 Property Inspector 的 **Logging** 下，选择 **Log signal data**。
 - greenLightRequested
 - greenLightLocked
 - Light1
 - Light2
- 在 **Simulation** 选项卡中，点击 **Run** .
- 在 **Simulation** 选项卡中的 **Review Results** 下，点击 **Data Inspector** .
- 在 Simulation Data Inspector 中，将记录的信号显示在单独的坐标区中。布尔信号 greenLightRequested 和 greenLightLocked 显示为数值 0 或 1。状态活动信号 Light1 和 Light2 显示为枚举数据，其值为 Green、Yellow、Red 和 None。



要在仿真过程中跟踪图活动，可以使用 Simulation Data Inspector 中的缩放和光标按钮。例如，下表详细说明了仿真前 300 秒内的活动。

时间	说明	Light 1	Light2	greenLightRe quested	greenLightLo cked
t = 0	在仿真开始时，两个交通信号灯都为红色。	Red	Red	false	false
t = 6	6 秒后 (REDDELAY)，两条街道上都有汽车在等待。两个交通信号灯都通过设置 greenLightRequested = true 来请求绿灯。	Red	Red	true	false

时间	说明	Light 1	Light2	greenLightRequested	greenLightLocked
t = 12	再经过 6 秒后 (REDDELAY): <ul style="list-style-type: none"> 信号灯 1 变绿, 设置 greenLightLocked = true 和 greenLightRequested = false。 信号灯 2 通过设置 greenLightRequested = true 来请求绿灯。 	Green	Red	false, 然后 true	true
t = 132	120 秒后 (MINGREENDELAY), 信号灯 1 变黄。	Yellow	Red	true	true
t = 147	15 秒后 (YELLOWDELAY): <ul style="list-style-type: none"> 信号灯 1 变红, 设置 greenLightLocked = false。 信号灯 2 变绿, 设置 greenLightLocked = true 和 greenLightRequested = false。 	Red	Green	false	false, 然后 true
t = 153	6 秒后 (REDDELAY), 信号灯 1 通过设置 greenLightRequested = true 请求绿灯。	Red	Green	true	true
t = 267	信号灯 2 在持续 120 秒绿灯 (MINGREENDELAY) 后变黄。	Red	Yellow	true	true
t = 282	15 秒后 (YELLOWDELAY): <ul style="list-style-type: none"> 信号灯 2 变红, 设置 greenLightLocked = false。 信号灯 1 变绿, 设置 greenLightLocked = true 和 greenLightRequested = false。 	Green	Red	false	false, 然后 true
t = 288	6 秒后 (REDDELAY), 信号灯 2 通过设置 greenLightRequested = true 请求绿灯。	Green	Red	true	true

重复循环, 直到仿真于 t = 1000 秒处结束。

另请参阅

相关示例

- “Model An Intersection Of One-Way Streets”

详细信息

- “创建层次结构来管理复杂系统” (第 2-22 页)
- “使用时序逻辑调度图动作” (第 2-45 页)
- “Monitor State Activity Through Active State Data”
- “Simplify Stateflow Charts by Incorporating Active State Output”
- “View State Activity by Using the Simulation Data Inspector”

使用时序逻辑调度图动作

要定义 Stateflow 图在仿真时间的行为，请在图的状态和转移动作中包含时序逻辑运算符。时序逻辑运算符是内置函数，可以告知状态保持激活的时间长度或布尔条件保持为 true 的时间长度。使用时序逻辑，您可以控制以下各项的时序：

- 各状态之间的转移
- 函数调用
- 变量值的更改

有关详细信息，请参阅“使用动作定义图行为”（第 2-17 页）。

时序逻辑运算符

最常用的绝对时间时序逻辑运算符是 **after**、**elapsed** 和 **duration**。

运算符	语法	说明
after	after(n,sec)	如果自关联状态激活以来经过的仿真时间达到 n 秒，则返回 true。否则，运算符返回 false。
elapsed	elapsed(sec)	返回自关联状态激活以来经过的仿真时间的秒数。
duration	duration(C)	返回自布尔条件 C 变为 true 以来经过的仿真时间的秒数。

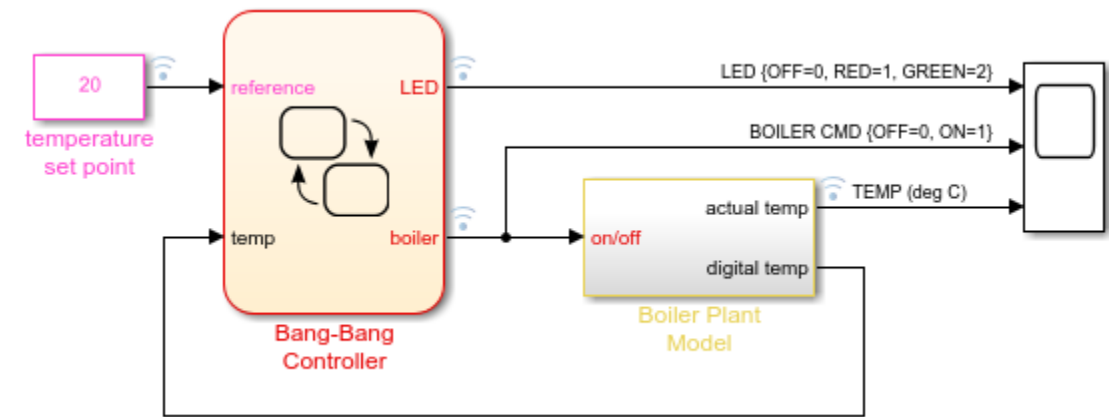
每个运算符都会在满足以下条件时将其关联计时器重置为零：

- 包含运算符的状态重新激活。
- 包含运算符的转移的源状态重新激活。
- **duration** 运算符中的布尔条件变为 false。

注意 某些运算符（如 **after**）支持以秒（sec）、毫秒（msec）和微秒（usec）计的基于事件的时序逻辑和绝对时间时序逻辑。有关详细信息，请参阅“使用时序逻辑控制图的执行”。

时序逻辑的示例

此示例使用时序逻辑对调节锅炉内部温度的 Bang-Bang 控制器建模。



该示例由 Stateflow 图和 Simulink® 子系统组成。Bang-Bang Controller 图将当前锅炉温度与参考设定值进行比较，并确定是否开启锅炉。Boiler Plant Model 子系统对锅炉内部的动态特性进行建模，根据控制器的状态升高或降低温度。然后，锅炉温度返回到控制器图，以进行下一步仿真。

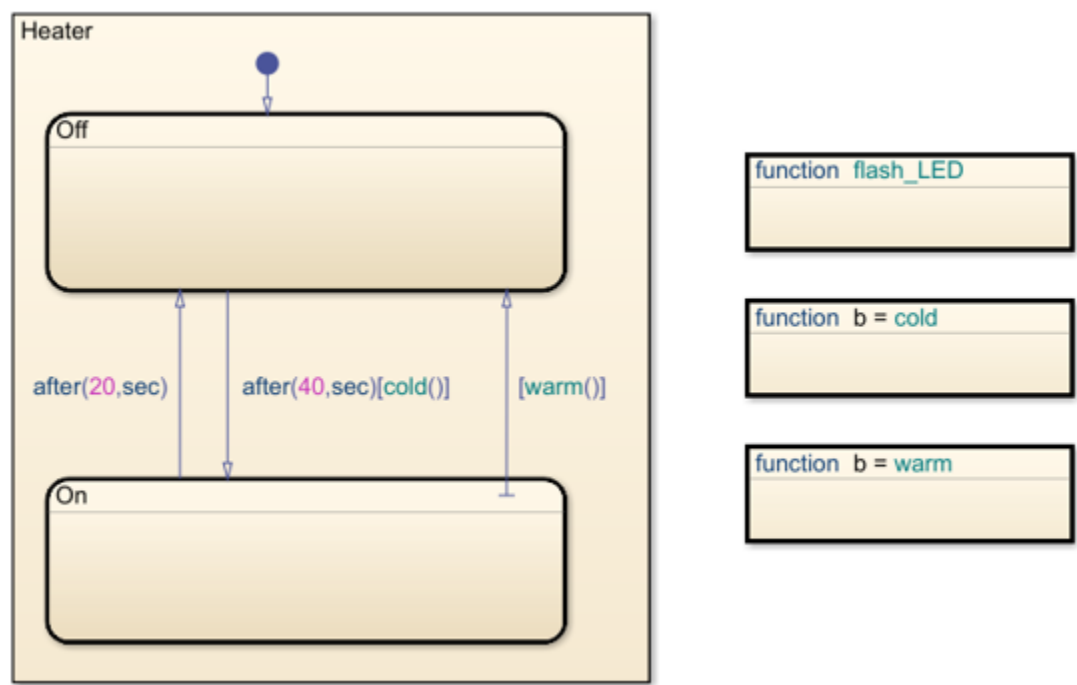
Bang-Bang Controller 图使用时序逻辑运算符 `after` 来实现以下目的：

- 当锅炉在开启和关闭之间切换时，调节 Bang-Bang 循环的时序。
- 根据锅炉的工作模式控制以不同速率闪烁的状态 LED。

定义锅炉行为和 LED 子系统行为的计时器彼此独立运行，并不阻断或中断控制器的仿真。

Bang-Bang 循环的时序

Bang-Bang Controller 图包含一对代表锅炉两种工作模式的子状态：`On` 和 `Off`。该图使用激活状态输出数据 `boiler` 来指示哪个子状态为激活状态。

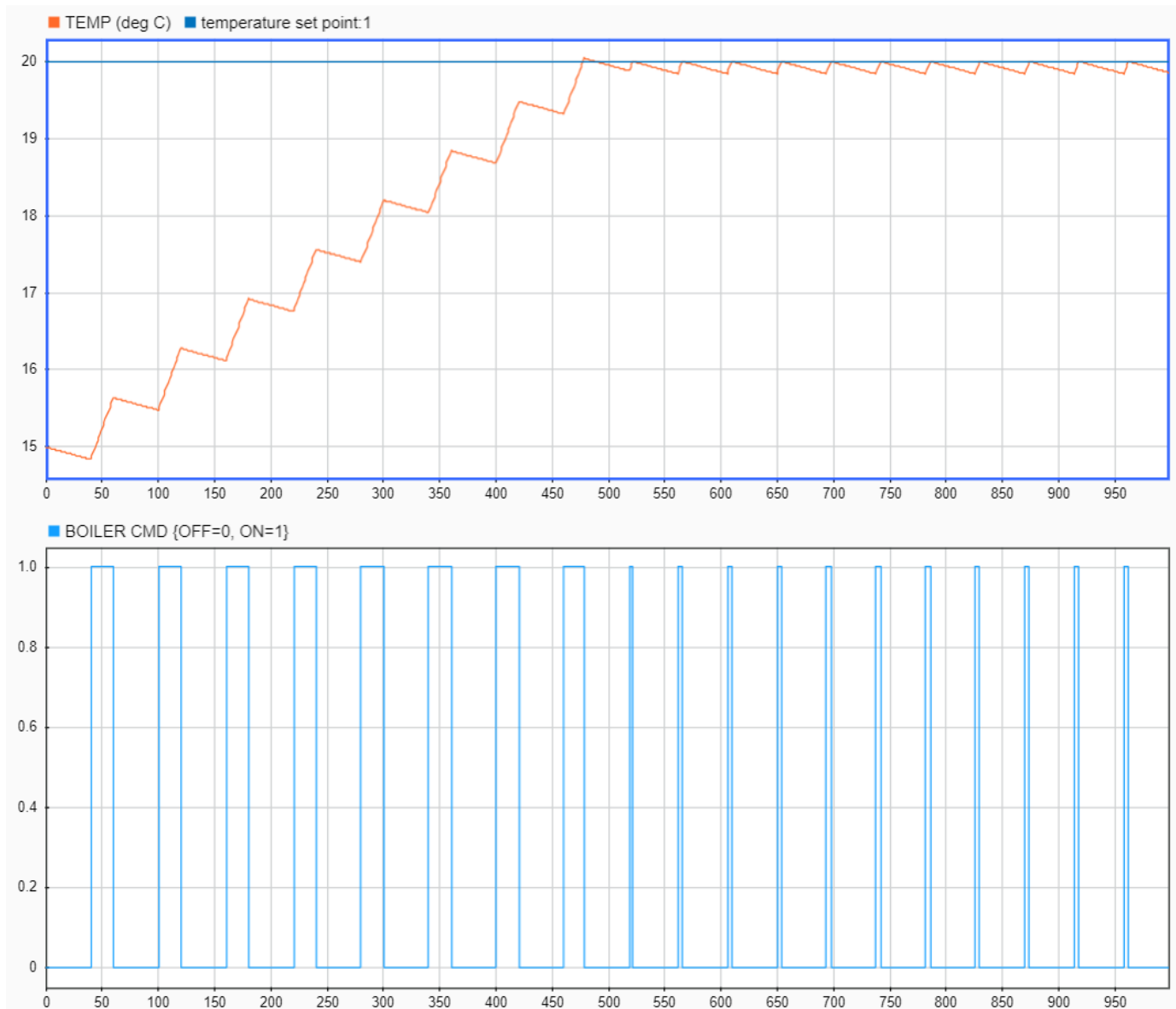


`On` 和 `Off` 子状态之间转移上的标签定义 Bang-Bang 控制器的行为。

转移	Label	说明
从 On 到 Off	<code>after(20,sec)</code>	在 <code>On</code> 状态下保持 20 秒后转移到 <code>Off</code> 状态。
从 Off 到 On	<code>after(40,sec)</code> <code>[cold()]</code>	当锅炉温度低于参考设定值时（即当图形函数 <code>cold()</code> 返回 <code>true</code> 时），在 <code>Off</code> 状态下至少保持 40 秒，然后转移到 <code>On</code> 状态。
从 On 到 Off	<code>[Heater.On.war</code> <code>m()]</code>	当锅炉温度等于或高于参考设定值时（即当图形函数 <code>Heater.On.warm()</code> 返回 <code>true</code> 时），转移到 <code>Off</code> 状态。

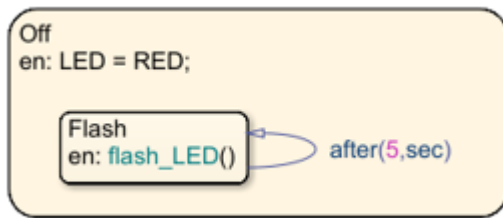
由于存在这些转移动作，Bang-Bang 循环的时序取决于锅炉的当前温度。在仿真开始时，锅炉较冷，控制器在 `Off` 状态下保持 40 秒，在 `On` 状态下保持 20 秒。在时间到达 `t = 478` 秒时，锅炉的温度达到参考

值。自这一时刻起，锅炉仅需补充在 **Off** 状态下的热损失。因而控制器在 **Off** 状态下保持 40 秒，在 **On** 状态下保持 4 秒。

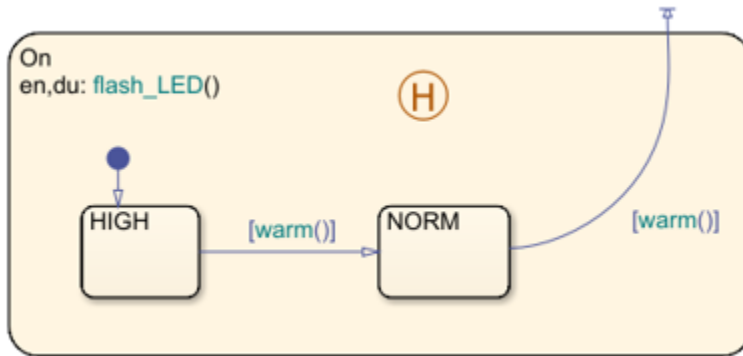


状态 LED 的时序

Off 状态包含子状态 **Flash**，其自环转移由动作 **after(5,sec)** 确定。由于这种转移，当 **Off** 状态被激活时，子状态将执行其 **entry** 动作并每隔 5 秒就调用一次图形函数 **flash_LED**。该函数使输出符号 **LED** 的值在 0 和 1 之间切换。

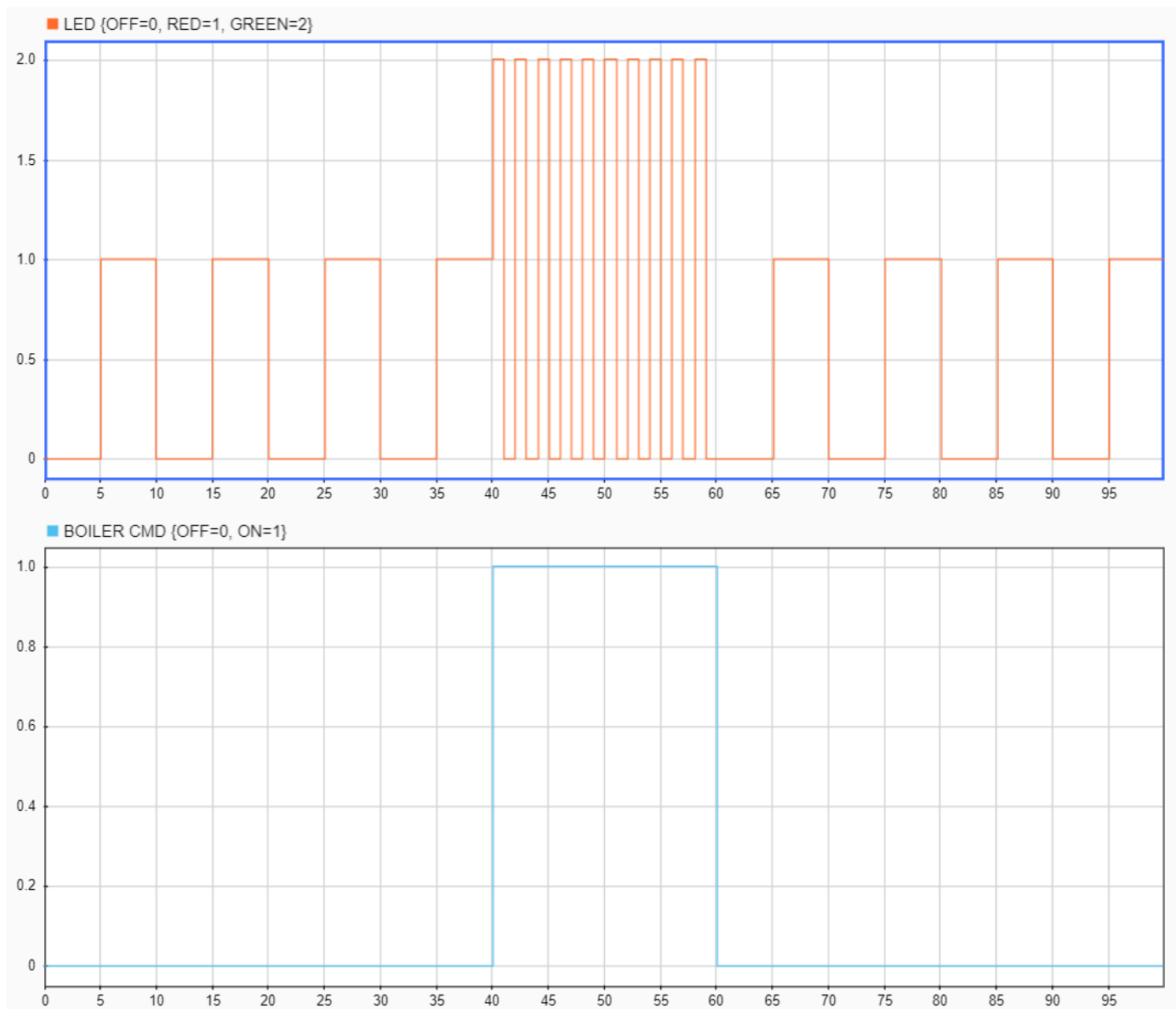


On 状态将图形函数 **flash_LED** 作为 **en,du** 类型的状态动作进行调用。当 **On** 状态被激活时，它会在仿真的每个时间步（本例中为每秒）调用该函数，使输出符号 **LED** 的值在 0 和 2 之间切换。



因此，状态 **LED** 的时序取决于锅炉的工作模式。例如：

- 从 $t = 0$ 秒到 $t = 40$ 秒，锅炉关闭，并且 **LED** 信号每隔 5 秒就在 0 和 1 之间交替一次。
- 从 $t = 40$ 秒到 $t = 60$ 秒，锅炉开启，并且 **LED** 信号每隔一秒就在 0 和 2 之间交替一次。
- 从 $t = 60$ 秒到 $t = 100$ 秒，锅炉再次关闭，并且 **LED** 信号每隔 5 秒就在 0 和 1 之间交替一次。



探索示例

使用其他时序逻辑来研究随着锅炉温度接近参考设定值，Bang-Bang 循环的时序如何变化。

1 输入调用 `elapsed` 和 `duration` 运算符的新状态动作。





- 在 On 状态下，让 `Timer1` 作为 On 状态被激活的时间长度：

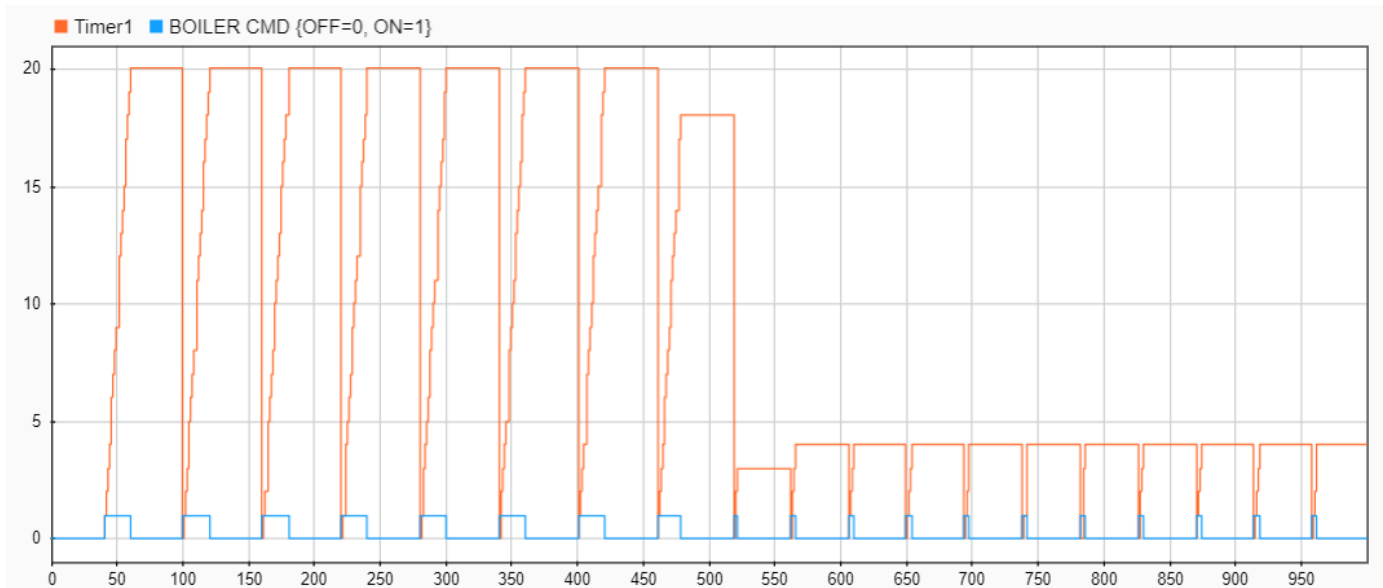
`en,du,ex: Timer1 = elapsed(sec)`

- 在 Off 状态下，让 `Timer2` 作为锅炉温度等于或高于参考设定值的时间长度：

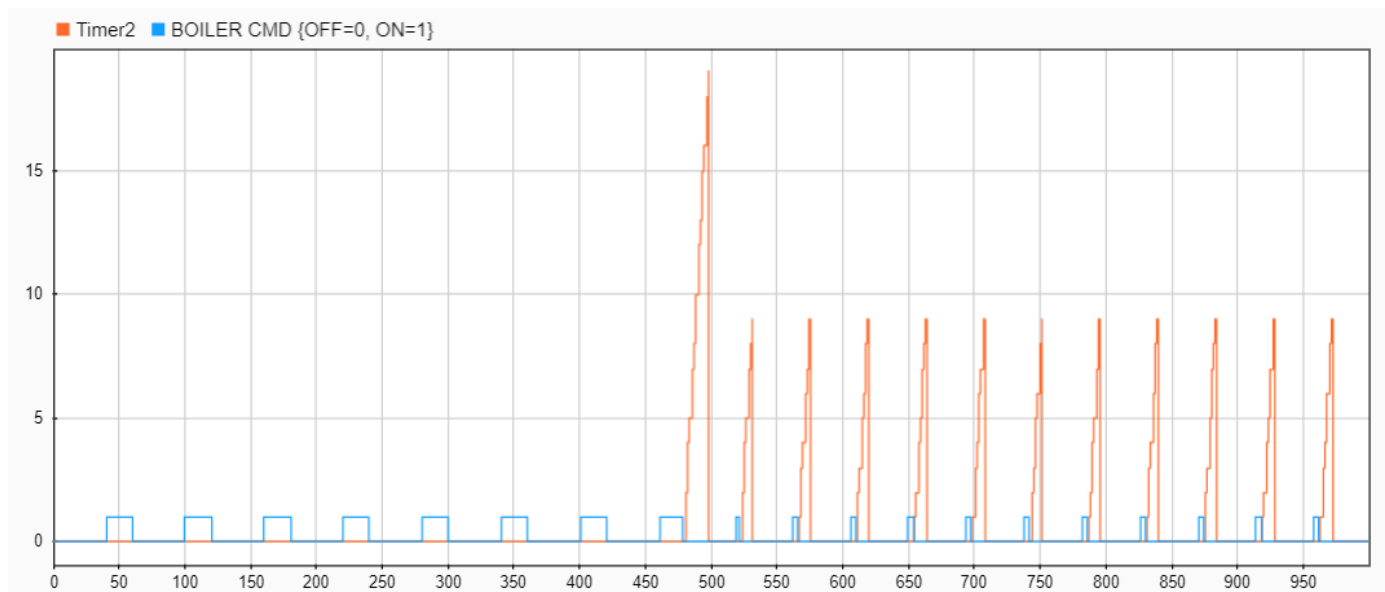
`en,du,ex: Timer2 = duration(temp>=reference)`


标签 `en,du,ex` 指示只要对应状态被激活，就会发生这些动作。

- 2 在 Symbols 窗格中, 点击 **Resolve Undefined Symbols** 。Stateflow Editor 将符号 **Timer1** 和 **Timer2** 解析为输出数据 。
- 3 为这些符号中的每个符号启用日志记录。在 Symbols 窗格中, 选择每个符号。在 Property Inspector 的 **Logging** 下, 选择 **Log signal data**。
 - **Timer1**
 - **Timer2**
- 4 在 **Simulation** 选项卡中, 点击 **Run** 。
- 5 在 **Simulation** 选项卡中的 **Review Results** 下, 点击 **Data Inspector** 。
- 6 在 Simulation Data Inspector 中, 在同一坐标区中显示信号 **boiler** 和 **Timer1**。绘图表明:
 - 通常情况下, 在锅炉冷时 Bang-Bang 循环的 **On** 阶段持续 20 秒, 在锅炉热时持续 4 秒。
 - 在锅炉第一次达到参考温度时, 循环会提前中断, 控制器在 **On** 状态仅保持 18 秒。
 - 在锅炉变热后, 第一个循环的时间比随后的循环时间略短, 在该循环中控制器在 **On** 状态仅保持了 3 秒。



- 7 在 Simulation Data Inspector 中, 在同一坐标区中显示信号 **boiler** 和 **Timer2**。绘图表明:
 - 在锅炉变热后, Bang-Bang 循环的 **Off** 阶段通常需要 9 秒才能冷却。
 - 锅炉第一次达到参考温度时, 用了超过两倍于 9 秒的时间 (19 秒) 来冷却。



较短的循环和较长的冷却时间是由于 **On** 状态内的子状态层次结构导致的。当锅炉首次达到参考温度时，从 **HIGH** 到 **NORM** 的转移会使控制器在一个额外的时间步内保持开启，导致锅炉温度超过正常值。而在后面的循环中，历史结点  导致 **On** 阶段以激活的 **NORM** 子状态开始。然后，控制器在锅炉达到参考温度后立即关闭，从而冷却锅炉。

另请参阅

after | duration | elapsed

相关示例

- “Model Bang-Bang Temperature Control System”

详细信息

- “使用动作定义图行为” (第 2-17 页)
- “使用时序逻辑控制图的执行”
- “通过定义图形函数重用逻辑模式”
- “历史结点”

其他学习工具

Stateflow 入门之旅

Stateflow 许可证附带三小时交互式培训课程

说明

为了帮助您快速学习 Stateflow 基础知识，Stateflow 入门之旅提供了自定进度的交互式教程。

完成 Stateflow 入门之旅后，您将能够使用 Stateflow 环境并根据真实示例构建 Stateflow 图。

为了循序渐进地讲授概念，Stateflow 入门之旅还安排了一些实操练习。提交任务后，您会收到自动评估和反馈。如果退出应用程序，您的进度将保存，因此您可以分多次完成培训。

Stateflow 入门之旅涵盖以下主题：

- 状态机可以：
- 创建状态图
- Stateflow 符号和数据
- 图动作
- 图执行
- 流程图
- Stateflow 中的函数
- 图层次结构

Stateflow 入门之旅帮助您在实践中练习从下列工程中学到的知识：

- Robotic Vacuum
- Robotic Vacuum Driving Modes

Training - Tasks

2.2 Running a Stateflow Chart

Task 1

At right is a Stateflow chart that models a traffic light. The light switches between red, yellow, and green based on a timer. If a fault occurs, the light switches to flashing red.

In Stateflow, the Run button simulates the chart.

TASK

Click Run to simulate the chart. The states and transitions will highlight blue as they become active.

Click Submit to assess your model, then click Next task to advance.

Hint | See Solution | Reset

Submit Next task

Task 2

Task 3

Task 4

Further Practice

Chart

StateflowOnramp ▶ Chart

There are two operating modes.

Normal: The traffic light rotates between red, yellow, and green based on a timer.

Fault: The traffic light turns the red lamp off and on based on a timer.

Symbols


TYPE	NAME	VALUE	PORT
	red	1	1
	yellow		2
	fault		1
	green	0	3

Training - Assessment

Requirements

Are the chart states, transitions, and Symbols all correct?

打开 Stateflow 入门之旅

- 在 Simulink Start Page 上, 点击 Stateflow Onramp 按钮  Stateflow Onramp。
- 打开一个 Stateflow 图, 在快速访问工具栏上, 点击 **Help > Learn Stateflow**。
- 在 MATLAB 命令提示符下, 输入 `learning.simulink.launchOnramp('stateflow')`。

另请参阅

主题

- “对有限状态机建模” (第 2-2 页)
- “构造并运行 Stateflow 图” (第 2-9 页)
- “使用动作定义图行为” (第 2-17 页)
- “创建层次结构来管理复杂系统” (第 2-22 页)

在 R2019b 中推出

