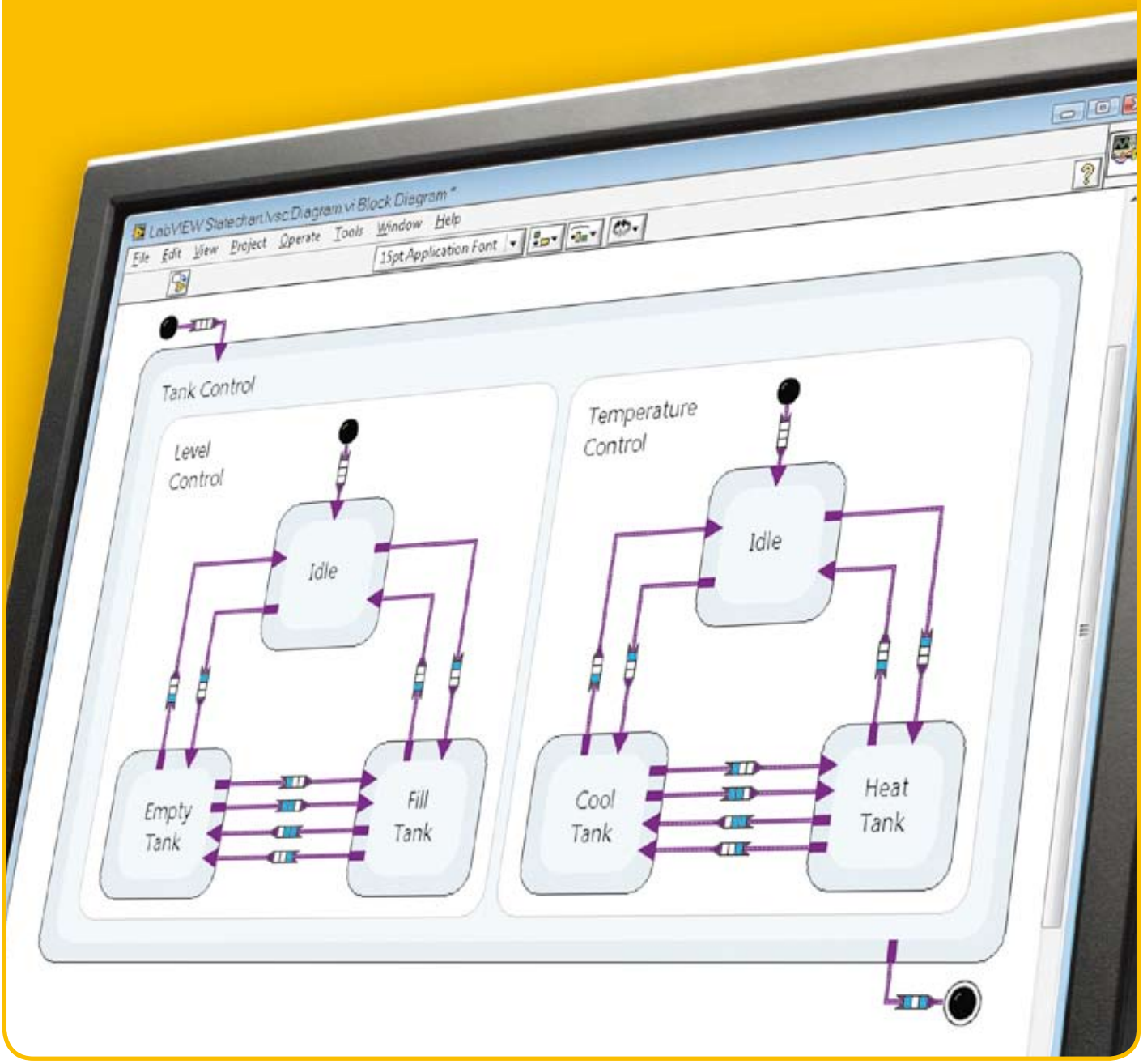
LabVIEW开发技术丛书

状 态 图 编 程 指 南



NATIONAL INSTRUMENTS

目录

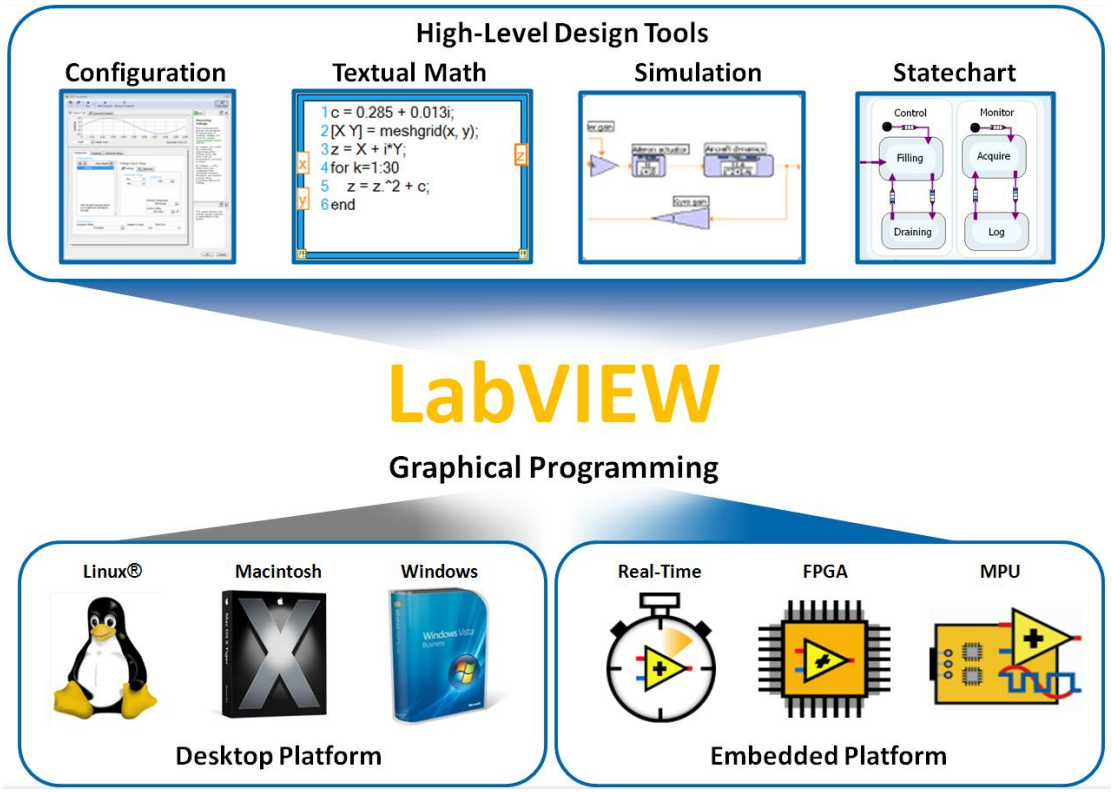
采用 LabVIEW 状态图模块开发应用程序

概览

本文档将解释状态图的定义，并说明 LabVIEW 状态图模块的基础。

引言

该模块在 LabVIEW 中增加了创建状态图的功能，以开发基于事件的控制与测试系统。状态图编程模块进一步补充了现有 LabVIEW 提供的数据流、文本数学、动态系统建模、基于配置的开发模型支持。你可以根据你的应用程序需求，选择合适的模型或模型组合来开发系统。



NI 的图形化系统设计平台中将 LabVIEW 的编程模块与现成的台式嵌入式控制器、测量 I/O 结合在一起。这样，你就拥有了一种集成的开发工具链，以进行系统的设计、原型化和实现。LabVIEW 状态图提供了一种高级的设计工具，具有强大的可扩展性，包含多种编程概念如嵌套、并发和事件等。因为状态图提供了一种系统级视图，所以可以将 LabVIEW 状态图用作一种可执行的应用程序。状态图编程模式特别适用于开发需要响应多种事件的复杂系统，例如嵌入式体统和通信系统。采用 LabVIEW 状态图模块，你可以将设计部署到各种硬件平台上——包括从台式 PC 机到 FPGA 的硬件平台。

注意：要获得完整的 LabVIEW 状态图模块文档，请参考配送文档。

状态图的历史

状态图是在 20 世纪 80 年代由 Weizmann 科学研究所的 David Harel 发明的。根据 Harel 所述，状态图的目的就是“扩展传统的状态转移图……以包括嵌套、并发和通信等概念。” Harel 在帮助设计一个复杂的航空系统的时候发明了状态图，想必就是为了弥补该航空系统的不足而找到了一些现成的工具。20 世纪 90 年代，UML 规范(Unified Modeling Language，统一建模语言) 将状态图归入为行为图，并广泛应用于嵌入式系统的建模。

状态图如何工作

要理解状态图(statechart)，最好先了解经典状态图(state diagram)，然后再了解嵌套、并发、事件等概念。经典状态图由两个主要结构组成：状态和状态转移。图 2 中的经典状态图描述了一个简单的饮料贩卖机，其中有 5 个状态和 7 个描述状态机运行方式的状态转移。机器从“空闲”状态开始，当投入硬币后，将转移到“硬币计数”状态。该经典状态图中还显示了贩卖机等待用户选择、送出饮料和找零这三个阶段的状态和转移。

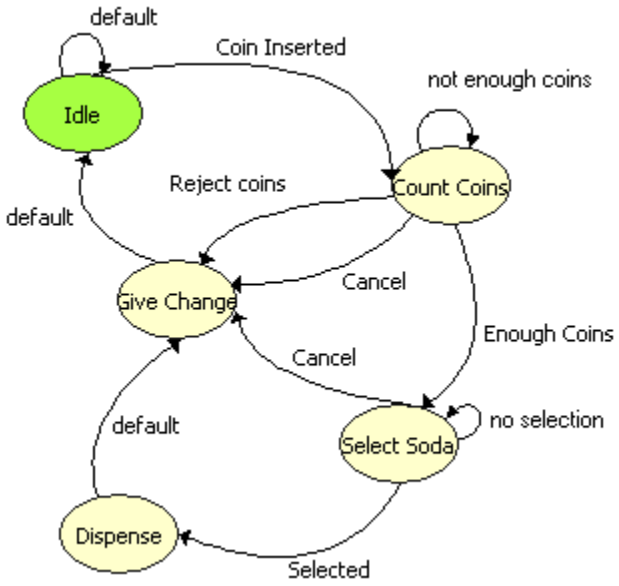
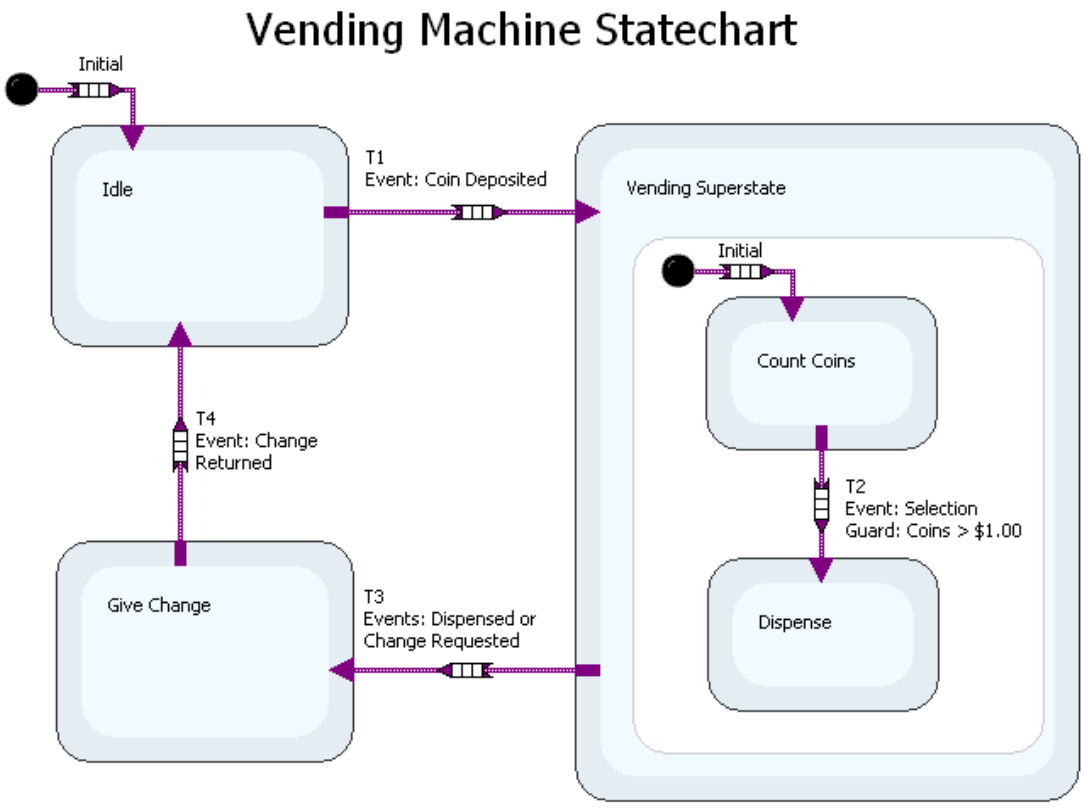
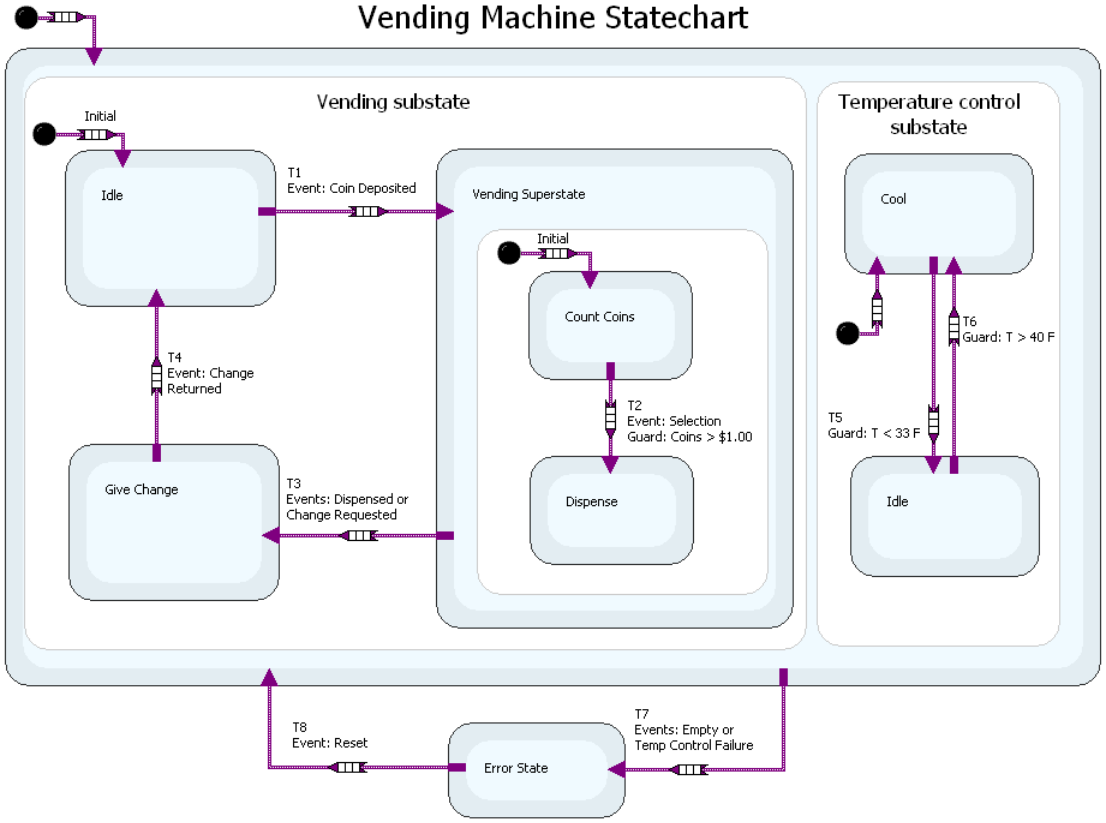


图 3 中的状态图描述了同一个饮料贩卖机的行为。请注意嵌套和事件怎样实现了状态和状态转移数目的减少。在状态图中，可以将“硬币计数”和“送出饮料”这两个状态组合在一个超状态中。你只需要在这两个状态中的任一状态和“找零”状态之间定义一个转移(T3)。T3 状态转移可以响应 3 个事件：饮料送出、请求找零或硬币弹出。另外，在经典状态图中，可以在状态转移 T2 中引入一个“警戒”条件，以省去“选择饮料”状态。要触发转移，警戒条件必须为 true。如果警戒条件为 false，则事件将被忽略，不触发转移。



这时，我们可以通过在贩卖机的软件中增加一个温度控制元件，来扩展该状态图，并说明并发的概念。图 4 中显示了如何将饮料贩卖逻辑与温度控制逻辑封装到一个与状态中。与状态所描述的系统能在同一时间处于两个彼此独立的状态中。T7 转移显示了状态图怎样定义两个子状态图的退出动作。



除了嵌套和并发外，状态图的其他一些特点对复杂系统的设计来说也非常有用。状态图中的“历史”允许一个超状态来“记录”它上一次的激活子状态。例如，假设某个超状态描述了一种机器，该机器在注入某种物质后对其加热。在机器注入物质的时候，暂停事件会暂停机器的注入操作；当恢复事件发生时，机器则会继续执行刚才的注入操作。

使用 LabVIEW 状态图

采用 LabVIEW 状态图模块，你可以采用状态图来设计软件模块，并采用数据流图形编程的方法来定义状态行为和转移逻辑。采用 LabVIEW 项目管理器(Project Explorer)将状态图完全集成到 LabVIEW 环境中。每个 LabVIEW 状态图都有好几个组件，可以用来配置设计的内容。图 5 中显示了一个示例状态图，记作 LVStatechart 1.lvsc。你可以创建一些触发来响应转移和状态反应，并编辑状态图中所使用的输入、输出数据变量的列表。

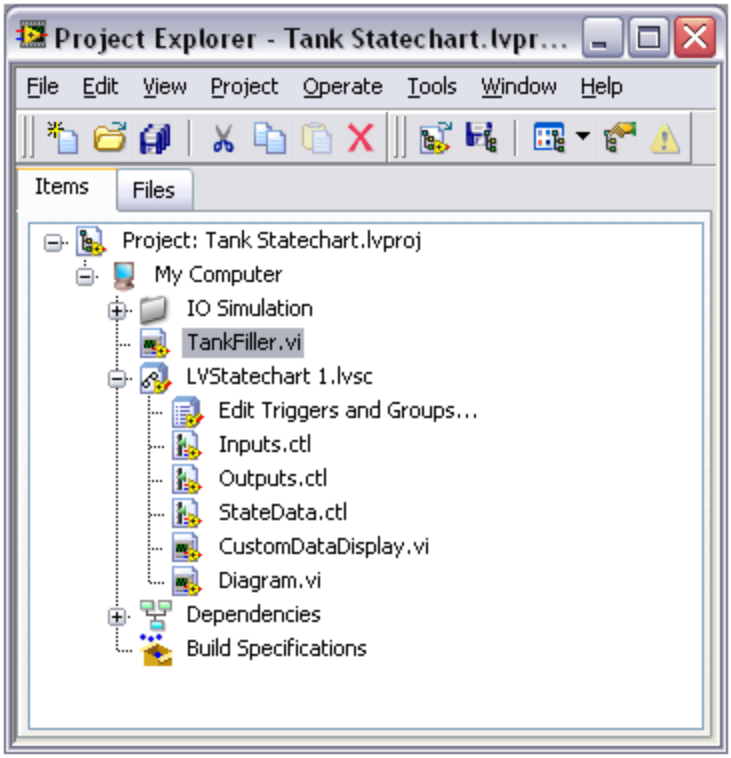
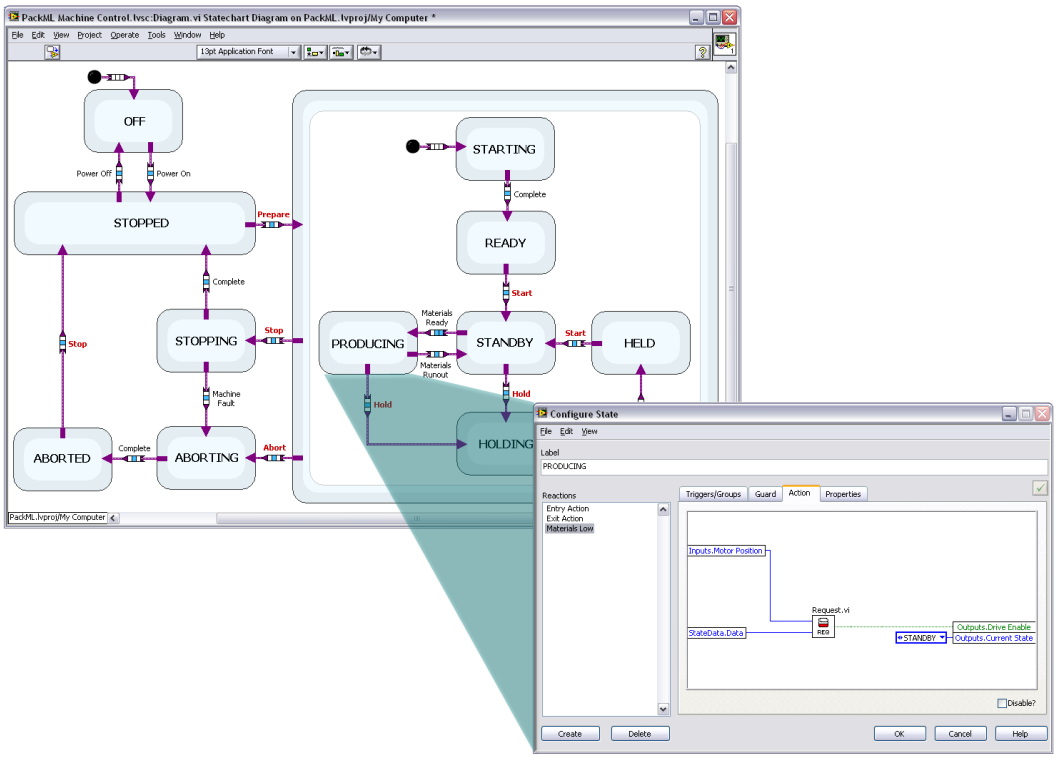
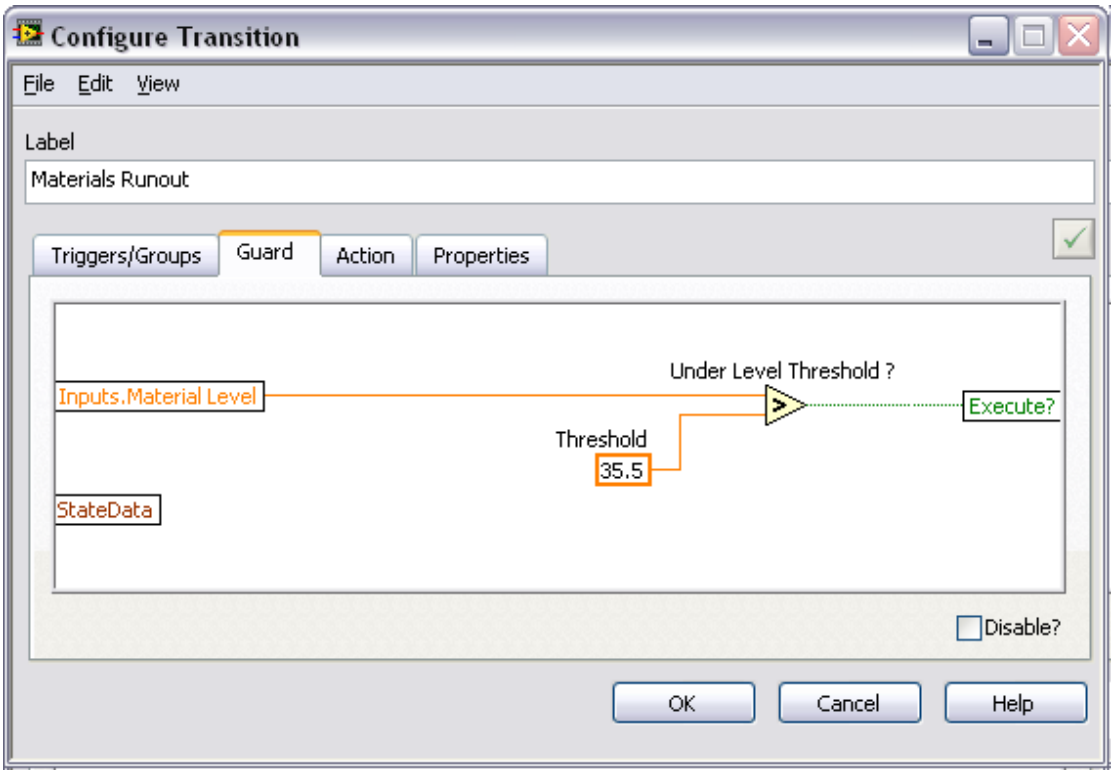


Diagram.vi 文件中包含了真实的状态图。在该图中，你可以创建系统的各个状态以及状态间的转移。状态图的一个主要优点在于可以直观地表达系统的行为，从而对软件进行自动注释。图 6 中显示了一个打包机器的状态图。从中可以很容易地看到机器的各个不同状态以及状态之间的转移。



状态图在描述被动反应系统的时候非常有用。每个状态都可以具有多个反应动作，以对应各种来自硬件设备或用户界面的触发或事件。反应动作可以采用 LabVIEW 的图形化编程实现。当系统处于“生产”状态而且触发事件“材料量低”发生的时候，所执行的代码如图 6 所示。触发器还可以导致两个状态之间的转移。触发转移的另一种方法是使用 LabVIEW 中计算警戒条件的代码。警戒条件描述了执行转移所必须满足的条件。图 7 中显示了“材料用完”转移逻辑的警戒条件代码。LabVIEW 代码确保了材料量的水平线必须低于 35.5，从而来执行从“生产”状态到“等待”状态的转移。



为了满足不同应用的需求，LabVIEW 状态图为两种执行模式生产代码：同步模式和异步模式。在同步模式中，状态图以相同的速率，不同的状态来描述控制器对 I/O 输入的响应行为。这种模式可以应用到嵌入式控制系统中，如引擎控制单元(ECU)、运动控制器、环境控制器等。异步模式则是用来实现具有外部事件的应用。在编程实现人机接面(HMI)和模型化时间驱动的系统和算法中，这种模式非常有用。

为状态图选择了合适的执行模式后，可以采用模块化 subVI 或函数调用的形式生成可执行代码。接着可以如图 7 所示，从 LabVIEW 数据流图中调用该 subVI。通过 LabVIEW 的加亮功能以及标准的调试工具如断点、探针(变量观察窗口)、单步执行等，来可视化调试状态图。

你可以为各种硬件平台生成状态图代码，包括桌面系统、人机接口(HMI)、可编程自动化控制器(PAC)如 NI CompactRIO 和 PXI、NI 硬件中的 FPGA(现场可编程门阵列)、任何 32 位的微处理器等等。LabVIEW 状态图模块可以在很多硬件平台中配置状态图，因而成为嵌入式系统开发和实现的高级设计工具。你还可以利用状态图和 LabVIEW 的控制设计与仿真模块，采用动态系统仿真对混合系统进行建模和评估。

状态图的优势

使用 LabVIEW 的状态图来设计系统，对软件开发人员来说有多个好处。状态图提供了一种系统级视图，包含了系统的每个可能状态，所以能够描述系统或应用的完整功能。在状态图中你必须考虑软件响应的每一种可能，所以状态图可以帮助降低软件“挂起”以及其它意外事件的可能性。如本篇已经讨论过的，状态图编程模型对于反应系统(这些系统的特点就是如何响应输入)尤其有用。所设计的系统可以根据事件的任意组合，灵活地处理多种状态反应和转移。在软件自我注释方面，状态图同图形化数据流编程比较类似，并且还可以促进开发人员之间的知识交流。设计小组中的新成员可以通过状态图迅速领会系统精髓。

总结

状态图为处理复杂的应用开发提供了一种完善的方法。状态图对于事件驱动的应用程序开发来说尤其有帮助，例如复杂的用户界面以及用于实现动态系统控制器、机器控制逻辑、数字通信协议等应用的高级状态机。采用新型的 LabVIEW 状态图模块，可以实现快速开发和 LabVIEW 平台的严密硬件集成。你可以将状态图增加到工具箱中，来编程实现复杂的应用程序。

LabVIEW 状态图模块中 UML 专用术语

概览

本文档将介绍一些与状态图相关的组成元素与专用术语，以及如何采用 NI LabVIEW 状态图模块来实现状态图。

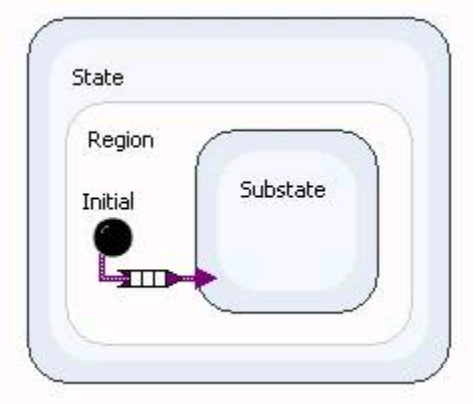
状态图介绍

David Harel 为了克服以前的经典状态机(state machine)描述方法的缺点，在状态机中增加了层次结构、并发和通信等概念，设计了状态图(statechart diagram)。UML 规范(Unified Modeling Language，统一建模语言)中将状态图归入为行为图。采用 NI LabVIEW 状态图模块，你可以使用状态图来创建应用程序。你可以利用状态图所提供的抽象功能，有效地开发出复杂的应用程序；同时使用 LabVIEW 来实现桌面系统、实时、FPGA 和嵌入式等对象上的应用。

状态图由**域(region)**、**状态(state)**、**伪状态(pseudostate)**、**转换(transition)**和**连接器(connector)**组成。LabVIEW 中已经集成了这些工具，允许用户在开发状态图时使用。

域

域是指包含**状态**的区域。顶层状态图是一个包含了所有状态的域。另外，你还可以在某个状态中创建域：即利用层次式设计的方法，在某个状态的内部创建其他状态。下图中描述了这种层次式设计功能：在一个状态的内部，通过域创建了一个子状态。每个域中都必须包含一个**初始**伪状态。



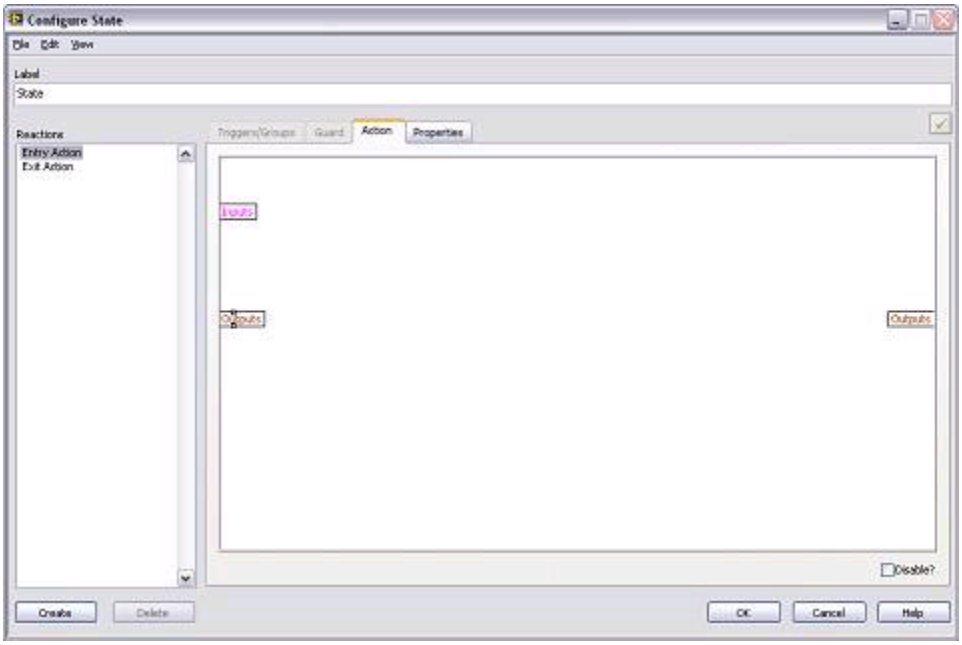
状态

状态是指状态图所能存在的某个阶段。状态必须位于域中，并且至少拥有一个进入的转换。

状态的进入和退出动作

每个状态都有一个相关的**进入**和**退出**动作。**进入动作**是指进入某个状态时所执行的 LabVIEW 代码。**退出动作**是指离开某个状态时(在转换到下一个状态之前)所执行的 LabVIEW 代码。每个状态都只能有一个进入和退出动作，而且这两个都是可选的。每次**进入**或**退出**某个状态时，都会执行进入与/或退出动作。

可以通过 Configure State 对话框来访问该代码。



状态的静态反应

可以进一步对状态进行配置，使之具有**静态反应**。静态反应是指状态没有执行任何进入或转出转换

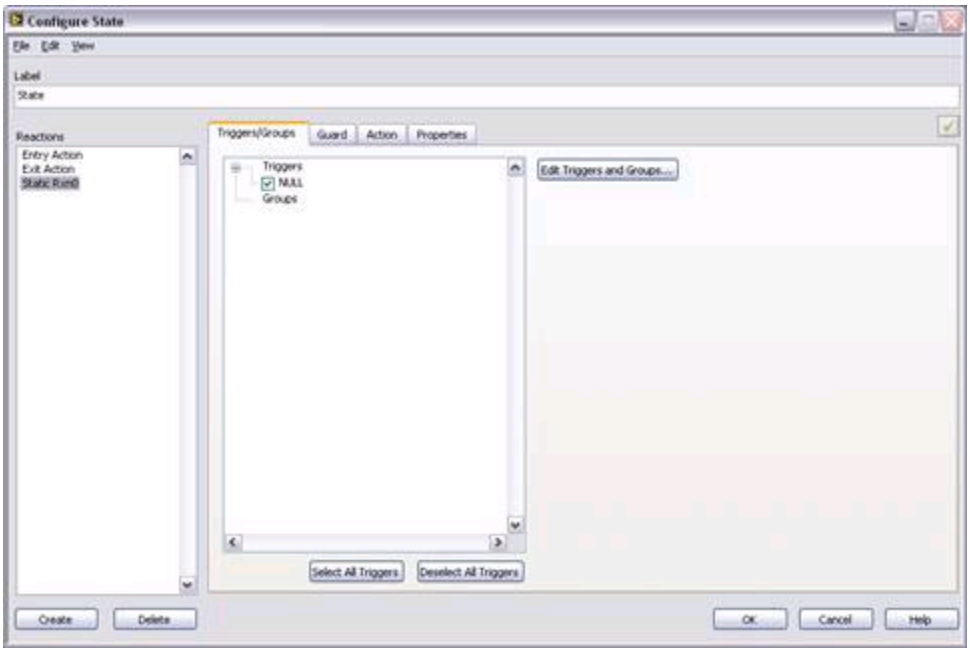
时所执行的动作。一个状态可以有多个静态反应，状态图的每次迭代中可能会执行这些静态反应。
Each static reaction comprises three components – trigger, guard, and action.
每个静态反应都由三个部分组成 – 触发器、紧戒条件和动作。

触发器是指触发状态图执行的事件或信号。**异步**状态图只有接收到触发器后才会执行 – 例如，按钮或其它用户界面交互可以产生一个触发器。触发器的值传递到状态图中，然后状态图基于触发器再执行相应动作。在**同步**状态图中，触发器则周期地自动传递到状态图中。触发器的默认值是 NULL。

监护条件是指在执行状态动作之前所测试的一段代码。如果监护条件为真，则将执行动作代码；如果监护条件为假，则不执行。

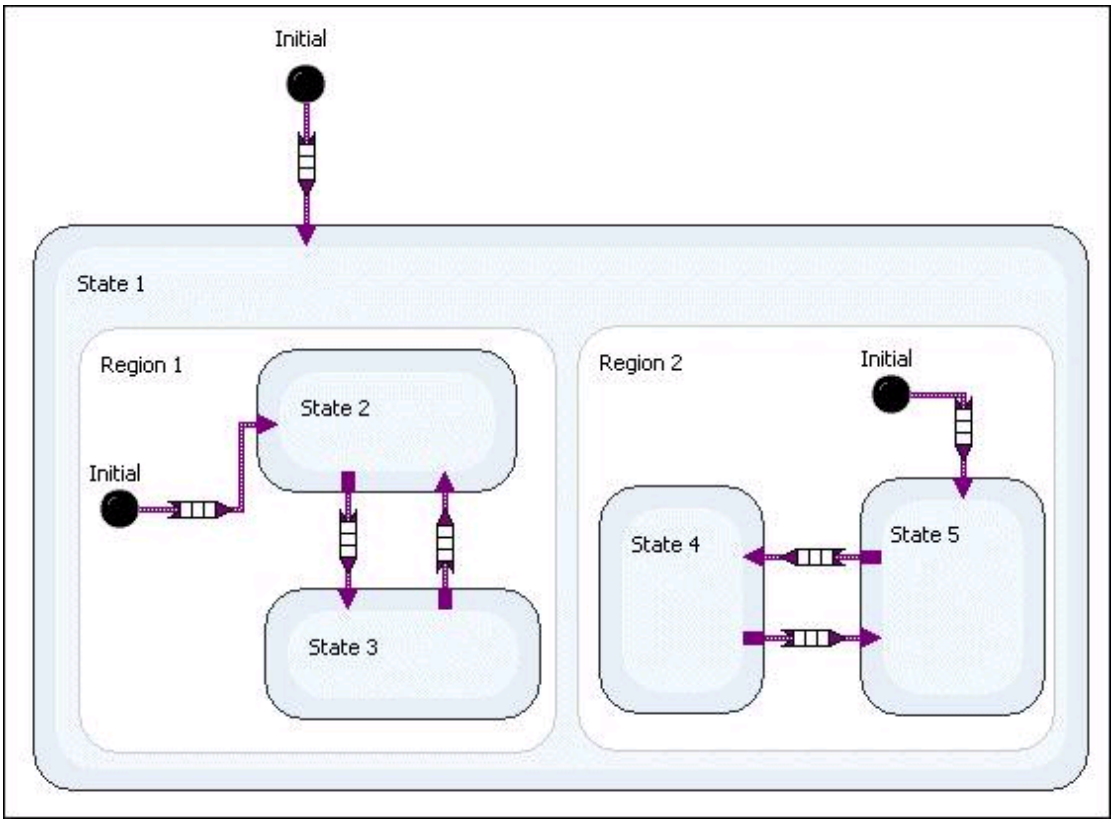
如果状态图接收到一个触发器(该触发器将由某个特定的静态反应来处理)并且监护条件的值为真，则将执行该动作代码。**动作**是指完成预期状态逻辑的 LabVIEW 代码，可以是输入，内部状态信息的读取以及相应的输出更改。

你可以通过 Configure State 对话框，新建一个反应动作来创建这种静态反应。一旦新建了一个反应动作，你就可以将它与触发器相关联并设计监护条件和动作代码。只有静态反应才能配置有触发器和监护条件。



正交域和并发

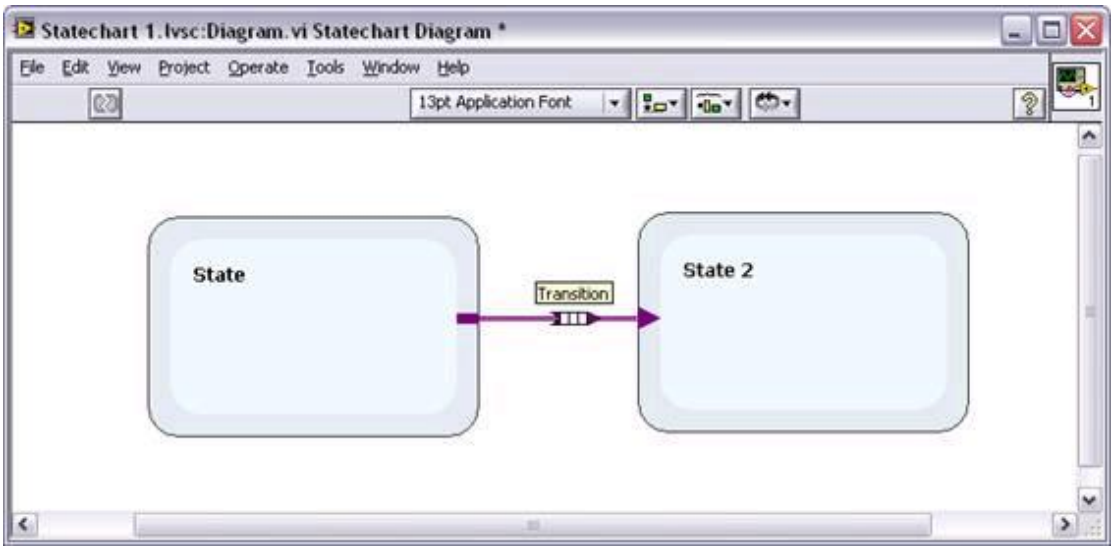
当状态具有两个或两个以上的域时，这些域就称为是正交的。下图中的域 1 和域 2 就是正交的。



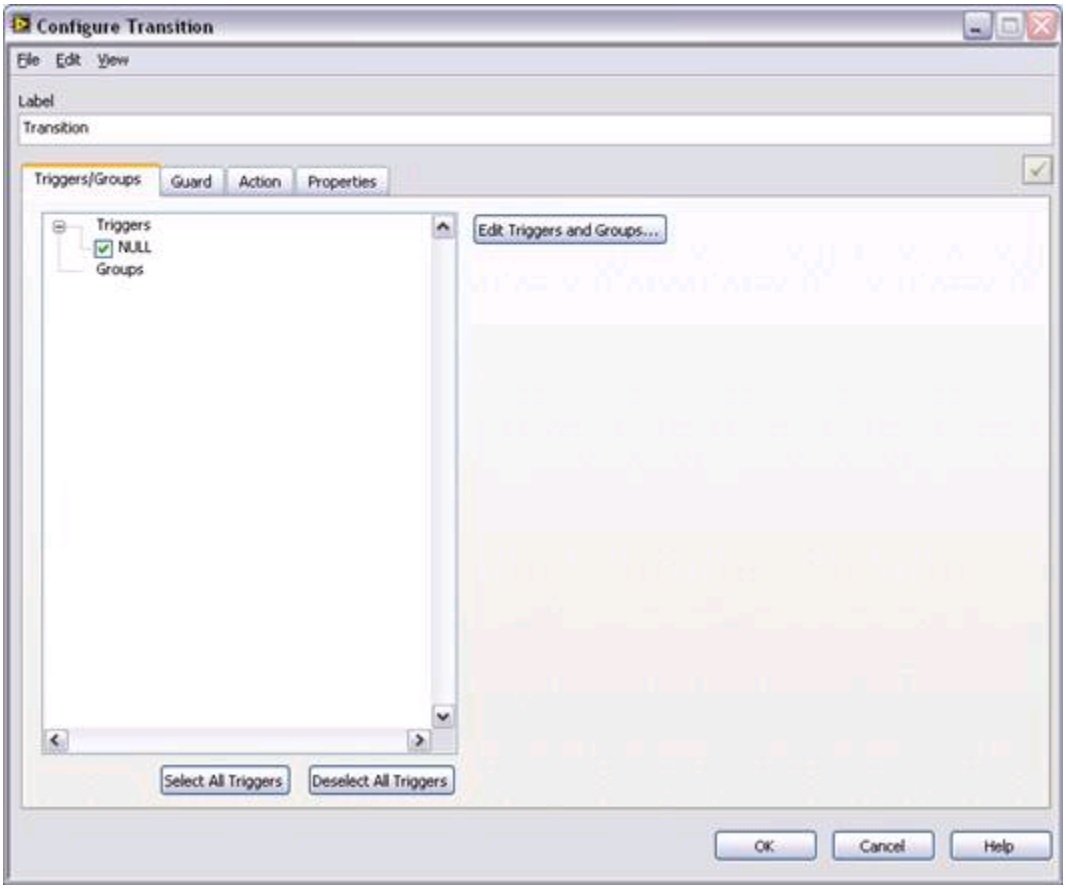
正交域中的子状态是并发的，也就是说当超状态都处于激活状态时，状态图在每次迭代中都可以进入每个正交域中的某个子状态。并发与并行是不一样的。在状态图的每次迭代中，并发的子状态是轮流被激活的，而并行子状态则是同时被激活的。LabVIEW 状态图模块不支持并行的状态激活。

转换的描述

转换定义了状态图在两个状态之间的转换条件。



转换由**端口**和**转换节点**构成。端口是指状态之间的连接点，而转换节点则基于触发器、监护条件和动作定义了转换的行为。用户可以通过 Configure Transition 对话框来配置转换节点。



转换中的**触发器**、**监护条件**和**动作**同状态中定义的触发器、监护条件和动作是一样的。触发器会触发转换的发生；如果监护条件为真，则将执行动作，而状态图也会转换到下一个状态。如果监护条件不为真，则不会执行动作代码，状态图也不会转到转换所指向的下一个状态。

伪状态和连接器

伪状态

伪状态是一种状态图对象，表示一种状态。LabVIEW 状态图模块中包括以下几种伪状态：

初始状态 – 是指进入域时首先出现的状态。每个域中都必须有一个初始状态。

终止状态 – 是指域中的最后一个状态，结束域中所有状态操作。

浅度历史 – 当状态图离开域然后再返回时，状态图重新进入在它退出域之前的最高一级的活动子状态。

深度历史 – 当状态图离开域然后再返回时，状态图重新进入在它退出域之前的最低一级的活动子状态。

连接器

连接器是一种状态图对象，将多个转换片段连接起来。LabVIEW 状态图模块包含以下几种连接器：

叉形 – 将一个转换片段分开成多个片段

合并 – 将多个转换片段合并到一个片段

连接 – 将多个转换片段连接起来

状态图迭代序列

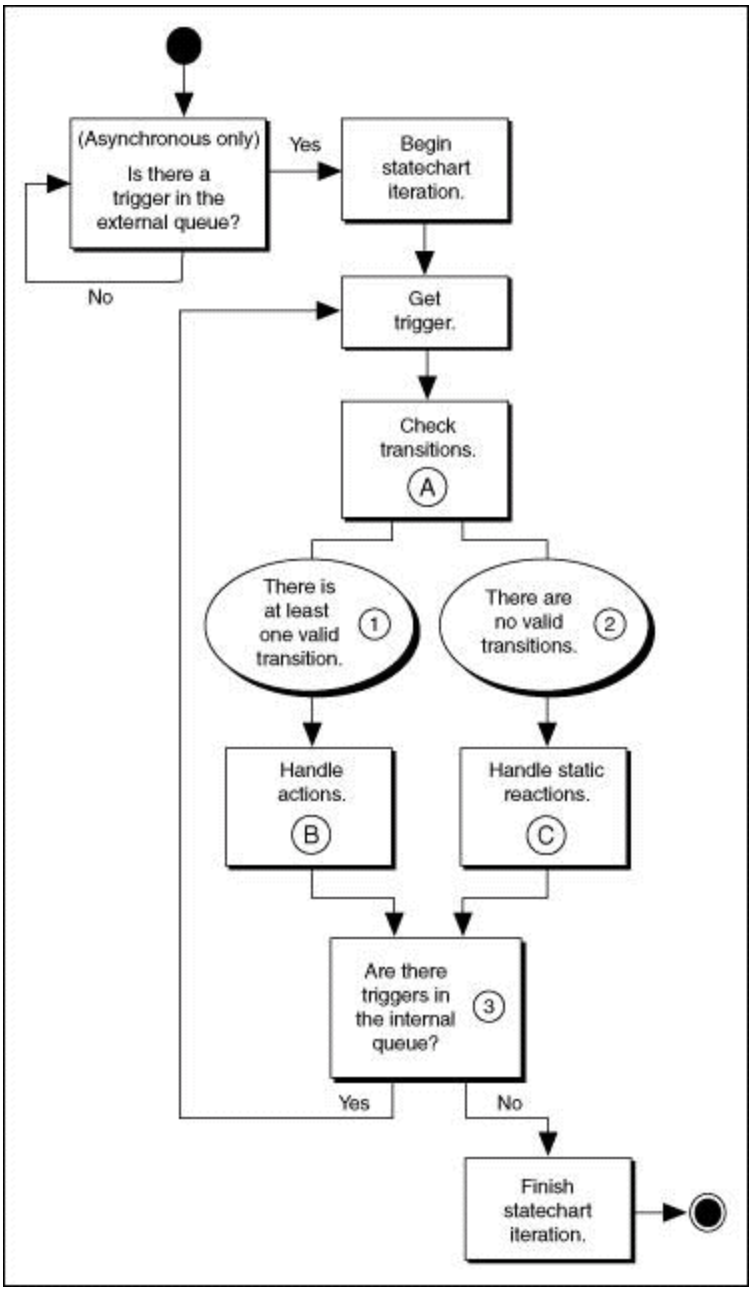
状态图的第一次迭代执行以下两步：

1. 转移到由顶级的初始伪状态所指定的状态中。
2. 执行该第一个状态以及任何指定子状态的进入动作。

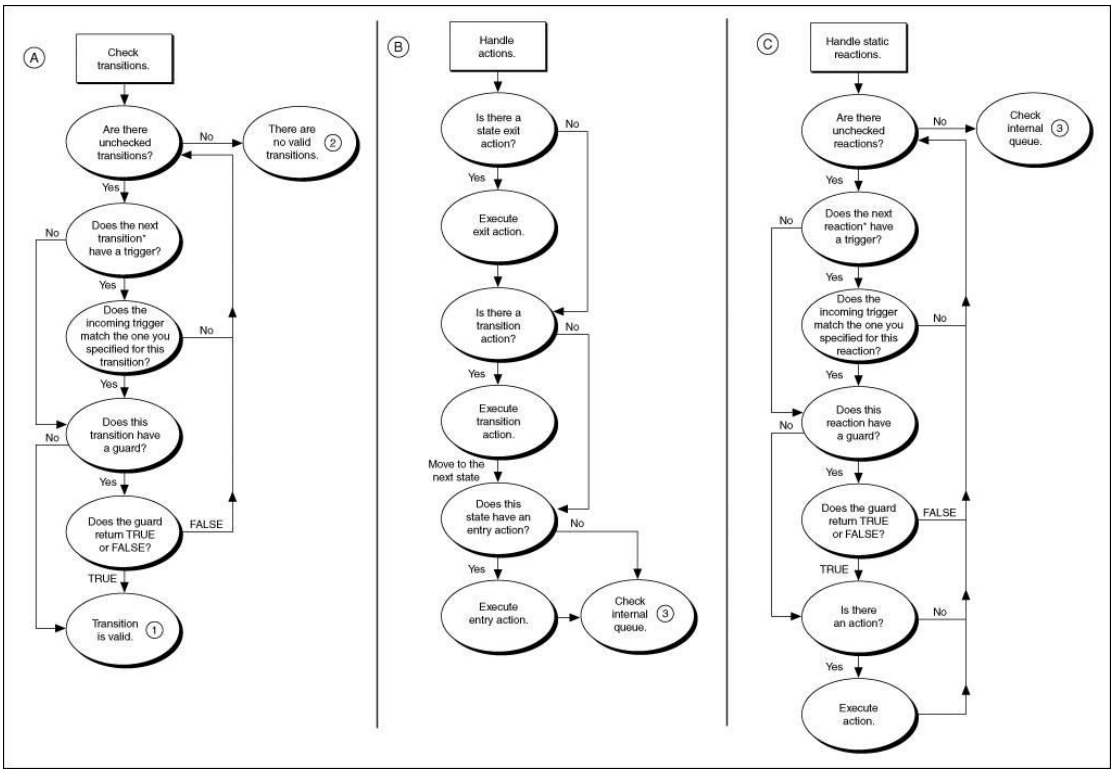
然后，当状态图接收到某个触发器后，执行以下动作：

1. 响应触发器
2. 检验内部触发器队列
3. 响应队列中的任何触发器
4. 再次检验内部触发器队列

下图显示了一个异步状态图的完整迭代过程。同步状态图与之类似，只不过它们不接收外部触发器。



下图详细描述了上图中的 A、B、C 过程。标注 1、2、3 的项是这些过程的最终结果。


A
Check transitions.
Are there unchecked transitions?
No
There are no valid transitions. 2
Yes
No
Does the next transition* have a trigger?
Yes
Does the incoming trigger match the one you specified for this transition?
No
Yes
No
Does this transition have a guard?
Yes
Does the guard return TRUE or FALSE?
FALSE
TRUE
Transition is valid. 1
B
Handle actions.
Is there a state exit action?
No
Yes
Execute exit action.
Is there a transition action?
No
Yes
Execute transition action.
Move to the next state
Does this state have an entry action?
No
Yes
Execute entry action.
Check internal queue. 3
C
Handle static reactions.
Are there unchecked reactions?
No
Check internal queue. 3
Yes
No
Does the next reaction* have a trigger?
Yes
Does the incoming trigger match the one you specified for this reaction?
No
Yes
No
Does this reaction have a guard?
Yes
Does the guard return TRUE or FALSE?
FALSE
TRUE
Is there an action?
No
Yes
Execute action.

如何对 LabVIEW 状态图应用程序进行调试

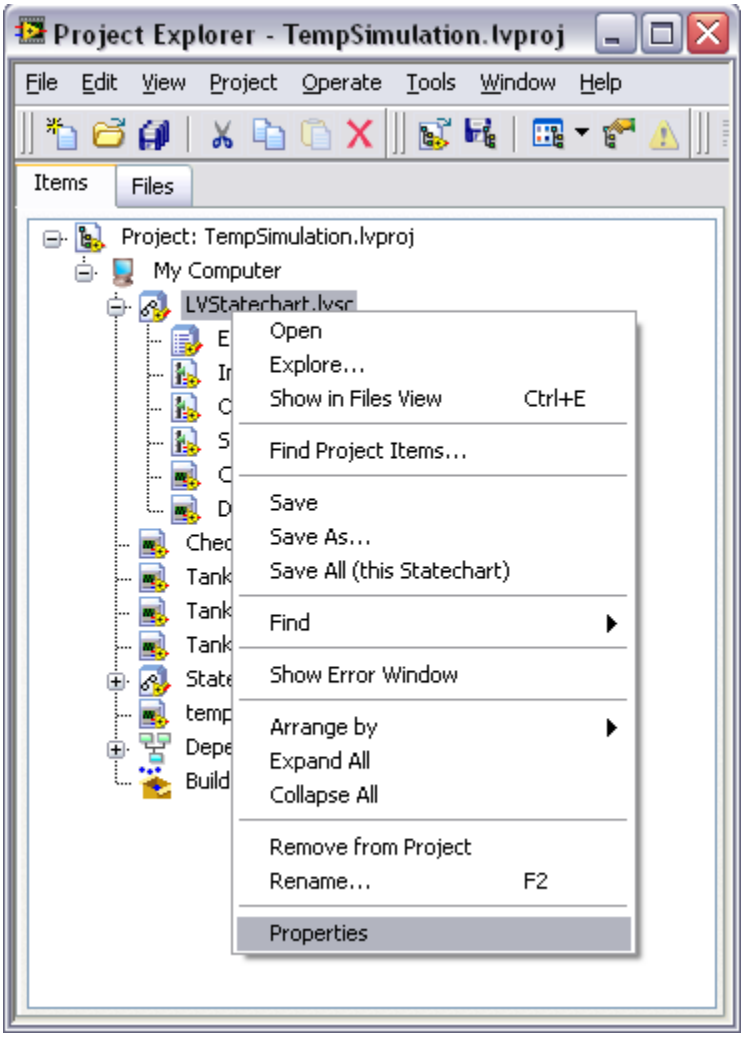
概览

任何应用程序在开发阶段都不可避免地包含一些缺陷。为了纠正这些软件缺陷，开发人员应当使用一些功能强大的调试工具来提高工作效率，从而深入地研究应用程序的具体运行情况。而 NI LabVIEW 状态图模块可以帮助开发人员有效地调试他们的程序。

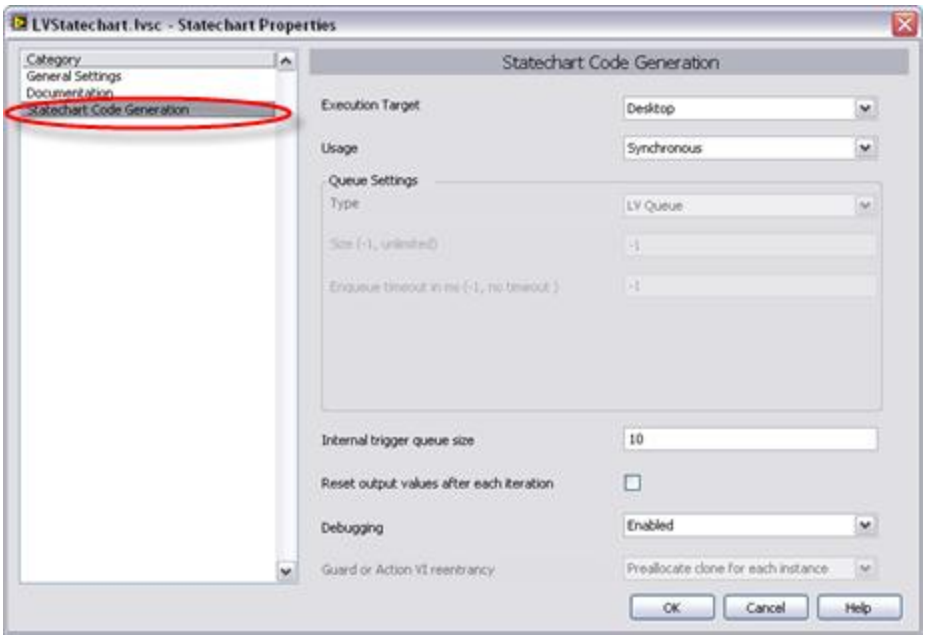
激活状态图调试功能

调试状态图的第一步：要确保已经激活了调试功能。

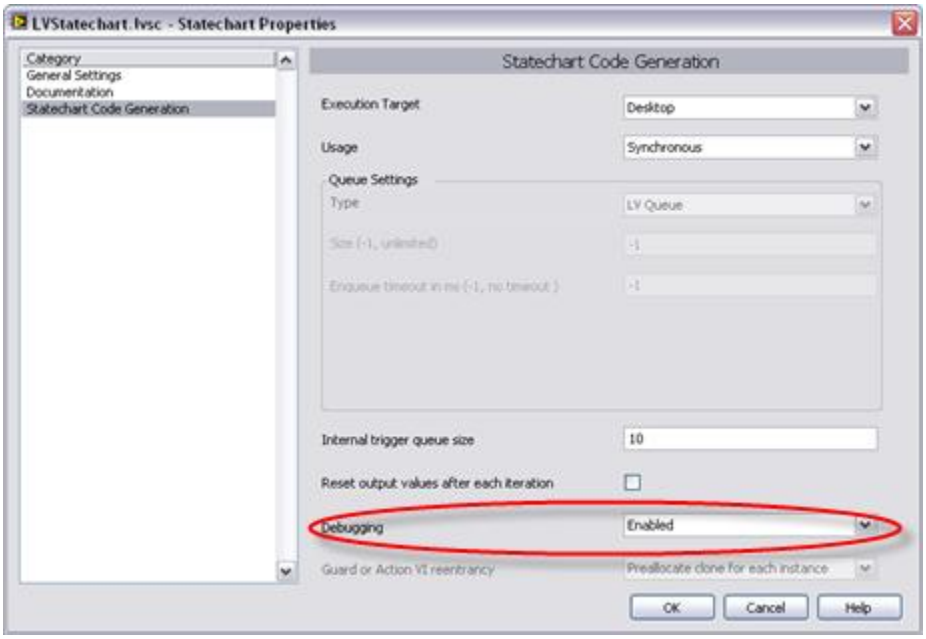
1. 从 Project Explorer 窗口中，右击状态图，并选择 Properties.



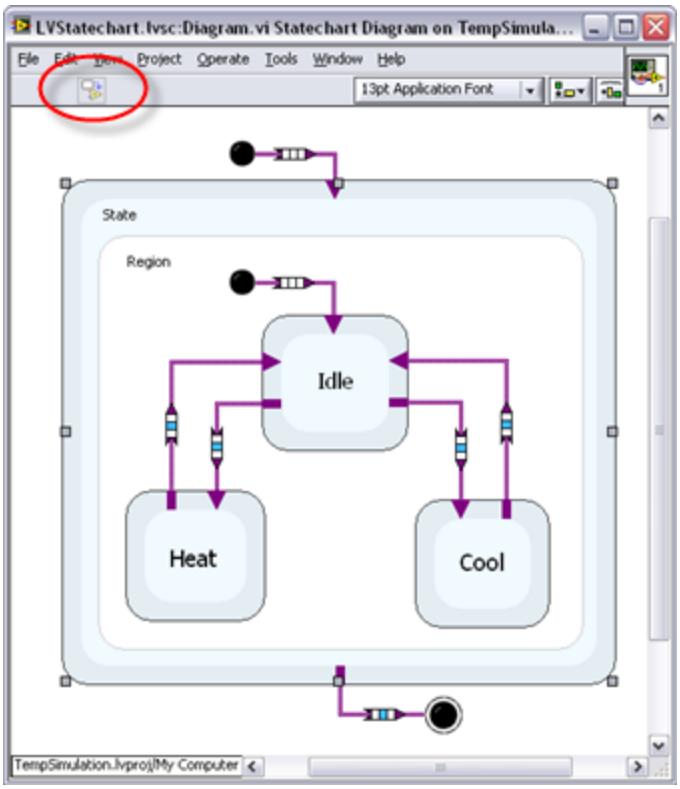
2. 在 Category 面板中，选择 Statechart Code Generation.



3. 在 Debugging 的下拉列表中，选择 Enabled。只有为台式机环境或实时环境生成状态图时，才具有调试功能。要调试 FPGA(现场可编程门阵列)或嵌入式硬件中的状态图，应当将状态图置于台式电脑中进行调试，以确保逻辑是正确的。



4. 这时可能需要重新生成状态图代码。

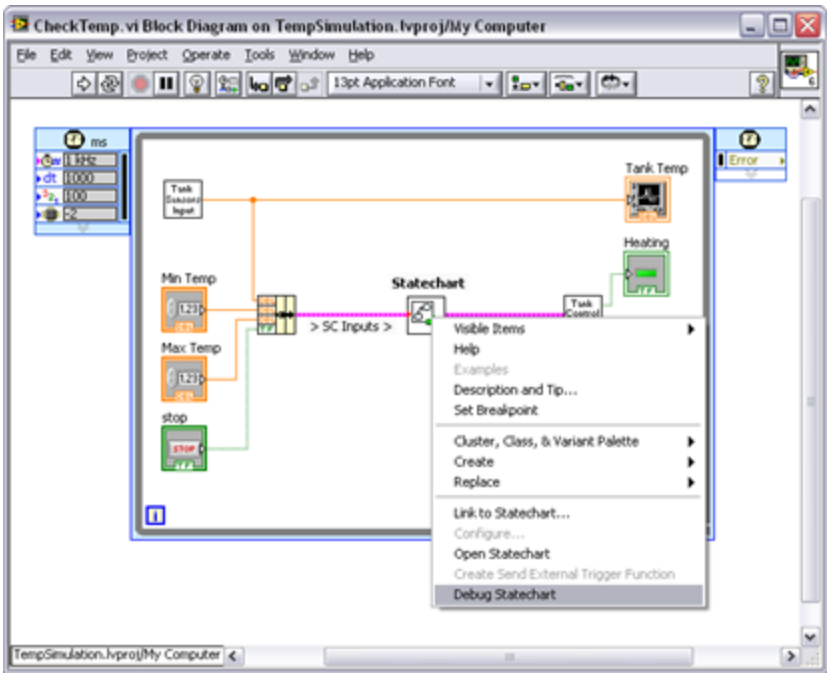


设置断点

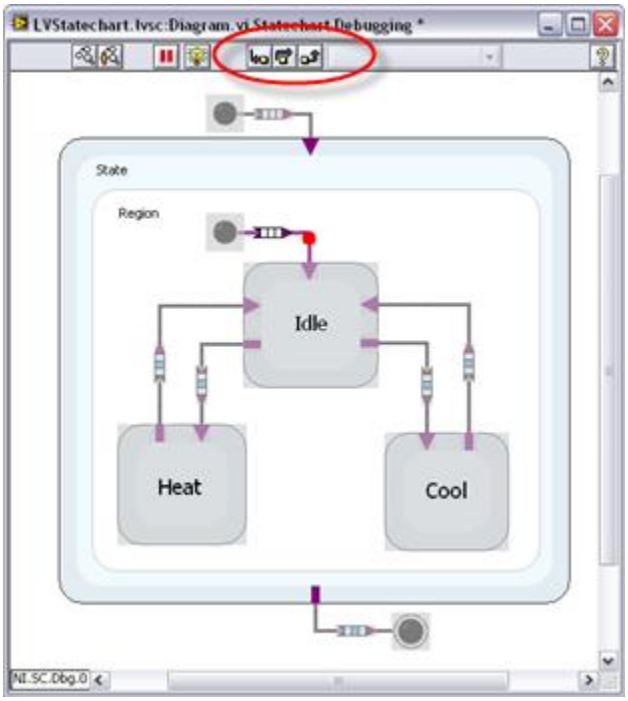
如同调试任何 LabVIEW 应用程序一样，你可以使用断点和探针来寻找程序中的错误。断点有两种设置方法：在状态图上设置或在状态/转移中设置。

以下步骤描述了如何在状态图上设置断点。

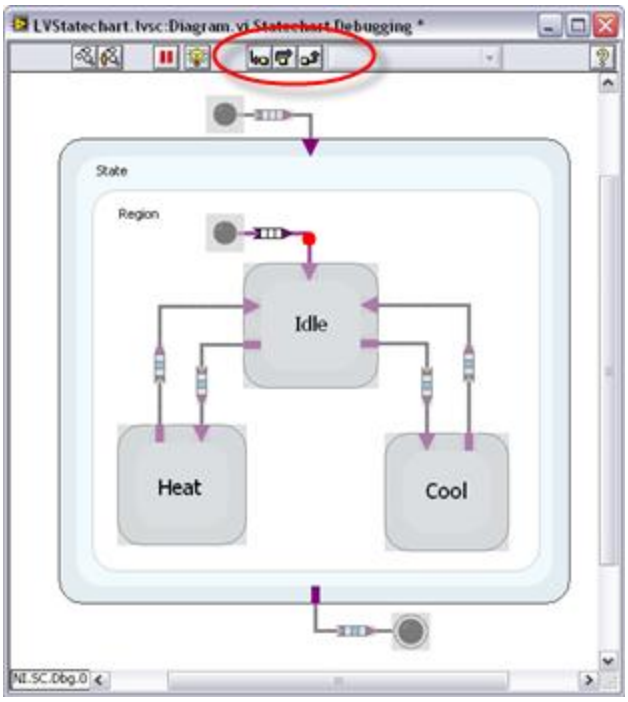
1. 右击链接到状态图上的 Run Statechart.vi，选择 Debug Statechart。



2. 这时状态图将显示于调试窗口中。在状态转移或连线上右击，选择 Set Breakpoint。



3. 使用 Single Step 按钮来单步执行代码。

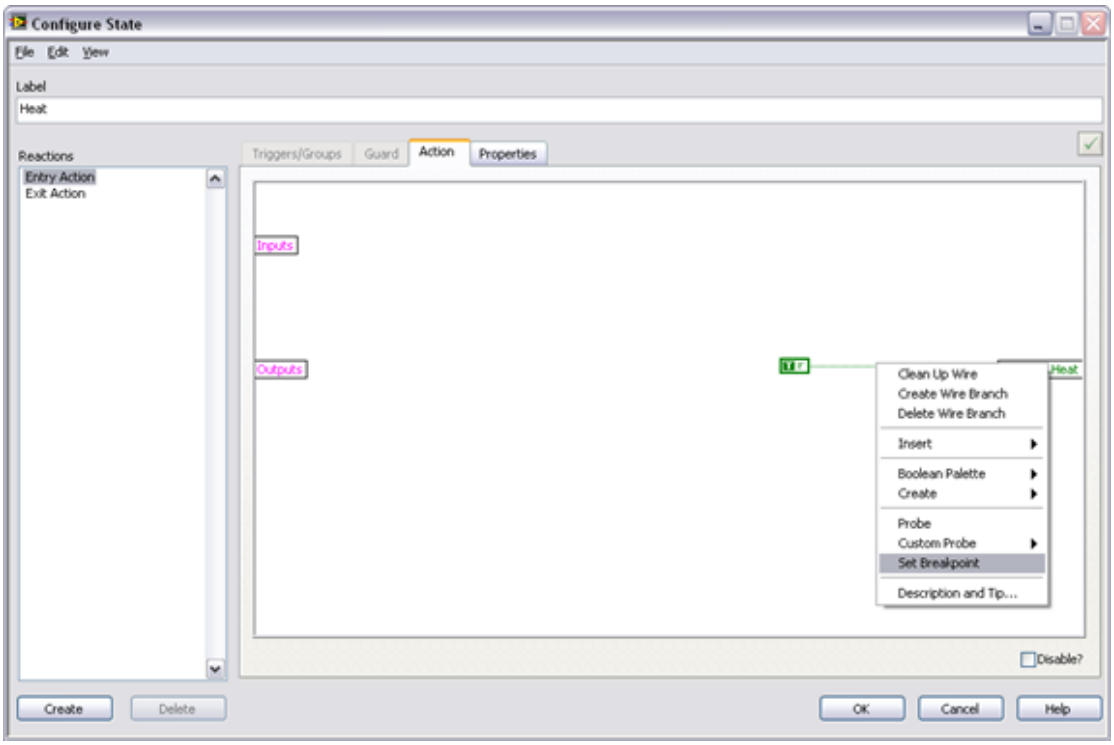


按照 LabVIEW 程序设置断点的方法，在所在状态中设置断点。

1. 打开状态图。

2. 在状态上右击并选择 Configure State... 。或者，在状态转移上右击并选择 Configure Transition...。

3. 状态或转移配置结束后，在连线上右击并选择 Set Breakpoint。状态图代码中的断点和探针的工作方式同 LabVIEW 应用中一样。

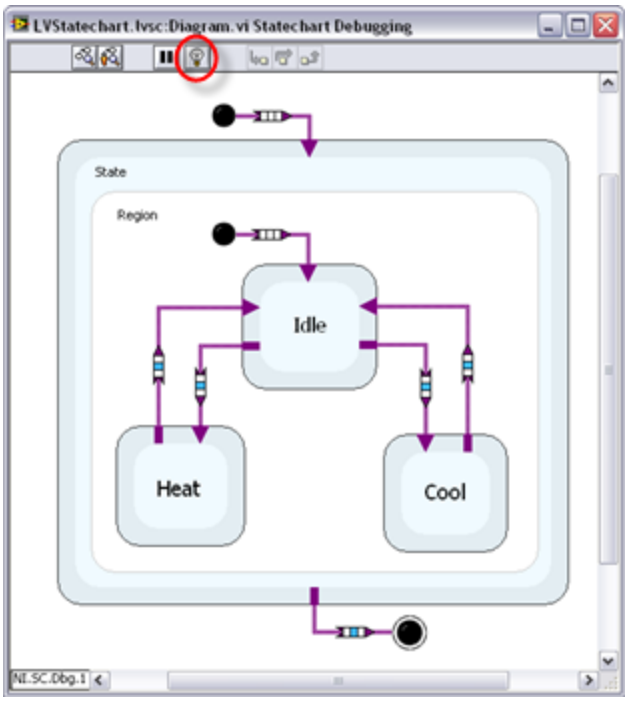


使用高亮化执行和状态图显示数据

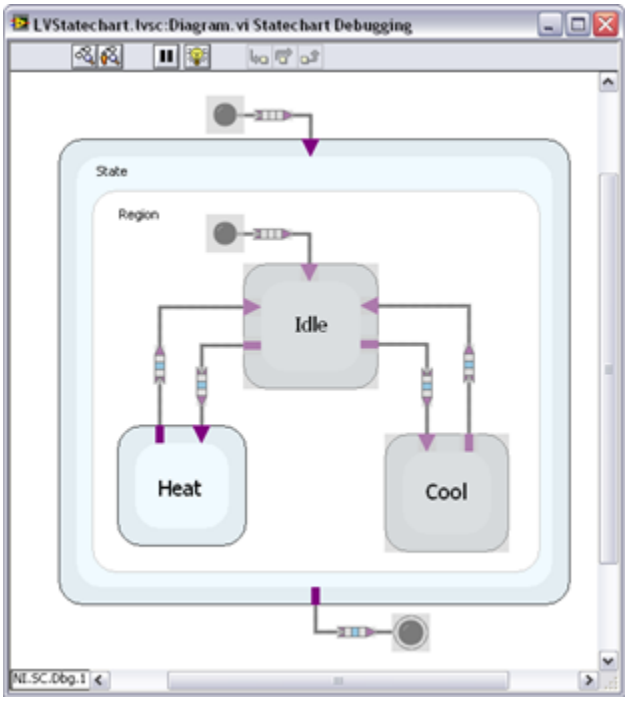
要查看状态图处于哪个状态中，可以采用加亮执行操作这个有效的方法。

1. 在 LabVIEW 应用中，右击 Run Statechart.vi 并选择 Debug Statechart。

2. 单击 Highlight Execution 按钮。运行该 VI。每次迭代中，正在执行的状态都会被加亮显示。



3. 如果必要，可以在 VI 运行的时候再次设置断点。

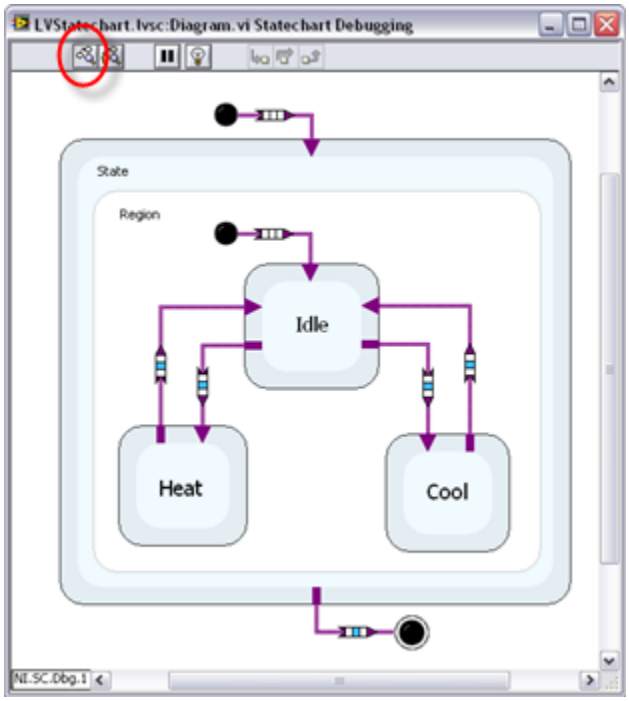


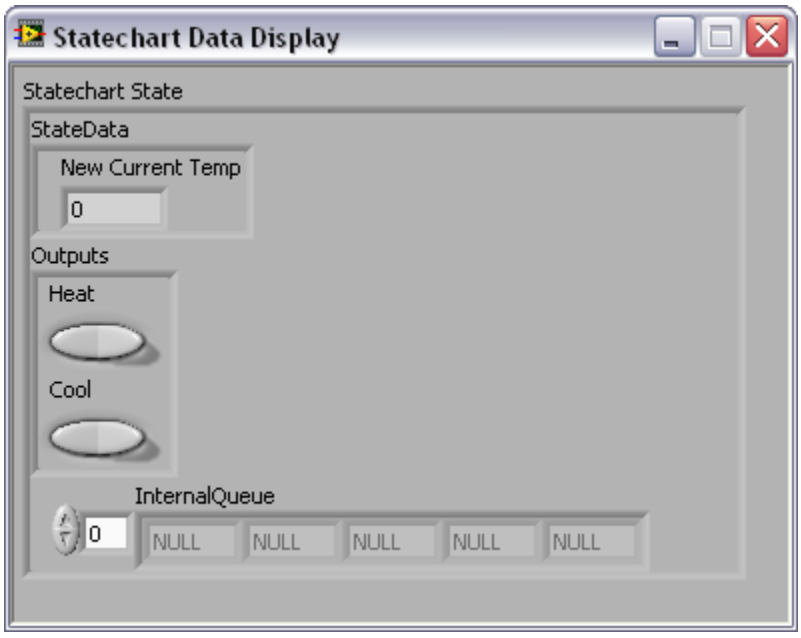
在调试状态图的过程中，另一个可能需要的功能是观察值特性。该功能将显示输出值、状态图的内部状态数据和内部触发器序列。可以通过以下步骤来实现：

1. 运行 VI。

2. 右击 Run Statechart.vi 并选择 Debug Statechart.

3. 单击 Open Statechart Data Display.





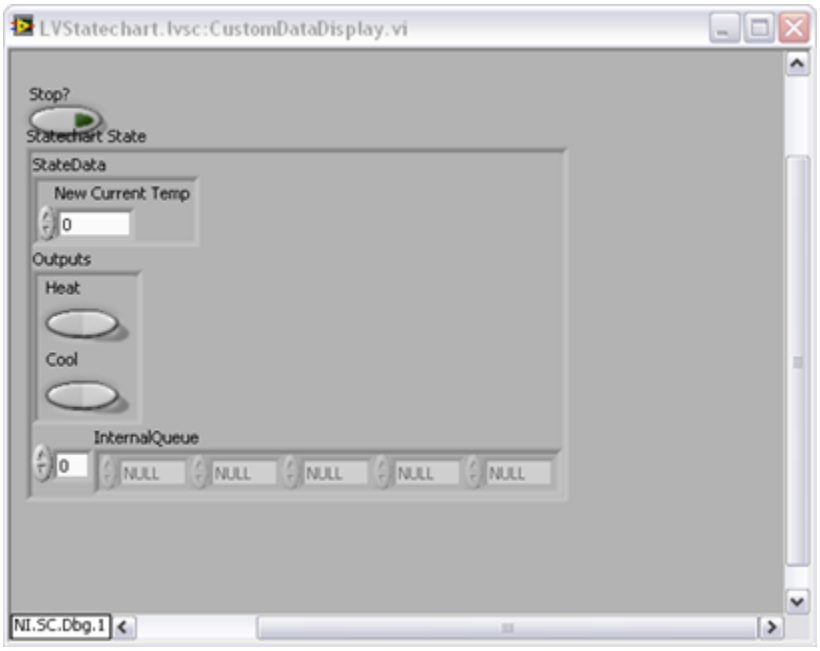
另外，你还可以创建自定义的状态图显示窗口，以便在窗口中实现更多的自定义功能。

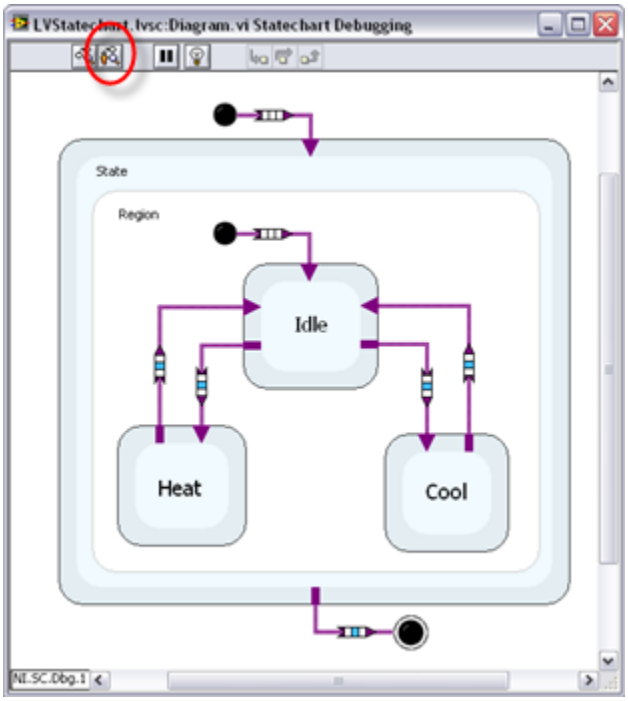
1. 在 Project Explorer 窗口中，扩展该状态图。

2. 打开 CustomDataDisplay.vi.

3. 你可以根据需要更改该观察窗口。也可以自定义该窗口，以包含在程序执行阶段希望能看到的其它信息。

4. 如果要在 VI 运行的时候显示该观察窗口，请调试状态图并选择 Open Statechart Custom Display。





总结

状态图为复杂应用程序的开发提供了一种更高级的，但是仍然具备调试程序功能的抽象工具。LabVIEW 可以提供能够高效调试状态图的应用程序，并排除软件开发中问题的技术。

LabVIEW 状态图模块生成代码概述

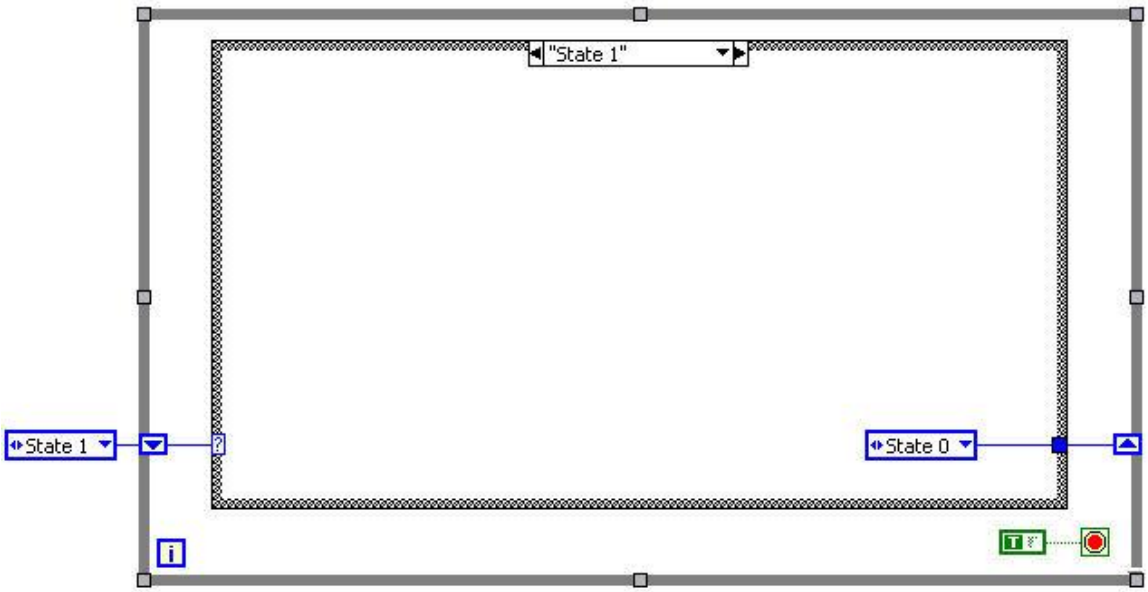
概览

本文档对 LabVIEW 状态图模块所生成的代码进行了概述。该代码从经典状态机开始发展，并包含了所有状态及其转移。

简介

本文档对 LabVIEW 状态图模块所生成的代码进行了概述。本文档着重讨论简化后的程序框图，而不是怎样具体完整的执行。

第一步是对简单状态机这一常见的 LabVIEW 体系结构进行描述。该结构包含了一个 While 循环和连接到移位寄存器的枚举型常量，如下图所示：

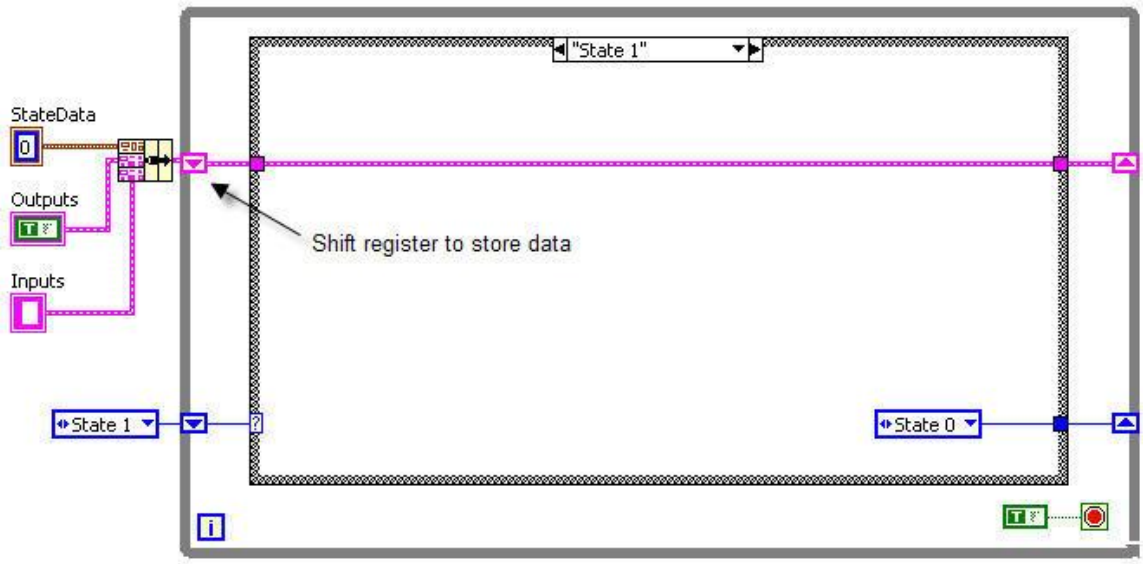


移位寄存器存储当前活动状态，而条件结构决定每个状态对应的执行代码。

状态图代码是从相同的基本体系结构中演变出的。另外，状态图代码包含以下附加结构并以此来处理每个部分。

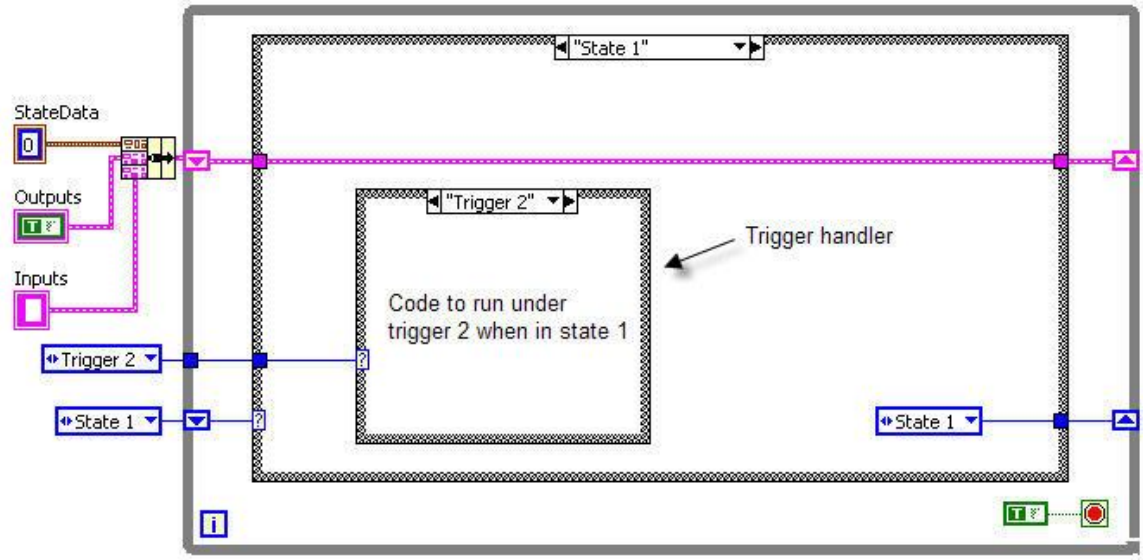
使用移位寄存器缓存数据

状态图提供了输入、输出和状态数据等概念。LabVIEW 使用移位寄存器对数据进行缓存，如下图所示：



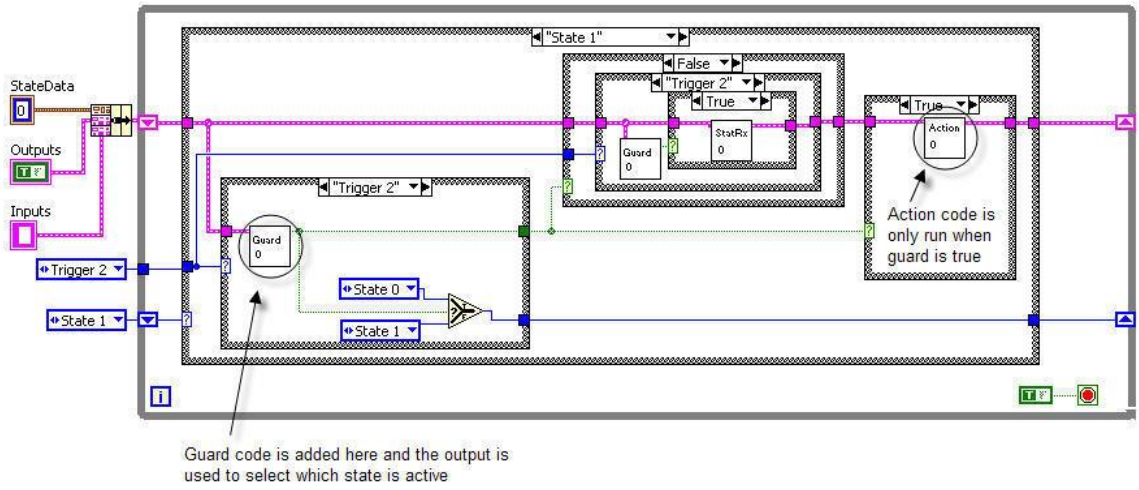
触发器决定运行状态

下一步引入了触发器，以决定在不同的活动状态下运行哪一个分支。下图显示了这部分代码：



转换包含保护和动作代码

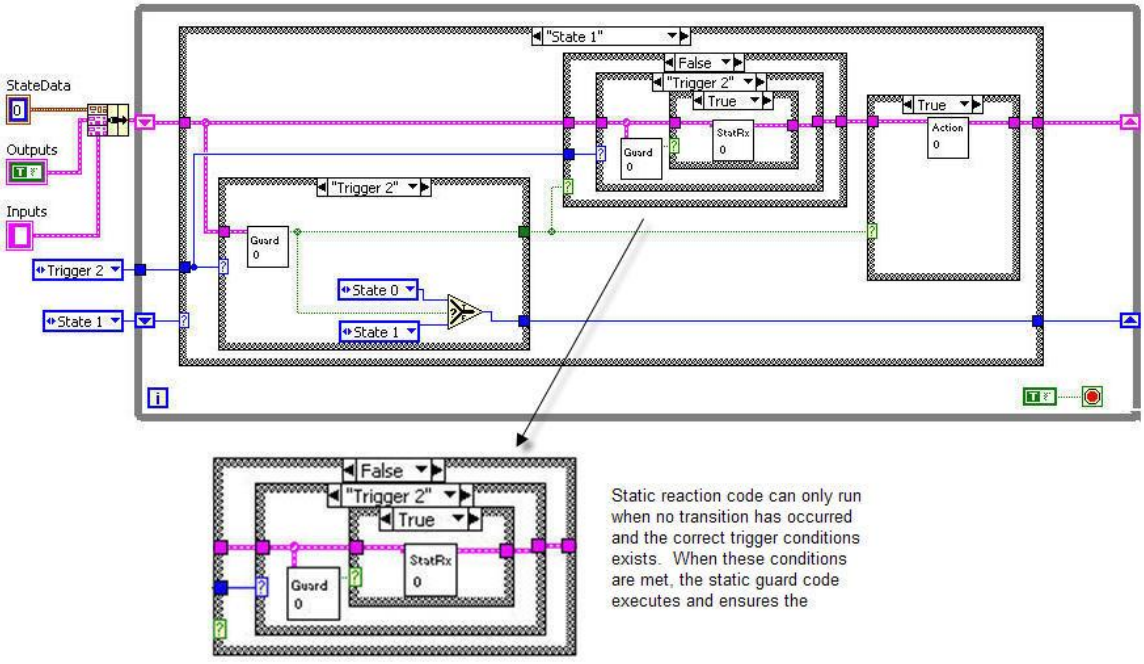
状态图程序框图使用触发器进行状态转换，每个转换都可能包含保护和动作代码。在评估某一个状态时，LabVIEW 首先执行保护代码来判断转换是否有效。在认为转换有效的情况下，LabVIEW 会在状态评估结束后执行动作代码。将一个单独的转换加入到状态机可以得到以下代码：



在相同触发器触发多次转换时，保护评估代码会按照转换优先等级在条件结构中排序。

静态响应

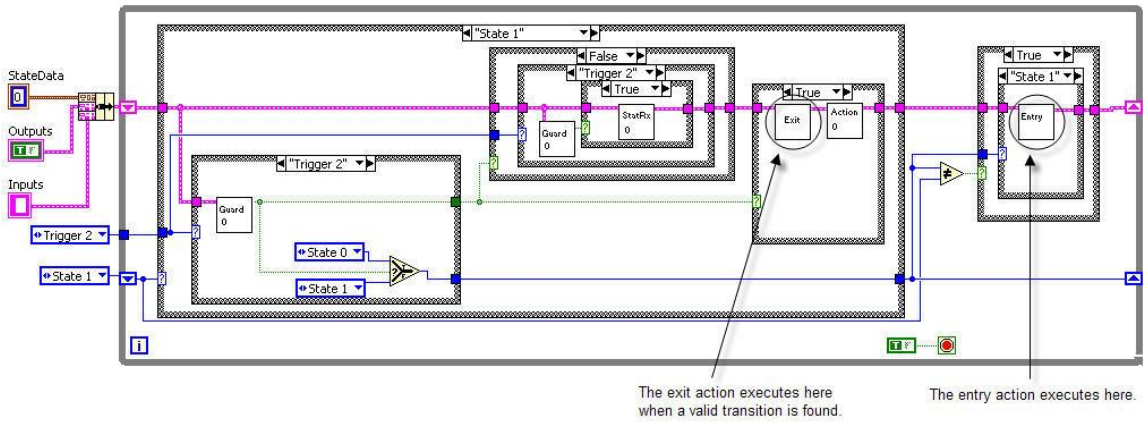
在状态图程序框图中，某个状态还可以有静态响应，该响应在状态活动的时候开始执行。静态响应可以包含保护和触发。并且只在没有有效转移、触发条件达到满足并且静态响应保护代码为真的情况下才被执行。



如果存在多个静态响应，代码将在条件结构中排列出来。

进入与退出动作

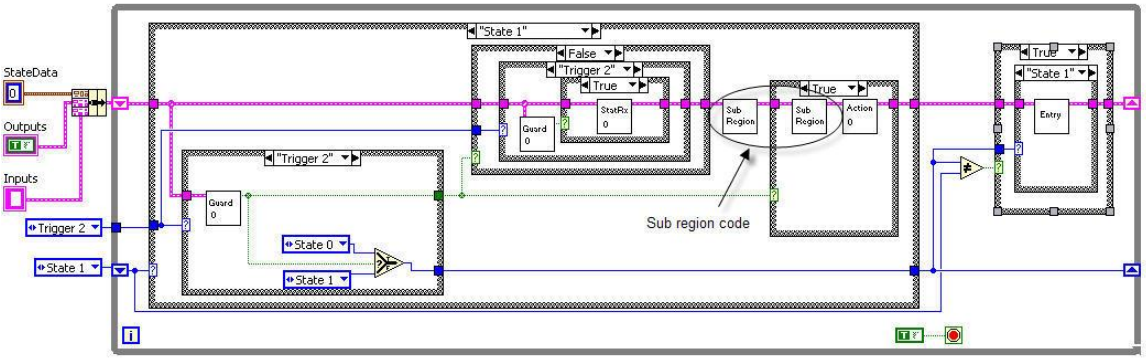
唯一剩下的部分是状态进入与退出动作。增加这些部分就可以完成这个简化的代码。最后得到的代码如下图所示，是带有转换和动作的状态机。



进入动作只在状态变化的时候进行评估。代码执行下一状态的进入动作，这样就可以满足状态图始终在已知状态处结束循环的要求。

层次结构

最后得到的代码是在单个区域内对一组状态的单采样实现。如果状态图使用层次结构，或在状态内部又包含了状态，那么状态图模块将会为每个区域生成代码，并从父状态调用代码。下图显示了这种情况：



本文讨论了大部分的生成代码。唯一需要补充的如下列出，并且对于层次结构来说十分必要：

1、状态图并不是使用枚举常量保存当前活动状态，实际上它保存活动状态的数组。

2、决定下一个活动状态的代码是状态数组而不是单一枚举变量。

使用 LabVIEW 状态图进行 FPGA 编程

概览

LabVIEW 状态图模块扩展了 NI 图形化系统设计平台，以满足日益复杂的应用需求。该 LabVIEW 软件平台的组件为 LabVIEW 添加了另一个设计模型，因此程序员可以在比过去更高的抽象层上来设计应用程序，同时快速将设计与实际 I/O 进行集成。使用 LabVIEW 状态图的设计功能和现有硬件，您可以快速地对数字通信系统、机器控制器和错误处理逻辑等嵌入式系统进行原型开发。

在 FPGA 中使用 LabVIEW 状态图

使用状态图程序框图，您可以通过定义状态、转移和事件进行系统设计。状态图对于例如通信协议实现、系统错误处理设计、创建用户界面和状态机开发等基于事件的设计而言，都是十分有用的。您可以将 LabVIEW 状态图程序部署到多种目标机器上，包括台式 PC、实时控制器和 FPGA。LabVIEW 状态图模块和 LabVIEW FPGA 模块无缝集成在一起，因此您可以在可重复配置的硬件上，使用状态图程序框图和数据流图形化编程进行数字逻辑设计，而无需使用底层描述语言或进行板卡级设计。

串行外围接口（SPI）

SPI 是通常应用于数据流（而不是读写）的同步协议，因此常被用于在微处理器和数字信号处理器之间进行通信。SPI 协议包含了在至少两个设备（主设备以及一个或更多从设备）之间发送的数据包。主设备控制两个数字信号，即片选信号（选择要通信的从设备）和时钟信号。从设备有一条来自于主设备的专用片选数据线，在包含多个从设备的情况下，这条线起连接所有设备的作用。如果应用程序包含主设备和从设备之间的双向通信，您就需要使用两条数据线——主设备输出从设备输入（MOSI）以及主设备输入从设备输出（MISO）。图 1 显示了用于在主设备和从设备之间进行通信的线路。要了解关于 SPI 协议的更多信息，请参考以下链接。

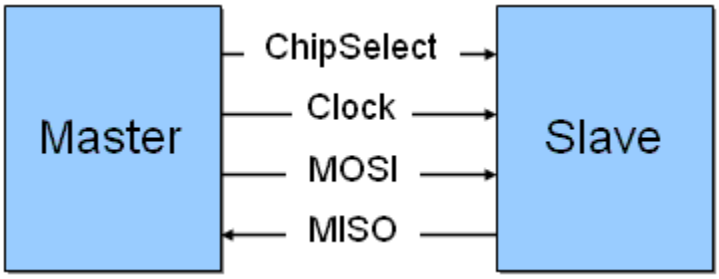


图 1：主设备与从设备之间 SPI 通信示意图

LabVIEW 状态图模块

在 LabVIEW 状态图模块发布之前，LabVIEW 中有两种常用的实现状态机的方法。在不使用附加软件包的情况下，在 LabVIEW 中实现状态机需要在 while 循环中使用条件结构。尽管这种方法是可行的，但您无法同时查看整个流程或是全部代码。LabVIEW 状态图工具包通过显示每个状态，将 LabVIEW 提升到了更高的抽象层次。LabVIEW 状态图模块将状态机设计带入了另一个层面，它在熟悉的 LabVIEW 用户界面中加入了状态机、分级结构、并发、触发和保护等功能，从而更加高级

全面。

对于这个应用程序而言，LabVIEW 状态图模块能够有效实现数字通信的编码与解码。有了 LabVIEW 状态图模块，定时框图可以在多个步骤中进行自然分解。这会将每个步骤进行隔离，从而降低了在实现过程中任意部分的程序复杂性。它还简化了一些通用操作，例如错误和异常处理等，从而帮助您从协议执行直至特殊错误处理状态中能够灵活发挥。LabVIEW 状态图模块还提供了更高等级的抽象，可以通过系统级视图简化复杂应用程序。

FPGA 设计

使用 LabVIEW FPGA，您可以使用与 LabVIEW 图形化数据流开发相似的方法，在不使用底层硬件描述语言和硬件板卡级设计的情况下，建立自定义的 I/O 测量和控制硬件。在系统设计中，硬件和软件相结合提供了很强的灵活性。SPI 通信协议就是这种结合的一个实例——您可以利用其建立并实现一种总线，从而与另一个系统进行交互。在添加了 LabVIEW 状态图模块之后，您还可以从代码中抽象出新的层次，让这种交互更易于可视化和实现。

图 2 是 SPI 通信的定时框图。在片选线路设置为低电平之后，通过 MOSI 线路将比特发送到特定的从设备上，每次发送一个比特，并与时钟线同步。

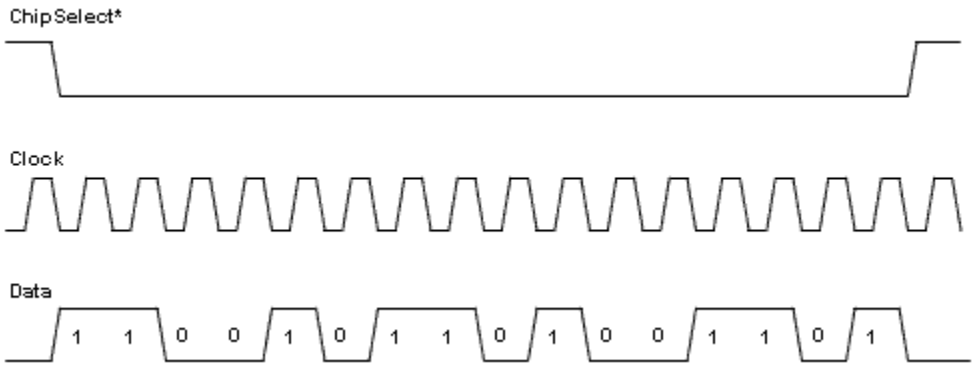


图 2：SPI 定时程序框图

对于 SPI 定时程序框图而言，以下步骤是将定时程序框图分解为状态机的一种方法：

1、设置片选为低电平

2、设置数据（0）

3、设置时钟为高电平

4、设置时钟为低电平

5、设置数据（1）

6、设置时钟为高电平

7、设置时钟为低电平

8、对于比特 2 至 15 重复数据和时钟

9、设置片选为高电平

从这些步骤可以简单地看出，通信协议是如何分解为状态图的有限个状态。请注意共有五个不同的步骤，并且其中有些步骤需要为每个数据比特重复。在状态图中，您可以为每个步骤和循环建立不同的状态。在这三个更新数据线和重置时钟线的重复步骤中，您可以配置状态图，将这个序列重复 16 次，然后进入最终步骤。

部署

现在请考虑在 LabVIEW FPGA 中使用 LabVIEW 状态图模块进行 SPI 通信的实际实现。代码是专为 NI PCI–7831R R 系列智能数据采集板卡设计的，但您可以在任何 NI CompactRIO、PXI 或是 PCI R 系列设备中使用 LabVIEW 状态图模块。图 3 显示了用于主设备的 LabVIEW 状态图。每个上面列出的定时程序框图步骤可以分解为状态图中的五个状态（将时钟重置与 MOSI 线路合并在一起）。它还包含闲置状态，在该状态下，主设备在收到发送指令之前始终保持等待状态，发送完 16 个比特之后，再次返回等待状态。

状态图包含多个定义执行的元素。状态图的初始终端是一个黑色圆圈，定义了状态图的进入点，并且是状态图进行初始化的伪状态。。之后，T1 是将状态图转换置入闲置状态。可以为每一个转换定义一个触发或保护。触发引起一个转换的发生，而保护是判断转换是否发生的条件执行代码。闲置状态等待来自主应用程序的发送指令，触发转换 T2。在写比特父状态内部，子状态配置实际的数据传送。每个状态对应设置或重置 FPGA 输出数字线路中的一条。

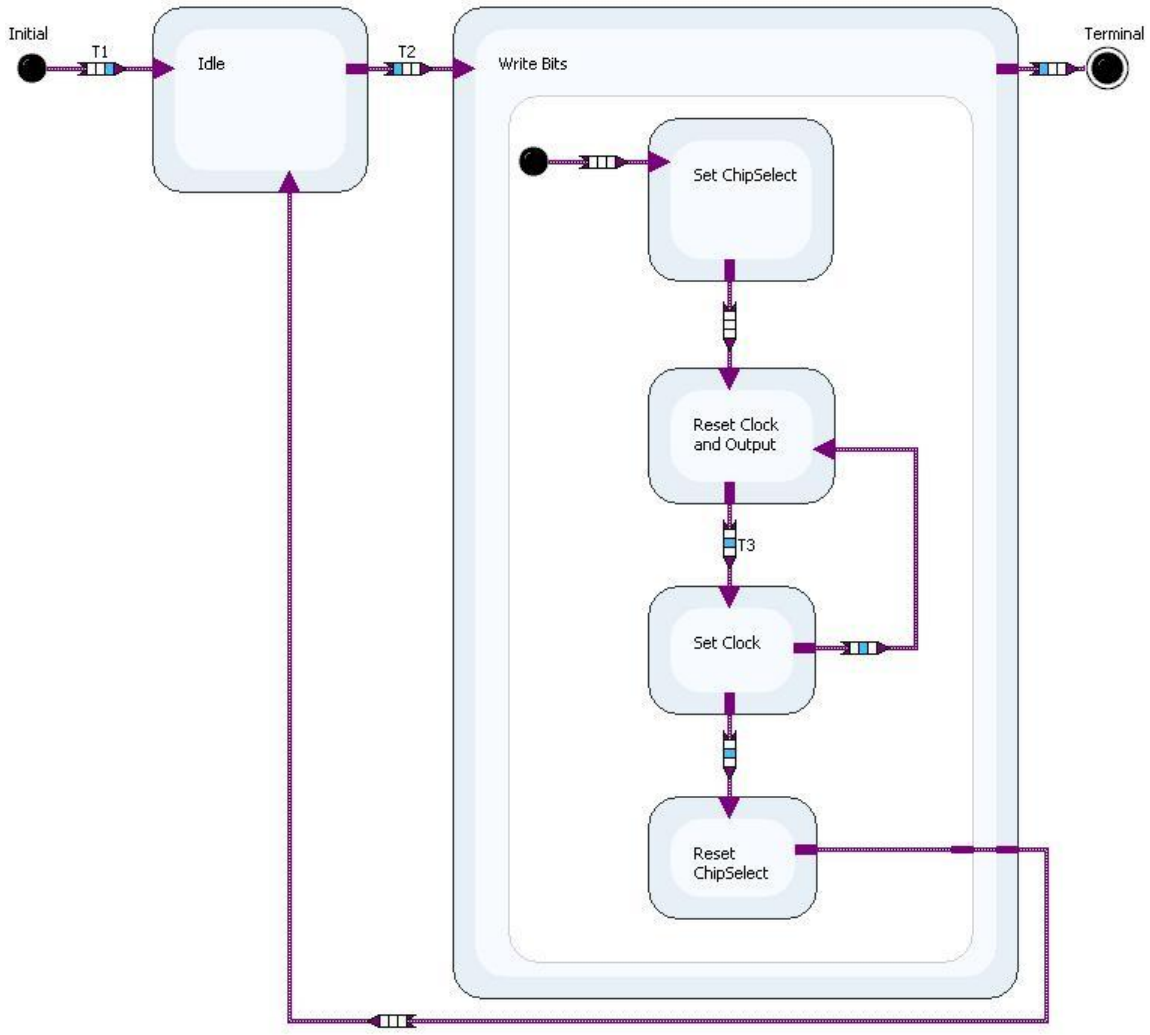


图 3：主设备写数据的状态图

状态内部的行为使用 LabVIEW 图形化代码定义。每个状态都包含了当系统进入状态或退出状态时以及当用户定义事件发生时所执行的动作代码。举例而言，进入重置时钟和输出状态的 LabVIEW 代码如图 4 所示。在这个状态中，当前比特从 16 比特数据数组中移出，写入 FPGA I/O 节点。之后，当前比特数增加，让下一个比特写入下一个循环中。与 SPI 定时保持一致，当时钟信号来的时候，时钟被设置为低电平，片选信号也被保持为低电平。

从重置时钟和输出状态向设置时钟状态的转换必须在半个时钟周期逝去的时候发生。图 5 显示了如何使用 LabVIEW 保护代码计算这种变换。当半个时钟周期等于滴答计数时，转换发生。退出端子代替了子区域中的所有状态，无论程序在这个子区域中处于何种状态，都将提示状态图在触发发生的时候退出。如果输出完成而状态图没有得到退出指令，它将返回闲置状态开始等待。

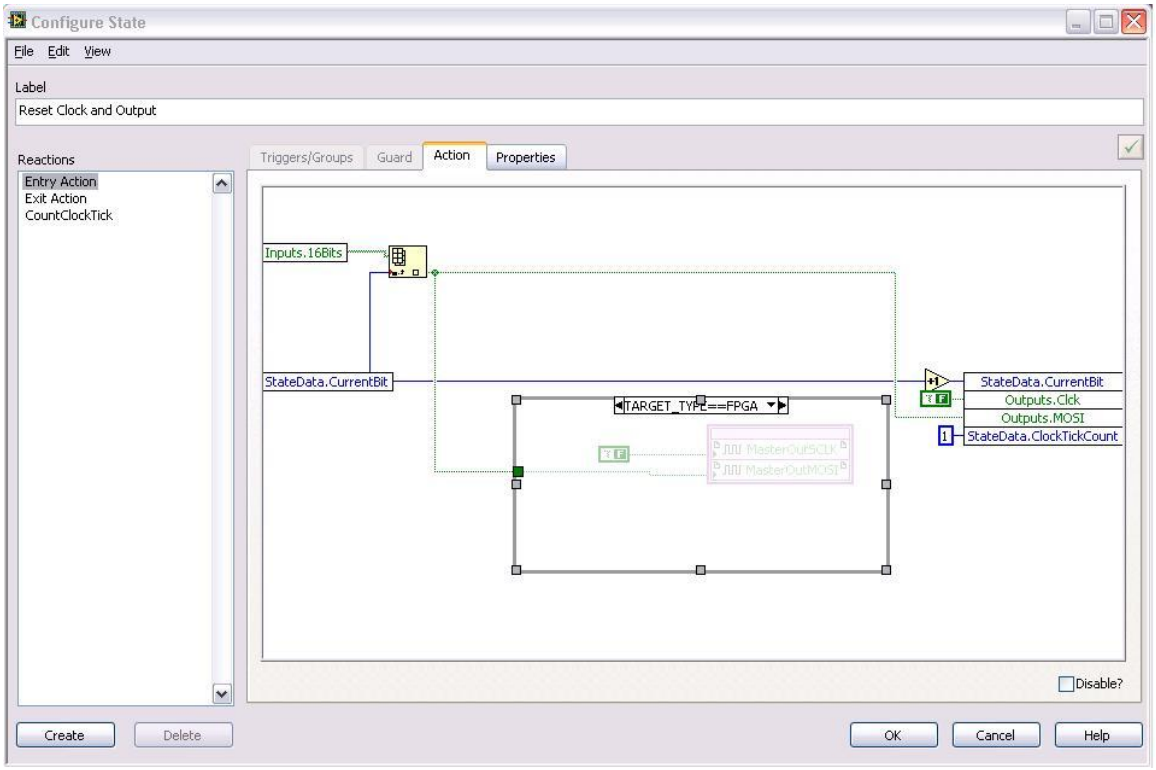


图 4：重置时钟和输出进入动作

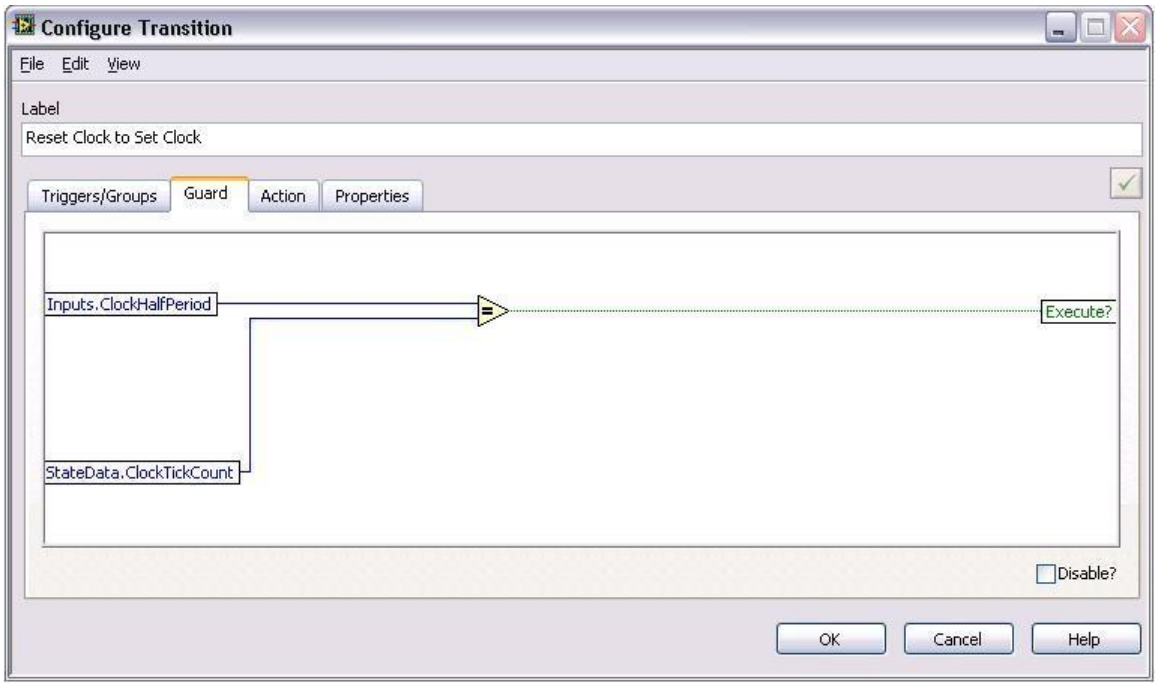


图 5：将重置时钟与输出转向设置时钟配置的保护代码

当您完成状态图的开发之后，您可以生成将程序框图和代码进行封装的 LabVIEW 子 VI。图 6 的程序框图中用红色圈出的部分显示了状态图 VI。它包含触发器输入和数据输入与输出。写触发是触发写比特动作开始时的状态图输入。状态图如同一个子 VI，您可以将数据传入或传出，还可以在另一个窗口中打开并进行编辑。

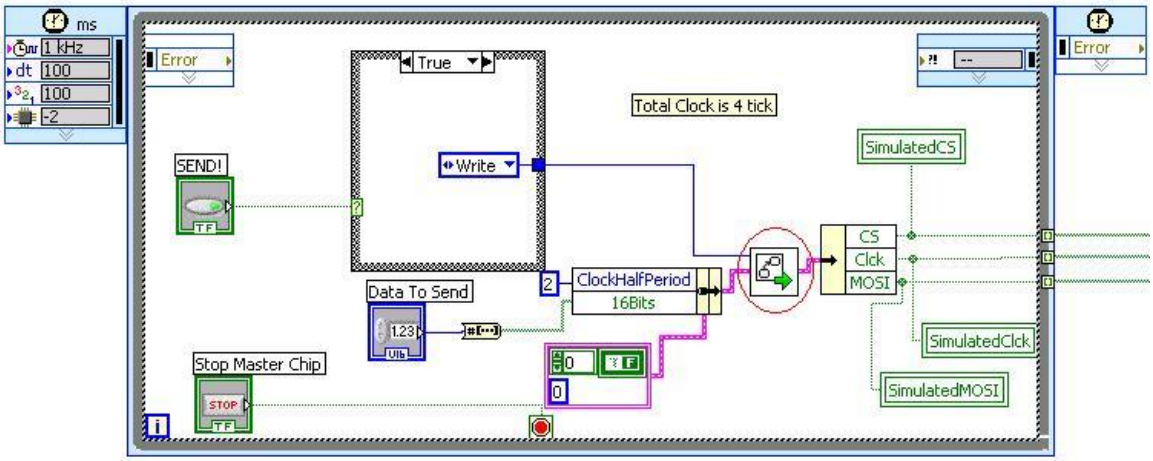


图 6：状态图用红色圈出的 LabVIEW FPGA 代码

调试

状态图代码的生成创建了在后台工作的 LabVIEW 代码。LabVIEW 状态图模块的主要优点是能够使用计算模型，在无需额外工作的情况下，将它自动翻译为 FPGA 代码，同时还能够得到包含抽象和扩展性在内的优点。此代码无需 FPGA 板卡即可进行调试和编译，因此开发员可以在 FPGA 上进行编译和运行之前，方便在台式机上对他们的设计进行验证。状态图还加强了编辑保护和动作时的 FPGA 语法规则，从而可以流畅地编译代码。

考虑

由于 FPGA 设计中需要考虑空间问题，需要注意的是 LabVIEW 状态图模块包含一定的开销。编译使用 LabVIEW 状态图实现的代码，与使用状态机的实现同样功能的普通 LabVIEW 代码相比，状态图代码要多使用 3%的系统资源。

状态图实现还会降低最大理论时钟速率。其原因是一个时钟对应一个检查退出变换的状态。如果一个状态有多条可能的输出路径，就需要多个时钟周期对它们进行计算，这样会成比例地降低最大理论时钟速率。但是，在之前描述的 SPI 实现中，理论最大值的降低并不会影响请求速率，因此不会导致任何问题。在希望开发快速系统或通信协议的时候很可能需要考虑到此因素。

比较

为了进行比较，图 7 显示了使用 FPGA 利用 LabVIEW 状态机编程实现相同的 SPI 输出。在这种表示中，您可以看到特定部分的代码位于状态或转换中，但您无法简单地对程序经过的各个状态进行可视化，这与从 LabVIEW 状态图模块得到的抽象效果是不同的。

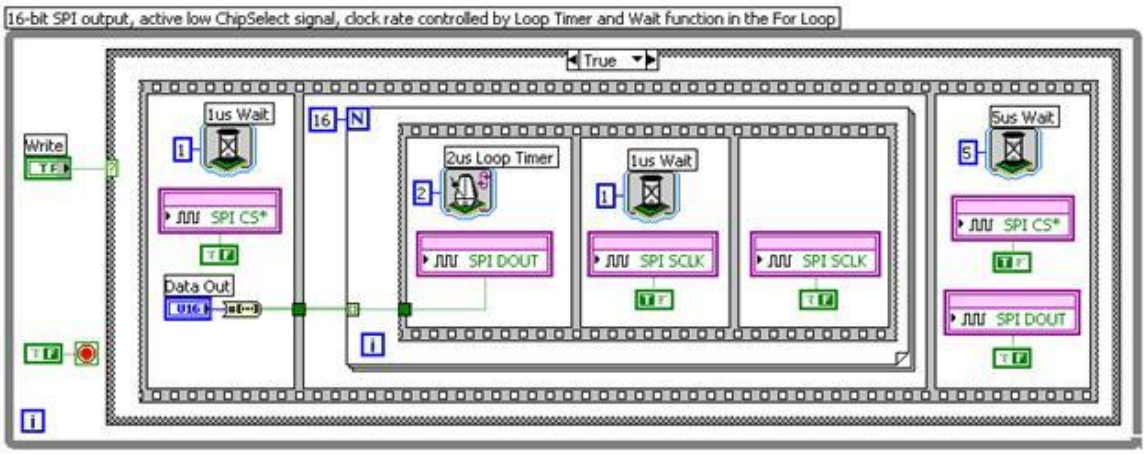


图 7：使用 FPGA 在 LabVIEW 中的非状态图实现

总结

使用不被支持或定制数字通信协议开发测试应用程序的工程师可以在基于 FPGA 的 NI R 系列可重复配置 I/O 硬件上，使用 LabVIEW FPGA 模块快速实现或原型开发不同的通信接口。全新的 LabVIEW 状态图模块通过增加系统级视图的抽象以及分级结构和并发，帮助您定义状态、转换以及事件，从而对状态机进行简化，例如通信协议实现等。LabVIEW 状态图模块将另一个计算模型添加到 LabVIEW 中，帮助您在比以前更高的抽象层次下设计应用程序，同时提高可维护性和可扩展性。使用这个全新的强大软件设计工具，NI 图形化系统设计平台能够帮助您解决愈加复杂的应用程序需求。

使用 NI LabVIEW 状态图搭建混合控制系统

混合控制系统简介

混合控制系统集成了与物理系统的动态特性相结合的离散逻辑。在“混合系统控制”中，Panos Antsaklis 与 Xenofon Koutsoukos 给出了混合控制系统的一个简单范例——一个配有通过一个自动化材料处理系统连接的多台机器的制造工厂[1]。部件的处理是在独立的机器上完成，但是，仅当数字传感器指示该部件传递到机器时才启动该处理。因此，完整的系统被描述为部件在机器间移动的事件驱动动态行为和机器内部对部件处理的时间驱动动态特性的组合。

Antsaklis 与 Koutsoukos 讨论了混合控制系统在制造、通信网络、自动驾驶仪设计、引擎控制、流量控制和化学过程控制中的应用[1]。他们还讨论了如何将混合控制应用于种类繁多的，与物理世界交互的嵌入式控制系统。在“基于逻辑的控制指南：切换控制系统”中，J. P. Hespanha 与 Daniel Liberzon 考察了混合控制系统的应用，并把重点放在了可重新配置系统、故障校正系统和某类参数自适应系统[2]。

在 NI LabVIEW 软件中，您可以利用一种称为 LabVIEW 状态图模块的基于事件的编程方式来描述离散逻辑。您可以利用 LabVIEW 控制设计与仿真模块全面描述该物理系统的动态特性。利用这两个模块的组合，您可以在单个环境下开发，仿真并实时实现一个完整的混合控制系统。

LabVIEW 状态图模块

David Harel 提出状态图是作为描述反应系统和状态机的一种比状态转换图或“状态图”更高级的方法[3]。状态图是包含表示状态的节点和表示状态转移的箭头的有向图。这些迁移箭头上标明了触发事件和保持条件。

状态图的一个主要缺陷是它没有层次性或模块性。因而，对于复杂的系统，它会变得非常庞大因而无法管理。

Harel 提出状态图是一种可简化复杂离散事件系统的可视化描述方式[3]。他描述了一个具有如下状态的航空电子系统范例：

1. 在所有的空运状态下，当黄色操纵杆推起时，座椅将被弹出
2. 变速箱的状态改变独立于制动系统
3. 当选择按钮被按下时，进入选择模式
4. 显示模式由时间显示、日期显示和码表显示组成。

范例 1 显示了将多个状态簇聚成一个超级状态的必要性。范例 2 描述了独立性或正交性。范例 3 强调了普遍的状态转移而不是单个事件标记的箭头。范例 4 描述了状态的细节。利用 Harel 开发的状态图方式，您可以表示模块性、状态簇、状态的正交性（或并发性）和状态的细节。您也可以在一个状态图内“缩放”以探究不同层次的具体状态内容。

面向离散事件系统的状态图概念被吸纳到由对象管理组（OMG）协会开发的统一建模语言（UML）规范中[4]。该 OMG 协会是一个国际性的、成员资格开放的、非盈利性计算机行业协会。LabVIEW 状态图模块与 UML 中关于状态图的规范相兼容。

LabVIEW 状态图模块可以在运行微软 Windows 操作系统的台式机上运行，或者是已经安装 LabVIEW 实时模块转化为专用实时目标平台的台式机。此外，LabVIEW 状态图模块还可以运行于种类繁多的嵌入式目标平台，其中包括 PXI 控制器、NI CompactRIO 或 CompactFieldPoint 可编程自动化控制器或者 NI 紧凑视觉系统。

LabVIEW 状态图模块也可以运行于 LabVIEW FPGA 目标平台，如 PXI/PCI 平台的 FPGA 设备、CompactRIO 可重新配置底板内的 FPGA 或紧凑视觉系统之上。您也可以通过 LabVIEW 微处理器 SDK、LabVIEW 触摸面板模块或 LabVIEW PDA 模块，将该模块与第三方硬件结合使用。

LabVIEW 控制设计与仿真模块

LabVIEW 控制设计与仿真模块包含完成线性系统分析、控制系统与预估程序设计所需要的工具，并且能够为受控系统以及非受控系统提供强大的数值仿真功能。

您可以利用数值的仿真来设计和分析物理组件、系统和进程。利用 LabVIEW 控制设计与仿真模块的仿真循环结构，您可以用标准模块框图的形式表示连续时间系统或离散时间系统。您也可以利用模块框图内连线的箭头标注反馈信号通路和前向信号通路。您还可以利用模块框图中的层次结构表示不同子系统的动态特性。您可以在该仿真循环中放置任意一个 LabVIEW VI，并把这个仿真循环放入一个大型的 LabVIEW 程序中。这一功能使创建批仿真或将仿真作为一部分来集成到一个大型的设计与分析任务中都得到了简化。

动态系统仿真功能包含对单输入单输出（SISO）系统和多输入多输出（MIMO）系统的仿真。其中包含面向连续时间系统和离散时间系统的线性系统模块。您可以利用微分方程、s–域传递函数或零极点增益模型描述连续时间系统，而利用差分方程、z–域传递函数或零极点增益模型描述离散时间系统。

非线性系统模块包含反冲、盲区、摩擦、量化器、饱和、限速程序、继电器和开关。您可以通过 1 维、2 维或 3 维表查询模块，或者通过在模块框图内添加文本公式添加定制的非线性系统。对于文本公式，您可以使用一个公式节点或数学脚本节点（在一个子 VI 内）。标准的信号算法的模块框图、信号发生、绘制、裁剪和线性化等功能都包含在这个工具模块中，当然，还包括了各种优化设计的方法。

通过 LabVIEW 控制设计与仿真模块与 LabVIEW 实时模块的结合使用，您可以把开发完成的离线仿真的 VI 直接应用到实时仿真或控制系统中去。因而，您可以使用一台台式机作为专用的目标平台，或者使用嵌入式目标平台，如 PXI、CompactRIO、CompactFieldPoint 或紧凑视觉系统。

案例研究：巡航控制系统仿真

在如下目录，您可以找到一个称为巡航控制用户界面 VI 的案例，它实现了 LabVIEW 状态图模块与 LabVIEW 控制设计与仿真模块的组合：

该巡航控制用户界面 VI 的前面板如图 1 所示。在此案例中，您测试驱动一个使用该用户界面的混合控制系统仿真， 该用户界面利用了离散逻辑仿真，而车辆动力学、路面状况和 PID 巡航控制系统则使用了连续时间仿真。

当您运行该 VI 时，您可以手动驱动该车辆，然后在加速计的滑动条上输入您所希望的速率。当该车辆沿着仿真路面前进时，请观测该车辆的速率（速率计上的红色指针），上坡时车速减慢，而下坡时车速加快。如果您将加速计设为零（换言之，把您的脚从油门上挪开），您可以观测到车速下降直至停止。如果您使用车闸，您也可以看到该车辆停在该路面的设定位置上。

为了激活巡航控制，请点击前面板的巡航控制部分上的“启动”按钮。该操作设置巡航控制状态为空闲。然后，您可以点击“设置”按钮以提供巡航控制系统的设定速率。此刻，速率计上的绿色指针上移至与设置点的速率相匹配。注意，加速计的滑动条自身开始移动，以指示巡航控制系统的控制操作。当车辆上坡和下坡时，速率变化幅度要小于（您）手动控制系统。

如果您希望关闭巡航控制系统并切换到手动控制，您可以再次点击“启动”按钮以关闭巡航控制系统。您可以点击前面板的巡航控制部分上的“重启”按钮，使巡航控制立即返回至上一次的速率设置点和“启动”状态。

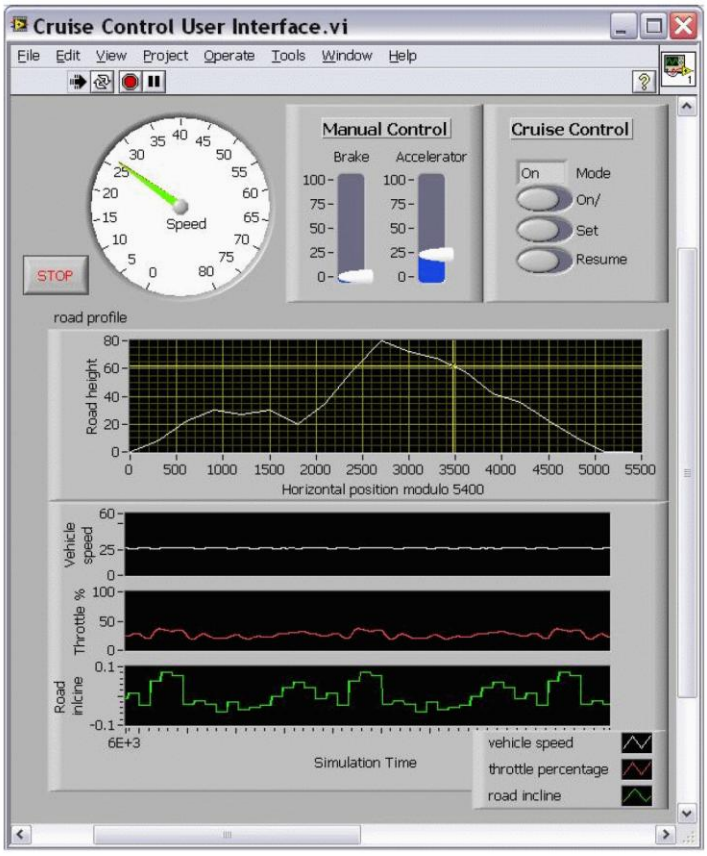


Figure 1. Front Panel for the Cruise Control User Interface VI

图 1 巡航控制用户界面 VI 的前面板

图 2 显示了巡航控制用户界面 VI 的顶层模块框图。它显示了一个仿真循环结构和循环中不同物理子系统（巡航控制、道路和车辆）的层次结构，以及巡航控制逻辑的运行状态图 VI。

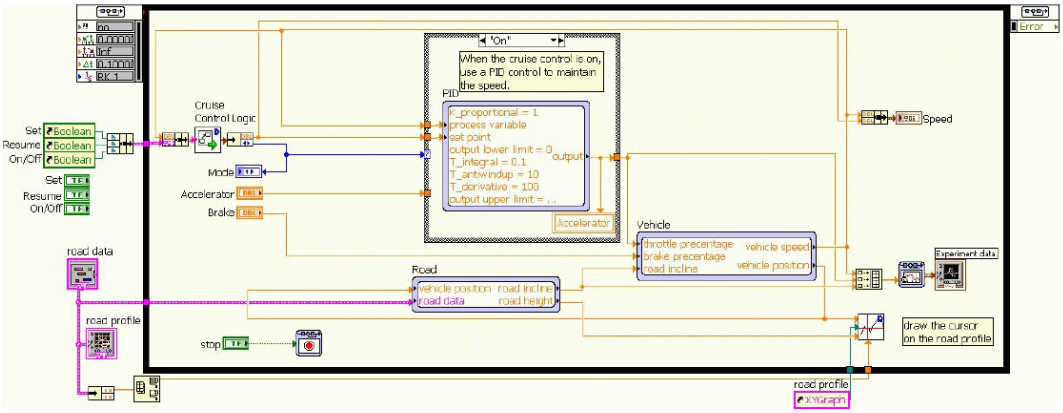


Figure 2. Top-Level Block Diagram for the Cruise Control User Interface VI

图 2 巡航控制用户界面 VI 的顶层模块框图

您可以通过右击该运行状态图 VI 并选择“打开状态图”选项，察看巡航控制逻辑状态图框图。图 3 显示了该例的状态图框图。它包含一个初始状态以及四个其他状态：关闭、空闲、工作和设置。

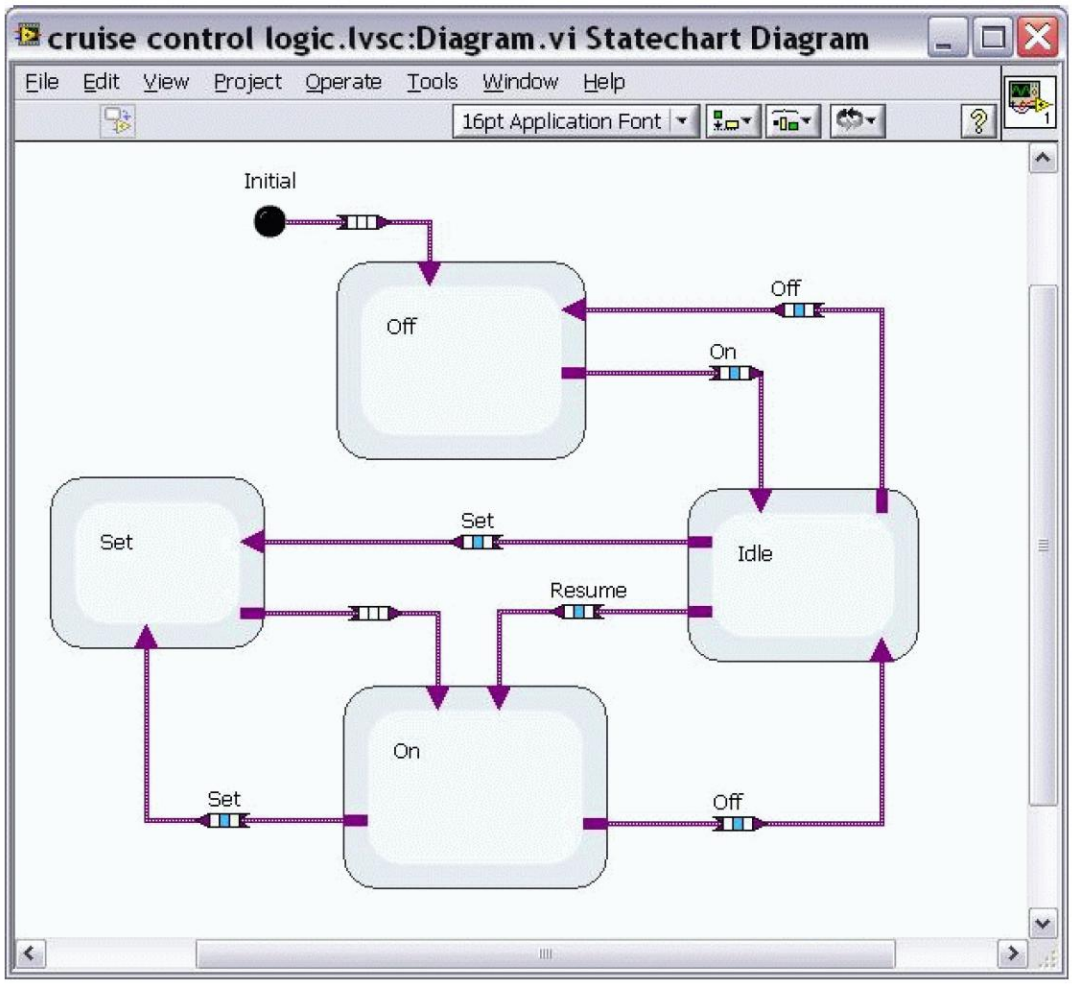


Figure 3. Cruise Control Logic Statechart Diagram Featuring Initial, Off, Idle, On, and Set States

图 3 巡航控制逻辑状态图框图的特征状态：初始、关闭、空闲、工作和设置状态

图 4 显示了连续时间仿真中的一个子系统——PID 控制系统——的范例。该控制系统的模块框图方便可视，并具有一个饱和模块以限制控制信号。

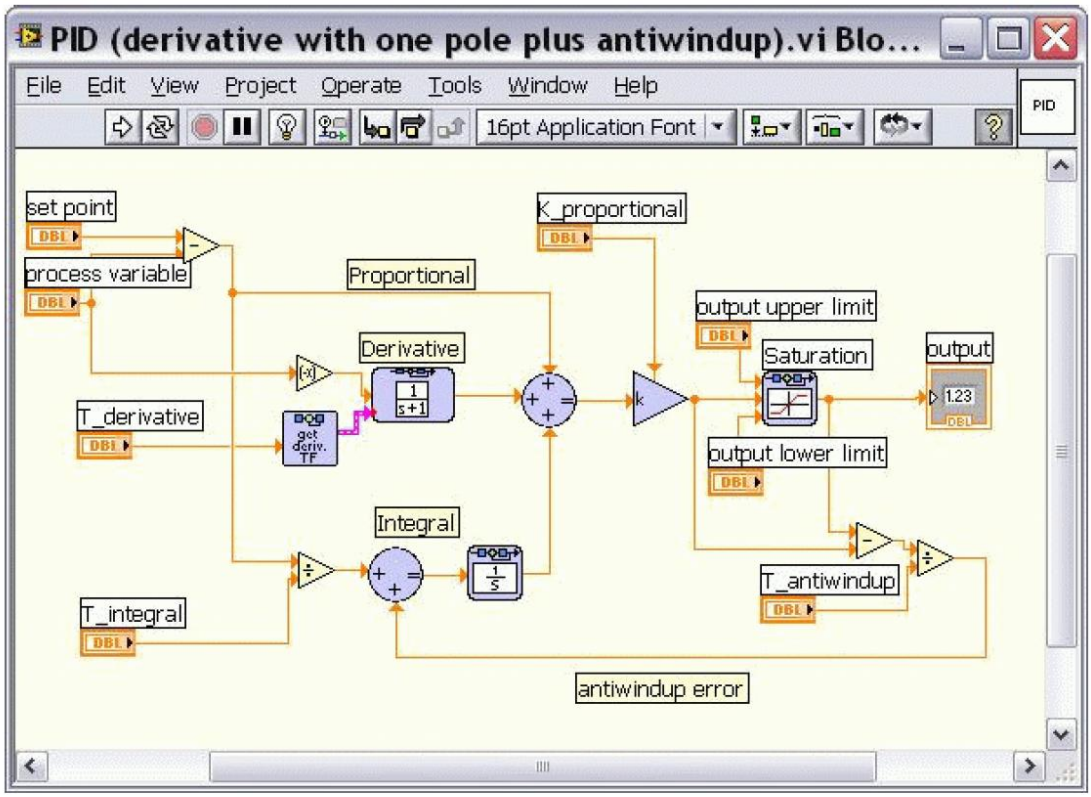


Figure 4. Subsystem in the Continuous-Time Simulation – The PID Control System

图 4。连续时间仿真中的子系统——PID 控制系统

总结

LabVIEW 使得将离散事件系统与物理系统模型组合在一个仿真或实时程序中变得非常方便。LabVIEW 用户界面在创建动态的、交互式程序方面提供帮助，您可以将该界面用作该完整系统设计的一部分。LabVIEW 状态图模块具有独特的原型构造和发布功能，它可以运行于 Windows、LabVIEW 实时目标平台或 LabVIEW FPGA 目标平台之上。LabVIEW 控制设计与仿真模块可以运行于 Windows 或 LabVIEW 实时目标平台之上，并且它也可以方便地与 LabVIEW FPGA 目标平台进行通信。