

Stateflow[®]

For Use with Simulink[®]

- Modeling
- Simulation
- Implementation

Getting Started

Version 6



How to Contact The MathWorks:



www.mathworks.com Web
comp.soft-sys.matlab Newsgroup



support@mathworks.com Technical support
suggest@mathworks.com Product enhancement suggestions
bugs@mathworks.com Bug reports
doc@mathworks.com Documentation error reports
service@mathworks.com Order status, license renewals, passcodes
info@mathworks.com Sales, pricing, and general information



508-647-7000 Phone



508-647-7001 Fax



The MathWorks, Inc. Mail
3 Apple Hill Drive
Natick, MA 01760-2098

For contact information about worldwide offices, see the MathWorks Web site.

Getting Started with Stateflow

© COPYRIGHT 2004 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.

MATLAB, Simulink, Stateflow, Handle Graphics, and Real-Time Workshop are registered trademarks, and TargetBox is a trademark of The MathWorks, Inc.

Other product or brand names are trademarks or registered trademarks of their respective holders.

Printing History: June 2004 First printing New for Version 6.0 (Release 14)

Introduction to Stateflow

1

What Is Stateflow?	1-2
How Stateflow Works	1-3
Stateflow Works with Simulink	1-3
Stateflow Represents Control Modes with States	1-6
Stateflow Changes Active States with Transitions	1-7
Stateflow Reacts to Events	1-8
Stateflow Chooses Destinations with Junctions	1-9
Stateflow Uses Data Variables	1-10
Watching Stateflow Execute During Simulation	1-12
Generating C Code for Targets	1-14
Simulation Targets	1-14
Real-Time Workshop Targets	1-14
Custom Targets	1-16
Ways That You Can Use Stateflow	1-17
Before You Get Started	1-19
Required User Knowledge Level	1-19
Required and Optional Software	1-19
Stateflow Installation	1-20
Setting Up Your Own Target Compiler	1-21
Using Stateflow on a Laptop Computer	1-22

Controlling with States and Transitions

2

Building an On-Off Control Model	2-2
Creating a Simulink Model with a Stateflow Block	2-3
Saving a Simulink Model	2-5
Opening the Diagram for a Stateflow Block	2-6
Drawing States in Stateflow Diagrams	2-7
Drawing Transitions in a Stateflow Diagram	2-10
Adding a Trigger Event to the Stateflow Diagram	2-14
Sending a Trigger Event to the Stateflow Chart	2-16
Simulating a Stateflow Diagram	2-22
Setting Up Diagram Simulation	2-22
Simulating a Stateflow Diagram	2-25
Using the Debug Tool During Simulation	2-32
Guarding Transitions with Event Triggers	2-37
Adding Event Triggers to Transitions	2-37
Adding Multiple Trigger Events to a Stateflow Chart	2-43
Sending Multiple Trigger Events to a Stateflow Chart	2-47
Modifying Output Data with Actions	2-51
Adding State Entry Actions	2-51
Adding Output Data to the Stateflow Chart	2-53
Sending Stateflow Output Data to Simulink	2-55
Simulating Event Triggers and Modified Output Data ...	2-56

Controlling with Junctions

3

Adding a Sensor to the Model	3-2
Adding a Sensor Event	3-2
Adding Sensor Data	3-6

Adding a Junction for Flow Control	3-8
Deleting, Copying, and Renaming Stateflow Objects	3-8
Adding and Connecting Junctions	3-13
Entering Transition Conditions	3-16
Adding a Graphical Function for Convenience	3-19
Adding a Graphical Function	3-19
Calling a Graphical Function	3-24
Simulating with a Sensor Event, Junction, and Function	3-25
Simulating Junction Behavior	3-31
Introducing Stateflow Semantics	3-36
Using Junctions in Flow Diagrams	3-37

Controlling with Superstates

4

Adding Superstates to Simplify Control	4-2
Adding a Superstate	4-3
Simulating the Superstate	4-9
Adding a History Junction to Save State	4-14
Adding a History Junction to a Superstate	4-14
Simulating with a History Junction	4-17
Creating Subcharts to Add More Substates	4-20
Converting a State to a Subchart	4-20
Simulating a Subchart	4-28
Controlling Objects with Parallel Superstates	4-31
Creating Exclusive States	4-31
Creating Parallel States	4-32
Simulating Parallel States	4-37

5

Controlling a Physical Plant 5-2

The Bang-Bang Boiler Demo 5-8

Where to Go from Here 5-13

Index

Introduction to Stateflow

This chapter introduces you to Stateflow[®] with preliminary information that you need before you start using Stateflow.

What Is Stateflow? (p. 1-2)

Get an overview of Stateflow features.

How Stateflow Works (p. 1-3)

Get an overview of how Stateflow operates in Simulink[®] models.

Watching Stateflow Execute During Simulation (p. 1-12)

See how you can easily modify your design, evaluate the results, and verify the system's behavior at any stage of your design.

Generating C Code for Targets (p. 1-14)

Learn how Stateflow automatically generates integer, floating-point, or fixed-point code directly from your design (requires Stateflow Coder).

Ways That You Can Use Stateflow (p. 1-17)

Shows you a variety of simulation types for which Stateflow is a preferred modeling tool.

Before You Get Started (p. 1-19)

Make sure that Stateflow is installed properly on your system.

What Is Stateflow?

Stateflow is a graphical design and development tool that works with Simulink. Stateflow is a suitable environment for modeling logic used to control and supervise a physical plant modeled in Simulink.

Stateflow integrates with its Simulink environment to model, simulate, and analyze your system. Stateflow lets you design and develop deterministic, supervisory control systems in a graphical environment. It visually models and simulates complex reactive control to provide clear, concise descriptions of complex system behavior using *finite state machine* theory, flow diagram notations, and state-transition diagrams all in the same diagram. Stateflow brings system specification and design closer together. It is easy to create designs, consider various scenarios, and iterate until the Stateflow diagram models the desired behavior.

How Stateflow Works

In Simulink, a Stateflow block uses a Stateflow diagram to represent an object with a discrete set of modes. These modes are known as *states*. The Stateflow finite state machine reacts to events by changing states for the controlled object. The behavior of the object depends on what state the object is in and how the object changes from one state to another.

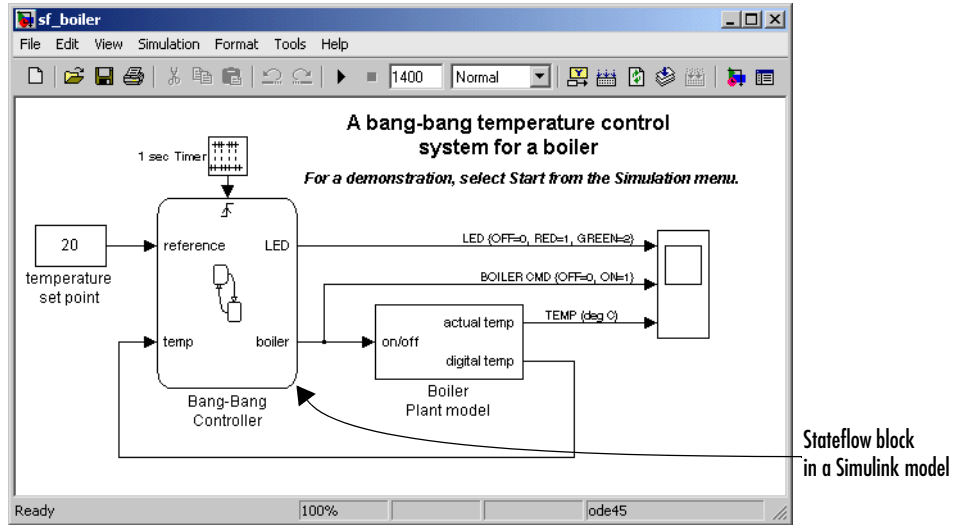
Use the following topics to get an introductory understanding of how Stateflow works in a Simulink model:

- “Stateflow Works with Simulink” on page 1-3
- “Stateflow Represents Control Modes with States” on page 1-6
- “Stateflow Changes Active States with Transitions” on page 1-7
- “Stateflow Reacts to Events” on page 1-8
- “Stateflow Chooses Destinations with Junctions” on page 1-9
- “Stateflow Uses Data Variables” on page 1-10

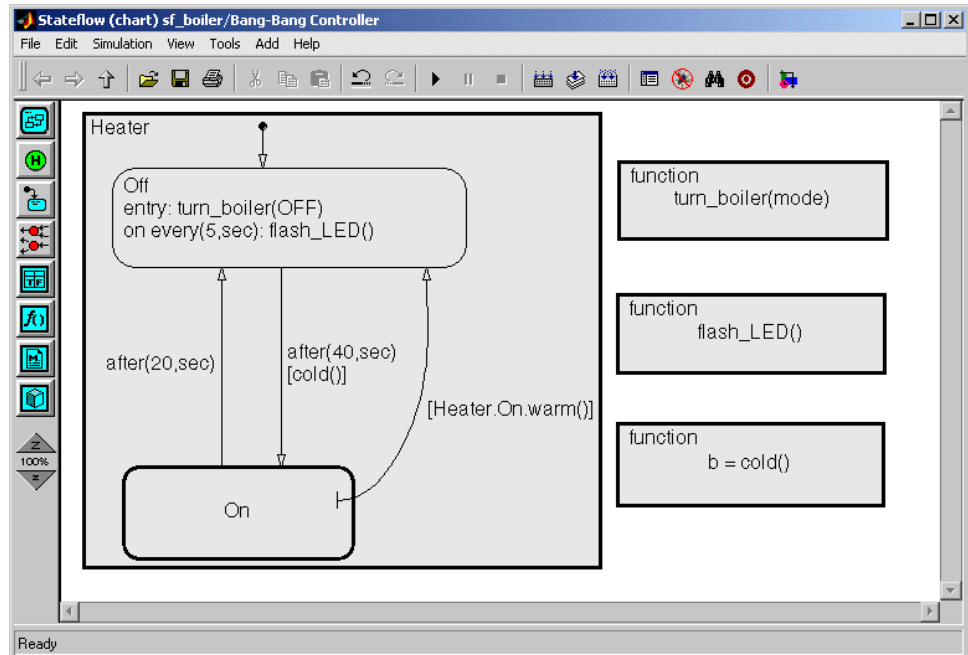
Stateflow Works with Simulink

Stateflow is a tool in Simulink that can be used to represent dynamic control of a control object in a physical plant that you model in Simulink. A control object can be a motor, a pump, or any device that changes the behavior of the model to control its operation. In Stateflow diagrams, you visually model reactions of the control object to physical events from sensors and switches. These reactions make decisions that change the behavior of the Simulink model.

The following diagram depicts a Stateflow block in the Stateflow demo model sf_boiler (type sf_boiler at the MATLAB® prompt):



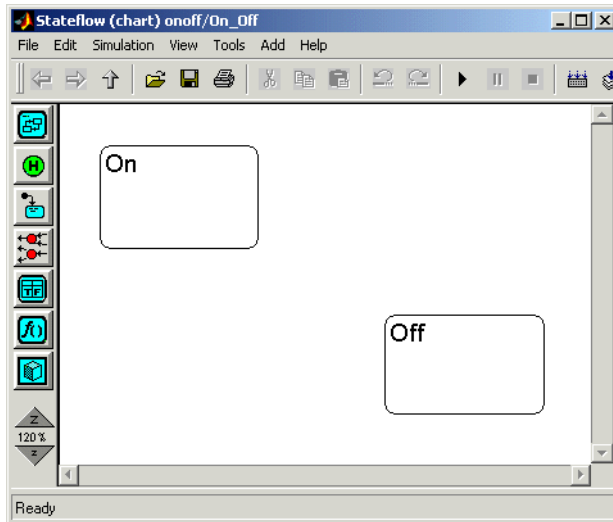
In the `sf_boiler` model, the Stateflow block controls a boiler simulated by the Simulink subsystem Boiler Plant Model. If you double-click the Stateflow block in this model, the Stateflow diagram that programs this Stateflow block appears in the Stateflow diagram editor window, as shown.



You can design a model starting with a Stateflow (control) perspective and then later build the Simulink model. You can also design a model starting from a Simulink perspective and then later add Stateflow diagrams. You might have an existing Simulink model that would benefit if you replace Simulink logic blocks with Stateflow diagrams. The approach you use determines how, and in what sequence, you develop various parts of the model.

Stateflow Represents Control Modes with States

The following example Stateflow diagram has two states, On and Off:

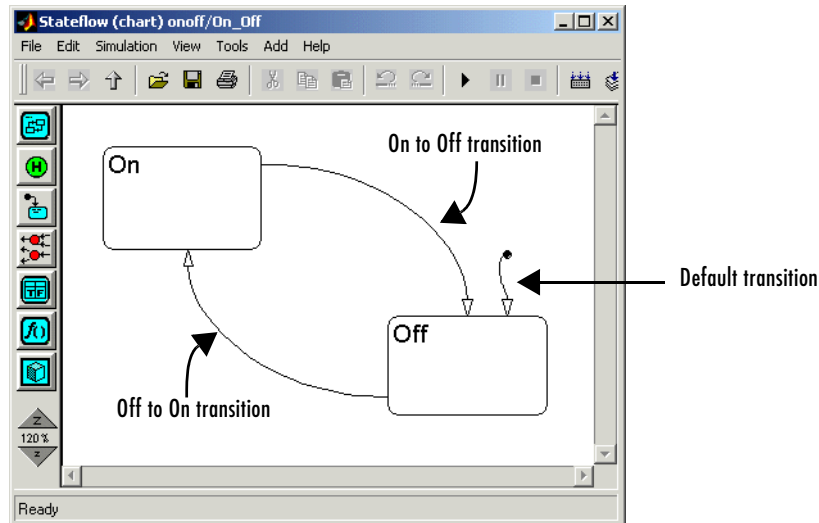


Finite states model objects that have a limited number of states. For example, a house fan or a blender can have states such as High, Medium, Low, and Off. Automatic transmissions in automobiles can have states such as Park, Reverse, Drive, and Low. The preceding example has just two states: On and Off. These states could model a light switch or a single-speed motor.

A state can be active or inactive. If a state is active in Stateflow, you can program activity to take place when it becomes active. For example, if the On state for a fan is made active, this can represent the fact that the fan is now on. If the Drive state for an automatic transmission is active, this can represent the fact that the vehicle using this transmission is moving in the forward direction in highest gear.

Stateflow Changes Active States with Transitions

States alone are not sufficient to model the behavior of an object that changes active states. Also needed are paths for changing active states. This service is provided by transitions. The following example adds three transitions to the example in “Stateflow Represents Control Modes with States” on page 1-6:



Notice that transitions are directional. They originate with a source state and terminate with a destination state. If the source state is active and the transition is taken, the source state becomes inactive and the destination state becomes active.

For example, if the Off state is active, the On state can become active through the Off to On transition. However, the model cannot use the same transition to activate the Off state if the On state is active. To make this change, an On to Off transition is required.

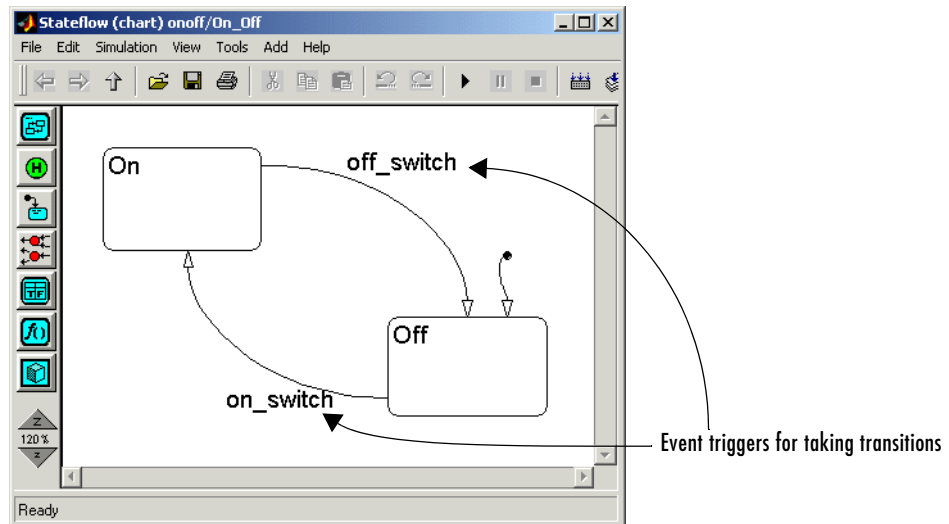
Also present is another type of transition, a default transition. This transition points to the state Off but has no apparent source state. The default transition is used to indicate which of the two states, On and Off, becomes active when the Stateflow diagram becomes active. In this case, the Off state becomes active when the diagram becomes active.

Stateflow Reacts to Events

Events provide the motivation that a model needs for taking a transition between states. An event can represent the point at which the temperature in a room exceeds a given temperature. An event can represent the point that water in a container reaches its top level requiring that the flow of water to it be shut off. Normally, the physical plant modeled in Simulink originates these events, which trigger a Stateflow block that responds to them.

An important thing to remember is that Stateflow diagrams become active only if an event occurs. Also, no transition from the active state to another state takes place without an event. In the preceding example, if the Off state is active and no later event is received, the Off state stays active.

Events can be specific in Stateflow, and the diagram can watch for specific events with the event triggers, as shown in the following example:

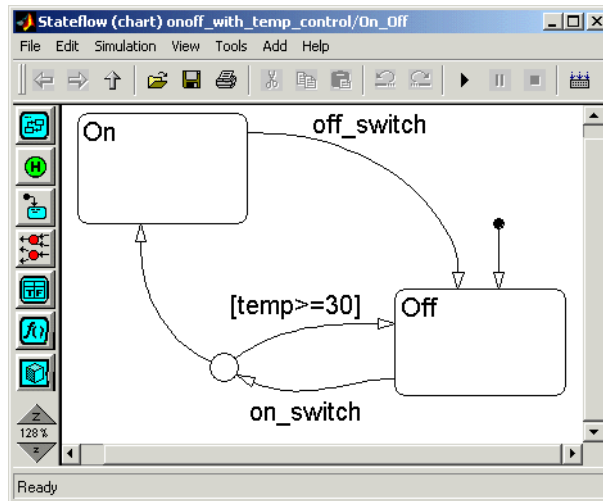


In this example, the transition from the Off state to the On state takes place only if the Off state is active and the event on_switch is received. Also, the transition from the On state to the Off state takes place only if the On state is active and the event off_switch is received.

Stateflow Chooses Destinations with Junctions

While states, transitions, and events are the basic building blocks of a state machine, Stateflow also provides decision points in transitions with junctions. Junctions provide an alternative path for transitions.

In the following example, a junction has been added to the preceding diagram.

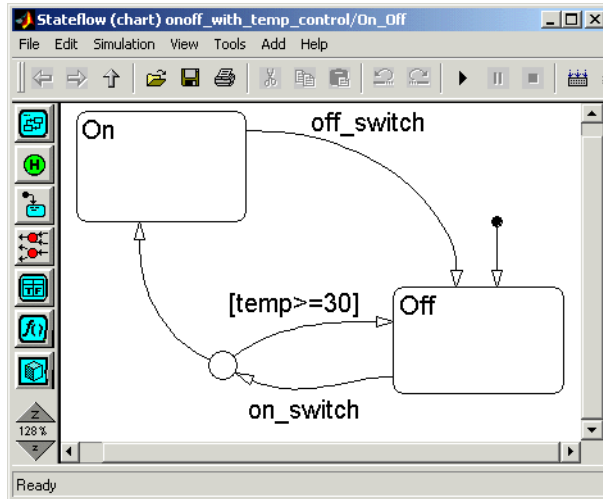


If the transition from Off to the junction is taken, one of the transitions out of the junction is taken thereafter. In this case, if the temperature is greater than or equal to 30 degrees, the Off state is reentered and the Off state stays active. If not, the transition to the On state is taken and the Off state is exited and the On state becomes active.

You can use flow diagram notation to create decision-making logic such as for loops and if-then-else constructs without the use of states. In some cases, using flow diagram notation provides a closer representation of the required system logic that avoids the use of unnecessary states.

Stateflow Uses Data Variables

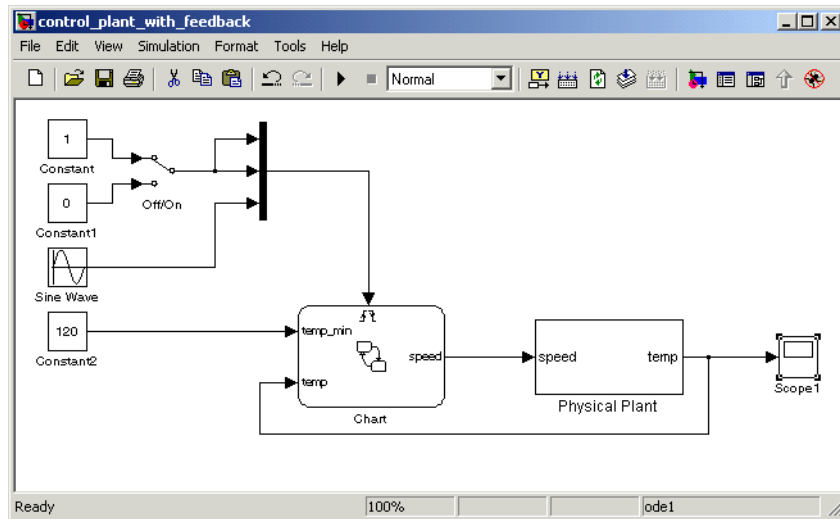
In the following example, discussed in “Stateflow Chooses Destinations with Junctions” on page 1-9, outside temperature is represented by the variable `temp`.



`temp` is an example of Stateflow data. Stateflow data are variables or constants that you define for Stateflow diagrams.

You can define local data used only within the Stateflow diagram or you can define input or output data that you receive from or send to the Simulink model, respectively.

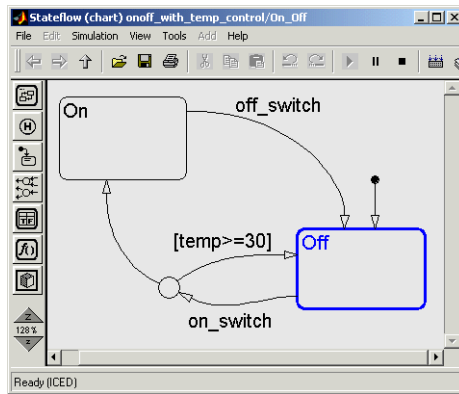
The following diagram shows two data inputs from Simulink.



In this example, the data `temp_init` receives its value from a Constant block in Simulink as the minimum setpoint temperature for the physical plant. `temp`, the temperature of the physical plant, is updated as input data from the physical plant in Simulink during simulation.

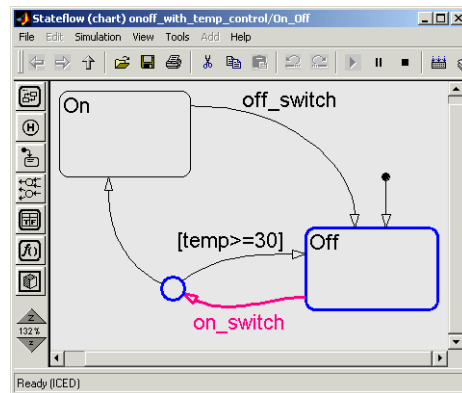
Watching Stateflow Execute During Simulation

Once you finish a Simulink model with a Stateflow block, you can simulate it. During simulation, Stateflow animates its Stateflow diagrams to show you how it responds to events, takes transitions, and changes states. In the animated diagram, active states are highlighted, as shown in the following example:

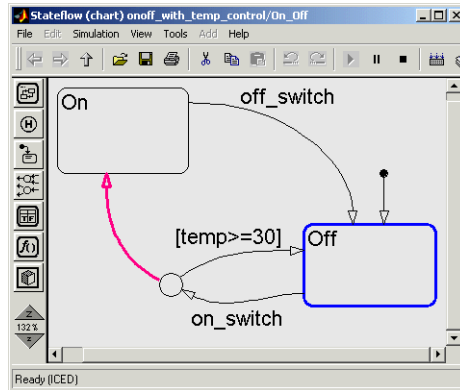


When events occur and transitions are taken, they too are highlighted. Continuing with the previous example, the following figures show what happens when an on_switch event occurs and the value of temp is less than 30:

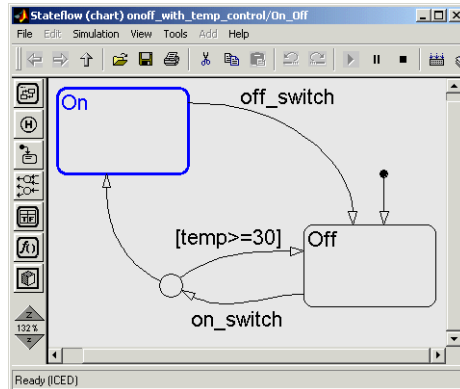
- 1 The transition to the junction is taken.



- 2 Because temp is less than 30, the transition to the On state is taken.



- 3 The Off state is exited; the On state is entered, becomes active, and stays active.



Generating C Code for Targets

When you are finished with your model, you might want to use the control part of your model as a source for code that you use in the actual controller for an actual physical plant. In this case, you may want to take advantage of Stateflow code generation.

Stateflow generates code for targets. A target is an object that belongs to a Stateflow machine. A machine is a container for all the Stateflow blocks in a Simulink model. A target object functions as a container for the generated code from the Stateflow blocks in a Stateflow machine.

There are three types of targets for a Stateflow machine, which are described in the following topics:

- “Simulation Targets” on page 1-14
- “Real-Time Workshop Targets” on page 1-14
- “Custom Targets” on page 1-16

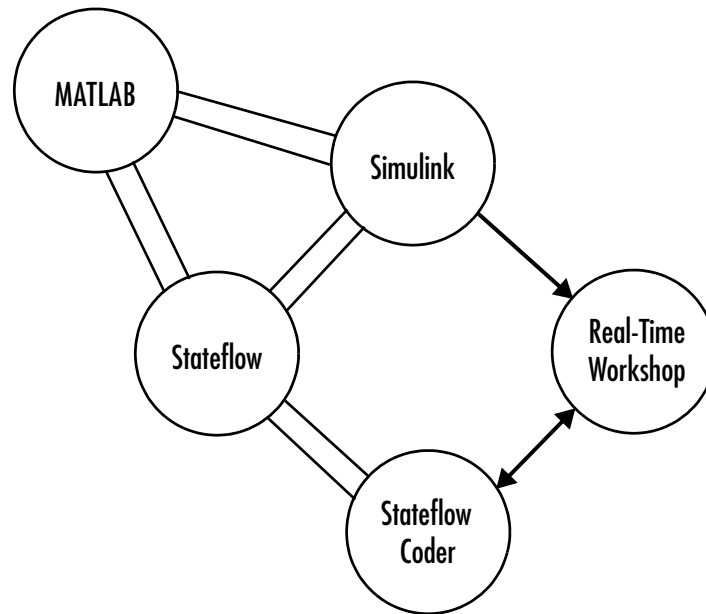
Simulation Targets

When you use Stateflow for simulation, Stateflow generates an S-function (MEX-file) for each Stateflow machine to support model simulation of Stateflow diagrams. The name for a simulation target in Stateflow is `sfun`.

Stateflow simulates its diagrams with generated code. Stateflow generates its own C-code to simulate Stateflow diagrams during simulation. When you generate code for simulation, this is referred to as generating a simulation target whose name is `sfun`.

Real-Time Workshop Targets

Real-Time Workshop[®] coordinates code generation from Simulink with Stateflow code generation to build an embeddable target, which is depicted in the following diagram.



Stateflow Coder generates integer or floating-point code based on the Stateflow machine. Real-Time Workshop generates code from the Simulink portion of the model and provides a framework for running generated Stateflow code in real time. The code generated by Stateflow Coder is seamlessly incorporated into code generated by Real-Time Workshop. You might want to design a solution that targets code generated from both products for a specific platform. If you have the Real-Time Workshop tool, you can take code from Simulink and Stateflow and run it as an application in another environment to control a process. For more information, see the Real-Time Workshop documentation.

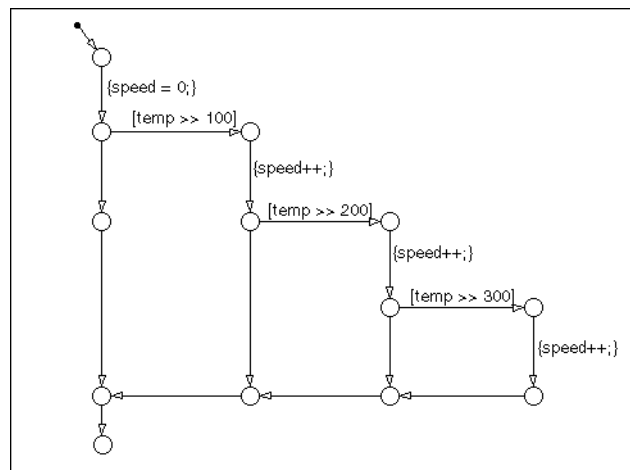
Custom Targets

If you have Stateflow Coder, you can generate code for target applications that you build in other environments, such as an embedded environment. Use custom targets with Stateflow Coder to generate code exclusively for the Stateflow machine portion of the Simulink model. Custom targets are uniquely named as something other than `sfun` or `rtw` in Stateflow. For more information, see “Configuring a Custom Target in Stateflow” in your Stateflow documentation.

Ways That You Can Use Stateflow

Here are a few of the ways that you can use Stateflow diagrams in your Simulink models.

- **Simulate plant control.** — The primary use of Stateflow is to simulate the control of control objects in a physical plant. For example, an automobile engine needs a fan to blow air through a radiator to cool water for cooling an engine. You model the engine with Simulink. You add a Stateflow block to monitor engine temperature and activate the fan and water pump for the engine.
- **Choose state equations for simulation.** — In models that simulate complex behavior based on complex physical relationships, you might need to change the equations that you use in response to plant circumstances. For example, if you are using Simulink to model materials during a collision, you might want to add a Stateflow block to change the equations of state depending on current values for stress, strain, and deformation.
- **Program complex logic visually.** — You can use Stateflow as a visual programming environment with flow diagrams that use only junctions and transitions. For example, the following Stateflow diagram performs the equivalent of a set of cascading if-then statements:



During simulation of your model, you can animate your visual programs and watch them execute. You can slow down execution or place breakpoints at different points in the program to examine outcomes, such as the value of data variables.

Here are some types of applications that benefit directly from the use of Stateflow:

- Embedded systems
 - Avionics (planes)
 - Automotive (cars)
 - Telecommunications (for example, routing algorithms)
 - Commercial (computer peripherals, appliances, and so on)
 - Programmable logic controllers (PLCs) (process control)
 - Industrial (machinery)
- Man-machine interface (MMI)
 - Graphical user interface (GUI)
- Hybrid systems
 - Air traffic control systems (digital signal processing [DSP] + control + MMI)

Demonstration examples have already been created for several of these application types. Once you learn Stateflow, be sure to examine the Stateflow demo models. See “Where to Go from Here” on page 5-13 for instructions on accessing the Stateflow demo models.

Before You Get Started

Before you start learning how to use Stateflow, check the following requirements:

- “Required User Knowledge Level” on page 1-19
- “Required and Optional Software” on page 1-19
- “Stateflow Installation” on page 1-20
- “Setting Up Your Own Target Compiler” on page 1-21
- “Using Stateflow on a Laptop Computer” on page 1-22

Required User Knowledge Level

Users of this Getting Started guide are presumed to have no knowledge of Stateflow. However, a minimum level of Simulink knowledge is required to draw the Simulink diagrams in the examples included in this guide.

Required and Optional Software

Stateflow is a multiplatform product that runs on Microsoft Windows and UNIX systems. Stateflow 6 requires the installation of the following software:

Required Product	Description
MATLAB	The Language of Technical Computing
Simulink	Design and simulate continuous- and discrete-time systems
C or C++ compiler	Used to compile Stateflow generated code for simulation. See “Setting Up Your Own Target Compiler” on page 1-21 for more information.

You can enhance the operation of Stateflow with the installation of the following optional software:

Optional Product	Description
Simulink Report Generator	Automatically generate documentation for Simulink and Stateflow models
Stateflow Coder	Generate C code from Stateflow diagrams
Real-Time Workshop	Integrate generated code from Stateflow diagrams with generated code for the Simulink elements of the model to create an embeddable application

For more information about any of these products, see one of the following:

- The online documentation for that product if it is installed
- The MathWorks Web site at www.mathworks.com; see the “products” section
- The Stateflow Web site at www.mathworks.com/products/stateflow

Stateflow Installation

Your platform-specific MATLAB installation documentation provides essentially all the information you need to install Stateflow.

Prior to installing Stateflow, you must obtain a License File or Personal License Password from The MathWorks. The License File or Personal License Password identifies the products you are permitted to install and use.

Stateflow and Stateflow Coder have certain product prerequisites that must be met for proper installation and execution.

Licensed Product	Prerequisite Product	Additional Information
Simulink 6	MATLAB 7 (Release 14)	Allows installation of Simulink and Stateflow in Demo mode.

Licensed Product	Prerequisite Product	Additional Information
Stateflow 6	Simulink 6	N/A
Stateflow Coder 6	Stateflow 6	N/A

If you experience installation difficulties and have Web access, look for license manager and installation information on the MathWorks support page at www.mathworks.com/support.

Setting Up Your Own Target Compiler

Building Simulink targets for Stateflow generated code requires a C compiler that MATLAB supports. The Microsoft Windows version of Stateflow comes with a C compiler (`lcc.exe`) and a make utility (`lccmake`). Both tools are installed in the directory `matlabroot\sys\lcc`. If you do not configure MATLAB to use any other compiler, Stateflow uses `lcc` to build targets.

For other compilers and for all non-Windows platforms, do the following:

- 1 Install the C compiler you want Stateflow to use to build targets on your system.

You can use any compiler supported by MATLAB for building MATLAB extension (MEX) files. See “Building MEX-Files” in the MATLAB External Interfaces documentation for information on C compilers supported by MATLAB.

Note Stateflow supports building targets with Microsoft Visual C/C++ 5.0 only if you have installed the Service Pack 3 updates for that product.

- 2 At the MATLAB prompt, type

```
mex -setup
```

to initiate prompts for entering the appropriate information about your compiler.

Stateflow uses the compiler that you specify to build MEX-files for building a Stateflow simulation target.

Using Stateflow on a Laptop Computer

If you plan to run the Microsoft Windows version of Stateflow on a laptop computer, you should configure the Windows color palette to use more than 256 colors. Otherwise, you can experience unacceptably slow performance.

To set the Windows graphics palette:

- 1 Click the right mouse button on the Windows desktop to display the desktop menu.
- 2 Select **Properties** from the desktop menu to display the Windows **Display Properties** dialog.
- 3 Select the **Settings** panel on the **Display Properties** dialog.
- 4 Choose a setting that is more than 256 colors from the **Color Palette** colors list.
- 5 Click **OK** to apply the new setting and dismiss the **Display Properties** dialog.

Controlling with States and Transitions

This chapter teaches you about Stateflow by guiding you through the creation and execution of a simple control model. This model consists of a single Stateflow block and some supporting Simulink blocks. The contents of the Stateflow block is a simple Stateflow diagram. As you proceed through the following sections, you improve both the Stateflow diagram and the Simulink model that calls it. In the process, you increase your knowledge and understanding of Stateflow states and transitions.

Building an On-Off Control Model
(p. 2-2)

Get started quickly building a control model with Stateflow.

Simulating a Stateflow Diagram
(p. 2-22)

Simulate the control model normally and with the **Stateflow Debugging** window.

Guarding Transitions with Event Triggers (p. 2-37)

Guard the transitions of a Stateflow diagram to execute only with the occurrence of specific events.

Modifying Output Data with Actions
(p. 2-51)

Send modified data from a Stateflow diagram to the Simulink model.

Simulating Event Triggers and Modified Output Data (p. 2-56)

Simulate the guarded transitions and modified output data of the improved control model.

Building an On-Off Control Model

In this section, you start learning how to use Stateflow by creating a Simulink model with a single Stateflow block that provides on-off control. This model is the basis for further improvements that you add and test in the remainder of this Getting Started guide. As you continue to build and improve the growing model, you learn how to use Stateflow in Simulink models.

Use the following procedure topics to create a simple control model with on-off control:

- 1 “Creating a Simulink Model with a Stateflow Block” on page 2-3 — Start a model in Simulink containing a Stateflow block.
- 2 “Simulating a Stateflow Diagram” on page 2-22 — Save and name the Simulink model as soon as you create it.
- 3 “Opening the Diagram for a Stateflow Block” on page 2-6 — Open the Stateflow diagram for the Stateflow block in the Simulink model.
- 4 “Drawing States in Stateflow Diagrams” on page 2-7 — Start editing the Stateflow diagram by drawing states.
- 5 “Drawing Transitions in a Stateflow Diagram” on page 2-10 — Draw transitions between the states of the Stateflow diagram.
- 6 “Adding a Trigger Event to the Stateflow Diagram” on page 2-14 — Define the input event from Simulink that triggers the execution of the Stateflow diagram in the Simulink model.
- 7 “Sending a Trigger Event to the Stateflow Chart” on page 2-16 — Program the Simulink model to send a trigger event to test the execution of the Stateflow diagram.

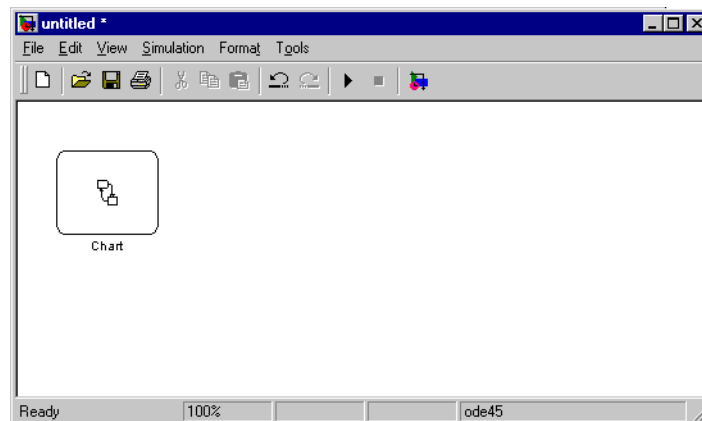
Creating a Simulink Model with a Stateflow Block

In Simulink, you use blocks to create a simulated model of a physical plant. Stateflow blocks are a part of Simulink models. You use Stateflow blocks in Simulink models to control objects in the physical plant, for example, a motor, a pump, or a fan.

In this topic, you begin the process of building an example model that uses Stateflow on-off control. Create and save a Simulink model with a single Stateflow block in it using the following steps:

- 1 At the MATLAB prompt, enter `sfnew`.

Stateflow displays the following untitled Simulink model window with a Stateflow block labeled Chart.

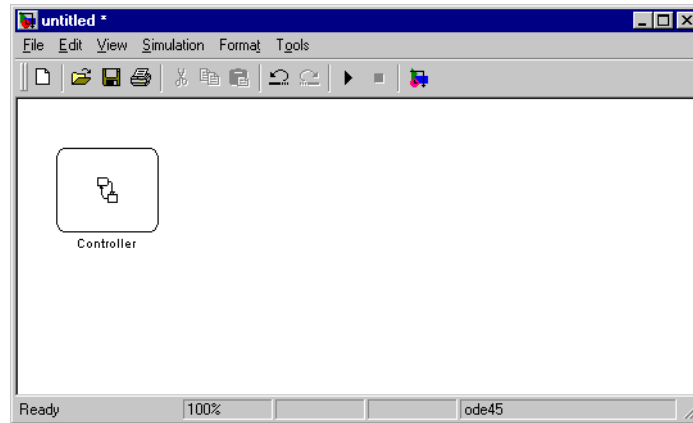


- 2 In the Simulink window, under the Stateflow block, click in the text Chart.

A text cursor appears for you to edit the label of the block.

- 3 Replace the text Chart with the text Controller and click outside of the block label when finished.

This gives the Stateflow block the new name Controller, as shown.



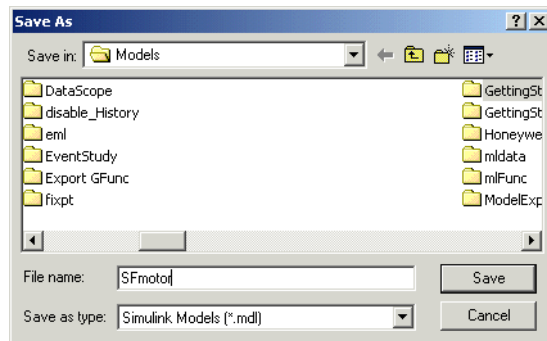
Now that the Stateflow block is labeled Controller, you refer to this block in the Simulink model as the “Controller Stateflow block” or just the “Controller block.”

Saving a Simulink Model

You start building a Simulink model in “Creating a Simulink Model with a Stateflow Block” on page 2-3. Before going any further, save your model with the following steps:

- 1 In the Simulink model window, from the **File** menu, click **Save As**.

The **Save As** dialog appears.



- 2 In the **File name** field, enter the file name SFcontro11.

The name of the file in which you save the model is the name of the Simulink model, which is now SFcontro11.

- 3 In the **Save in** field and the pane below, select a directory in which to save the new Simulink model.

- 4 Click the **Save** button to save the model.

When you save your model, Stateflow creates an `sfprj` directory in the directory holding the project file. This directory is used to hold model information.

Once you have saved your model file the first time, you can save it again by selecting **Save** from the **File** menu.

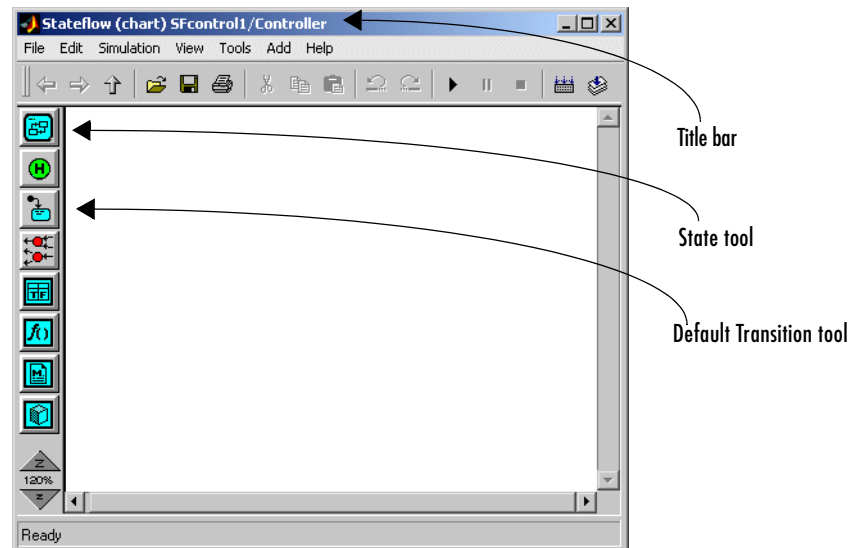
Opening the Diagram for a Stateflow Block

You use Stateflow blocks to program the control of an object in the physical plant that you model in Simulink. For example, you might want the Stateflow block to decide if plant pressure or temperature is too high and make a decision to turn on a motor-driven device to reduce plant temperature or pressure. You program this control in a Stateflow diagram for a Stateflow block in Simulink.

Open the Stateflow diagram for the Controller block as follows:

- 1 In the Simulink window, double-click the Controller block.

An empty Stateflow diagram editor window appears, as shown.




In the empty window you just opened, notice the following:

- The name of the Stateflow block for the diagram you are editing (Controller) appears in the title bar of the window preceded by its owning Simulink model (SFcontrol1).
- The toolbar on the left is for drawing the different graphical objects that compose a Stateflow diagram. Take note of the drawing tools that you use in drawing a Stateflow diagram for the Controller block in this section: the State tool and the Default Transition tool.

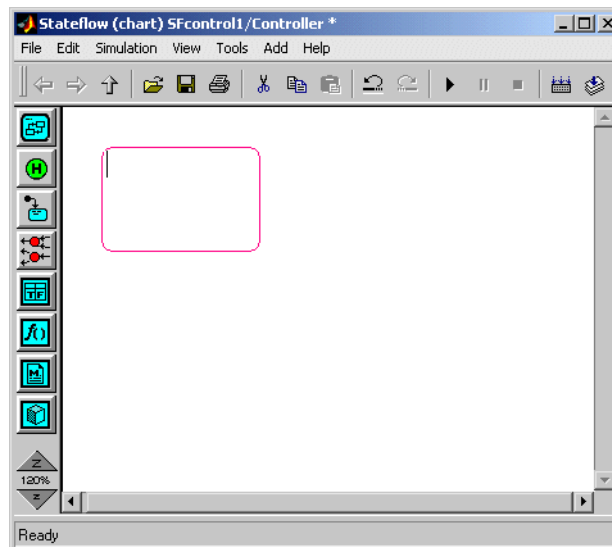
Drawing States in Stateflow Diagrams

You program the control behavior for a Stateflow block in its Stateflow diagram. In “Opening the Diagram for a Stateflow Block” on page 2-6, you open the empty Stateflow diagram for the Controller Stateflow block. Now you need to graphically program it with its control behavior.

States are objects in a Stateflow diagram that represent modes of operation for a control device. In this example, a control device has two states of operation: on and off. Start programming the Stateflow diagram for the Controller block by drawing the states On and Off in the following steps:

- 1 In the drawing toolbar on the left, click the **State** tool .
- 2 Move the cursor into the drawing area and notice that the cursor takes the shape of a box with rounded corners.
- 3 In the upper-left corner of the drawing area, click to place a new state, as shown.

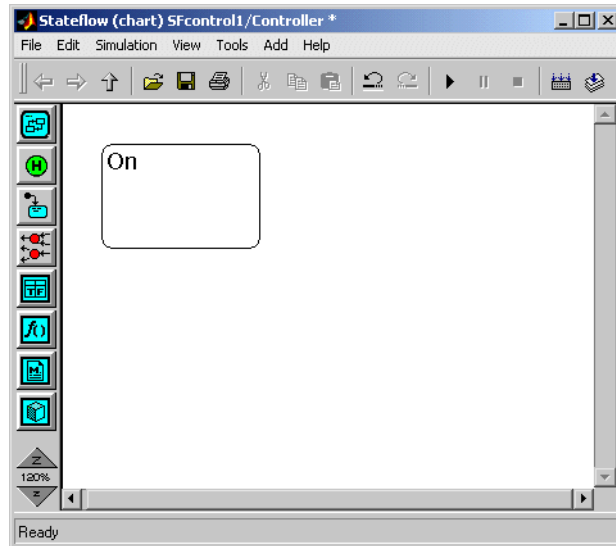
A highlighted state appears with a blinking text cursor in the upper-left corner of the state, as shown.



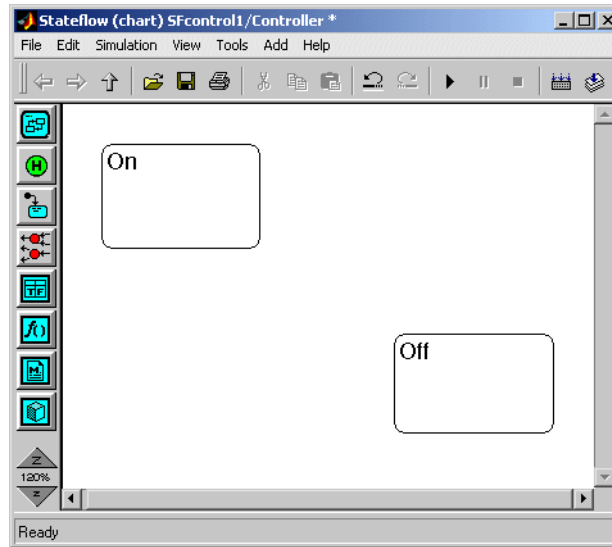
- 4 Enter the text On and click outside of the state.

This gives the new state the name On. You enter the name for a state at the beginning of its label.

The Stateflow diagram editor should now have the following appearance:



- 5 Use the preceding steps to draw and label a new state in the lower-right corner of the Stateflow diagram editor with the name Off, as shown.



Stateflow diagrams are a form of a finite state machine. Finite state machines use states to represent the possible control modes of an object. You have just created two states named On and Off. These two states represent the two control modes for a single-speed fan: on and off. But they can also represent the control modes of a host of other devices that can be on or off, such as a fan, a heater, and so on.

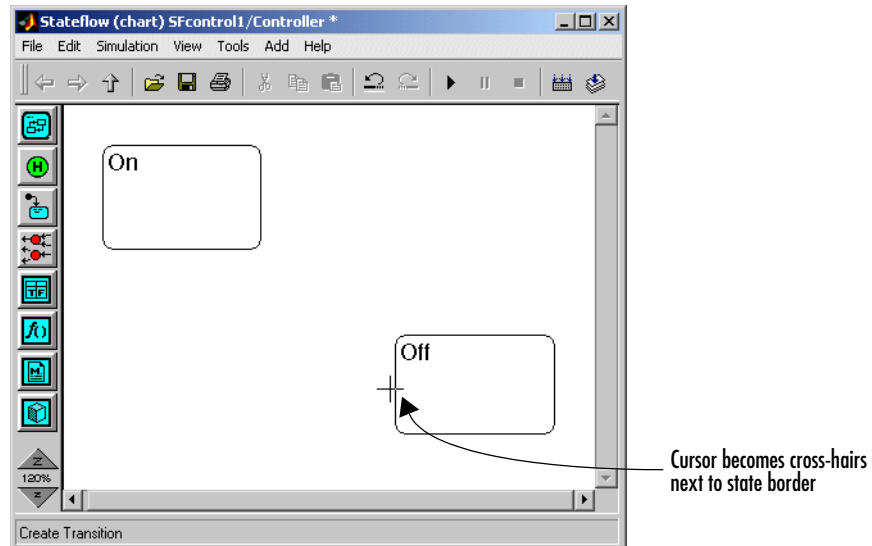
Drawing Transitions in a Stateflow Diagram

States in a Stateflow diagram can be active or inactive. In “Drawing States in Stateflow Diagrams” on page 2-7, you draw an On state and an Off state for the SFcontrol1 model. An active Off state indicates the device you are controlling is off, and an active On state indicates that the device is on.

Because a device such as a motor or a fan cannot be on and off at the same time, only one state can be active at a time. The diagram needs to define a way to change the control state of the device from Off to On and from On to Off.

Transitions in a Stateflow diagram define a path along which an active state can become inactive and another state can become active. In this topic, you draw transitions between the Off state and the On state with the following steps:

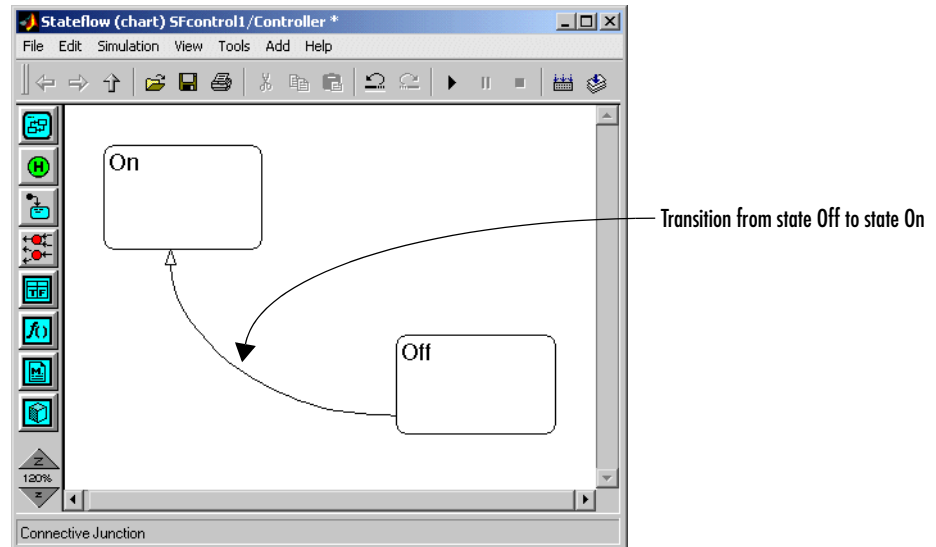
- 1 Position the cursor at a straight portion of the left border of the Off state as shown.



Notice that cross-hairs appear in place of the cursor. Cross-hairs do not appear if you place the cursor on the corner of a state. There, an angled double-arrow appears for resizing the state. Corners are used exclusively for resizing.

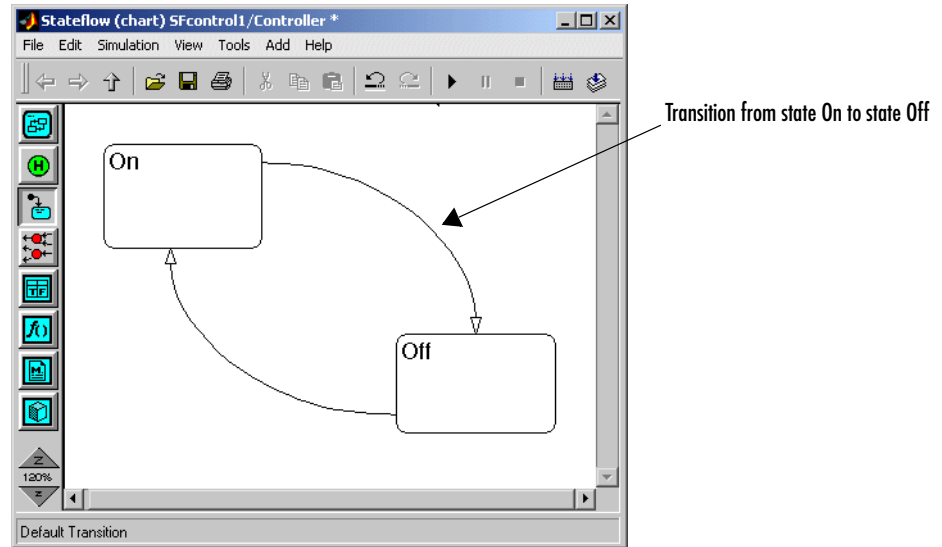
- 2 When the cursor changes to cross-hairs, click-drag the mouse to the bottom border of the On state and release the mouse.

The Stateflow diagram now has a transition similar to the following:




Notice that a transition is directional. This transition proceeds from the Off state to the On state. This means that if the Off state is active, the On state can become active and the Off state inactive through this transition.

- 3 Use the preceding steps to draw a transition from the right side of the state On to the top of the state Off as shown.



This transition proceeds from the On state to the Off state. This means that if the On state is active, the Off state can become active and the On state inactive through this transition.

You need to add one last transition to tell the Stateflow diagram which of the two states, Off or On, is active when the Stateflow diagram starts operating. You use a default transition to specify the Off state as the first active state with the following steps:

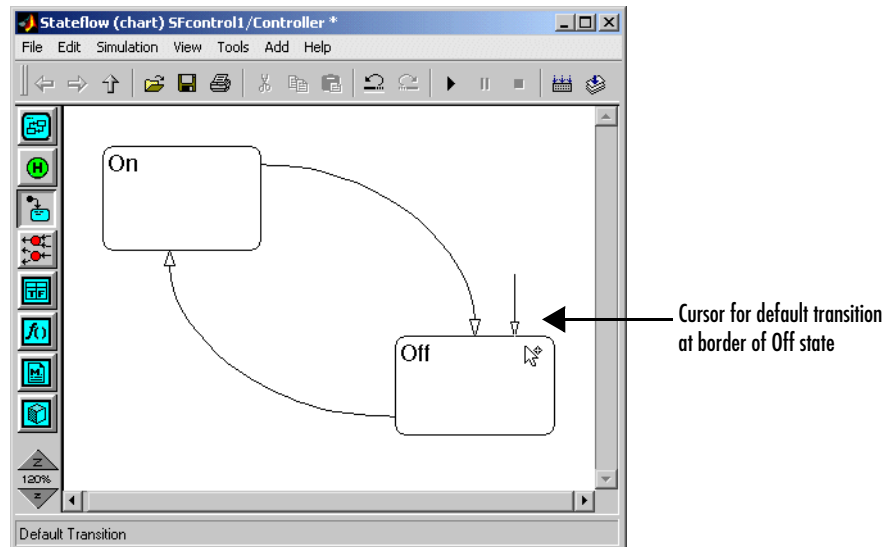
4 In the drawing toolbar, click the Default Transition tool .

5 Move the cursor into the drawing area.

Notice that the cursor is an arrow pointing down and to the right.

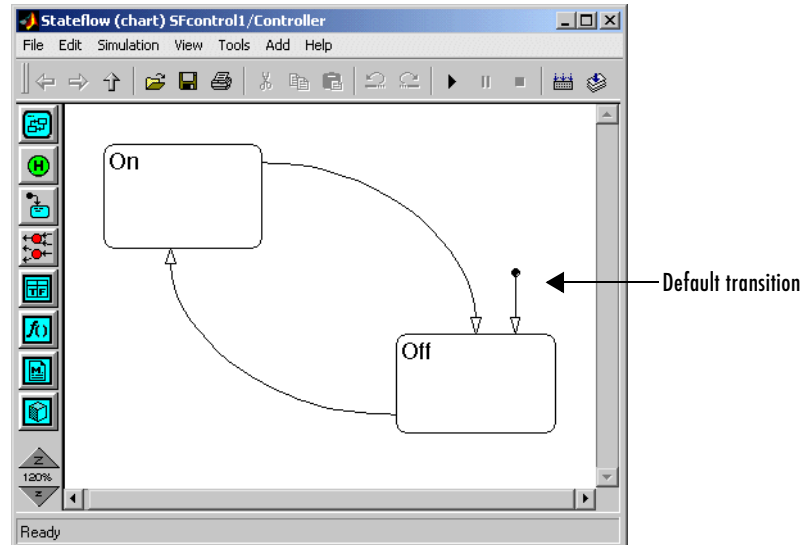
6 Move the cursor to the right side of the top border of the Off state.

When the cursor's arrowhead touches the top border of the Off state, it snaps straight up and down, as shown.



- 7 Release the mouse and click an empty part of the diagram.

You should now see a default transition pointing to the Off state, as shown.



Adding a Trigger Event to the Stateflow Diagram

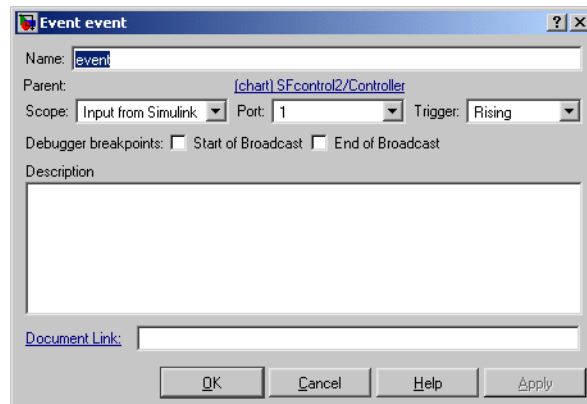
A Stateflow diagram updates (executes) only in response to an event that it receives from Simulink. In “Drawing Transitions in a Stateflow Diagram” on page 2-10 you provide transitions for the On and Off states to alternately become active and inactive during execution of the Controller Stateflow diagram. Now you need to provide events to Stateflow to update the Stateflow diagram and power its transitions.

In this topic, you define *trigger events* for the example Stateflow diagram. There are other ways to send events that update Stateflow diagrams. For example, you can tell the Stateflow diagram to execute every time Simulink samples the model. However, trigger events give you the greatest individual control over individual events sent to a Stateflow diagram, and that makes them the best events for studying Stateflow behavior.

Use the following steps to define a trigger event for updating the Controller diagram:

- 1 In the Stateflow diagram editor, from the **Add** menu, select **Event**.
- 2 In the resulting submenu, select **Input from Simulink**.

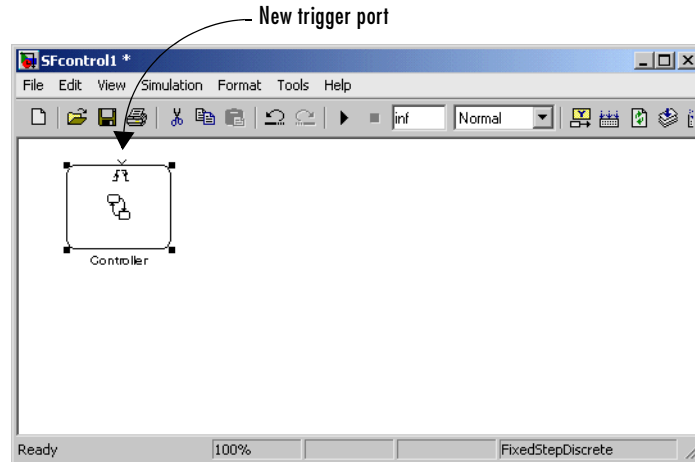
The property dialog for the new event appears.



By default, the name of the event is set to event in the **Name** field of the **Event properties** dialog.

- 3 Select **Either Edge** as the **Trigger** type.
- 4 Click **OK** to apply the changes and close the **Event properties** dialog.

The Stateflow block in Simulink now has a trigger port for the event event as shown.

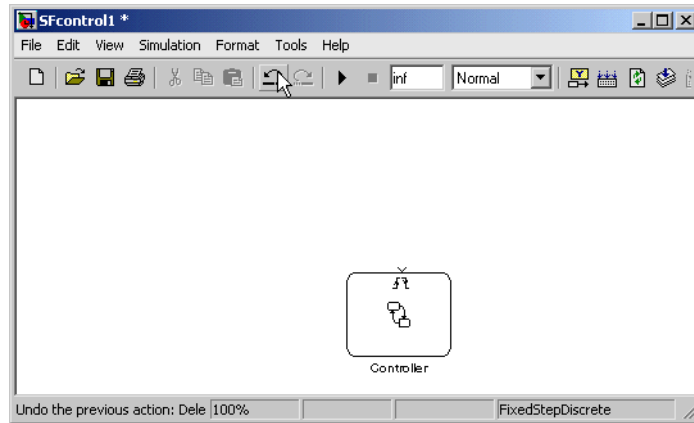


Sending a Trigger Event to the Stateflow Chart

You define a trigger event for the Controller diagram in “Adding a Trigger Event to the Stateflow Diagram” on page 2-14. The Controller block in Simulink now has a trigger port for the event event. To update the Controller diagram, the Simulink model must send a signal to the trigger port to trigger the event in the Controller diagram.

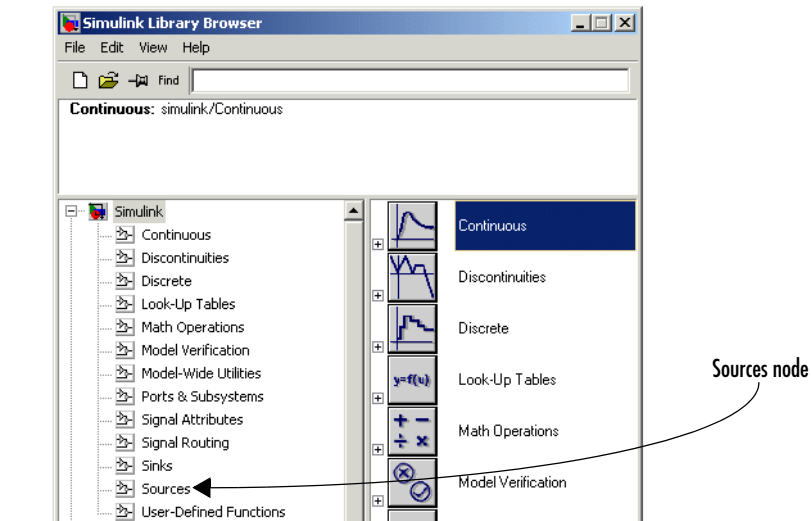
You defined the trigger event event with an **Either Edge** trigger for the Controller diagram. This event occurs when a control signal attached to the trigger port for the Controller diagram rises or falls in value as it passes through zero. In the following steps, use a manual switch to provide a control signal from Simulink that rises or falls to trigger the event event in the Controller diagram.

- 1 In the Simulink window, click-drag the Controller block down and to the right as shown.



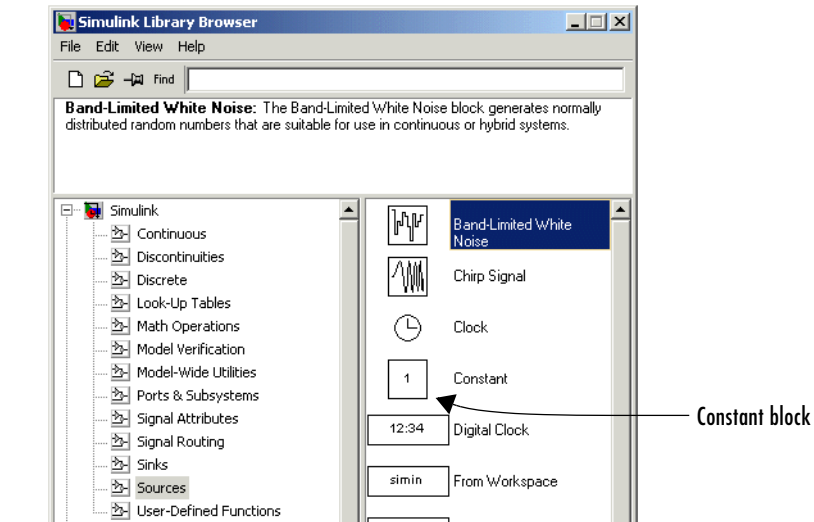
- 2 From the Simulink **View** menu, select **Library browser**.

The **Simulink Library Browser** window opens with the **Simulink** node expanded.



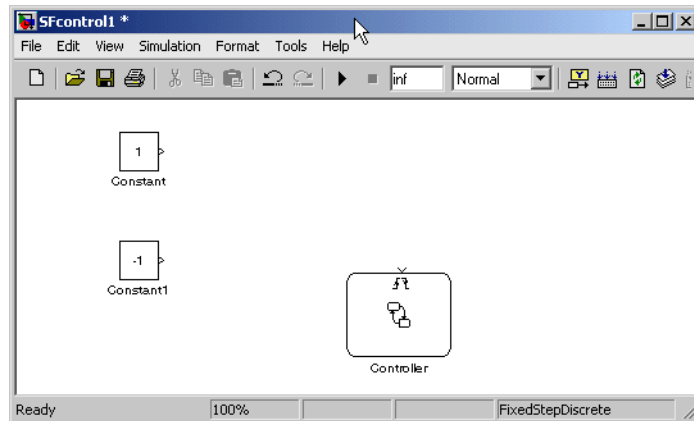
- 3 Under the **Simulink** node, select the **Sources** node.

The right pane of the **Simulink Library Browser** window displays the block members of the **Sources** library.



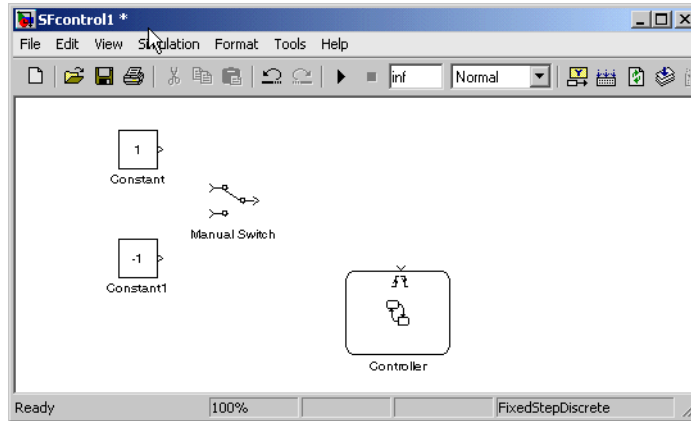
- 4 From the right pane of the **Simulink Library Browser** window, click-drag the Constant block to the left of the Stateflow block in the Simulink model.

- 5 Add another Constant block to the left of the Controller as shown.

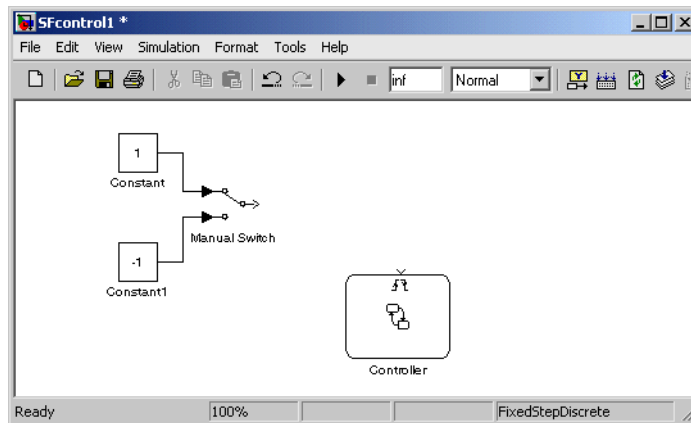


- 6 Double-click the bottom Constant block.
- 7 In the resulting **Block Parameters** dialog, in the **Constant value** field, enter a value of -1 and click **OK** to close the dialog.
- 8 In the **Simulink Library Browser** window, select the **Signal Routing** node.

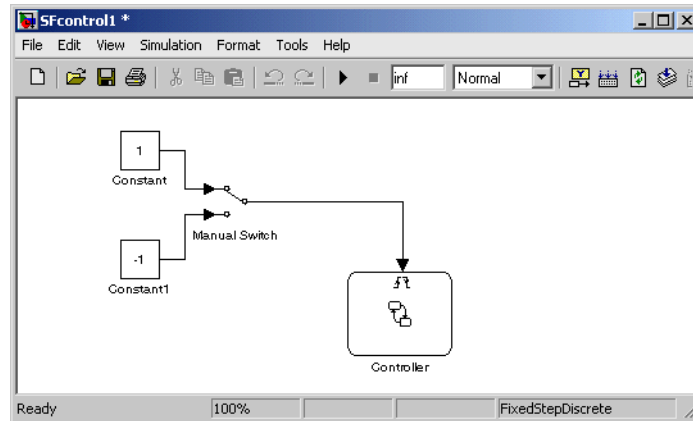
- 9 From the contents of the **Signal Routing** library in the right pane of the **Simulink Library Browser** window, click-drag a Manual Switch block to the right of the Constant blocks as shown.



- 10 Connect the Constant blocks to the poles of the Manual Switch as shown.



- 11 Connect the output of the Manual Switch to the Controller block trigger port as shown.



During testing, you change the pole position of the Manual Switch (from -1 to 1 or from 1 to -1). This sends a rising or falling signal to the Controller diagram that crosses zero. Because you defined the event in the Controller diagram with an **Either Edge** trigger, either signal is recognized by the Controller diagram as the event event, and the diagram executes.

- 12 Save the model (as SFcontrol1).

Simulating a Stateflow Diagram

After you build the Simulink model described in “Building an On-Off Control Model” on page 2-2, you naturally want to test it. You test Simulink models in a process called *simulation*. During simulation, Stateflow diagrams generate a graphical display that informs you of the transition that is executing and the currently active state. In this way, you can find out if a Stateflow diagram is behaving the way you expect it to, and make appropriate changes, if required.

Use the following procedure topics to simulate the Controller Stateflow diagram in the SFcontrol1 model:

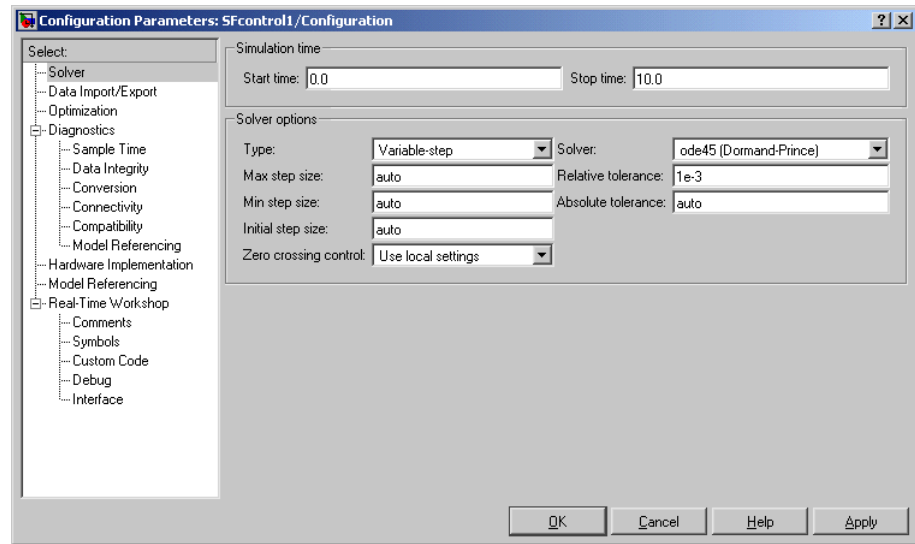
- 1 “Setting Up Diagram Simulation” on page 2-22 — Define simulation parameters for the Simulink model.
- 2 “Simulating a Stateflow Diagram” on page 2-25 — Go through the steps of simulating your model.
- 3 “Using the Debug Tool During Simulation” on page 2-32 — Use the debug tool to slow down and gain step-by-step control of simulation.

Setting Up Diagram Simulation

By default, simulation is enabled in Simulink and Stateflow models. To be sure that it is enabled for the SFcontrol1 model, use the following steps to set up simulation for the Controller Stateflow diagram:

- 1 In the Stateflow diagram window, from the **Simulation** menu, choose **Configuration Parameters**.

The Simulation Parameters dialog appears as shown.

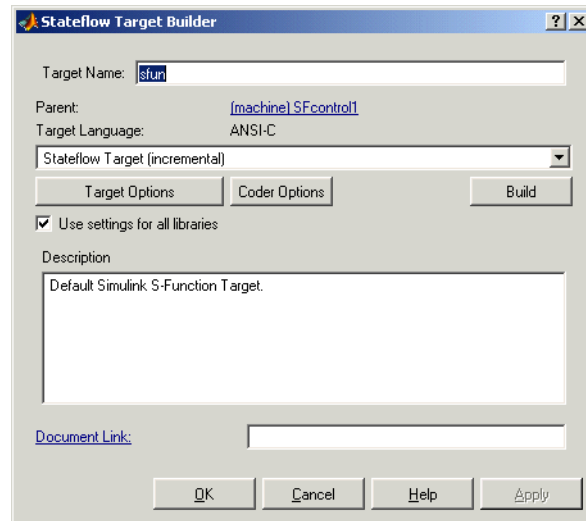


- 2 In the **Stop time** field, enter `inf` and click **OK** to apply the changes and close the dialog.

The stop time value of `inf` tells Simulink to simulate your model indefinitely until you tell it to stop.

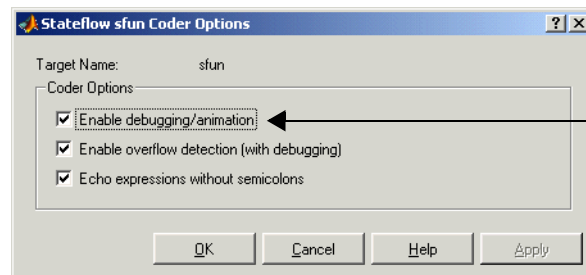
- 3 In the Stateflow diagram editor, from the **Tools** menu, select **Open Simulation Target**.

The **Simulation Target Builder** dialog appears as shown.



- 4 In the **Simulation Target Builder** dialog, click **Coder Options**.

The **Simulation Coder Options** dialog appears as shown.

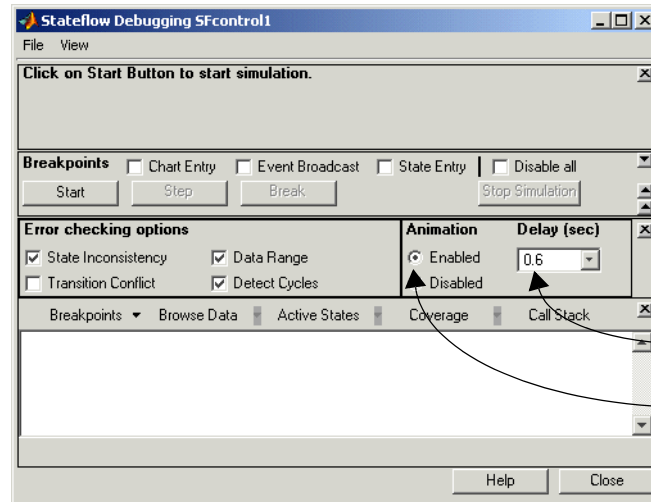


Make sure this option is checked

- 5 Make sure that the check box to **Enable debugging/animation** is selected, then click **OK** to apply the change and close the **Simulation Coder Options** dialog.
- 6 In the remaining **Simulation Target Builder** dialog, click **OK** to close it.

- 7 From the Stateflow diagram editor **Tools** menu, select **Debug**.

The **Stateflow Debugging** dialog appears as shown.



Select a delay value of 0.6

Make sure **Animation** is set to **Enabled**

- 8 In the **Animation** section, make sure that the **Enabled** radio button is selected and the **Delay (sec)** field is set to **0.6**.

Entering a larger value in the **Delay (sec)** field slows down simulation. A smaller value speeds simulation up.

- 9 Click **Close** to apply the change and close the **Stateflow Debugging** window.

Simulating a Stateflow Diagram

Stateflow allows you to observe the behavior of Stateflow diagrams in Simulink models during simulation of the model. Before you simulate a Stateflow diagram, you make sure that simulation is enabled for your model as you did in “Setting Up Diagram Simulation” on page 2-22.

When you simulate a Simulink model containing Stateflow blocks, Stateflow builds an application for each Stateflow block that graphically executes its Stateflow diagram by highlighting the active state and the executing transition. By simulating your Stateflow diagrams, you understand the behavior that you specify for them, and find out if they are behaving as you expect them to.

Simulate the SFcontrol11 model as described in the following steps:

- 1 Before you begin simulating the SFcontrol11 model, position the Simulink model window and the Controller Stateflow diagram window so that they can both be seen at once with no overlap.
- 2 From the Stateflow diagram editor **Simulation** menu, select **Start** to start simulation of the model and notice the following:

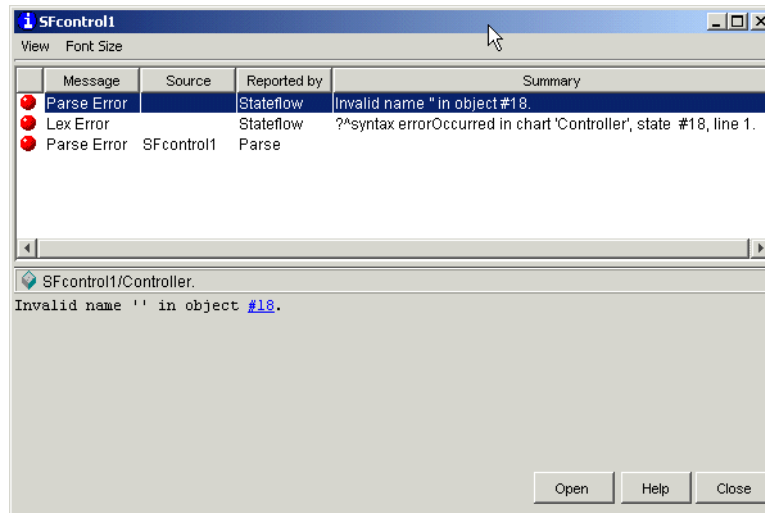
- Stateflow temporarily sets the model to read-only to prevent accidental modification while the model is simulating.

This is displayed as “Ready (ICED)” at the bottom of the Stateflow diagram window, where “ICED” is an internal Stateflow designation.

- Stateflow parses the Controller Stateflow diagram for errors.

Parsing the Stateflow diagram ensures that the notations you specified are valid and correct. If any critical errors are found, they are displayed in a Stateflow **Builder** window and simulation is halted. If no errors are found, the window does not appear, and simulation continues.

The following example window results from an error introduced into the Controller diagram by removing the name of the On state.



Critical error messages are displayed with a red button. You can display the full text of an error message in the bottom pane by clicking on it in the top pane. If you double-click an error message in the top pane, the Stateflow diagram editor appears with the error-causing object highlighted.

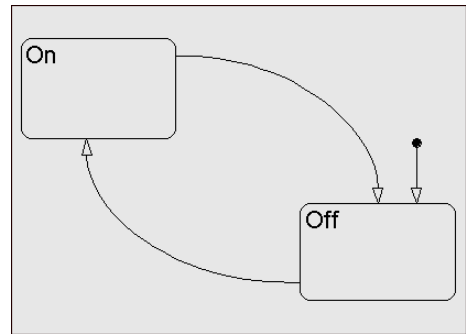
- Stateflow displays status messages in the MATLAB Command Window while it builds the simulation application.

The simulation application that Stateflow generates for simulation is referred to as an `sfun` target. Stateflow creates a directory called `sfprj` in the current MATLAB directory if the directory does not already exist.

Stateflow uses this directory to store the generated files that make up the sfun target.

Simulation commences when the Stateflow diagram editor background becomes shaded as shown.

Stateflow begins in simulation mode. The Controller diagram is active, but nothing in it is.

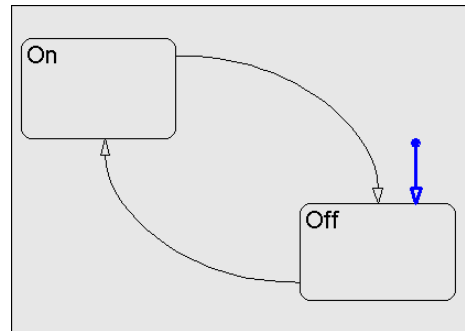


- 3 In the Simulink window, double-click the Manual Switch.

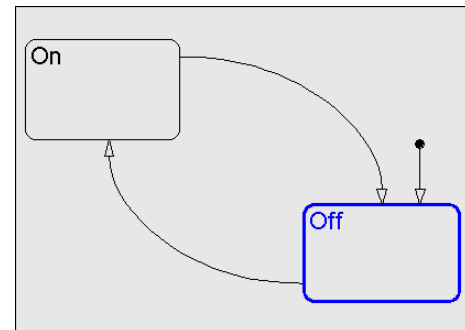
This causes the Manual Switch to switch between its inputs, -1 and 1. Because the trigger event for the Controller block is set to **Either Edge**, changing the switch in either direction (1 to -1 or -1 to 1) sends a falling edge or rising edge signal to the Controller block that recognizes it as the event event.

The event causes the Stateflow diagram to execute and look for the first available transition, which is the default transition into the state Off. You see the following simulation sequence:

The only available transition, the default transition, is taken.



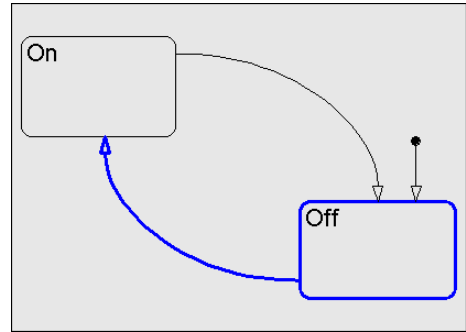
The destination state of the default transition, the state Off, is entered, becomes active, and stays active until another event occurs.



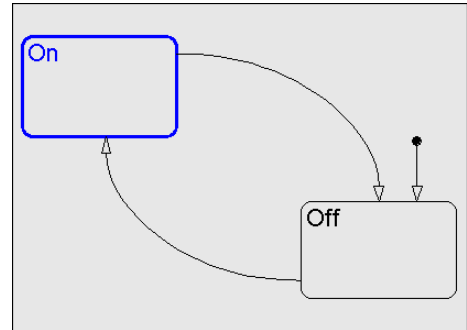
- 4 In the Simulink window, double-click the Manual Switch.

This sends another event event that causes the Stateflow diagram to look for an available transition. You see the following continued simulation behavior:

The only available transition from the active state Off to another state is the transition to the state On. The transition to On is taken.



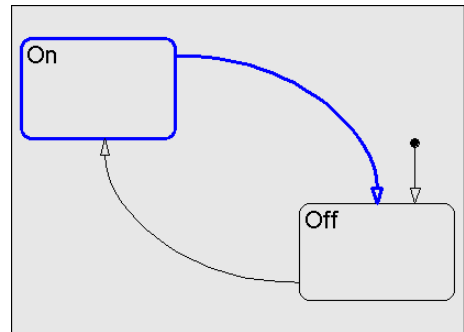
The Off state is exited and the destination state On is entered, becomes active, and stays active.



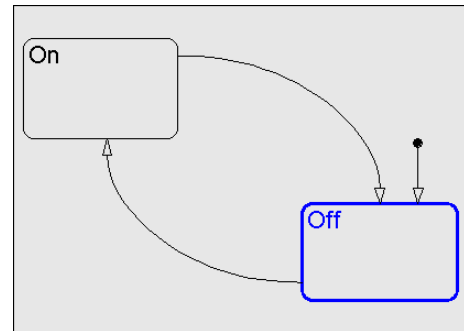
- 5 In the Simulink window, double-click the Manual Switch.

This sends another event event that causes the Stateflow diagram to look for an available transition. You see the following continued simulation behavior:

The only available transition from the active state On is the transition to the Off state. The transition to the state Off is taken.



The On state is exited and the destination state Off, is entered, becomes active, and stays active.



- 6 Continue sending events by double-clicking the Manual Switch in the Simulink window to repeat the sequence in the preceding steps 4 and 5.
- 7 Choose **Stop** from the graphics editor **Simulation** menu to stop the simulation.

Once the simulation stops, Stateflow resets the model to be editable.

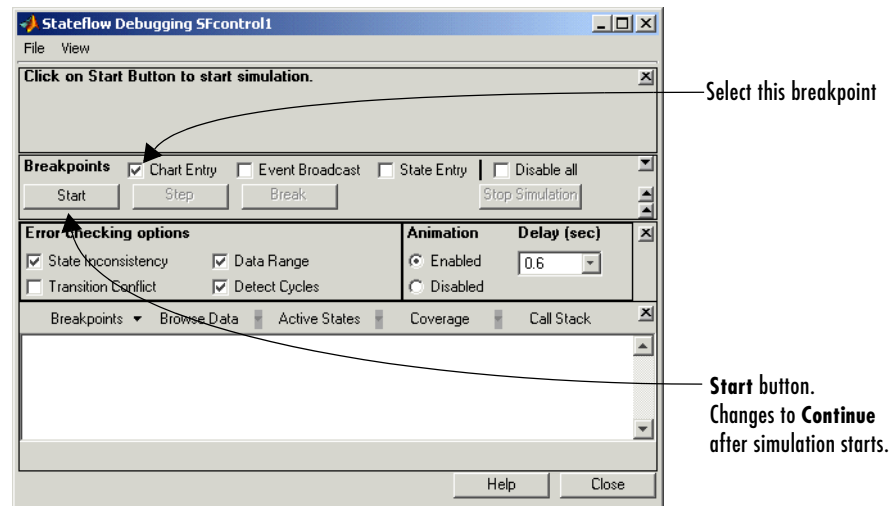
Using the Debug Tool During Simulation

In “Simulating a Stateflow Diagram” on page 2-25, Stateflow highlights executing transitions and active states. You can enhance normal Stateflow simulation with the **Stateflow Debugging** tool. This tool lets you step through individual Stateflow actions that are not identified to you in normal simulation. By identifying all Stateflow diagram actions, the Debugging tool gives you insight into Stateflow behavior, and helps eliminate any doubts you may have about how your Stateflow diagrams work.

The following steps show you how to step through a simulation of the Controller Stateflow diagram using the **Stateflow Debugging** tool:

- 1 In the Stateflow diagram editor, from the **Tools** menu, select **Debug**.

The **Stateflow Debugging** window appears as shown.



- 2 Select the **Chart Entry** breakpoint.

This pauses execution when the diagram is entered after it is triggered by an event from the Manual Switch.

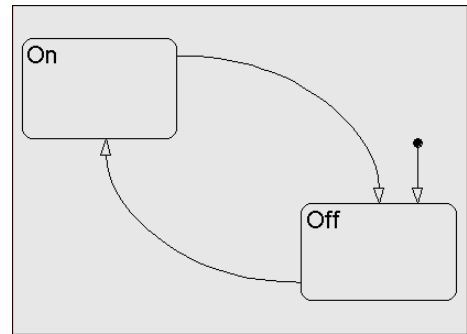
- 3 Position the **Stateflow Debugging** window so that you can see it, the Simulink model window, and the Controller Stateflow diagram window at the same time.

- 4 In the **Stateflow Debugging** window, click **Start** to start the simulation.

When simulation begins, the **Start** button is relabeled to a **Continue** button and both the **Continue** and **Step** buttons are disabled. This means that the Controller diagram is inactive (asleep) and waiting for an event.

Simulation running....

This message appears in the top pane of the Debugging window. It indicates that the simulation is running, but there are no events to process. The Controller diagram is active, but asleep.



- 5 In the Simulink window, double-click the Manual Switch block to send an event to the Controller diagram.

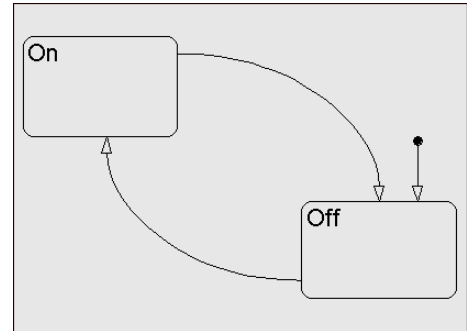
The Debugging window now displays information in its top pane with the following labels:

- **Stopped** — What is being processed in the current step.
- **Executing** — The Stateflow diagram executing. In this example, this will always be Controller.
- **Current Event** — The current event being processed. In this example, this will always be the event event.
- **Simulink Time** — The simulation time of the model.

For now, pay attention only to the **Stopped** message. This message indicates what is taking place in the current simulation step. It is included with the examples that follow.

Stopped: During Chart Controller

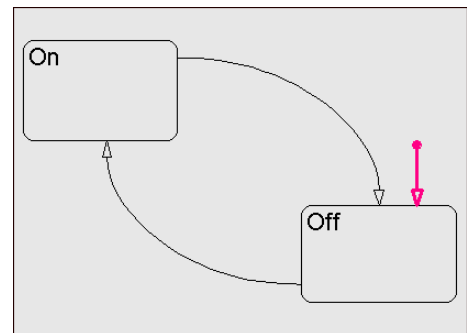
Indicates that this step is taking place while the Controller diagram is active and ready to respond to events.



- 6 In the **Stateflow Debugging** window, click **Step** to advance simulation by a step.

Stopped: Processing Transition?

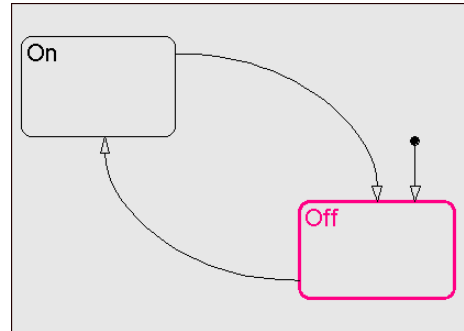
The Controller diagram is processing its default transition. The **Debugging** window refers to the identity of the current transition by its label. Because the Default Transition has no label, its label is referred to with a question mark character (?).



7 Click **Step** to advance simulation by a step.

Stopped: Just after activation of State Off.

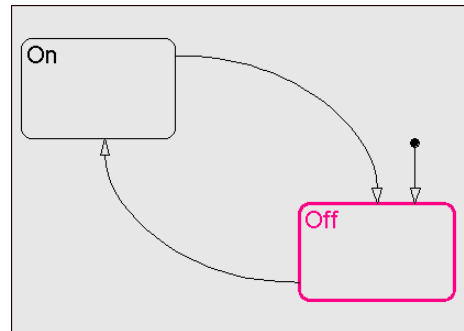
State Off is entered and is now active.



8 Click **Step** to advance simulation by a step.

Stopped: After broadcast of input event event.

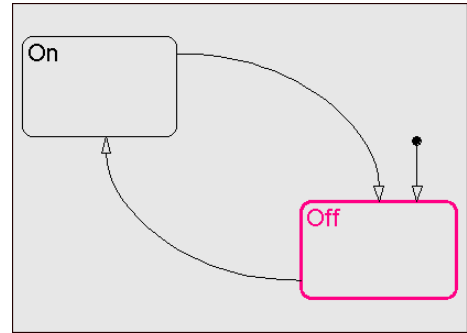
All response activity to the event is finished. The event has been consumed.



- 9 Click **Step** to advance simulation by a step.

Simulation running....

The Controller diagram is active but asleep. In this phase, it is waiting for an event to wake it and be processed. The **Continue** and **Step** buttons are disabled until another event occurs.



- 10 In the Simulink model, continue indefinitely to double-click the Manual Switch block to send another event to the Controller diagram and advance simulation to the next active state with the **Step** button.
- 11 When you are finished debugging, in the **Debugging** window, click **Stop Simulation** to stop the simulation.

Once the simulation stops, the model becomes editable.

- 12 In the **Debugging** window, click **Close** to close it.

The **Stateflow Debugging** window is an important tool for studying Stateflow *semantics*, the description of the behavior of Stateflow objects in Stateflow diagrams. Even though the **Stateflow Debugging** window is not used throughout the remainder of this guide, you are encouraged to use it whenever you want to clarify Stateflow semantics in your models.

Guarding Transitions with Event Triggers

The transitions for the Controller Stateflow diagram in the SFcontro11 model that you build in “Building an On-Off Control Model” on page 2-2 react to the event event. However, these transitions react the same to any event, whether it is named event, x, y, z, or whatever. This means that any event sent to the Controller Stateflow diagram results in taking a transition.

You can restrict transitions to react only to specific events with *event triggers*. In the following procedure topics, you add *event triggers* to the Controller Stateflow diagram and test them with trigger events that you add to the Simulink test interface:

- 1 “Adding Event Triggers to Transitions” on page 2-37 — Use event triggers to restrict the transitions between the On and Off states to the events on_switch and off_switch.
- 2 “Adding Multiple Trigger Events to a Stateflow Chart” on page 2-43 — Add the events on_switch and off_switch to the Stateflow diagram.
- 3 “Sending Multiple Trigger Events to a Stateflow Chart” on page 2-47 — Send the on_switch and off_switch events to the Controller Stateflow diagram from the Simulink interface for testing.

Adding Event Triggers to Transitions

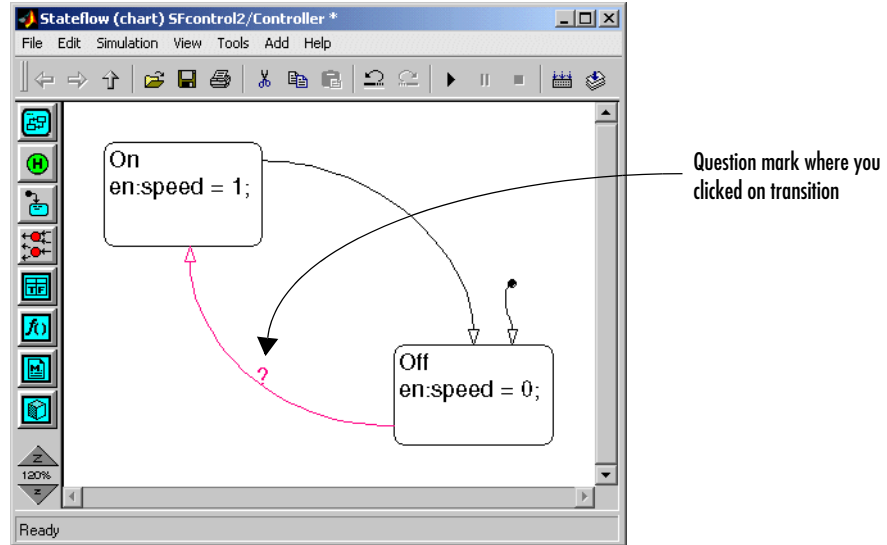
If you want a transition to respond only to a specific event, you specify an event trigger for the transition in its label. An event trigger limits the response of its transition to the occurrence of the named event.

In this topic, you add an event trigger to each of the transitions between the On and Off states of the Controller Stateflow diagram with the following steps:

- 1 If not already loaded, load the Simulink model SFcontro11 you build in “Building an On-Off Control Model” on page 2-2 and save it as SFcontro12.

- 2 Click the transition from the state Off to the state On.

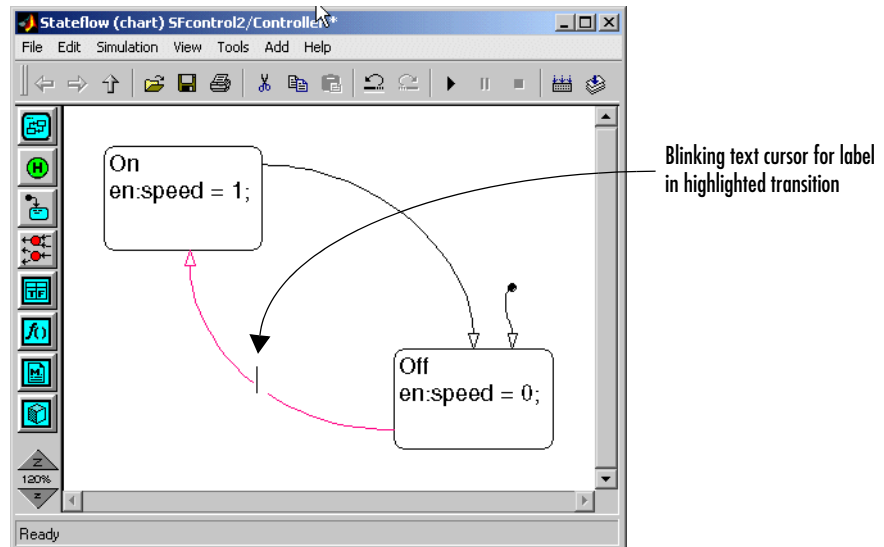
The transition highlights to show that it is selected. Also, a question mark character (?) appears next to the point on the transition that you clicked as shown.



- 3 Place the mouse cursor over the question mark character.

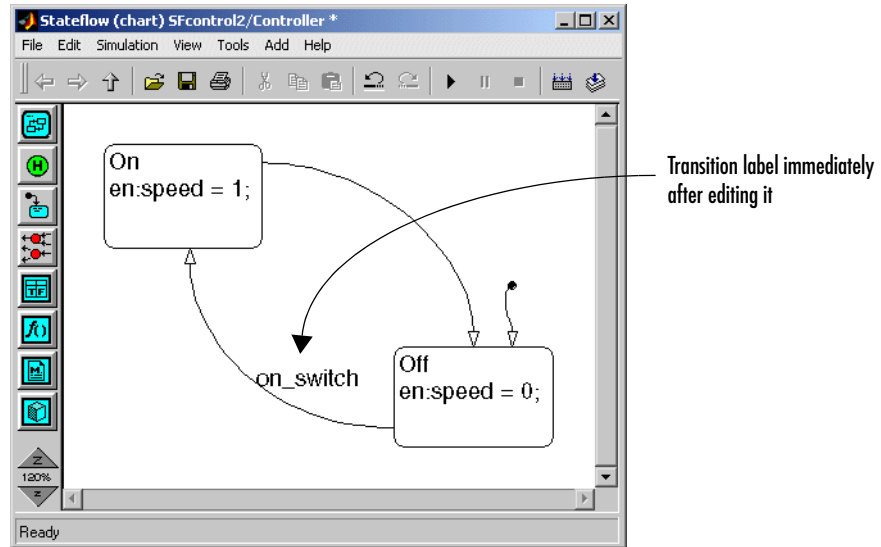
- 4 When the mouse cursor changes to a text cursor, click the mouse.

A blinking text cursor appears in place of the question mark, as shown.



- 5 Type the text `on_switch` and click a location outside the transition.

You should now see the following label for the transition from the state Off to the state On.

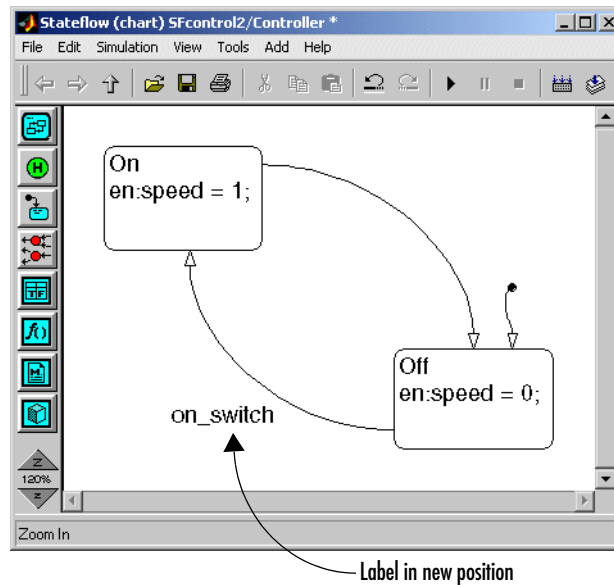


The transition label that you entered (`on_switch`) is an event trigger. Event triggers are part of the syntax for transition labels. A transition event trigger watches for the event it is named after. If the event `on_switch` occurs and the Off state is active, the transition to the On state is taken. The transition is not taken for any other event. This provides the Stateflow diagram the ability to change the active state from the Off state to the On state when it receives a specific event that is meant to turn the device on.

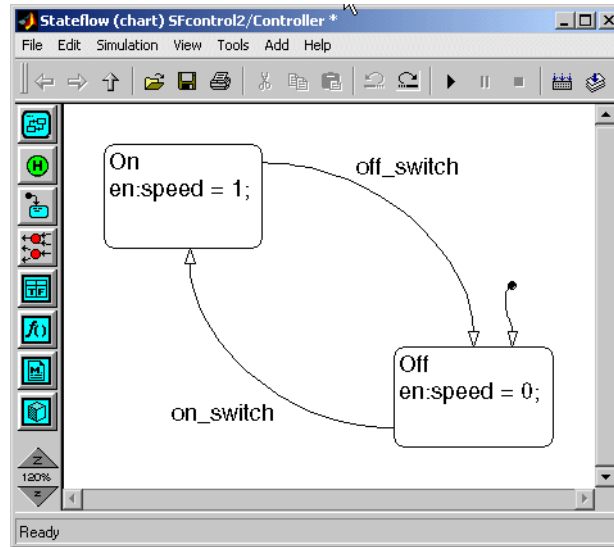
- 6 Click-drag the `on_switch` label to a point outside its transition and then click an empty part of the diagram.

You can click-drag the label for a transition to any point in the Stateflow diagram to change the appearance of the diagram. If the two get separated, you can always tell what transition a label belongs to by clicking on the label. When you do, both the label and its transition become highlighted. Similarly, when you click on the transition, both the transition and its label become highlighted.

You should now see something like the following:



- 7 Use the preceding steps to label the transition from the state On to the state Off with `off_switch` as shown.



Event triggers control the execution of the transitions that they guard. Before you added the event triggers, the transitions from the Off to the On state and the On to the Off state take place in response to any event. Now the Stateflow diagram responds only to the events named in the event triggers.

Note Never add an event trigger to the default transition for the diagram. By definition, the default transition for a diagram transitions to its destination state when the diagram is started. If the first event that starts the diagram does not trigger the default transition, it is never taken, and the diagram does not execute.

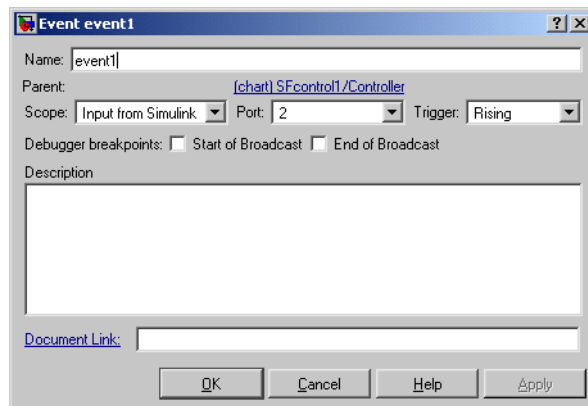
Adding Multiple Trigger Events to a Stateflow Chart

In “Adding Event Triggers to Transitions” on page 2-37, you use the names of the events to specify event triggers for transitions that execute only with the occurrence of one of these events. To use these event triggers, you must also define the events that enable them for the Controller Stateflow diagram.

Define the two events `on_switch` and `off_switch` for the Controller Stateflow diagram with the following steps:

- 1 In the Stateflow diagram editor, from the **Add** menu, select **Event**.
- 2 In the resulting submenu, select **Input from Simulink**.

The **Event** dialog for a new event, event1, appears.



Notice that the middle port field is set to a value of 2.

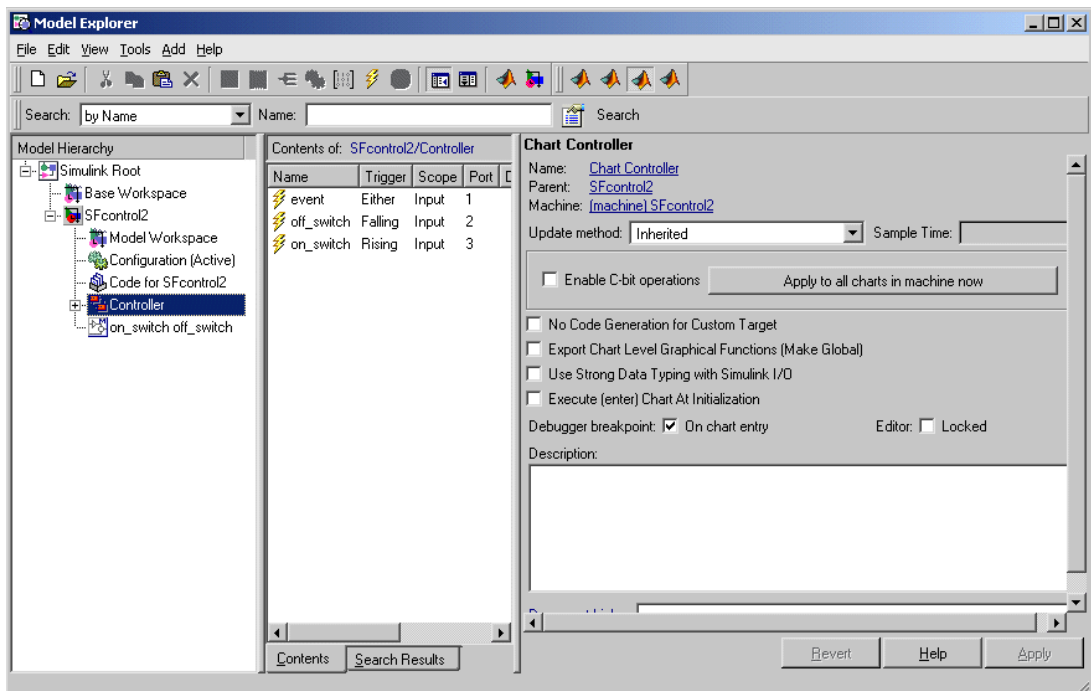
- 3 In the **Name** field, enter `off_switch`.
- 4 In the **Trigger** field, select **Falling Edge**.
- 5 Click **OK** to apply the changes and close the **Event** dialog.

- Repeat step 1 through step 5 to add the event on_switch with a scope of **Input from Simulink**, and a **Rising Edge** trigger.

You have now defined the events on_switch and off_switch in the Controller Stateflow diagram. The event off_switch has a **Falling Edge** trigger and the event on_switch has a **Rising Edge** trigger. The event off_switch occurs only when the control signal attached to the Stateflow block trigger port falls in value as it passes through zero. The event on_switch occurs only when that control signal rises as it passes through zero.

- In the Stateflow diagram editor, from the **Tools** menu, select **Explore**.

The **Model Explorer** window appears, as shown.



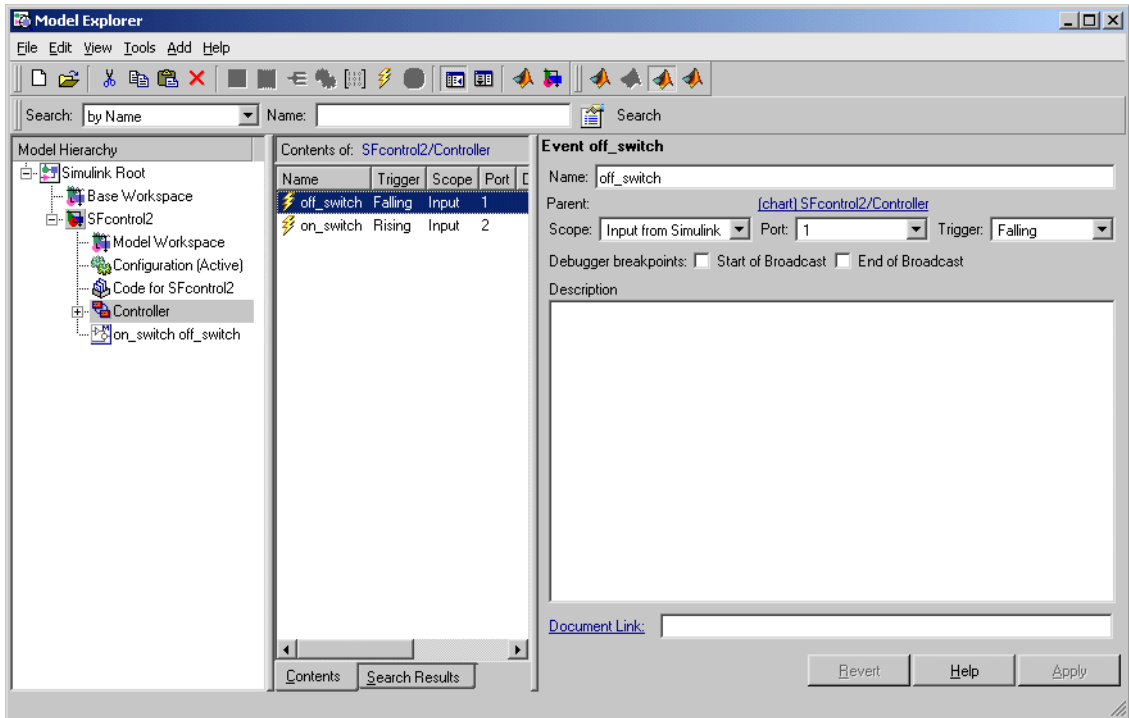
Model Explorer is a tool that lists nongraphical objects for all Simulink objects, including Stateflow objects. Events are nongraphical objects. Model Explorer is the only place to change or delete events.

The left **Model Hierarchy** pane lists the Stateflow objects for the model SFcontrol2, which includes the model itself. To see some of these objects, you need to expand the Controller diagram that contains them. The right **Contents** pane lists the nongraphical objects owned by the highlighted object in the **Model Hierarchy** pane.

The Controller diagram is highlighted in the **Model Hierarchy** pane and the **Contents** pane shows the events that you have added to it. These include the off_switch and on_switch events and the event event you added in the previous section. Notice that these events are indexed in the **Port** column in the order that they were added: (1) event, (2) off_switch, and (3) on_switch.

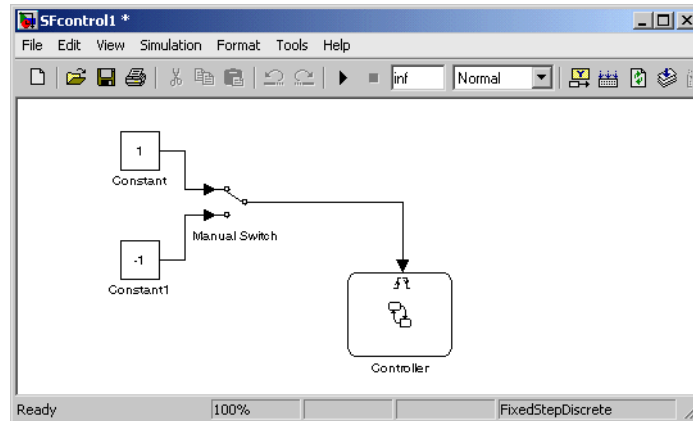
- 8 In the **Contents** pane, click the row for the event event to highlight it.
- 9 Press the **Delete** key.

The row for the event event disappears and the event is deleted. Notice that the indexing for the remaining events is automatically adjusted from three to two elements: (1) off_switch, and (2) on_switch.



Sending Multiple Trigger Events to a Stateflow Chart

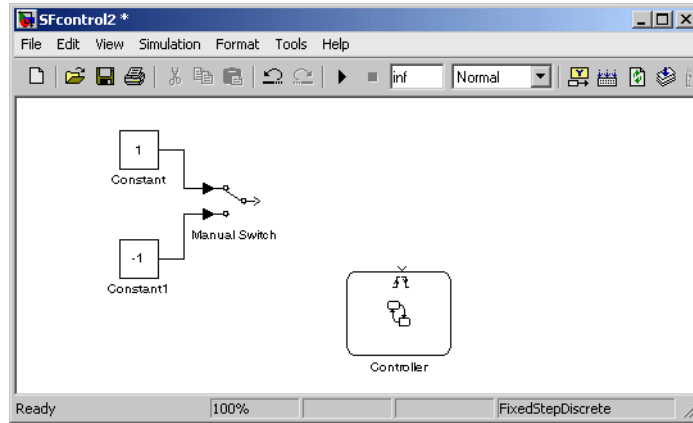
In “Adding Event Triggers to Transitions” on page 2-37, you replace the trigger event event with two trigger events: off_switch and on_switch. Even though you added multiple events, the trigger port in the Simulink test interface for the Controller block has the same appearance as before.



Each event requires a trigger signal to generate its event with a rise or fall in signal. However, Stateflow blocks have only one trigger port for receiving trigger signals. This means that each signal must be indexed into an array of two trigger signals.

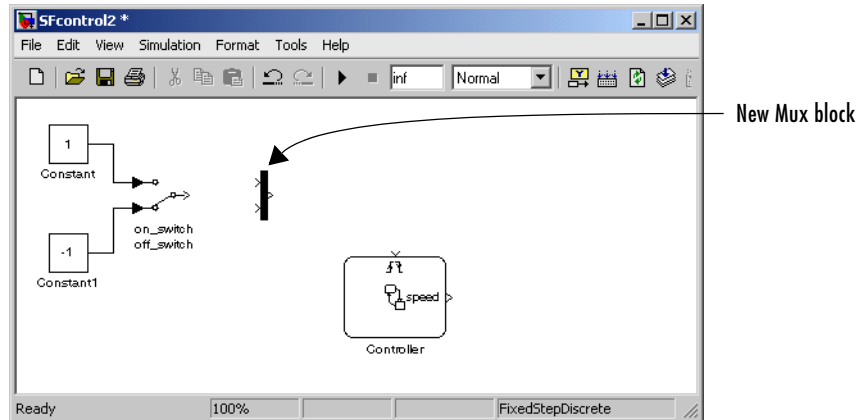
Provide two trigger signals to the single trigger port of the Controller Stateflow block with the following steps:

- 1 Click the signal line from the Manual Switch to the Controller block and press **Delete** to delete it as shown.



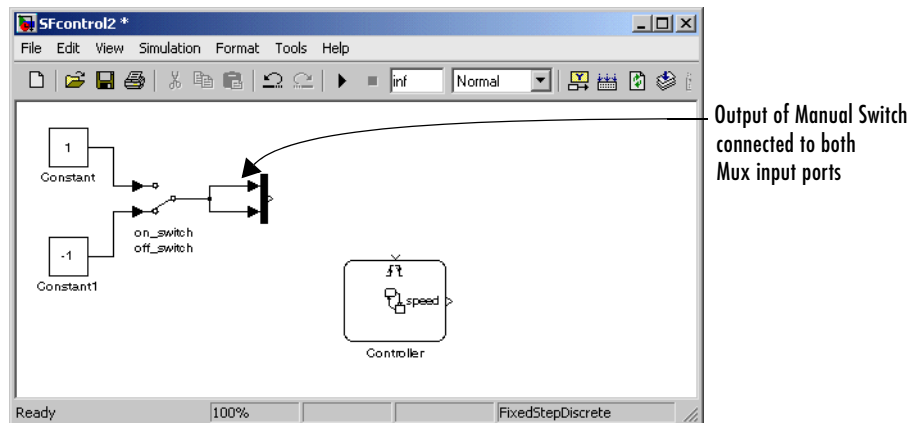
- 2 Click on the Manual Switch block label and rename it as follows:
on_switch
off_switch

- 3** In the **Simulink Library Browser**, under the **Simulink** node, in the **Signal Routing** library, click-drag a Mux block to the right of the Manual Switch block.



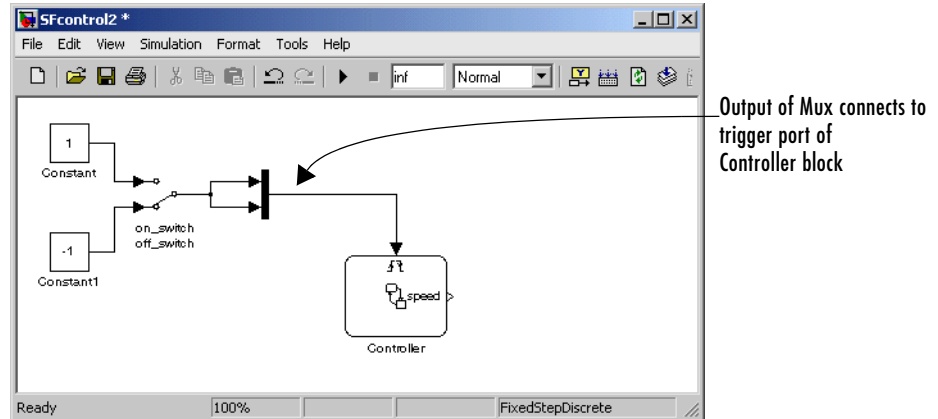
By default, the Mux block has two inputs that are joined in an array to a single output.

- 4** Connect the output of the Manual Switch to both inputs of the Mux block.



This sends the same signal from the Manual Switch to the input ports on the Mux.

- 5 Connect the output side of the Mux block to the trigger port as shown.



The top input port of the mux now connects with the trigger event of index 1, off_switch. The bottom port connects with the trigger event of index 2, on_switch. During simulation, when you toggle the Manual Switch from -1 to 1, you send a rising trigger signal to both signal inputs of the trigger port. Only the on_switch event port reacts by sending an on_switch event to the Controller Stateflow diagram. Likewise, when you toggle the Manual Switch from 1 to -1, you generate an off_switch event.

- 6 Save the model (as SFcontrol2).

Modifying Output Data with Actions

If you are simulating a model with a control device such as a motor, the Stateflow diagram needs to hand over to Simulink some indication of the state that the motor is in to continue the simulation. You can imagine that such information might inform the model that the motor is on and now providing cooling or heating to the model, or that the motor is off and plant temperature or pressure is increasing accordingly.

Use the following procedure topics to output modified data from the Controller Stateflow diagram to the Simulink test interface:

- 1 “Adding State Entry Actions” on page 2-51 — Program the Controller Stateflow diagram to modify data from Simulink in the entry action of the active state.
- 2 “Adding Output Data to the Stateflow Chart” on page 2-53 — Pass Stateflow modified data from the Controller diagram to the Simulink model.
- 3 “Sending Stateflow Output Data to Simulink” on page 2-55 — Receive modified data from the Controller Stateflow diagram in the Simulink model and display it.

Adding State Entry Actions

You modify data in Stateflow with actions. Actions are programming lines that you add to states or transitions in Stateflow diagrams. In this topic, you use state entry actions to change a data value that provides control information to the Simulink model. For now, assume that your diagram defines a data named speed that you pass to the Simulink model. You add a state entry action to modify the value of speed to indicate the speed of a device (for example, a motor) that you are controlling to the Simulink model.

Add entry actions to the labels of the states On and Off to modify the data speed with the following steps:

- 1 Place the mouse cursor over the name of the On state.

Notice that the mouse cursor changes to a text cursor over the state label.

- 2 Click when the text cursor is at the end of the name of the On state.

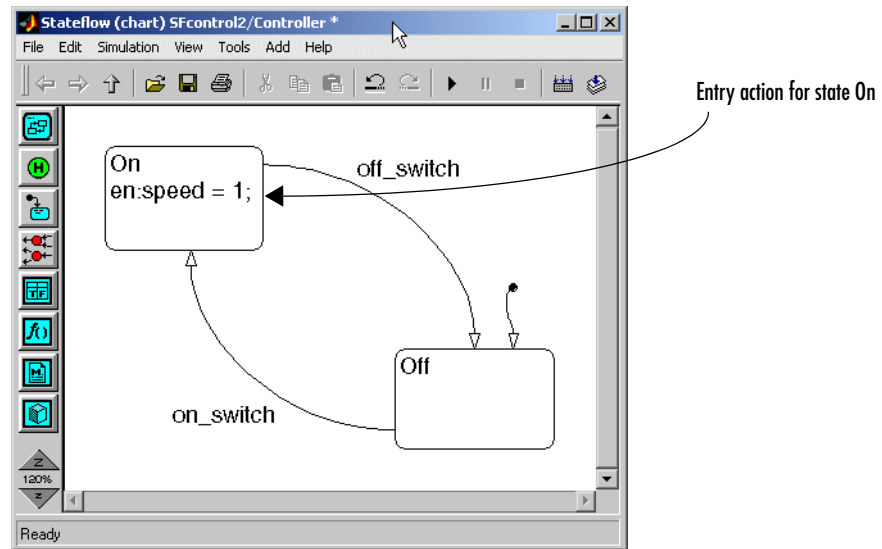
This highlights the state and its label and a blinking text cursor appears at the end of the label, which contains only the name On.

- 3 Press **Enter** and enter the following text line.

```
en: speed = 1;
```

- 4 Click outside of the state when finished entering text.

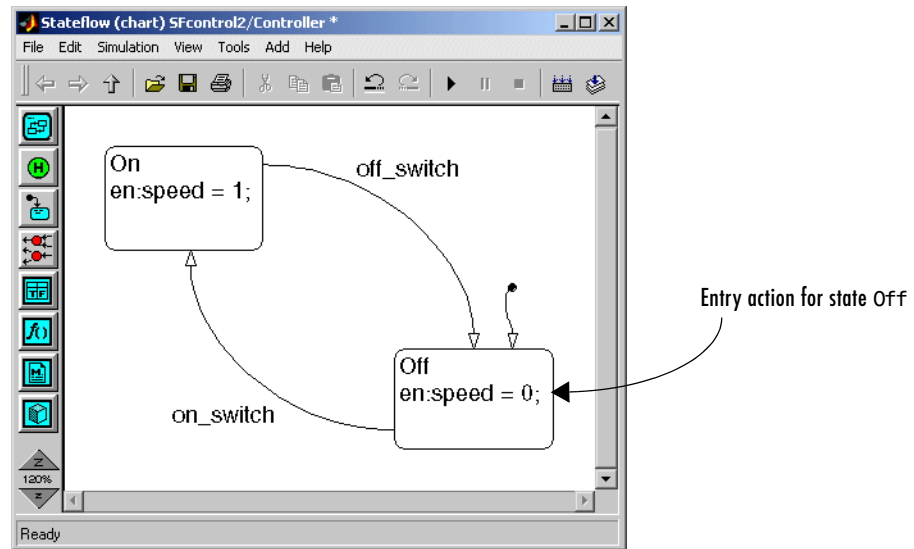
The Controller Stateflow diagram should appear similar to the following:



The label for the On state now includes a name for the state (On) and an entry action, which is identified by the prefix `en:`. This entry action assigns the data `speed` a value of 1 when the state On becomes active.

- 5 Repeat the preceding steps to add the following entry action to the state Off:
`en:speed = 0;`

The Controller Stateflow diagram should appear similar to the following:



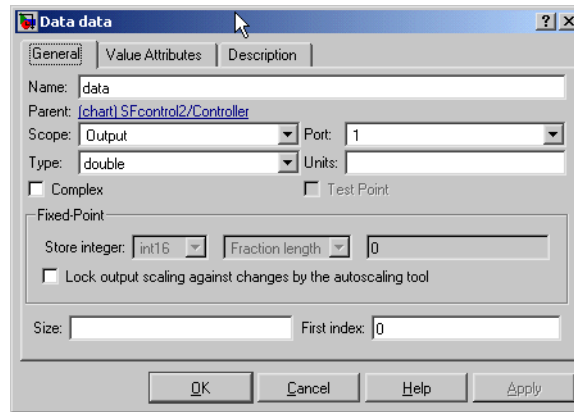
Adding Output Data to the Stateflow Chart

In “Adding State Entry Actions” on page 2-51, you modify the data `speed` in state entry actions that you intend to pass to the Simulink model. Before you can use `speed` as data in the Controller diagram, you must define it to the Controller diagram as data passed to the Simulink model.

Use the following steps to define the data `speed` as output to Simulink for the Controller Stateflow diagram:

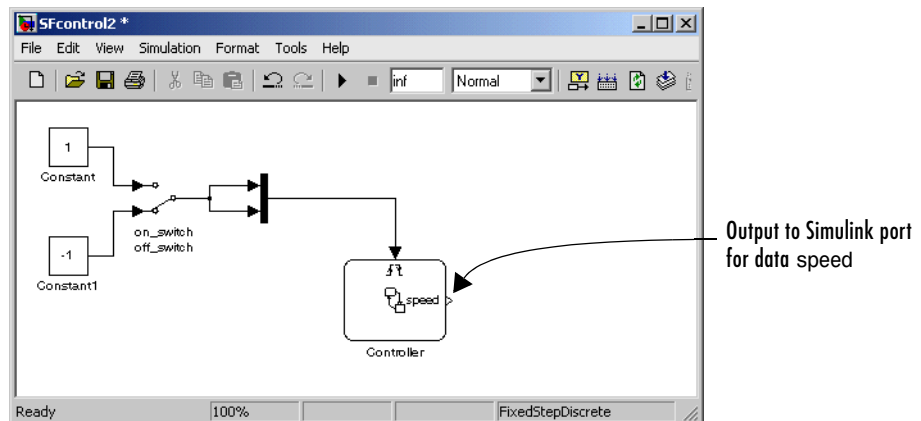
- 1 In the Stateflow diagram editor for the Controller block, from the **Add** menu, select **Data**.
- 2 In the resulting submenu, select **Output to Simulink**.

The property dialog for the new data appears.



- 3 In the **Name** field of the data properties dialog, enter speed.
- 4 Click **OK** to apply the changes and close the window.

An output port for the data speed now appears on the right side of the Controller Stateflow block in the Simulink model as shown.

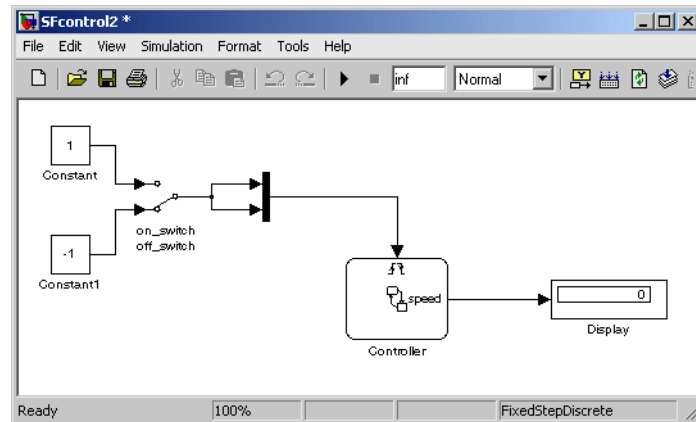


Sending Stateflow Output Data to Simulink

In “Adding Output Data to the Stateflow Chart” on page 2-53, you define data output to the Simulink model for the Controller Stateflow diagram. This creates an output port for the data speed that now appears on the right side of the Controller Stateflow block in the Simulink model. This output port gives the Simulink model access to the data speed defined for, and modified in, the Controller Stateflow diagram.

In this topic, you demonstrate that the Simulink model receives the data speed from the Controller Stateflow diagram by displaying it in the Simulink model with the following steps:

- 1 From the **Simulink Library Browser**, in the Simulink **Sinks** library, add a Display block to the right of the Controller Stateflow block as shown.
- 2 Connect the output port for the data speed on the Controller Stateflow block to the Display block as shown.



When you simulate the model, the Controller block modifies the value of the data speed, which it passes on to the Simulink model for display.

- 3 Save the model (as SFcontrol2.mdl).

Simulating Event Triggers and Modified Output Data

In this section, you test the features that you add to the Controller diagram and the Simulink block of the SFcontro12 model in the following previous sections:

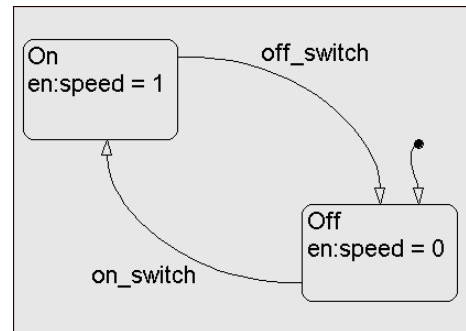
- “Guarding Transitions with Event Triggers” on page 2-37
- “Modifying Output Data with Actions” on page 2-51

It is important to test new features that you add to the model to make sure that they behave as you expect. Test the new features by simulating the Controller diagram of the SFcontro12 model with the following steps:

- 1 In the Simulink diagram, make sure that the Manual Switch points downward to the -1 pole. If not, double-click it to change its position.
- 2 Choose **Start** from the Stateflow diagram editor **Simulation** menu to start simulation of the model.

After a small delay during which the model builds, simulation of the Controller Stateflow diagram begins in the Stateflow diagram editor with the introduction of a gray background as shown.

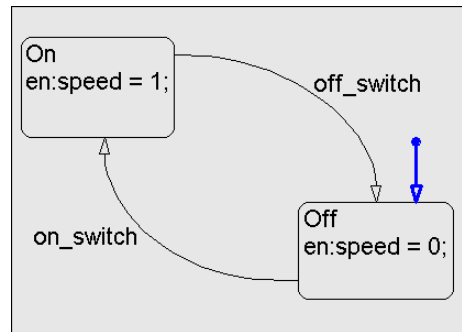
Stateflow begins in simulation mode. The Controller diagram is active, but nothing in it is.



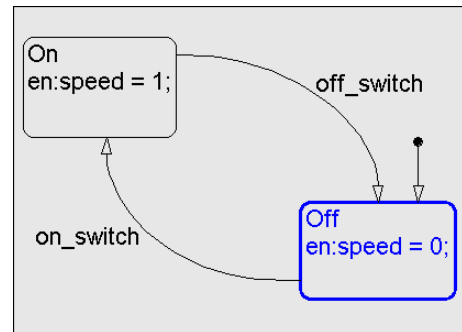
- 3 In the Simulink model, double-click the Manual Switch to move it from the -1 pole to the 1 pole.

Since `on_switch` is a rising edge triggered event, the rise in signal from -1 to 1 sends an `on_switch` event to the Controller block. The Controller diagram responds as follows:

The `on_switch` event causes the Controller diagram to look for an available transition. The only available transition is the default transition. Because this transition is not guarded, it reacts to any event, and is therefore taken.



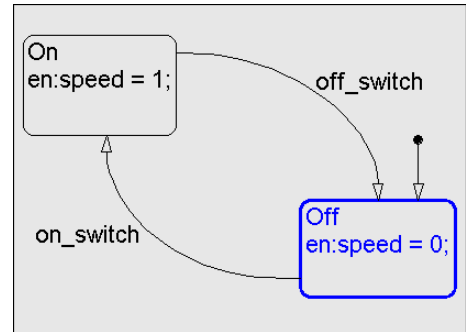
The destination state of the default transition, the state Off, is entered. Also, the data `speed` is set to a value of 0 as part of the entry action for the state Off. The state Off remains active until another event is sent to the Controller diagram.



- 4 In the Simulink model, double-click the Manual Switch to move it from the 1 pole to the -1 pole.

Since `off_switch` is a falling edge triggered event, the drop in signal from 1 to -1 sends an `off_switch` event to the Controller block. The Controller diagram responds as follows:

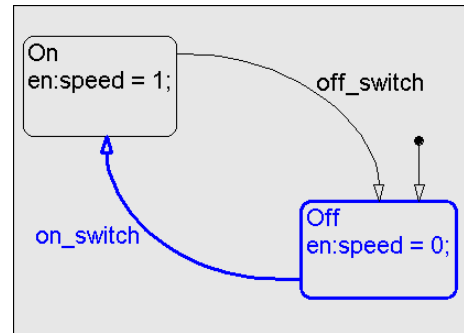
The `off_switch` event causes the Controller diagram to look for an available transition. Only the transition to the On state is available. However, it is guarded by an `on_switch` event trigger. No transition takes place and the Off state remains active.



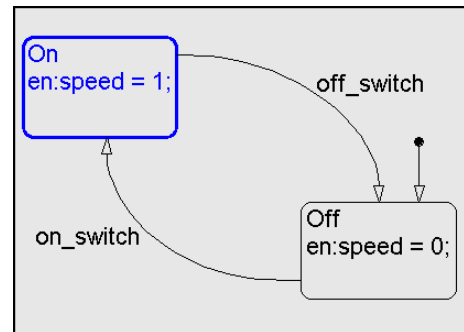
- 5 In the Simulink model, double-click the Manual Switch to move it from the -1 pole to the 1 pole.

Since `on_switch` is a rising edge triggered event, the rise in signal from -1 to 1 sends an `on_switch` event to the Controller block. The Controller diagram responds as follows:

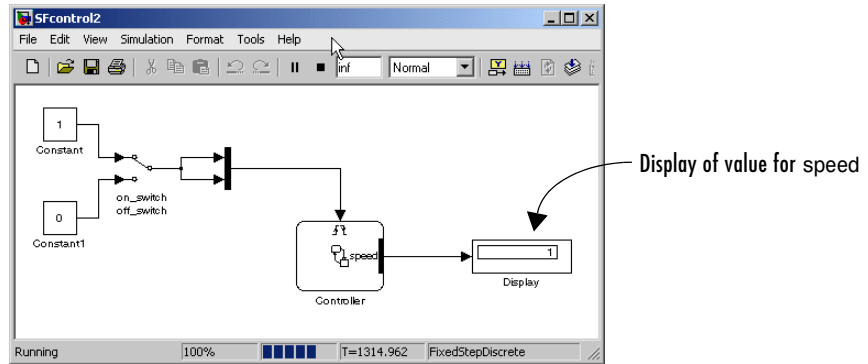
The `on_switch` event causes the Controller diagram to look for an available transition. Only the transition to the On state is available. Because it is guarded by an `on_switch` event trigger, this transition takes place.



The Off state is exited and the On state is entered and becomes active. Also, the data `speed` is set to a value of 1 as part of the entry action for the state On. The state On remains active until another event is sent to the Controller diagram.



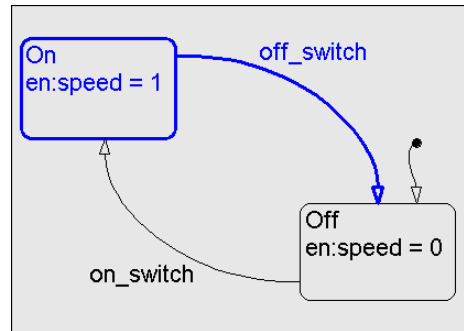
The value for speed is displayed in the Display block of the Simulink model as shown.



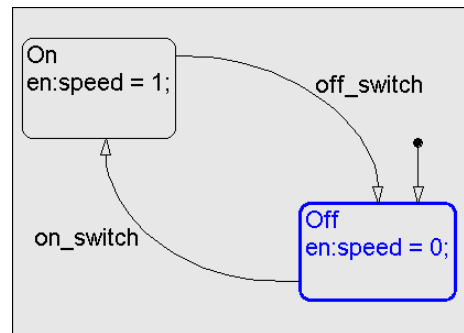
- 6 In the Simulink model, double-click the Manual Switch to move it from the 1 pole to the -1 pole.

The fall in signal from 1 to -1 sends an `off_switch` event to the Controller block. The Controller diagram responds as follows:

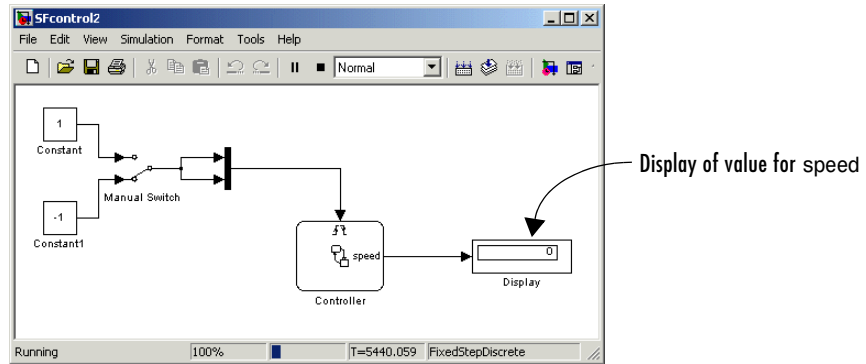
The `off_switch` event causes the Controller diagram to look for an available transition. Only the transition to the Off state is available. Because it is guarded by an `off_switch` event trigger, this transition takes place.



The On state is exited and the Off state is entered and becomes active. Also, the data `speed` is set to a value of 0 as part of the entry action for the state Off. The state Off remains active until another event is sent to the Controller diagram.



The value for speed is displayed in the Display block of the Simulink model as shown.



- 7 Continue to send events by double-clicking the Manual Switch in the Simulink window to repeat the sequence in the preceding steps 5 and 6.
- 8 Choose **Stop** from the graphics editor **Simulation** menu to stop the simulation.

Once simulation stops, Stateflow resets the model to be editable.

Controlling with Junctions

This chapter introduces you to new Stateflow objects that include junctions, functions, temporal events, and superstates. These objects extend the use of states and transitions that form the basis of Stateflow capability. Use the following sections to add new control features to the models you build in Chapter 2, “Controlling with States and Transitions”:

Adding a Sensor to the Model (p. 3-2)	Add the equivalent of a sensor to the Simulink model as data and a regularly occurring event.
Adding a Junction for Flow Control (p. 3-8)	Modify a Stateflow diagram to use junctions to add control that chooses between state destinations. While doing this, also learn how to edit diagrams.
Adding a Graphical Function for Convenience (p. 3-19)	Use a graphical function to make a Stateflow diagram more readable.
Simulating with a Sensor Event, Junction, and Function (p. 3-25)	Simulate the junction and graphical function you added to the example model.
Simulating Junction Behavior (p. 3-31)	Demonstrate junction behaviors with small changes to the model.
Using Junctions in Flow Diagrams (p. 3-37)	Use junctions without states to create visible programming flow diagrams.

Adding a Sensor to the Model

In this topic, you add a temperature sensor to the model that you use to control the speed of a device such as a motor, fan, or pump. A sensor has two primary attributes: a value for the property it is measuring, and a regular time interval during which a new value is reported. Normally, the value of a sensor is reported as a voltage that must be converted to useful units such as degrees, flow, capacity, and so on. For the sake of example, assume that the temperature sensor reports the temperature directly in degrees Fahrenheit.

Use the following procedure topics to add a reporting sensor to the `SFcontrol2` model you completed in Chapter 2, “Controlling with States and Transitions”:

- 1 “Adding a Sensor Event” on page 3-2 — Use the fluctuation in a signal to send a periodic event to Stateflow, mimicking the periodic reporting that a sensor makes of its value.
- 2 “Adding Sensor Data” on page 3-6 — Add data to Stateflow that represents the value of a sensor.

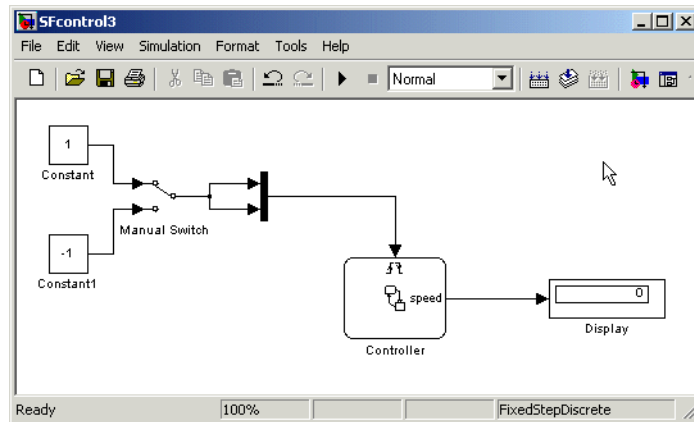
Adding a Sensor Event

To take advantage of sensors monitoring a physical plant, you need to check the sensor value at regular intervals. Proper control of a physical plant can be maintained only if a sensor measurement reports at regular intervals. For example, a controlling application might need to monitor the temperature or pressure of a device every second. If it waits for an hour before the next measurement is taken, undesired consequences can result.

You can simulate a sensor reading in Simulink with a regularly occurring sensor event and sensor data representing the value reported by the sensor. In Simulink, you use a regularly changing source to output a trigger event for determining times during which you read the sensor value.

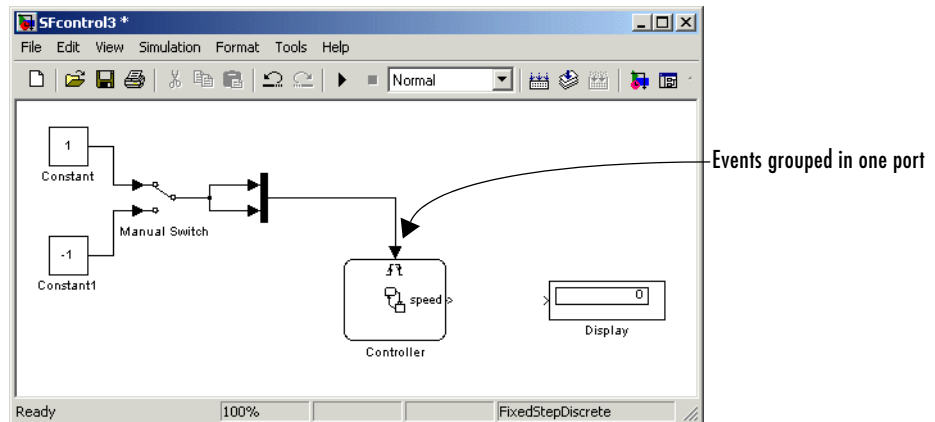
Use a sine wave source in Simulink to add a sensor event to trigger a Stateflow block in the following procedure:

- 1 If not already loaded, load the Simulink model `SFcontrol2` you save at the end of “Modifying Output Data with Actions” on page 2-51 and save it as `SFcontrol3`.

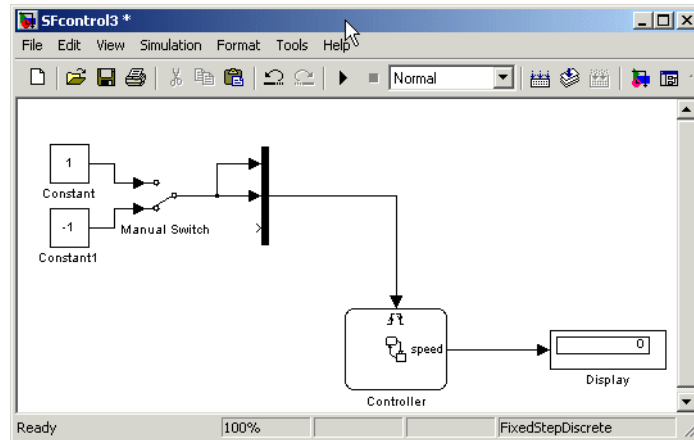


- In the Stateflow diagram editor, from the **Add** menu, select **Event** and add an event named **temp_event** on port 3 with the scope **Input from Simulink**, and a **Rising Edge** trigger type.

A port for the new event **temp_event** does not appear in the Simulink model because all input events are grouped into the trigger port at the top of the **Controller** block.

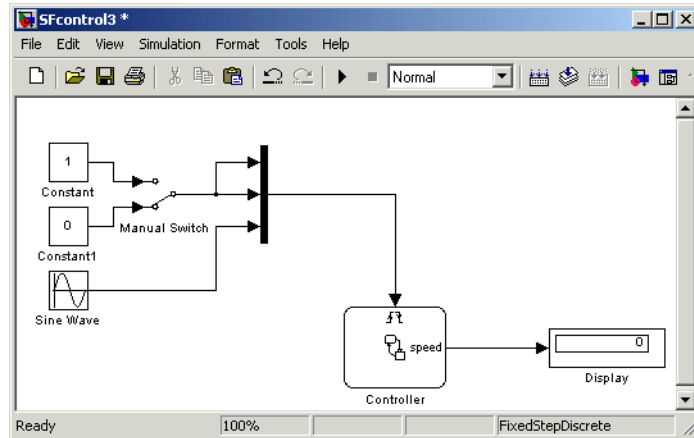


- 3 Double-click the Mux block and change its **Number of inputs** to 3.
- 4 Click-drag the top and bottom ends of the Mux block bar to show all three ports clearly, as shown.



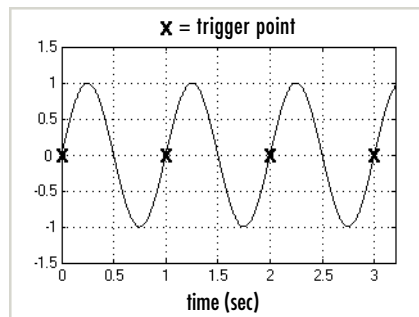
- 5 In the Simulink library browser, in the Sources library, click-drag a Sine Wave block to a position under the Constant blocks that feed the Manual Switch.

- 6 Connect the Sine Wave block output to the bottom input of the Mux block.



- 7 Double-click the Sine Wave block and change the **Frequency** to 2π .

Because the event `temp_event` is a rising edge trigger, it is triggered (broadcast) every time its input signal crosses zero while rising. Because the Sine Wave block now outputs a complete sine wave every second, `temp_event` is broadcast to the Stateflow diagram every second, as shown.

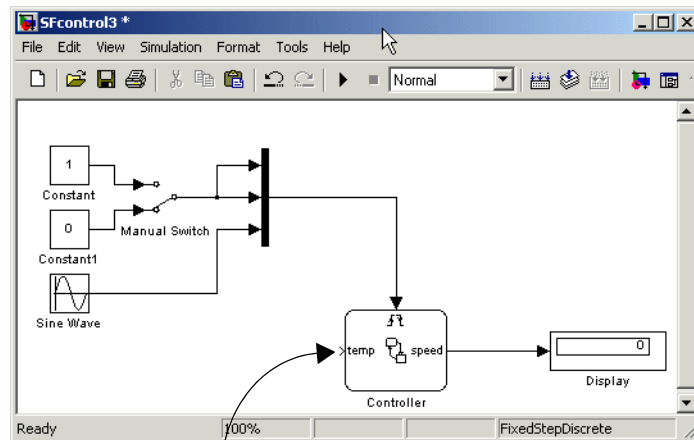


Adding Sensor Data

You can simulate sensors in Simulink with a sensor value and a time interval for which the sensor reports its value. In “Adding a Sensor Event” on page 3-2, you added an event to simulate the reporting time for a sensor. Now you need to add a data value to the Stateflow block to simulate the value of the sensor in the following steps:

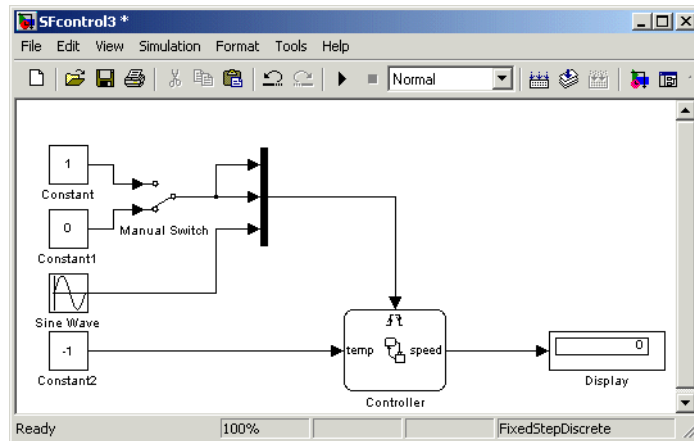
- 1 In the Stateflow diagram editor, from the **Add** menu, select **Data** and add a data named **temp** with scope **Input from Simulink**.

A port appears on the left side of the Controller block in the Simulink model for the new data **temp** as shown.



Input data port
for temperature sensor

- 2 Right-click the bottom Constant block and drag-copy a new Constant block under the Sine Wave block.
- 3 Connect the output of the new Constant block to the input port for the data temp on the left side of the Controller block as shown.



The source for the value of the data temp is the value of the Constant block. You can change this value during simulation by double-clicking the Constant block and entering a new value in the **Constant value** field in the resulting **Block Parameters** dialog and clicking **Apply**.

- 4 Save the model (as SFcontrol3.mdl).

Adding a Junction for Flow Control

In “Adding a Sensor to the Model” on page 3-2, you add a simulated temperature sensor to the Simulink model. The sensor has a value for the temperature provided by the input data `temp`, and a periodic event `temp_event`, that acts as a reporting time for the temperature. In this section, you add more selective control based on the new temperature sensor, which reports the value of the temperature to the Controller Stateflow diagram every second.

Add temperature control to the `SFcontro13` model in the following procedure topics:

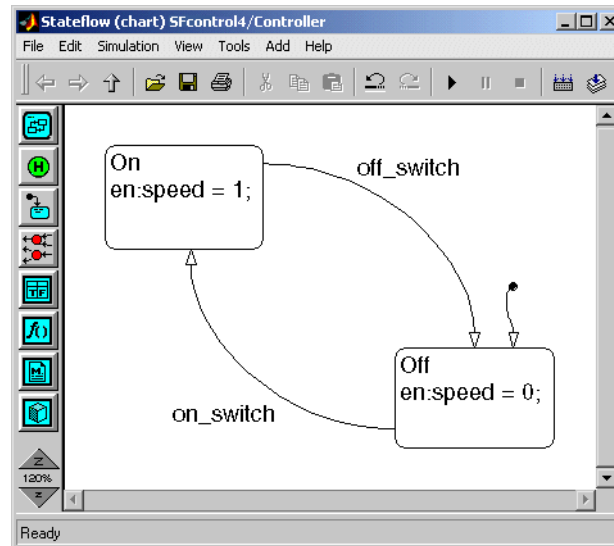
- 1 “Deleting, Copying, and Renaming Stateflow Objects” on page 3-8 — Alter the Stateflow diagram by deleting a transition, copying a state, and renaming two states.
- 2 “Adding and Connecting Junctions” on page 3-13 — Connect a junction in the Stateflow diagram.
- 3 “Entering Transition Conditions” on page 3-16 — Enter conditions for transitions and transition segments.

Deleting, Copying, and Renaming Stateflow Objects

Begin preparation of the Stateflow diagram for the `SFcontro13` model by removing a transition from it in the following steps:

- 1 If necessary, load the `SFcontro13` model you built in “Adding a Sensor to the Model” on page 3-2, and save it as `SFcontro14`.

- 2 Double-click on the Stateflow block Controller to open the Stateflow diagram as shown.

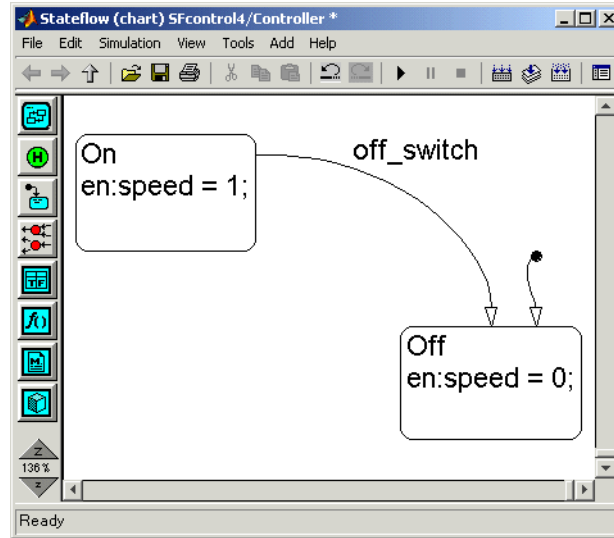


- 3 Select the transition from the Off state to the On state by clicking it at any point in the transition.

The transition and its label highlight to a different color. You can select any graphical Stateflow object by clicking it.

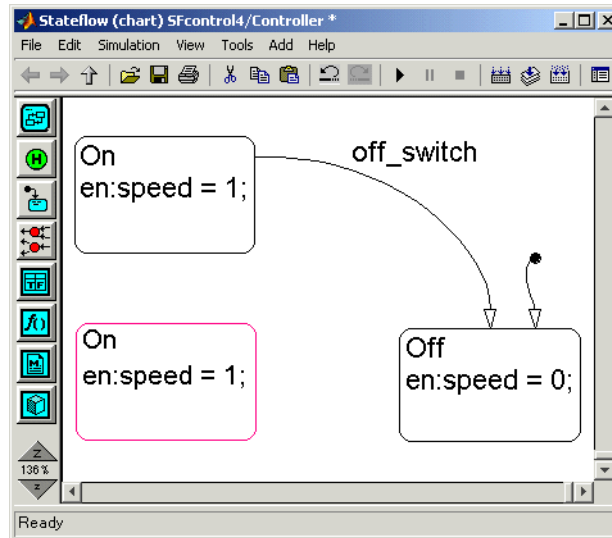
4 Press **Delete**.

The transition disappears as shown.



Continue altering the Stateflow diagram by copying and renaming states in the following steps.

- 5 Right-click on any point in the On state and hold down the mouse key while you drag a copy of the On state to a position just below as shown.



- 6 Click on either line of the label for the top On state to place a cursor in it.

- 7** Rename the top On state to the Hi state with the following label:

```
Hi  
en: speed=2;
```

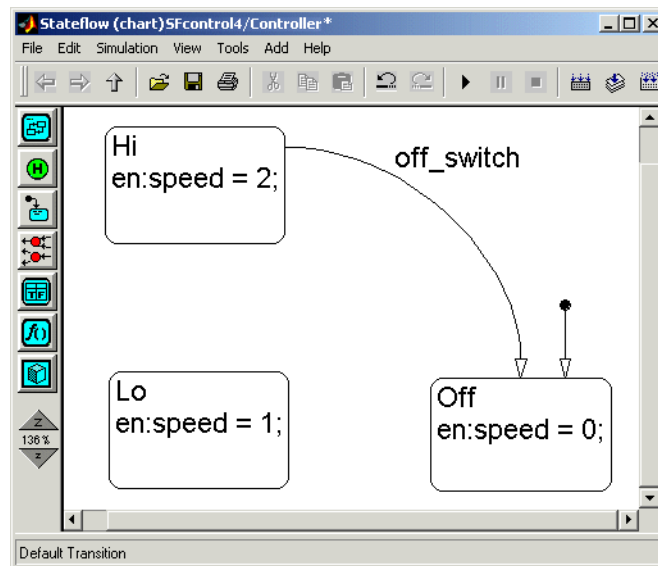
The On state becomes the Hi state. When Hi becomes active, its entry action changes the value of the data speed to 2.

- 8** Rename the bottom On state as follows:

```
Lo  
en: speed=1;
```

The state Lo represents a low speed state for the controlled device, such as a motor. When it becomes active, it changes the value of the data speed to 1.


The Stateflow diagram should now have the following appearance:

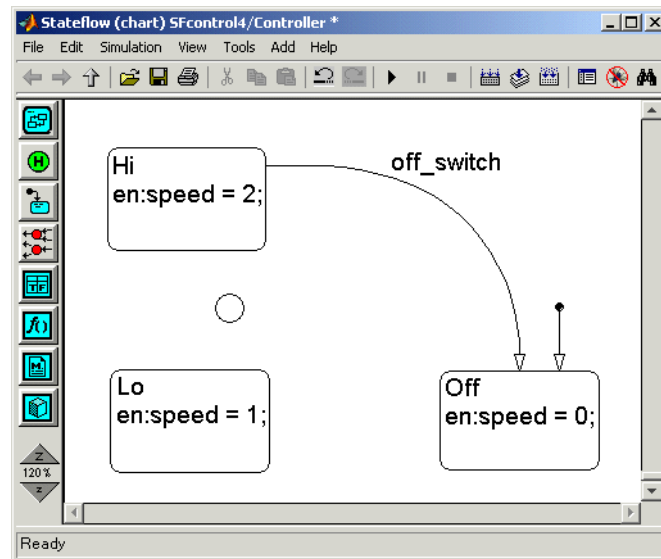


Adding and Connecting Junctions

You use junctions to provide alternate paths for transitions. This lets the Stateflow diagram choose the destination state for a transition based on a condition or set of conditions. In this section, you add a junction that chooses between two destination states for a transition based on the value of the temperature. This represents the choice of one of two possible speeds for the device you are controlling, such as a motor.

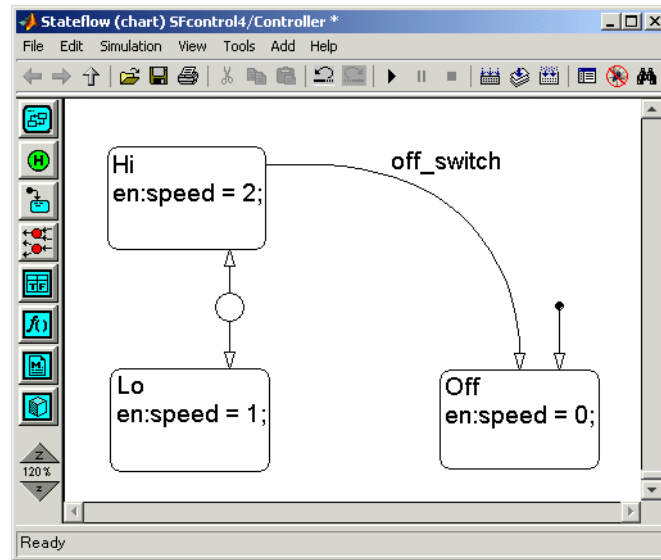
Place a junction between the two states Hi and Lo and connect it with the following steps:

- 1 From the drawing toolbar on the left, click the **Junction** tool .
- 2 Move the cursor into the drawing area and place a junction between the states Hi and Lo as shown.

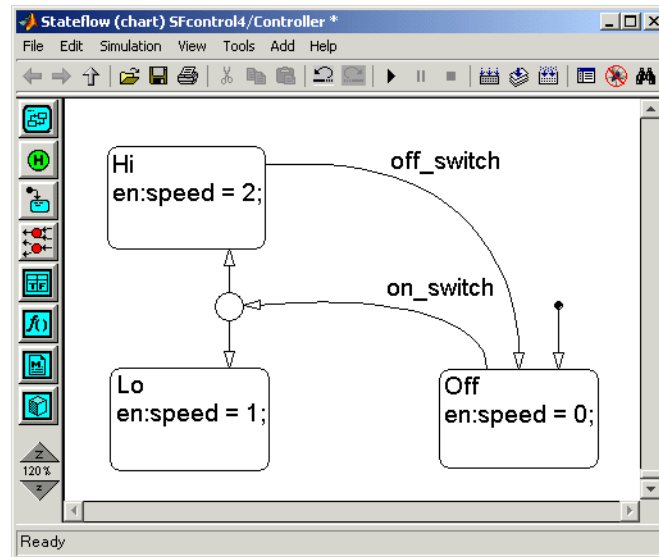


- 3 Draw a transition from the top border of the junction to the bottom border of the state Hi just as you would between two states.
- 4 Draw another transition from the junction down to the top of the state Lo.

The Stateflow diagram now has the following appearance:



- 5 Draw a transition from the top of the state Off to the right side of the junction and label it with the event trigger `on_switch` as shown.



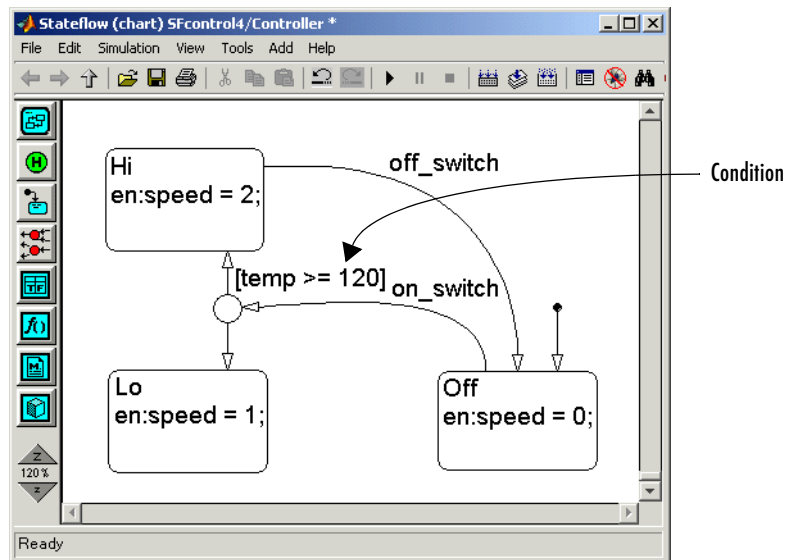
The transition from the Off state now has two possible destinations: Hi and Lo. To make a choice between these destinations, you need to add a condition to one of the transitions out of the junction in “Entering Transition Conditions” on page 3-16.

Entering Transition Conditions

Transitions to and from junctions are referred to as transition segments. Transitions, which, by definition, go from state to state, use transition segments to make a complete transition. The transition segments you add in “Adding and Connecting Junctions” on page 3-13 define two possible destinations for the transition from Off: Hi or Lo. To know which of the two destinations to take, you need to provide a logical decision.

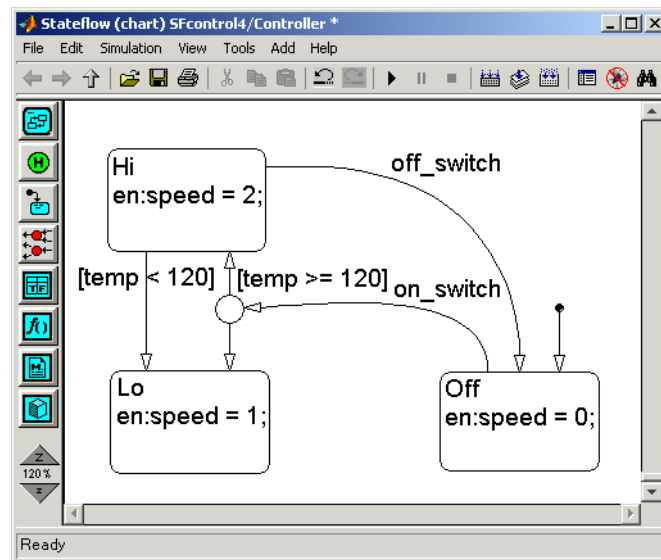
Use the following steps to enter conditions on the transition segments from a junction:

- 1 Label the transition from the junction to the state Hi with the condition `[temp >= 120]` as shown.



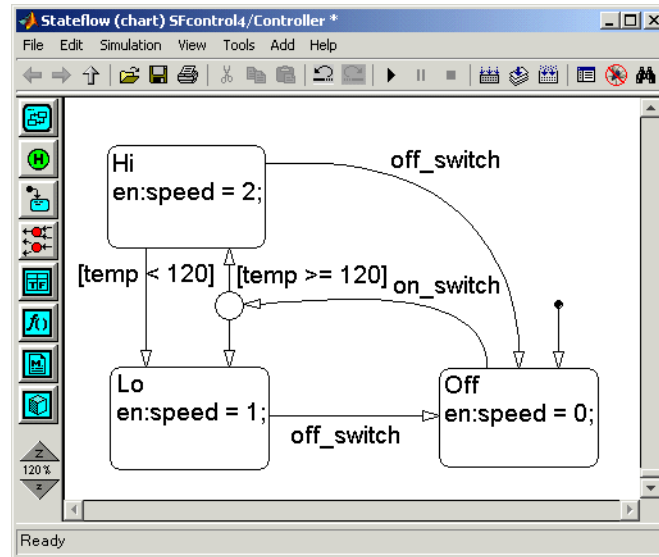
If the Off state is active and an `on_switch` event occurs, the transition segment to the junction is taken. Once the junction becomes active, Stateflow searches for a valid outgoing transition segment. Stateflow always gives higher priority to transitions that have conditions over other transitions that have no conditions. This means that if the temperature of the attic, `temp`, equals or exceeds 120 degrees, the attic fan enters the Hi state. If not, the attic fan enters the Lo state. Once the Lo or Hi state becomes active, no more transitions are taken in the diagram until another event occurs.

- 2 Draw a transition from the Hi state to the Lo state and label it with the conditions `[temp < 120]` as shown.



This transition allows the device you are controlling to transition to the Lo state when the temperature of the controlled drops below 120 degrees. Because this transition is from state to state, it requires a regular event provided by the temperature sensor event, `temp_event`, which occurs every second.

- 3 Draw a transition from the Lo state to the Off state and label it with the event trigger `off_switch` as shown.



The Controller needs to stop the control device if it receives an `off_switch` event and either the Lo or Hi state is active. The transition from Hi to Off already takes care of part of this requirement. You add a similar transition from the state Lo to the state Off with an `off_switch` event for the rest of the requirement.

- 4 Save the current model (as SFcontrol14.mdl).

Adding a Graphical Function for Convenience

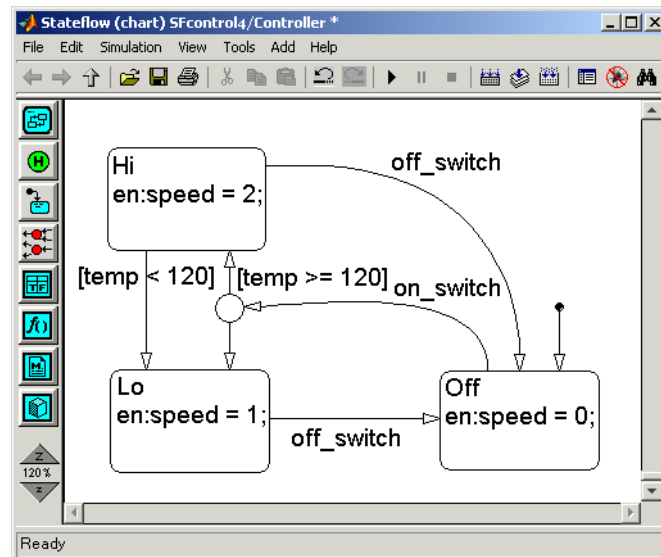
In “Adding a Junction for Flow Control” on page 3-8, you add a junction to the Stateflow diagram to provide two-speed control to a control device such as a motor. You can improve the compactness and readability of this diagram with a graphical function that you add in the following procedure topics:

- 1 “Adding a Graphical Function” on page 3-19 — Add and program a small graphical function to replace some lengthy conditions.
- 2 “Calling a Graphical Function” on page 3-24 — Call a graphical function from the action language of a transition.


Adding a Graphical Function

You can shorten conditions on transitions with a graphical function. Add a graphical function to the Stateflow diagram of `sf_control14.mdl` with the following steps:

- 1 If not already loaded, load the Simulink model `SFcontrol14` you save in “Adding a Junction for Flow Control” on page 3-8 and double-click the Stateflow block `Controller` to open it.

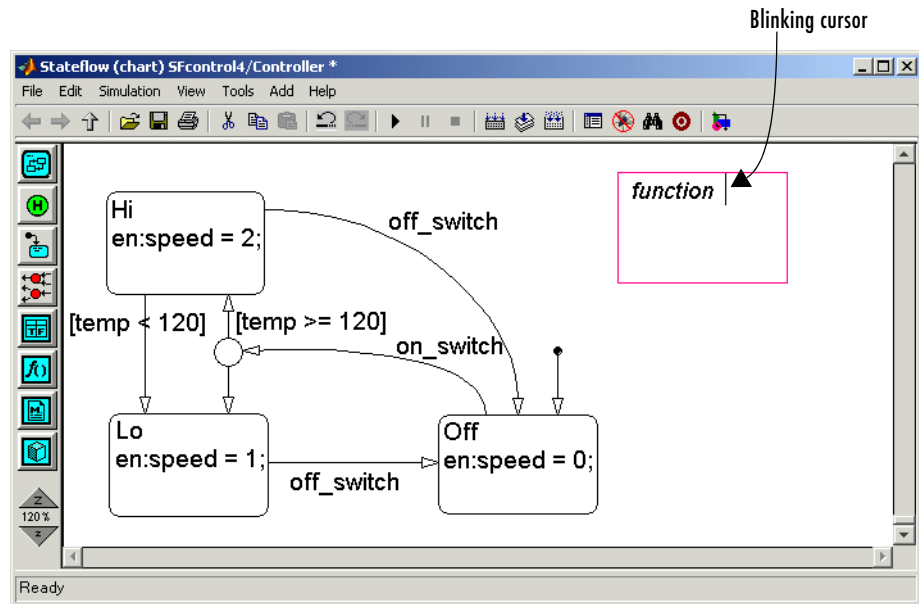


2 Resize the window by click-dragging the right border of the Stateflow diagram to the right by about 3 inches.

3 In the Stateflow diagram editor, click the Graphical Function tool .

4 Place the cursor, now in the shape of the box, on the right side of the diagram.

A highlighted function box appears with a blinking cursor in its upper-right corner as shown.



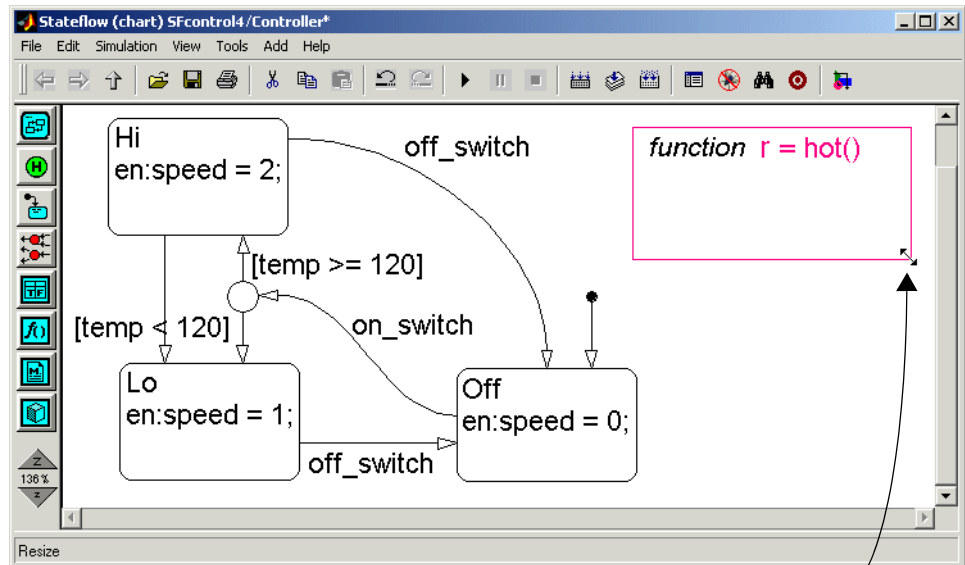
5 Enter the text

```
r = hot();
```


and click outside the function box.

6 Place the mouse cursor over the lower-right corner of the function box.

The mouse cursor now appears as a double arrow.

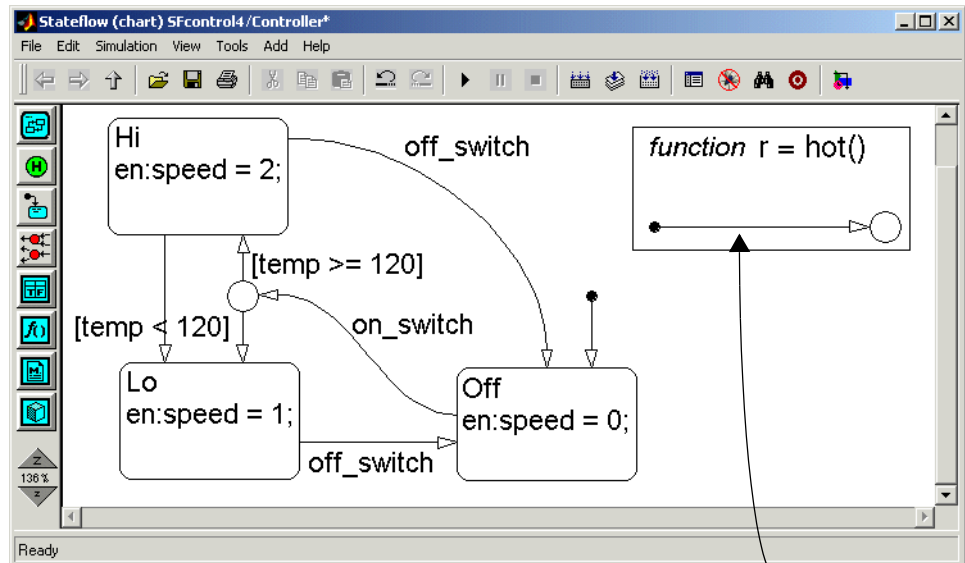
7 Click-drag the function box corner to the right about three inches to resize the function box as shown.

Resizing cursor

- 8 Select the Default Transition tool  and place the mouse cursor inside the function box in the lower-right corner and click.

This places a junction with an incoming default transition in the lower-right corner of the function box as shown.

- 9 Click-drag the source of the default transition to the left side of the function box as shown.



Default transition into junction

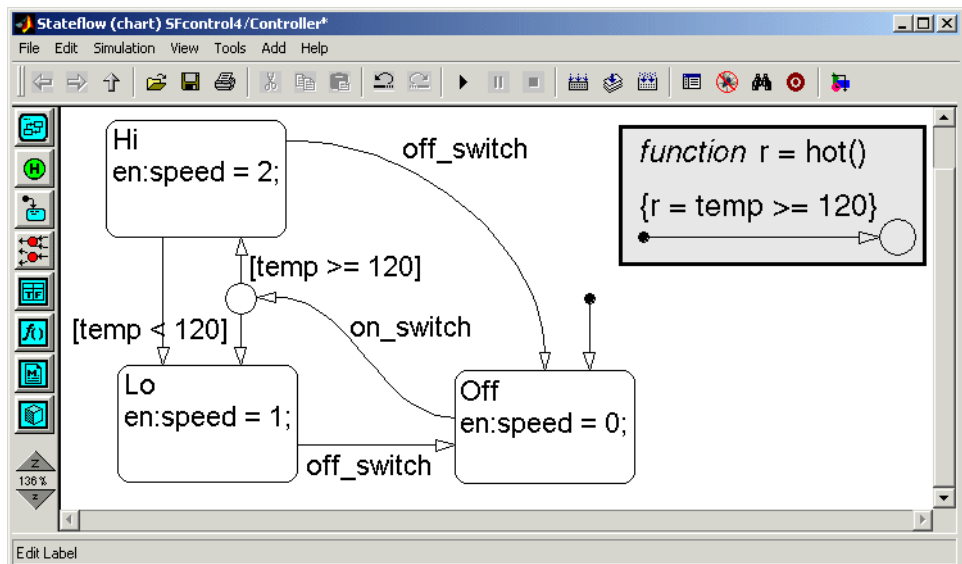
- 10** Click the question mark (?) on the default transition and label it as follows:
- ```
{r = temp >= 120;}
```

The statement `{r = temp >= 120;}` is an example of a transition condition action. A condition action is executed when the condition for its transition is true. Because the default transition has no condition, its condition action is executed automatically and the default transition is taken into the junction. In this case, when the function `hot` is called, the logical result of the comparison of the temperature with 120 is stored in the return data `r`.

A default transition into a junction is a type of flow diagram. Flow diagrams use transitions and junctions. You can use more complicated flow diagrams to represent common code structures like for loops and if-then-else constructs without the use of states. Flow diagrams are discussed in more detail later on in “Using Junctions in Flow Diagrams” on page 3-37.

- 11** Double-click the function box to group it.

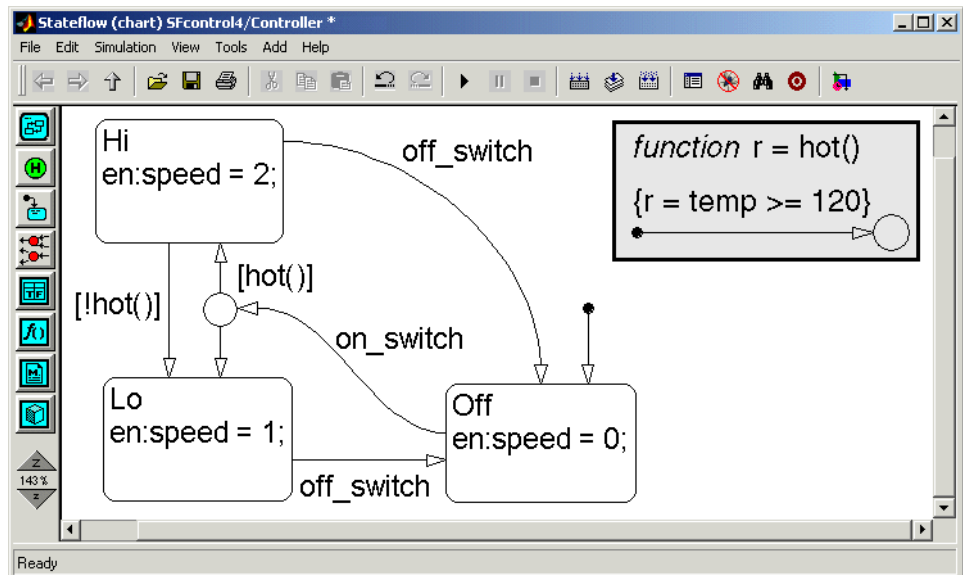
This joins the function and the objects it contains into a group of objects that you can move as a unit. The finished diagram should now appear similar to the following:



## Calling a Graphical Function

Once you make a function, you call it from a transition condition or one of the actions of a state or transition. The function you added in “Adding a Graphical Function” on page 3-19 returns the comparison value that you test for in the transition segment from the junction to the state Hi. Use the following steps to call the function `hot` in the condition of that transition segment:

- 1 Click on the transition from the junction to the state Hi.
- 2 A question mark character (?) appears on the transition.
- 3 Click the question mark and edit the label to be `[hot()]`.
- 4 Edit the label for the transition from state Hi to state Lo to be `[!hot()]`, as shown.



- 5 Save the model (SFcontrol14).

## Simulating with a Sensor Event, Junction, and Function

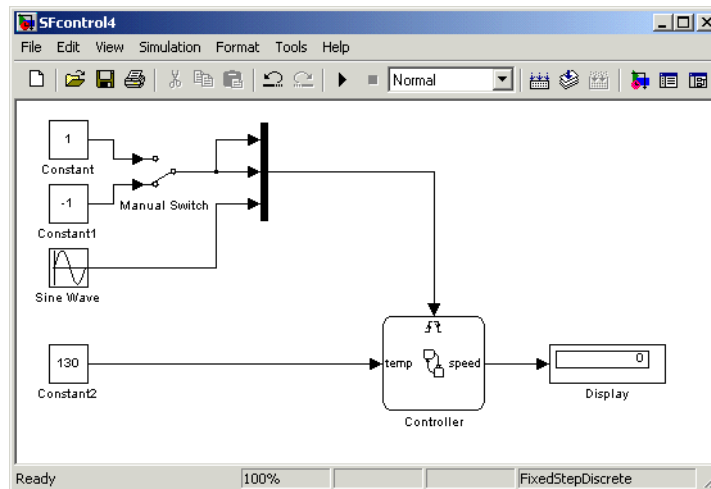
In this section, you test the features that you add to the Controller diagram and the Simulink block in the SFcontro14 model in the previous sections:

- “Adding a Sensor to the Model” on page 3-2
- “Adding a Junction for Flow Control” on page 3-8
- “Adding a Graphical Function for Convenience” on page 3-19

Test the new features by simulating the Controller diagram of the SFcontro14 model with the following steps:

- 1 In the Simulink diagram,
  - Make sure that the Manual Switch points downward to the -1 pole.
  - Set the Constant block feeding the port temp on the Stateflow Controller block to a value of 130.

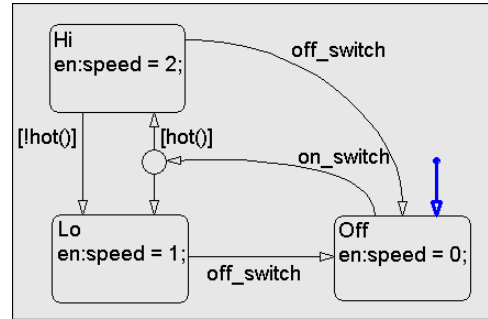
The Simulink diagram should have the following appearance:



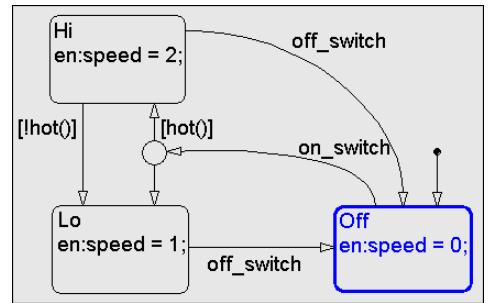
The Stateflow diagram editor for the Controller block should already be open. If not, double-click the Controller block.

- 2 Choose **Start** from the Simulink window **Simulation** menu to start simulation of the mode.

The Stateflow diagram editor updates with every broadcast event. Because the temperature sensor event, temp\_event, is broadcast every second, the Stateflow diagram updates and the default transition to state Off is taken.



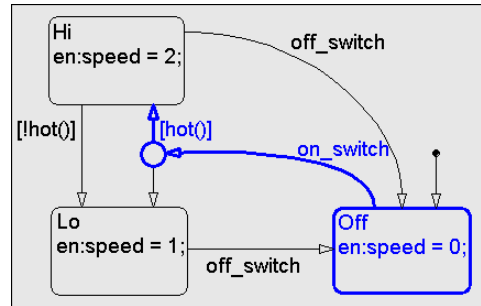
The only outgoing transition from Off requires an on\_switch event. So Off stays active until you send it that event.



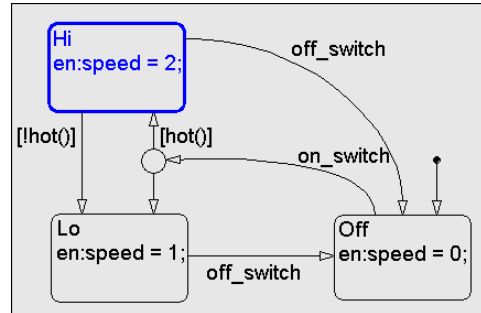
- 3 In the Simulink model, double-click the Manual Switch to move it from the -1 pole to the 1 pole.

This sends an on\_switch event to the Controller block. The Controller diagram responds as follows:

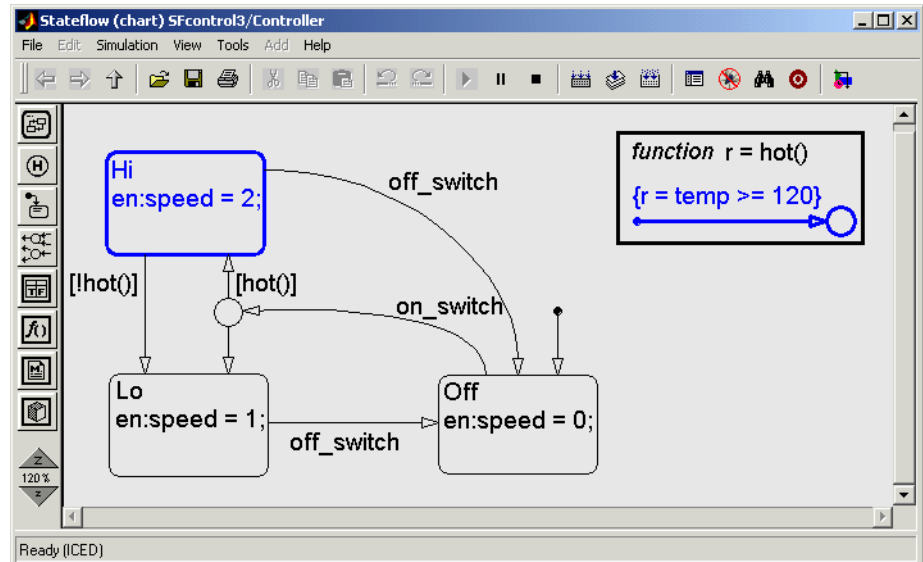
The on\_switch event causes the Controller diagram to take the transition to the junction guarded by an on\_switch event trigger. The junction gives priority to the outgoing transition with a condition, which it tests. Because hot returns a value of true, it is taken.



The value for the data speed is set to 2 (see Simulink Display block) and the Hi state becomes active, and stays active.



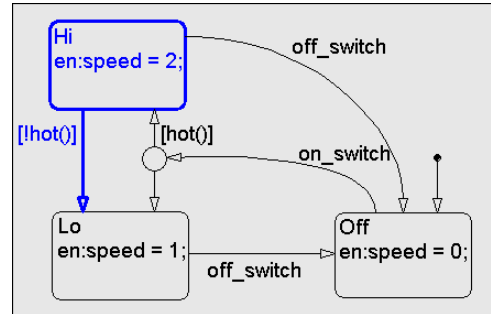
Notice that the function `hot` continues to be highlighted while the Hi state is active.



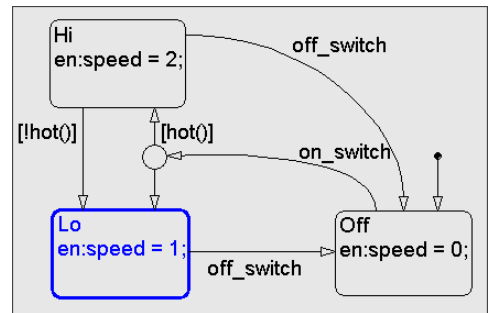
This occurs because the event `temp_event` is broadcast to the diagram every second. When a broadcast occurs, the diagram is updated, and the Hi state looks for an outgoing transition. The only outgoing transition is guarded by the condition `[!hot()]`. Because `hot()` returns a value of `true`, `!hot()` is false, and the transition is not taken. This computation is taking place every second.

- 4 In the Simulink diagram, change the value of the Constant block attached to the data temp input port to 110.

When the value of temp is changed to 110, hot() returns a value of false and !hot() is true. This means that the update event temp\_event causes the transition to Lo to be taken.



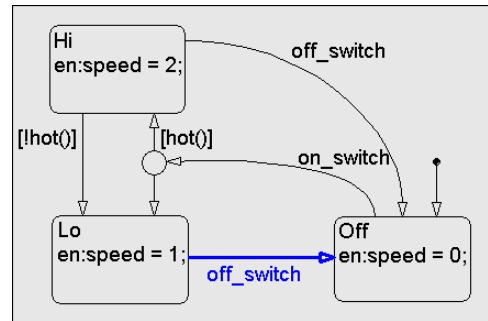
The value for the data speed is set to 1 (see Simulink Display block) and the state Lo becomes active. It stays active because its only outgoing transition requires an off\_switch event.



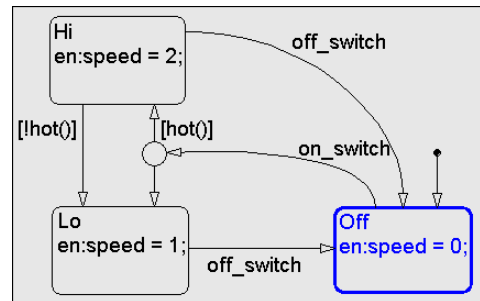
- In the Simulink model, double-click the Manual Switch to move it from the 1 pole to the -1 pole.

Because off\_switch is a falling edge triggered event, the drop in signal from 1 to -1 sends an off\_switch event to the Controller block. The Controller diagram responds as follows:

The off\_switch event causes the Controller diagram to look for an available transition. Because the transition to the Off state requires an off\_switch event, it is taken.



The value for the data speed is set to 0, and the Off state is active, and stays active, until an on\_switch event occurs. This puts the model simulation back at the beginning of its behavior cycle.

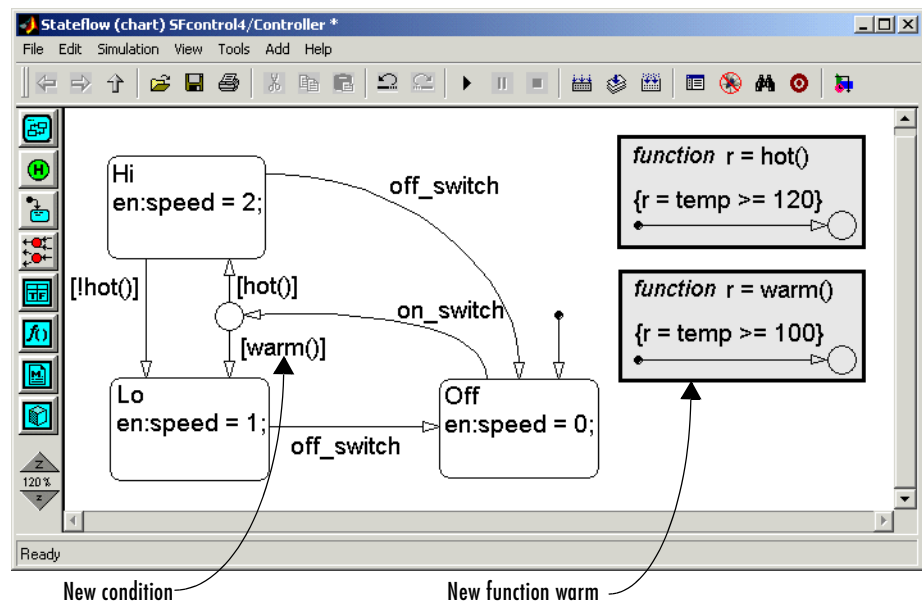




## Simulating Junction Behavior

In this section, you change the model SFcontro14 you build in “Simulating with a Sensor Event, Junction, and Function” on page 3-25 to demonstrate important junction behavior. Use the following steps to add and test a new function `warm` and the transition condition that calls it:

- 1 Add the function `warm` and a condition that calls it to the SFcontro14 model as shown.

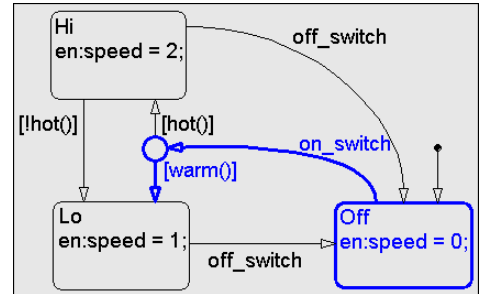


Notice that the functions `warm` and `hot` are not complementary. Both functions are true if the temperature is greater than or equal to 120. To determine the outgoing transition segment from the junction, Stateflow applies the following rule:

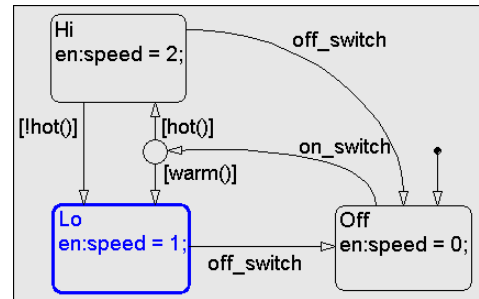
If several outgoing transitions with conditions are equally valid, Stateflow chooses the outgoing transition with the smallest clock position.

- 2 Leave the value of the Constant block input to the input data temp at 130 and simulate the model with an on\_switch event to observe the following behavior:

Because both hot and warm return a value of true, Stateflow chooses the 6 O'Clock transition segment over the 12 O'Clock one.



The value for the data speed is set to 1, and the Lo state is active, and stays active, until an off\_switch event occurs.

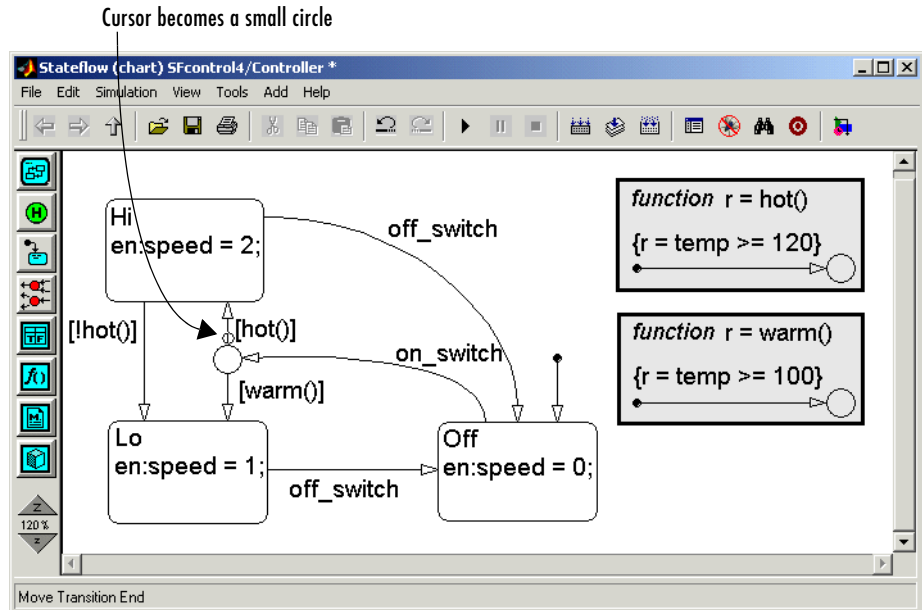


You can change this behavior by making a small change in the location of the outgoing transition in the following steps:

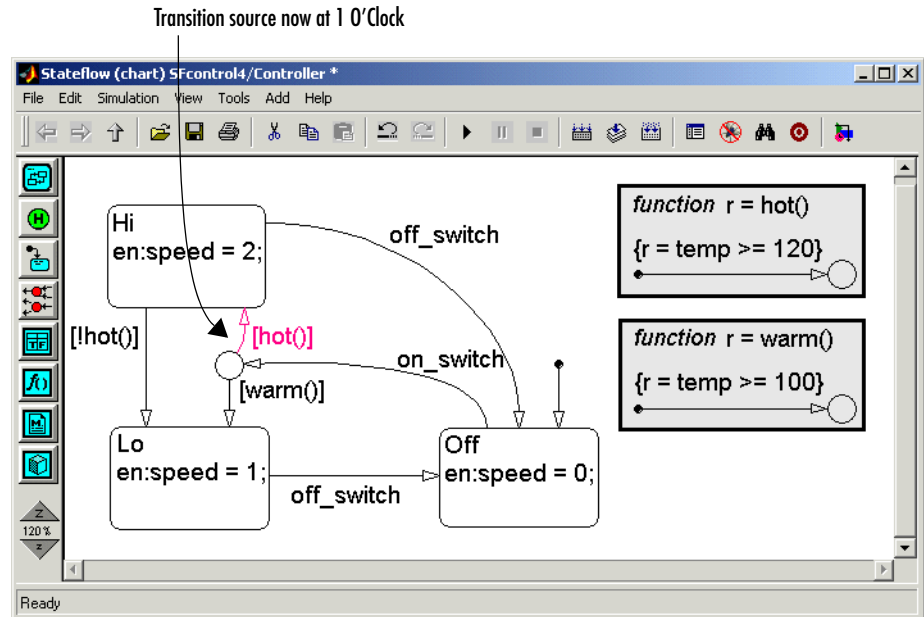
- 3 Stop simulation.

- 4 Position the cursor over the source point of the outgoing transition on the junction.

The cursor changes to a small circle as shown.



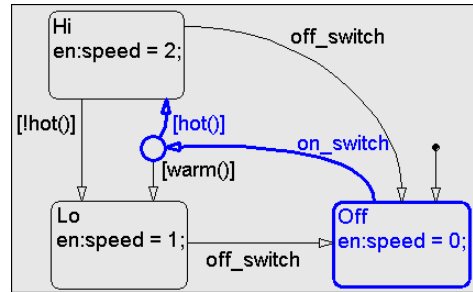
- 5 When the circle forms, click-drag the transition source over to the right and release in a position similar to the following:



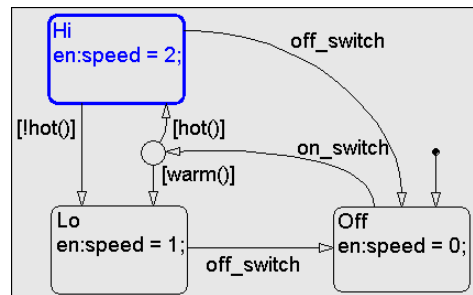
- 6 Start simulation again.

- 7 Send an on\_switch event by double-clicking the Manual Switch block to move it from the -1 pole to the 1 pole.

Both hot and warm return a value of true. This time Stateflow chooses between the two transition segments by favoring the 1 O’Clock transition segment over the 6 O’Clock one.



The Hi state becomes active, and stays active until an off\_switch event occurs (try it).



## Introducing Stateflow Semantics

The behavior applied to the outgoing transition from a junction in the example in “Simulating Junction Behavior” on page 3-31 also applies to states. For outgoing transitions of equal validity from a state or a junction, Stateflow makes the decision based on clock position.

Stateflow follows a rule like this to make sure your Stateflow diagrams are *deterministic*. If Stateflow does not have this kind of rule, it could get stuck determining the right thing to do. Instead, Stateflow has a set of rules called *semantics* that it uses to make sure your model always runs and never gets stuck.

Here are some of the semantics that you have learned in this Getting Started guide to this point:

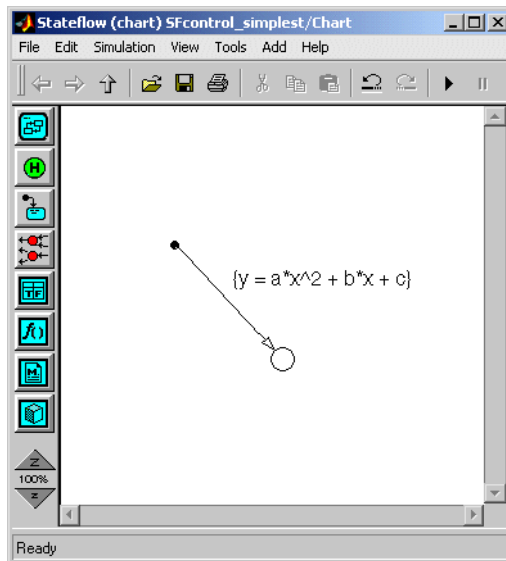
- Stateflow updates only in response to a broadcast event.
- Transitions can be guarded to take place only in response to the specified event under the specified condition.
- States can have entry actions: actions that are performed when the state is entered.
- In testing for outgoing transitions or transition segments from a junction or a state, Stateflow gives preference to transitions with conditions.
- If two or more outgoing transitions test equal in validity, Stateflow prefers the transition with the smallest clock position.

Stateflow has other semantics to make your diagrams behave deterministically. This subject is dealt with in much greater detail in the “Stateflow Semantics” chapter of Stateflow documentation. This is a good place to go to further your education on Stateflow after you have completed this Getting Started guide.

## Using Junctions in Flow Diagrams

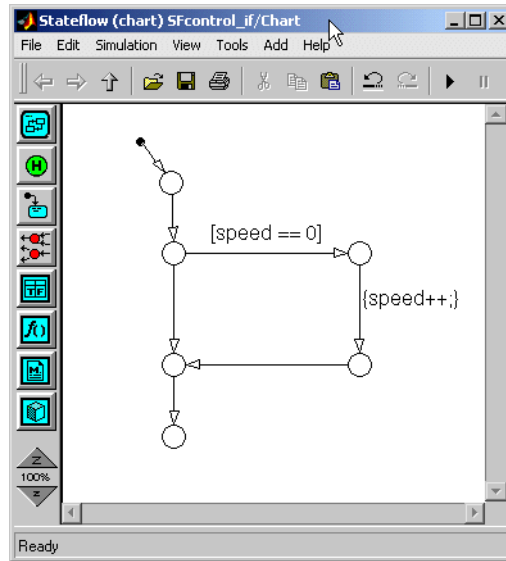
In “Simulating Junction Behavior” on page 3-31, you see the convenience of a junction in providing alternate paths to complete a transition between states. But you can use junctions without states to provide flow diagrams. Flow diagrams are “visible programming” that let you perform logical operations without using a state. In this section, you see a few examples of flow diagrams and how they can be useful to you.

The simplest flow diagram is a default transition into a junction. You have already seen this flow diagram in “Adding a Graphical Function for Convenience” on page 3-19. You use it to execute whatever actions you attach to the default transition. The following is an example of the simplest flow diagram:



When this diagram receives an update, the default transition is taken into the junction. Because there is no other flow out of the junction, it is a terminating junction. When Stateflow encounters a terminating junction, the chart is exited altogether. When the next update occurs, execution starts over again with the default transition. This is different from a state that stays active between updates.

The benefit of flow diagrams is their ability to perform complex logic in a visible environment. For example, the following flow diagram behaves like an `if` statement:



This flow diagram would have the following code representation:

```
if (speed == 1){
 speed++;
}
```

To test the behavior of this flow diagram, do the following:

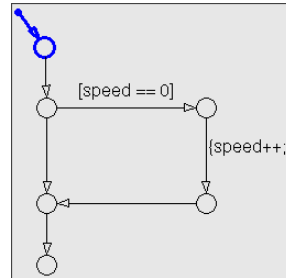
- 1 Open the model `SFcontrol12.mdl`.
- 2 Open the Stateflow block and clear the diagram.
- 3 Enter the preceding flow diagram.
- 4 Save the model as `SFcontrol_if.mdl`.

Note that the data `speed` is already defined for `SFcontrol12.mdl` (and now `SFcontrol_if.mdl`) and initialized to zero by default.

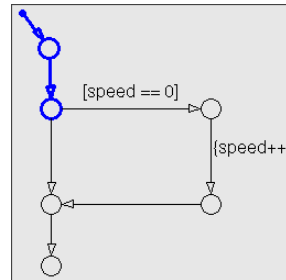


- 5 Choose **Start** from the Simulink window **Simulation** menu to start simulation of SFcontrol\_if.mdl.
- 6 In the Simulink model, double-click the Manual Switch to move it from the -1 pole to the 1 pole to send an update event:

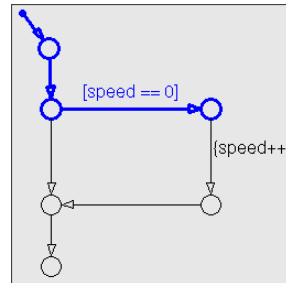
The default transition is taken into the first junction.



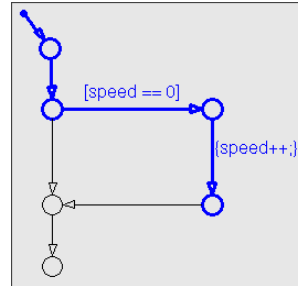
The only outgoing transition from the first junction has no condition, so it is taken.



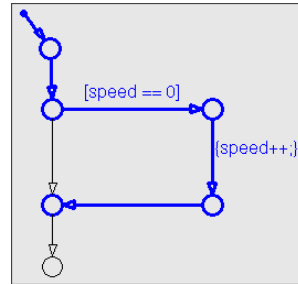
Remember that transitions with conditions have preference over unconditional transitions. Because  $\text{speed} = 0$ , the condition  $[\text{speed}==0]$  is true, and the outgoing transition to the right is taken.



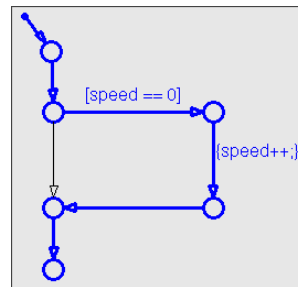
The only outgoing transition has no condition, so it is taken. Taking it executes the condition action {speed++}, which adds 1 to the value of speed, which is now 1.



The only outgoing transition has no condition, so it is taken.

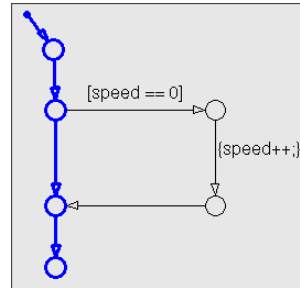


The only outgoing transition has no condition, so it is taken. Its destination is a terminating junction, a junction with no outgoing transitions. This ends execution of the diagram.

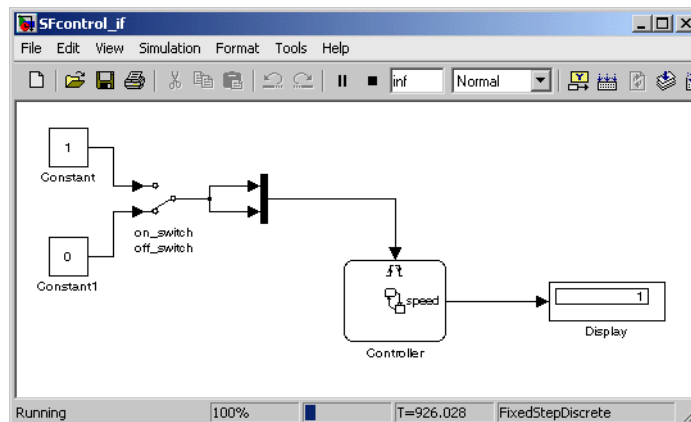


- 7 In the Simulink model, double-click the Manual Switch to move it from the 1 pole to the -1 pole to send another update event.

Because speed is no longer equal to 0, when the chart is updated again, the alternate path is taken.



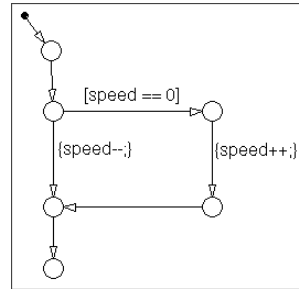
The Display block in Simulink reflects the resulting value for speed.



Besides the flow diagram for an if statement, there are a number of other flow diagram types. For example, you can add an action to the alternate path of the if diagram to create an if-else diagram.

#### IF-ELSE

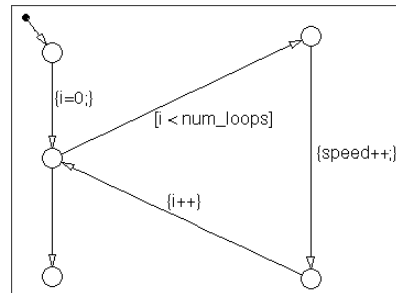
```
if (speed == 0){
 speed++;
}
else{
 speed--;
}
```



Here are a few examples of other flow diagrams that you can try on your own. Don't forget to define the counter data i.

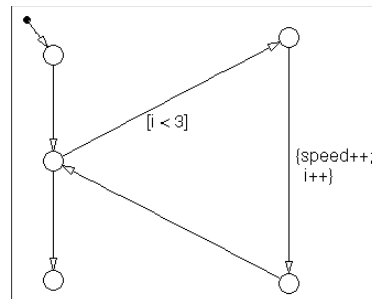
#### FOR loop

```
for (i=0;i<num_loops;i++){
 speed++;
}
```



#### WHILE

```
while (i < 3){
 speed++;
 i++;
}
```



# Controlling with Superstates

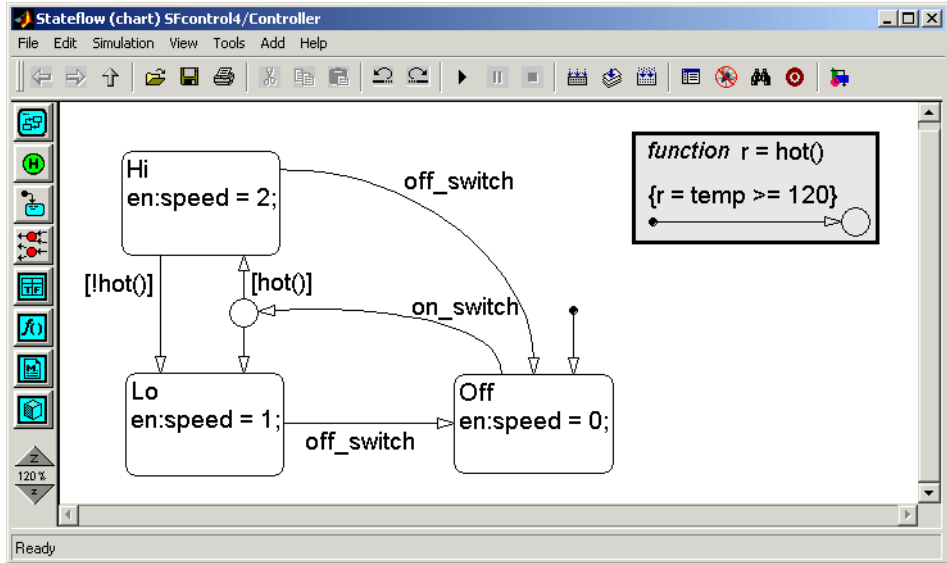
---

This chapter introduces you to superstates. These objects extend the use of states and transitions by defining states that contain other Stateflow objects. Superstates form the basis of Stateflow containment hierarchy. They also help you to simplify your Stateflow diagrams. Use the following sections to add superstates to the SFcontro14 model you build in Chapter 3, “Controlling with Junctions”:

|                                                         |                                                                                                      |
|---------------------------------------------------------|------------------------------------------------------------------------------------------------------|
| Adding Superstates to Simplify Control (p. 4-2)         | Use a junction and an additional state to delay turning on a control device for temperature changes. |
| Adding a History Junction to Save State (p. 4-14)       | Use a history junction to remember the active state of a superstate when it becomes active again.    |
| Creating Subcharts to Add More Substates (p. 4-20)      | Convert a superstate into a subchart to gain room for more substates.                                |
| Controlling Objects with Parallel Superstates (p. 4-31) | Use parallel states to control two independent devices together.                                     |

## Adding Superstates to Simplify Control

You create a junction in Chapter 3, “Controlling with Junctions” to choose between destination states in the following Stateflow diagram:



While this does give you more destination control states, it also adds more complexity to the Stateflow diagram. Instead of having one transition from the On state to the Off state, now you need transitions from both the Hi and Lo states to the Off state. In fact, for every control state that you add (for example, Medium, Medium-High, and so on), you need a transition back to the Off state.

To solve this problem, you create a superstate for the Hi and Lo states. Superstates can contain many states for a given state and give you the simplicity of only one exit point. In the following procedure topics, you create and test the On superstate that you add to contain the states Hi and Lo:

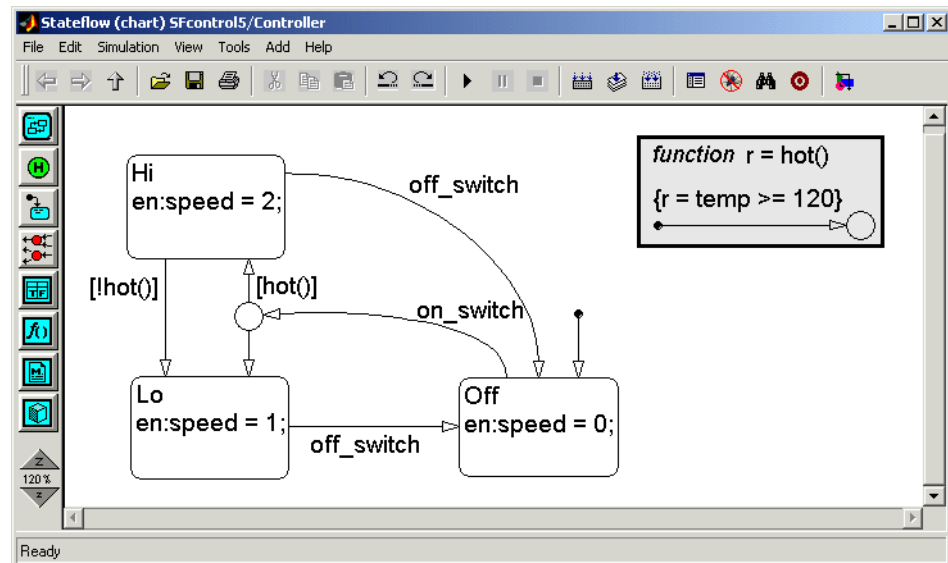
- 1 “Adding a Superstate” on page 4-3 — Add a superstate to contain the states Hi and Lo, which then become substates.
- 2 “Simulating the Superstate” on page 4-9 — Simulate the superstate to examine its behavior.

## Adding a Superstate

You add superstates to contain other objects to give them the simplicity of one start point and one exit point. In this topic, you add a superstate to contain the states Hi and Lo in the Stateflow diagram of the Simulink model SFcontro14 that you created in, Chapter 3, “Controlling with Junctions.”

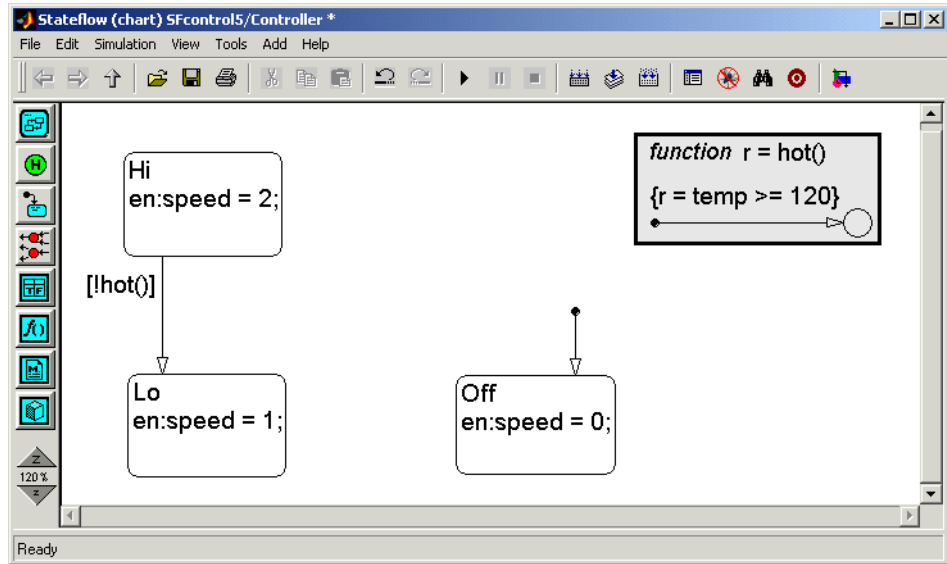
Use the following steps to add the superstate On:

- 1 If not already loaded, load the Simulink model SFcontro14 you save in “Adding a Graphical Function for Convenience” on page 3-19 and save it as SFcontro15.
- 2 Double-click on the Stateflow block Controller to open it, if not already open.

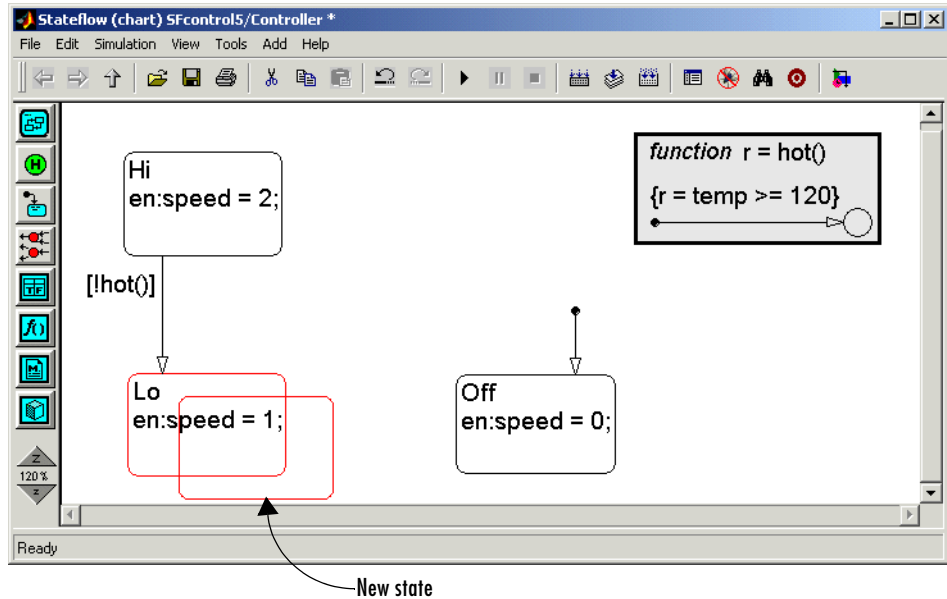


- 3 Delete all the transitions except the default transition into the Off state and the transition from Hi to Lo.

The Stateflow diagram now has the following appearance:



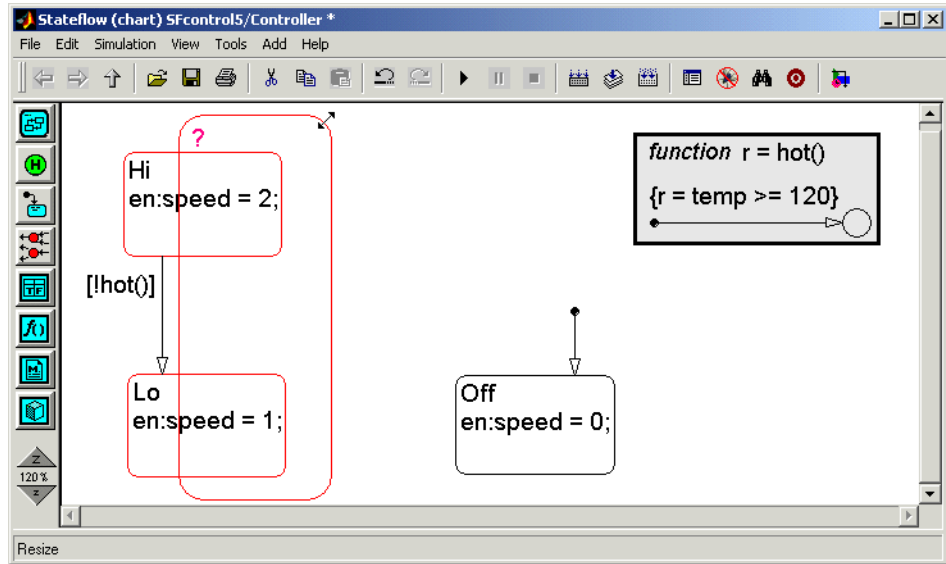


**4** Place a new state below the Lo state as shown.

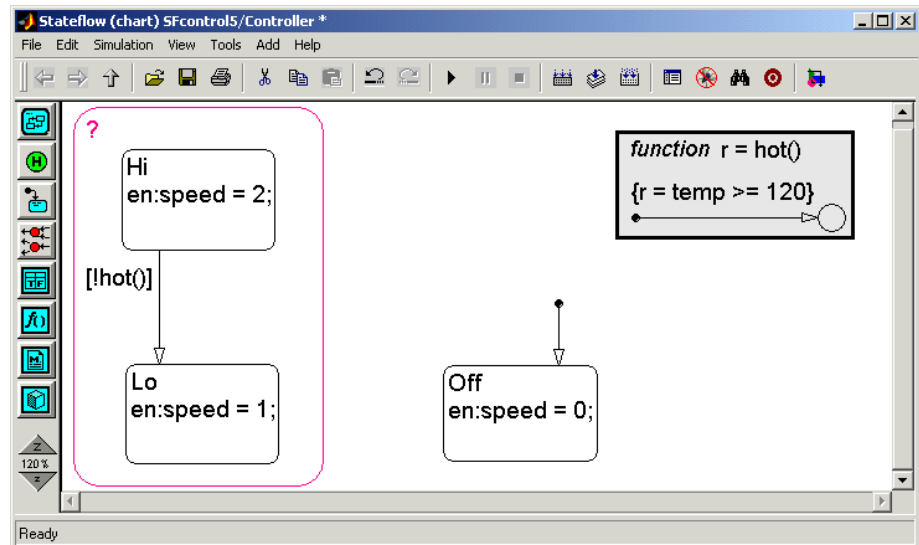
Notice that both the new state and the Lo state are highlighted. This kind of highlighting occurs for conflicts between states. Two states cannot permanently overlap each other like this. However, that is soon remedied in the following steps.

## 4 Controlling with Superstates

- 5 Place the cursor over the upper-right corner of the new state until it turns into a set of diagonal arrows and click-drag the upper-right corner of the new state above the state Hi as shown.



- Click-drag the upper-left corner of the new state to the left of Hi and release the mouse as shown.

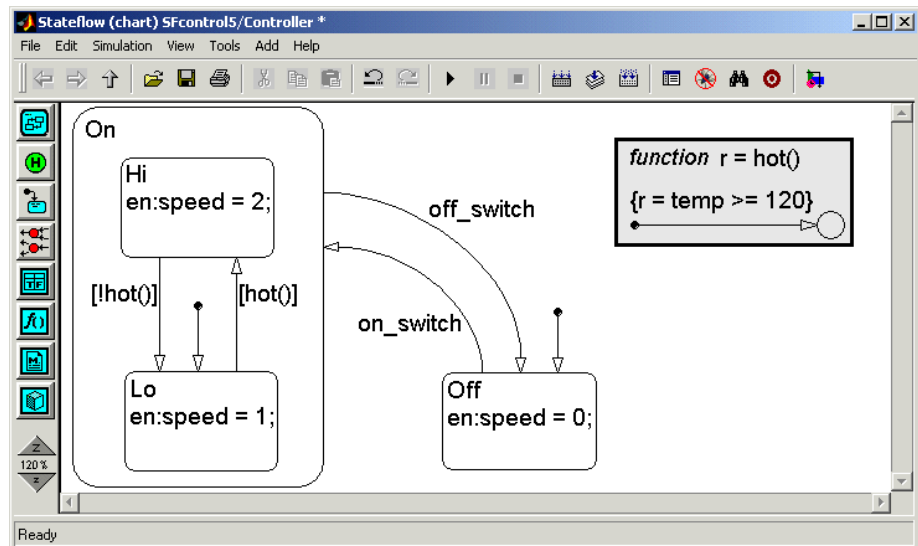


Notice that even though the new state is highlighted, the states Hi and Lo are not. These states are no longer in conflict with the new state, which now contains them.

Finish the diagram with the following steps:

- 7 Name the new state On.
- 8 Draw a default transition to the state Lo.
- 9 Click-drag a transition from Lo to Hi and label it with `[hot()]`.
- 10 Click-drag a transition from On to Off and label it with `off_switch`.
- 11 Click-drag a transition from On to Off and label it with `on_switch`.

This is what the final Stateflow diagram should look like:



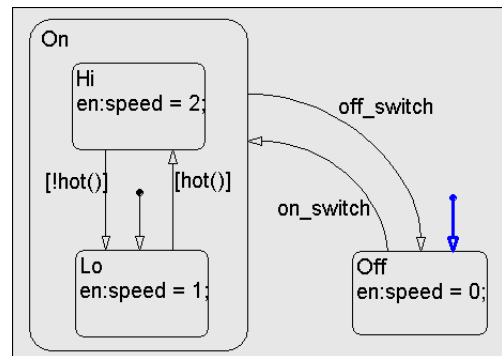
- 12 Save the model (SFcontrol15).

## Simulating the Superstate

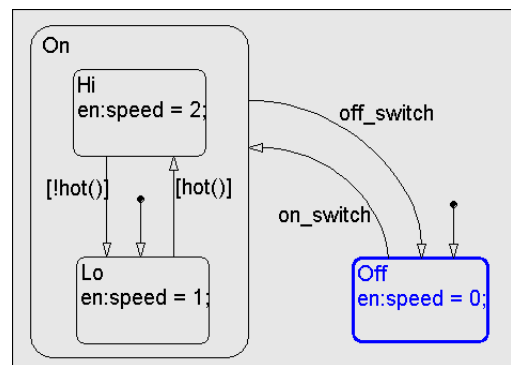
In this section, you test the superstate that you add to the Controller diagram of the model SFcontro15 in “Adding Superstates to Simplify Control” on page 4-2. Test the new superstate with the following steps:

- 1 In the Simulink diagram,
  - Make sure that the Manual Switch points downward to the -1 pole.
  - Set the Constant block feeding the port temp on the Stateflow Controller block to a value of 110.
- 2 Choose **Start** from the Simulink window **Simulation** menu to start simulation of the model and observe the following:

The Stateflow diagram updates with the first temp event and the default transition to state Off is taken.

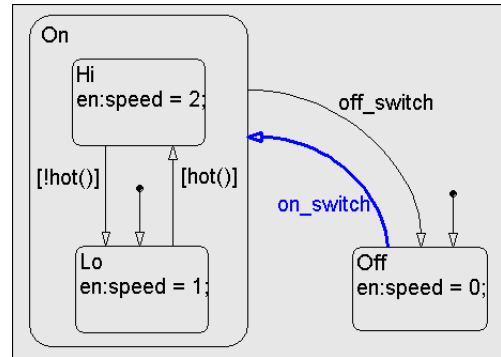


Off is entered and stays active.

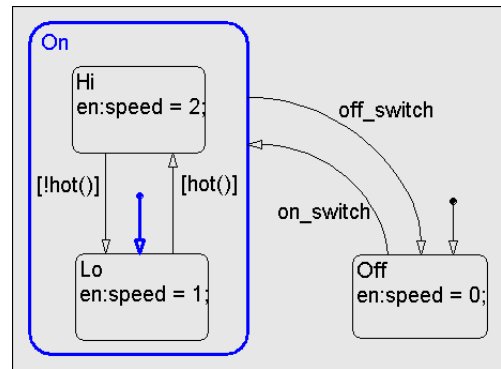


- 3 In the Simulink model, with the Manual Switch block, send an on\_switch event to the Controller block.

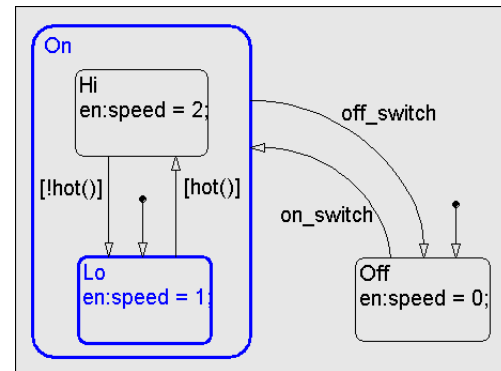
The transition to the On state is taken.



The On state is entered and becomes active. The default transition to substate Lo is taken.



The Lo state is entered. Both On and its substate Lo remain active.

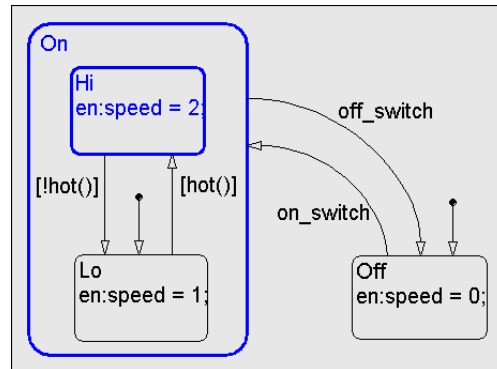
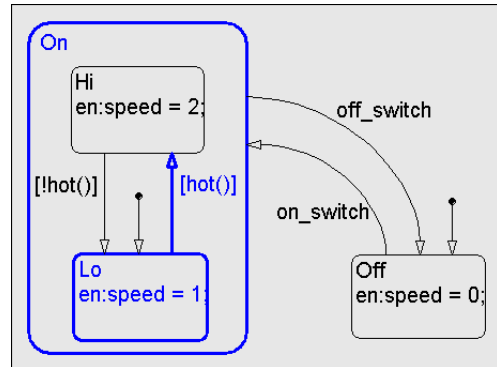


- 4 In the Simulink diagram, change the value of the Constant block attached to the data temp input port to 130.

The Stateflow diagram updates with a temp event. Both the On and Lo state test for valid outgoing transitions. Because the value of temp is changed to 130, the function Hot() returns a value of true and the transition from Lo to Hi is taken.

The transition from On to Off responds only to the event off\_switch and is, therefore, not taken.

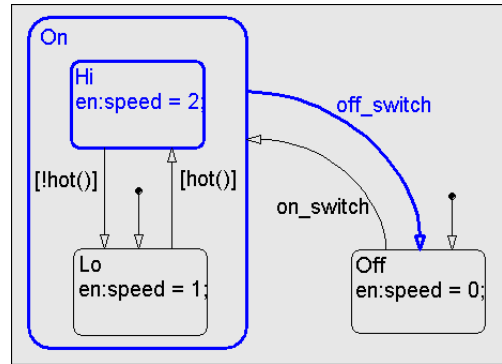
The value for the data speed is set to 2 and the state Hi becomes active and stays active, along with its parent state, On.



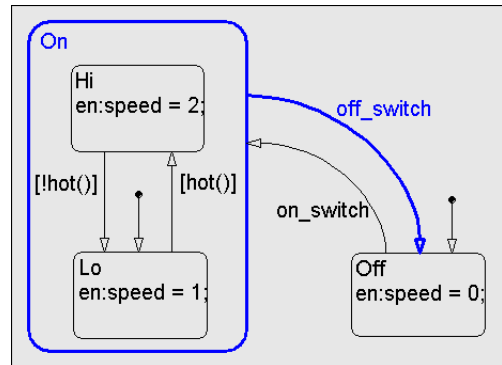
- 5 Alternate the value for temp between 110 and 130 and observe the transitions from the Hi to the Lo state and from the Lo to the Hi state.

- 6 In the Simulink diagram, use the Manual Switch block to send an off\_switch event trigger to the Stateflow block Controller.

With the broadcast of the off\_switch event, the transition from On to Off is taken.

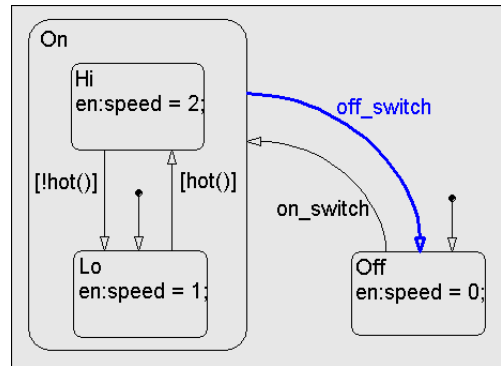


Before On is exited, its active substate, Hi, is exited and is no longer active.

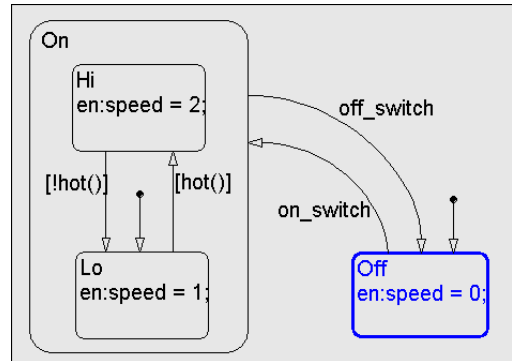




On is exited and is no longer active.



Finally, the data speed is set to 0 and the state Off is entered and becomes active.



### Adding a History Junction to Save State

If a Stateflow diagram exits and reenters a superstate like the one in “Simulating the Superstate” on page 4-9, the destination substate of the default transition becomes the first active substate. While this behavior might be acceptable for some superstates, for others, it might be preferable to reenter the most recently active substate.

For example, suppose that a superstate represents the state of a clothes washer and its substates represent its different cycles: soak, wash, rinse, and spin. If you lift the lid of a washer while it is in the spin cycle, the washer stops in response. If you drop the lid, the washer resumes its spin cycle. If, instead, the soak cycle started again, it might take you a long time to get your clothes washed!

History junctions extend the ability of charts and superstates by recording the activity of their substates. You use a history junction in a chart or a superstate to indicate that its last active substate becomes active when the chart or superstate is reentered. Add and simulate the effects of a history junction in the following procedure topics:

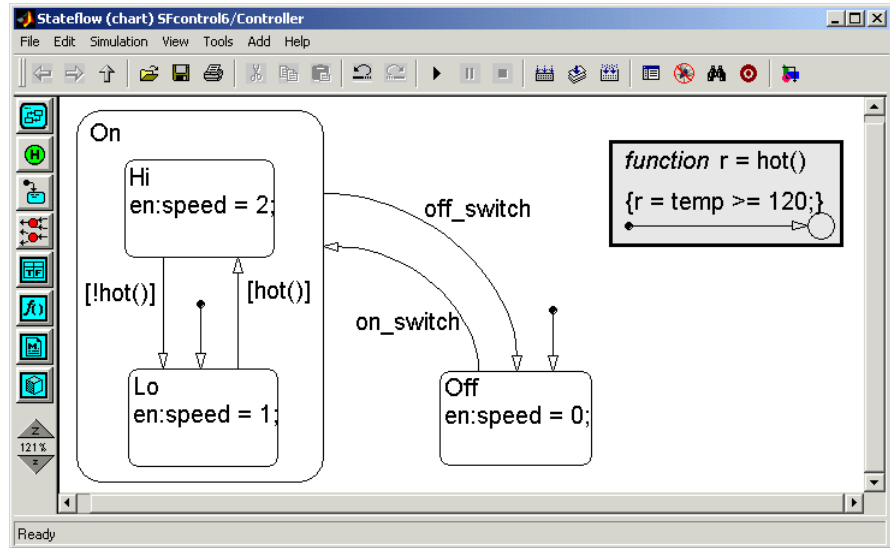
- 1 “Adding a History Junction to a Superstate” on page 4-14
- 2 “Simulating with a History Junction” on page 4-17


### Adding a History Junction to a Superstate

History junctions store the most recently active substate of a superstate. When the superstate is exited and reentered, this is the substate that becomes active. In this topic, you add a history junction to a superstate in the following steps:

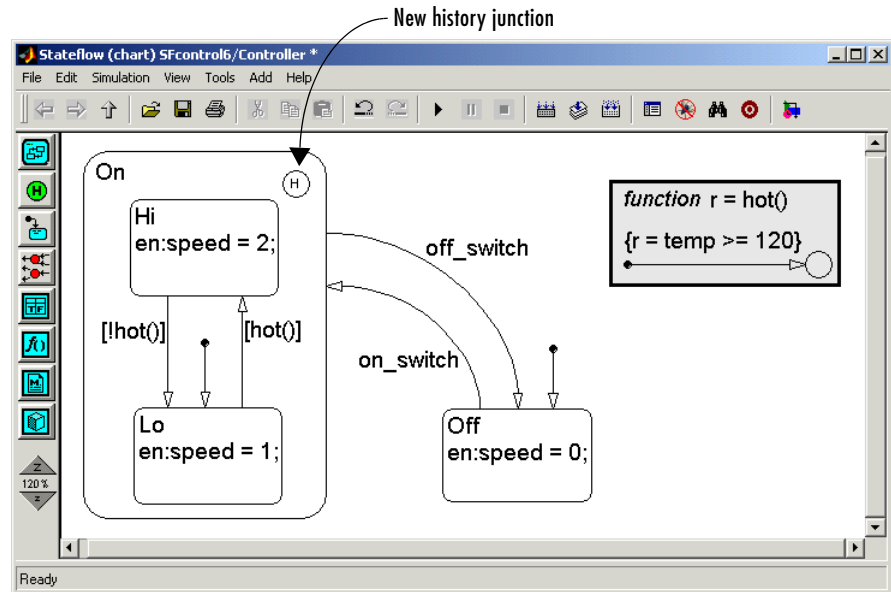
- 1 If not already loaded, load the Simulink model SFcontro15, which you save in “Adding Superstates to Simplify Control” on page 4-2.
- 2 Save the model as SFcontro16.

- 3 Open the Stateflow diagram for the Controller block as shown.



4 In the drawing toolbar, click the History Junction tool .

5 Move the cursor to the upper-right corner of the superstate On and click to place a history junction as shown.



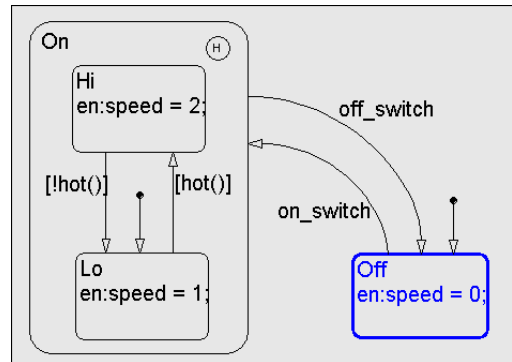
6 Save the model (SFcontrol16).

## Simulating with a History Junction

Observe the behavior of the substates of the superstate On during simulation of the Stateflow diagram in the SFcontro16 model.

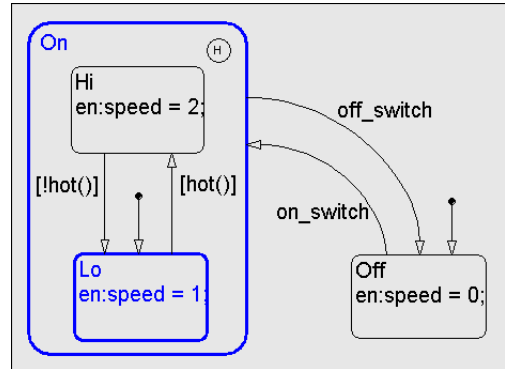
- 1 Start simulation.

When simulation starts, the first temp\_event event updates the diagram and the state Off is entered.

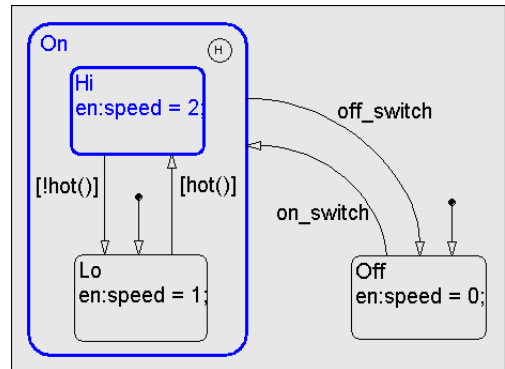


- 2 Use the Manual Switch to send an on\_switch event to the Controller block.

The on\_switch event triggers the transition to On. Both On and its default state, Lo, become active.

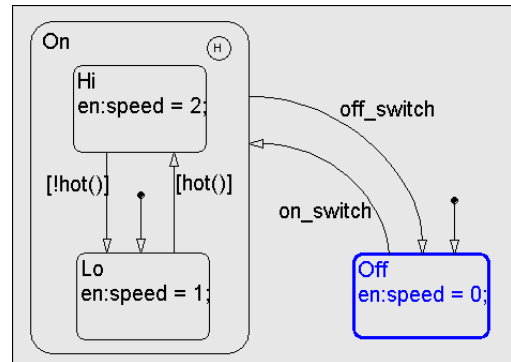


The periodic event temp\_event updates the diagram every second. Because the data temp is set to 130, the function hot returns a true and the transition from Lo to Hi is taken. State Hi becomes active.



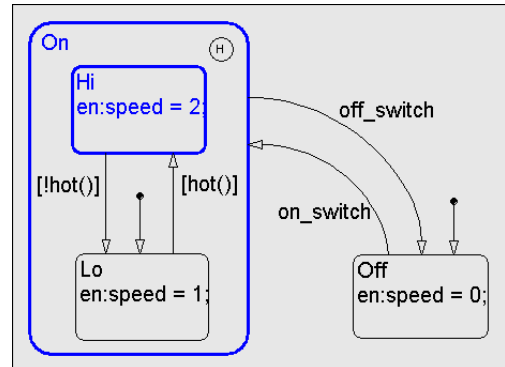
- 3 Use the Manual Switch to send an `off_switch` event to the Controller block.

The `off_switch` event triggers a transition from On to Off. Off becomes active.



- 4 Use the Manual Switch to send an `on_switch` event to the Controller block.

The `on_switch` event triggers a transition from Off to On. This time, Hi is the active substate, and not Lo, the default substate.



### Creating Subcharts to Add More Substates

Many superstates have more substates of control than just Hi and Lo. For example, an On state for a washer must have states for soaking, washing, rinsing, and spinning. The On state for the transmission of a car must have substates park, reverse, drive, lo, and overdrive. In this section, you add more substates to the superstate On in the model SFcontro16 that you create in “Adding Superstates to Simplify Control” on page 4-2. But adding substates can make your diagrams crowded. To make your diagrams more readable, Stateflow diagrams let you create subcharts.

Create more substates of the state On by making it a subchart in the following procedure topics:

- 1 “Converting a State to a Subchart” on page 4-20
- 2 “Simulating a Subchart” on page 4-28

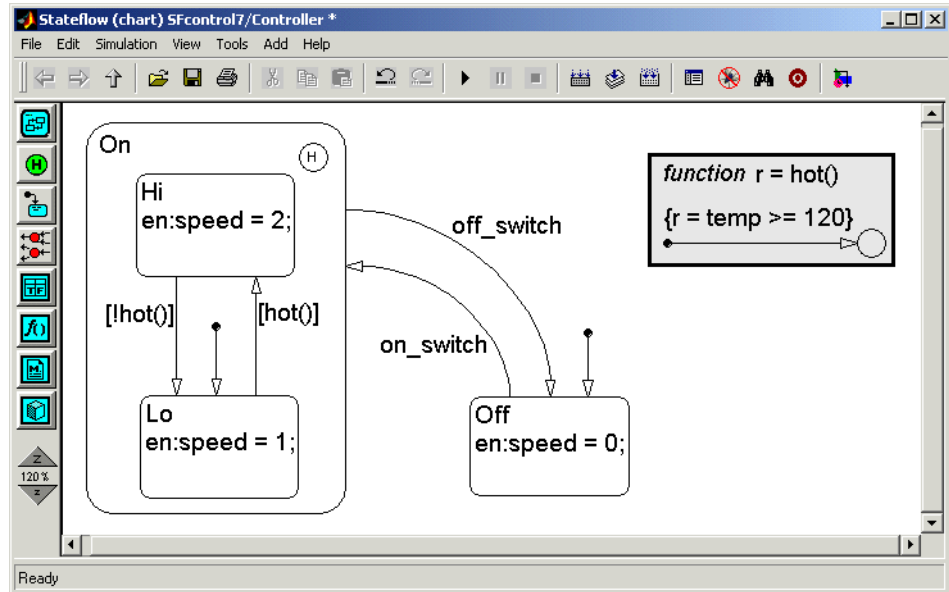
#### Converting a State to a Subchart

Let’s suppose that you want to add more substates to the superstate On that you create in “Adding a Superstate” on page 4-3. You can always click-drag the edges of the Stateflow diagram editor window to make it as big as possible. However, if you add enough substates, you could easily run out of room and start to impinge on other elements of the diagram. Instead, you can turn the On state into its own chart, a subchart, with the following steps:

- 1 If not already loaded, load the Simulink model SFcontro16 you save in “Adding Superstates to Simplify Control” on page 4-2 and save it as SFcontro17.



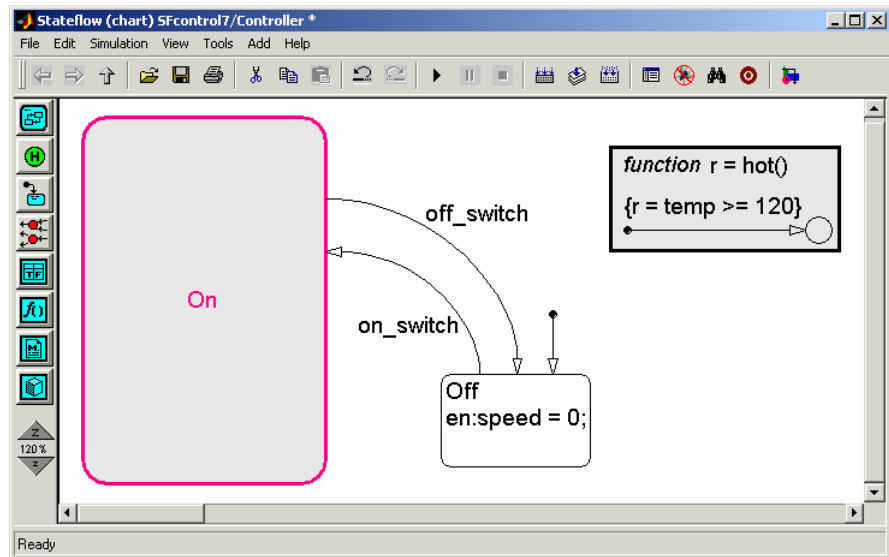
2 Double-click on the Stateflow block Chart to open it.



3 Right-click on the state On.

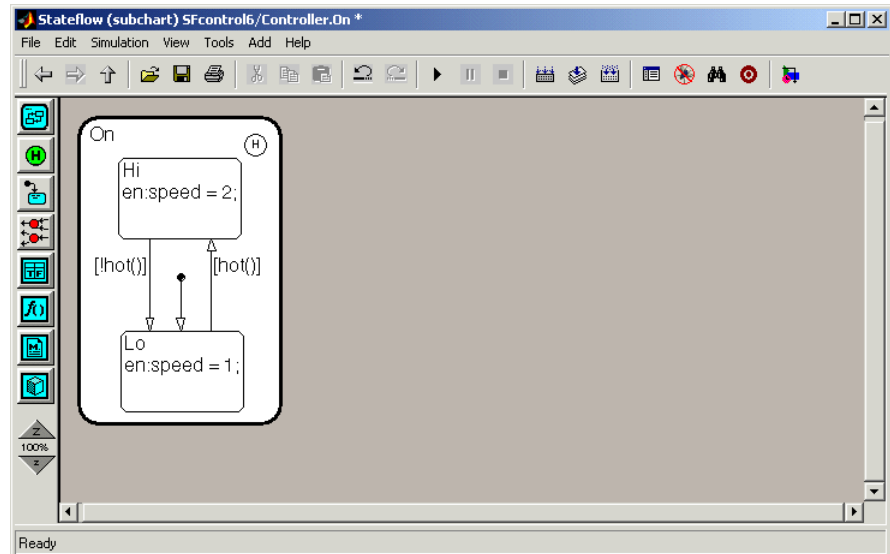
- 4 In the resulting pop-up menu, select **Make Contents** followed by **Subcharted** in the submenu that results.

The state **On** now becomes opaque as shown.

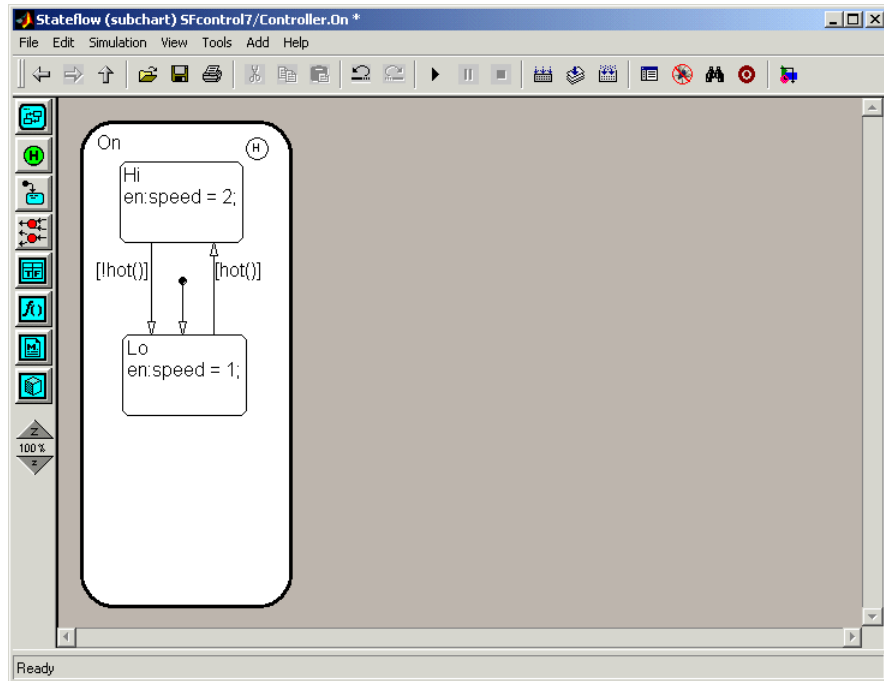


**5** Double-click the On state.

The On state and its contents now appear in a chart of their own, a subchart.



- Vertically enlarge the subchart window and the state On to have the following appearance:



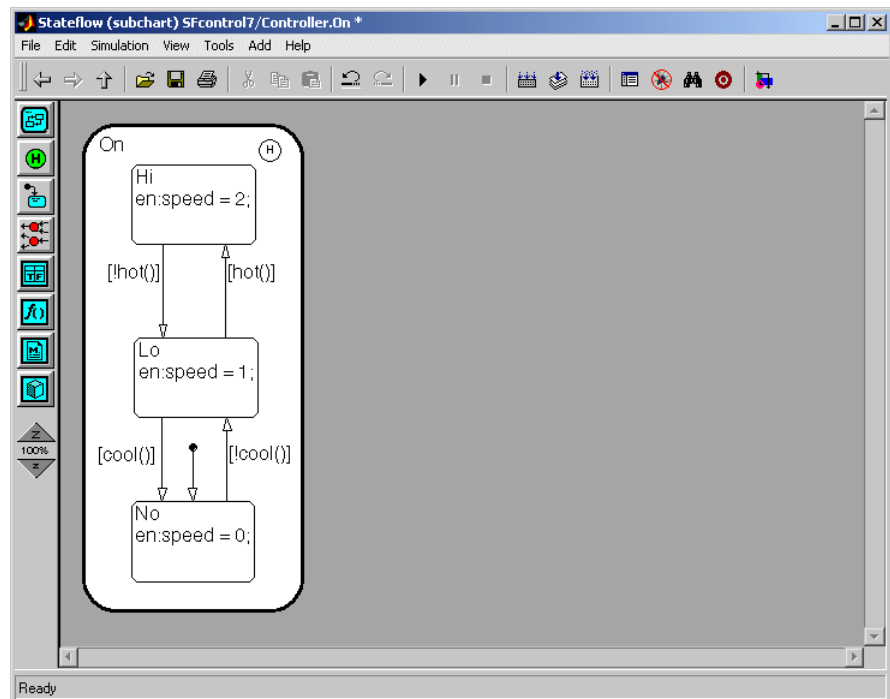
As a subchart, the state On can be any size.


- 7 At the bottom of state On, add a new state named No with the entry action `speed = 0;`.
- 8 Delete the default transition to the state Lo and add a new default transition to the state No.

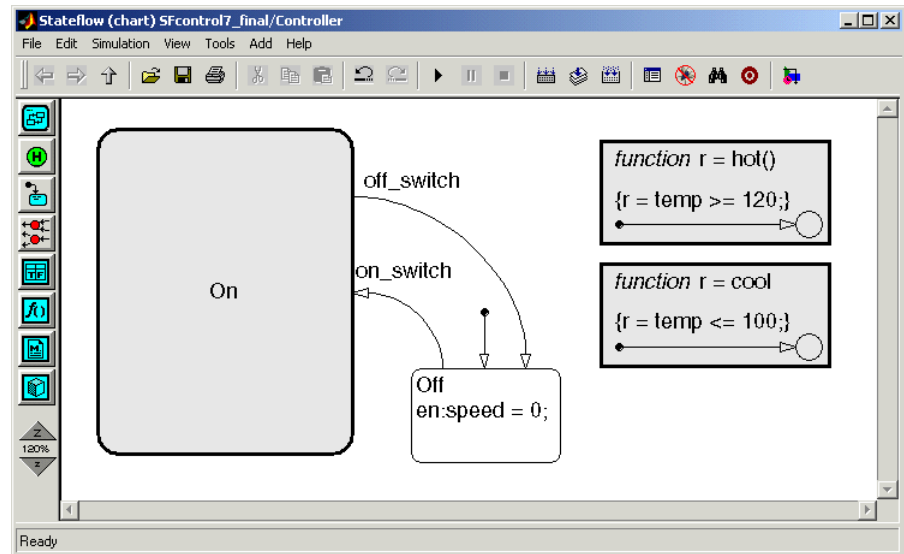
Only one substate can be the default state for a superstate.

- 9 Draw a transition from No to Lo and label it `[!cool()]`.
- 10 Draw a transition from Lo to No and label it `[cool()]`.

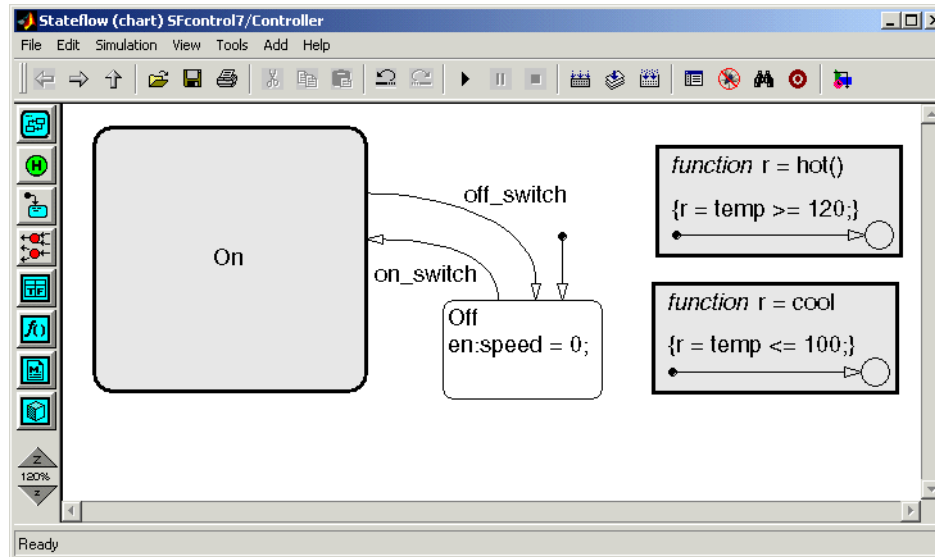
This is what you should now see:



- 11 In the Stateflow diagram editor, click the Up to Controller tool . Focus returns to the original Stateflow diagram.
- 12 Add the function `cool()` and group it, as shown.



- 13** You can now resize the state On and move other diagram objects to make the diagram appear a little neater, as shown.



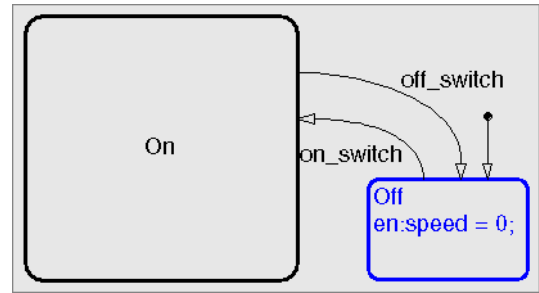
- 14** Save the model (SFcontrol17).

## Simulating a Subchart

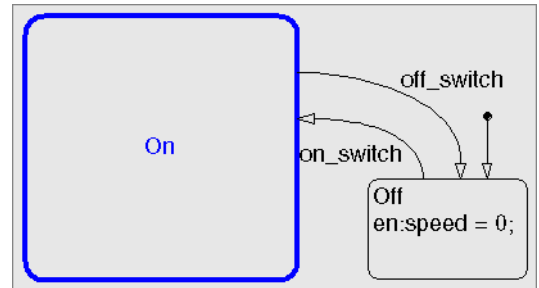
In “Converting a State to a Subchart” on page 4-20, you edit a Stateflow diagram to turn the superstate On into a subchart and add substates to it. Observe the behavior of the subchart On and its substates during simulation of the Stateflow diagram in the SFcontrol17 model in the following steps:

- 1 Start the simulation and change the value of the input data temp to 90.

When simulation starts, the first temp event updates the diagram and the state Off is entered.



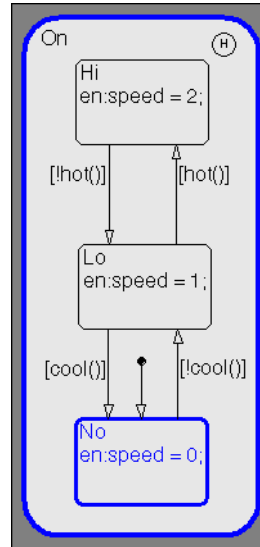
- 2 Send an on\_switch event with the manual switch.





**3** Double-click the On state.

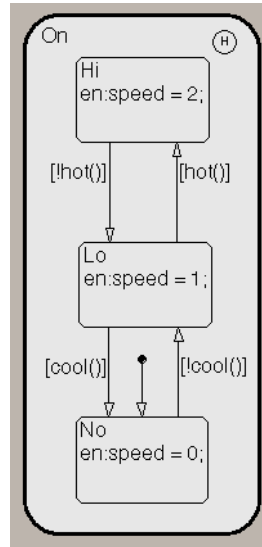
This opens the subchart On to expose the activity of its child states.



**4** Change the input value for temp to take the transitions between the states No, Lo, and Hi.

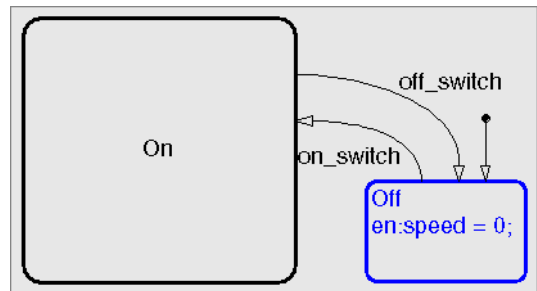
- 5 Send an off\_switch event with the Manual Switch block.

The active substate On exits.



- 6 Click the Up to Controller tool .

The transition from the On state is taken to the Off state, and the Off state becomes active.



## Controlling Objects with Parallel Superstates

In this section, you control two control objects instead of one. For example, instead of controlling a single fan with two speeds, you use two single-speed fans, each with its own control. Because both fans have their own control and must run concurrently, you use another form of hierarchy to control the fans in Stateflow: parallel states.

You use parallel states in Stateflow to control concurrent processes. Concurrent processes are control processes that take place simultaneously, but independently. Because each fan is controlled separately, as if it were the only fan running, you control them concurrently.

Use the following procedure topics to create a model with parallel states and test them in simulation:

- 1 “Creating Parallel States” on page 4-32 — Create a model with two Stateflow diagrams, each executing in its own parallel state.
- 2 “Simulating Parallel States” on page 4-37 — Test the parallel states you create in simulation.

### Creating Exclusive States

The states that you have seen so far in this Getting Started guide are exclusive states. Exclusive states have solid borders and are sometimes referred to as AND states. You create them by assigning the parent of the states *exclusive decomposition*. Because this is the default setting for Stateflow charts, all the states you have created so far have been exclusive.

The distinguishing feature about exclusive states is that only one state among brother and sister states can become active at any one time. For example, if you have a chart with the states On and Off, only one of these states can become active when you update the chart. You specify that state with a default transition.

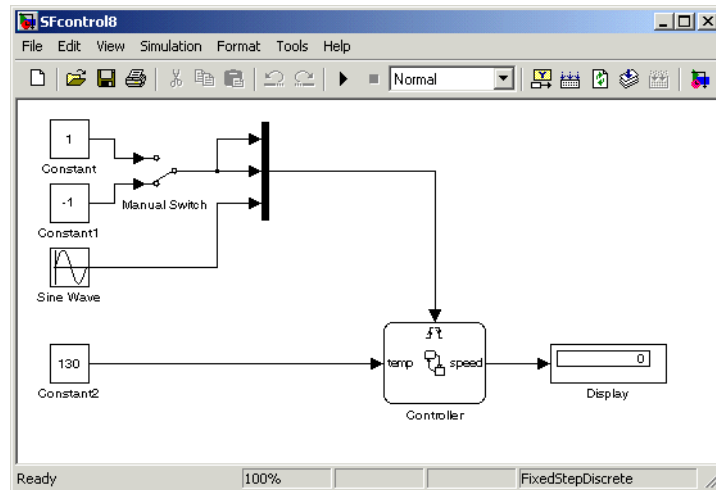
### Creating Parallel States

In “Creating Exclusive States” on page 4-31, you learn about exclusive states, the default state of Stateflow, the type of state that you have been creating in all the previous material of this Getting Started guide. In this topic, you learn about parallel states, the only other type of state in Stateflow.

Parallel states have dashed borders and are sometimes referred to as OR states. You create them by assigning the parent of the states *parallel decomposition*. The distinguishing feature about parallel states is that all brother and sister parallel states are active at the same time. You could say that these states are actively concurrent with each other.

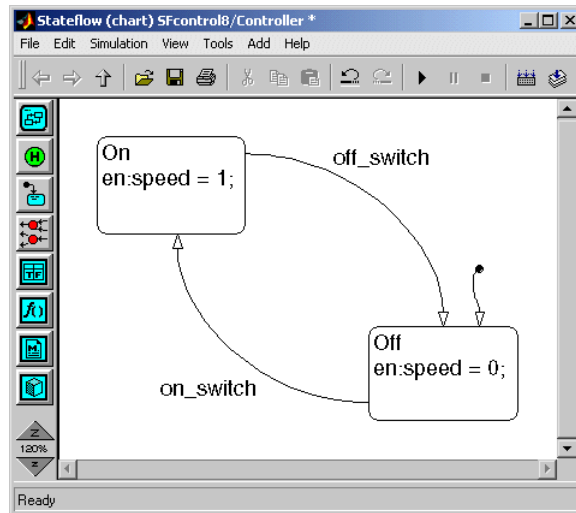
In this topic, you create parallel states to provide control over two individual control objects in the following steps:

- 1 Load the Simulink model SFcontrol13 you save in “Adding a Sensor to the Model” on page 3-2 and save it as SFcontrol18.

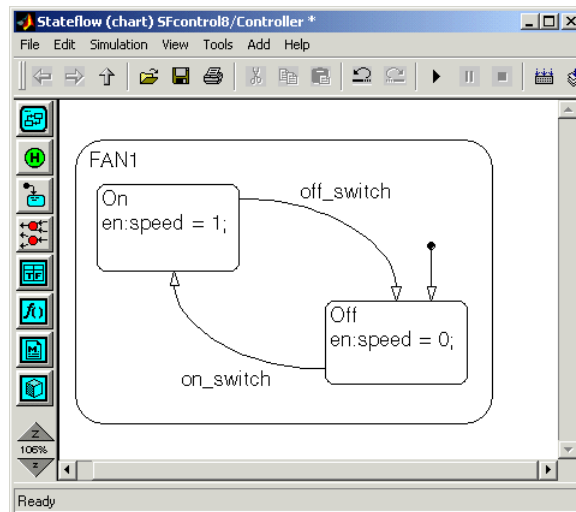


- 2 Change the Constant block value for the input data temp to 130.

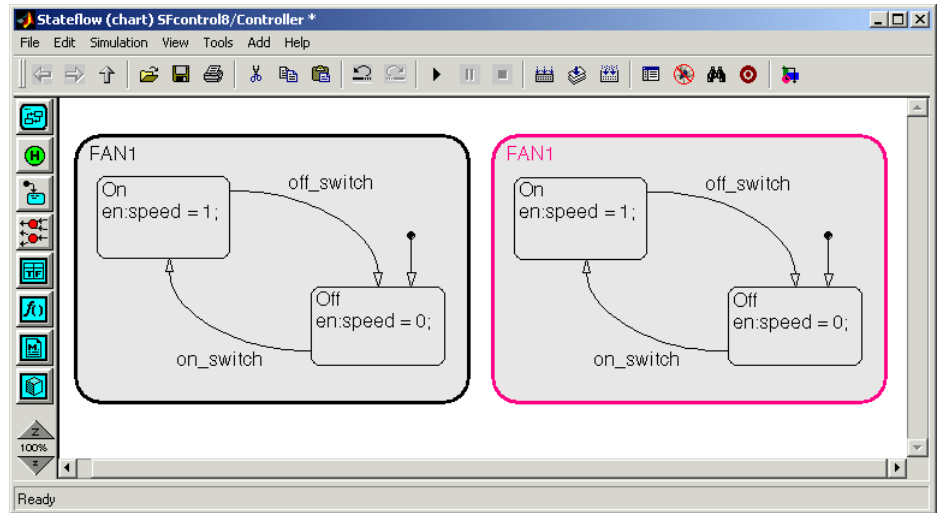
- 3 Open the Stateflow diagram for the Stateflow block Controller.



- 4 Use zoom to decrease the size of the diagram and draw a superstate around the diagram, as shown.



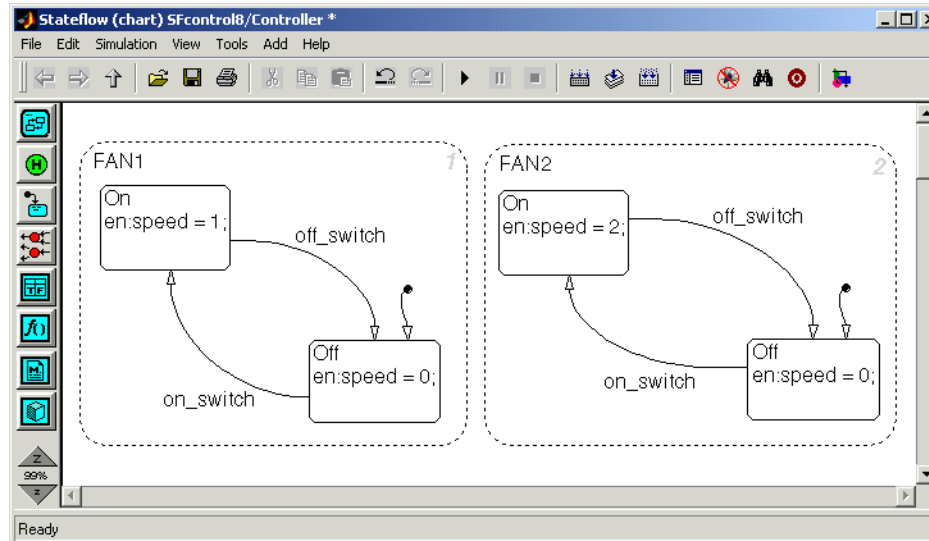
- 5 Label the superstate FAN1 and double-click it to group it.
- 6 Increase the Stateflow diagram window to twice its horizontal size.
- 7 Place the cursor in state FAN1 and right-click and drag a copy of it to the right as shown.



- 8 Rename the copied superstate on the right to FAN2.

- 9 Right-click empty space in the diagram and, from the resulting context menu, select **Decomposition** followed by **Parallel(AND)**.

Notice that the states FAN1 and FAN2 now have dashed borders.



Notice that both states have numbers in their upper-right corner. These numbers give the priority of one parallel state over another. Even though parallel states have concurrent behavior, they are not completely concurrent. They are executed according to their priority number.

Priority numbers are assigned to parallel states automatically by Stateflow based on the position of the state. Parallel states that have a higher position in the Stateflow diagram editor receive a higher priority. Parallel states on the left have priority over states on the right with the same vertical position.

- 10** Move the FAN2 state up or down and notice the change in priority number for both FAN1 and FAN2.

Notice that the Stateflow diagram has two states named On and two states named Off. These states do not conflict because each set of On and Off states is in its own namespace. States must be named uniquely, but only within their namespace.

- 11** In the parallel state FAN2, label the transition from Off to On as follows:

```
[in(FAN1.On) & (temp>120)]
```

The expression `in(FAN1.On)` checks if the On state in FAN1 is active before the transition is taken. This use of the built-in function `in` is referred to as an *implicit event*. In this case, if the state Off in FAN2 is active and the state On in FAN1 is active, `in(FAN1.On)` is true and the transition from Off to On in FAN2 is taken.

---

**Note** You use namespace dot notation to refer to an object in another namespace.

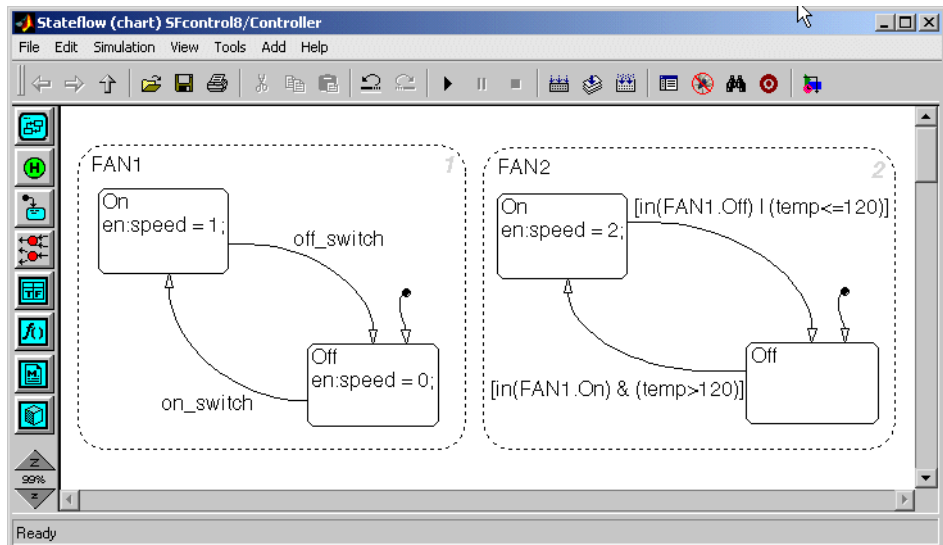
---



**12** Continue making the following changes to the diagram in parallel state FAN2:

- Label the transition from On to Off as follows:  
[in(FAN1.Off) | (temp<=120)]
- Change the entry action of state On to speed = 2;.
- Remove the entry action of state Off.

After these changes are made, this is what you should see.



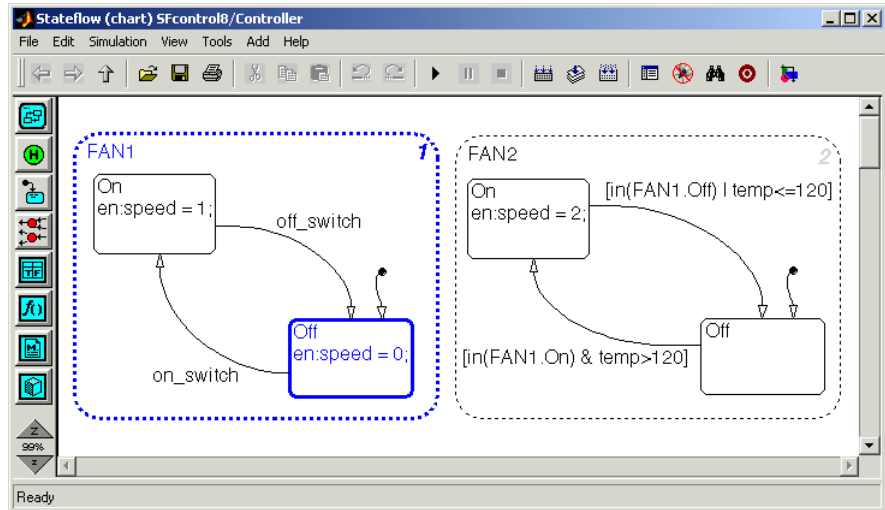
**13** Save the model (SFcontrol8).

## Simulating Parallel States

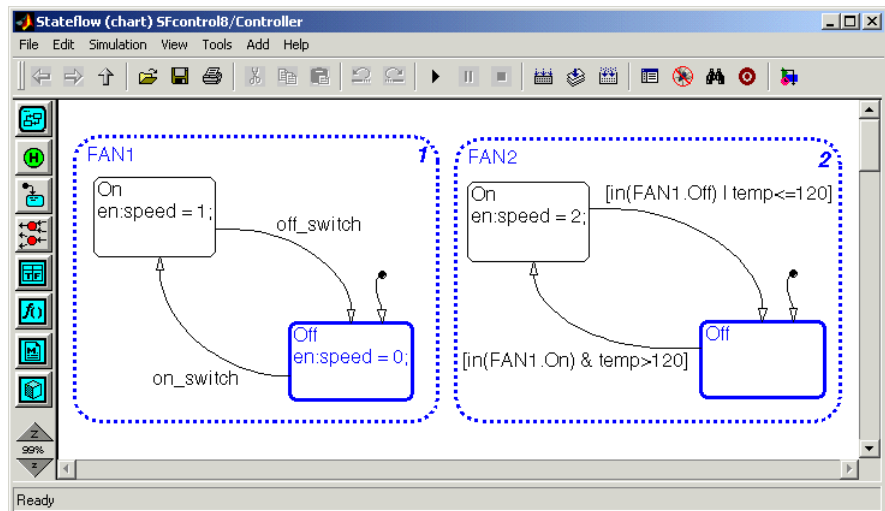
In “Creating Parallel States” on page 4-32, you create parallel states, that each have their own Stateflow diagram. Observe the behavior of each parallel state during simulation of the Stateflow diagram in the model SFcontrol8 in the following steps.

- 1 Start the simulation.

Parallel state FAN1 becomes active followed by its default substate Off.

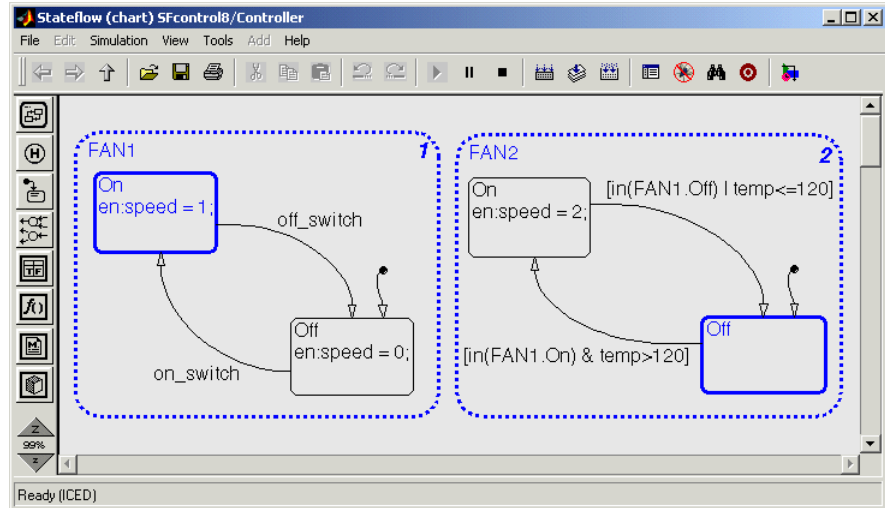


Parallel state FAN2 becomes active followed by its default substate Off.

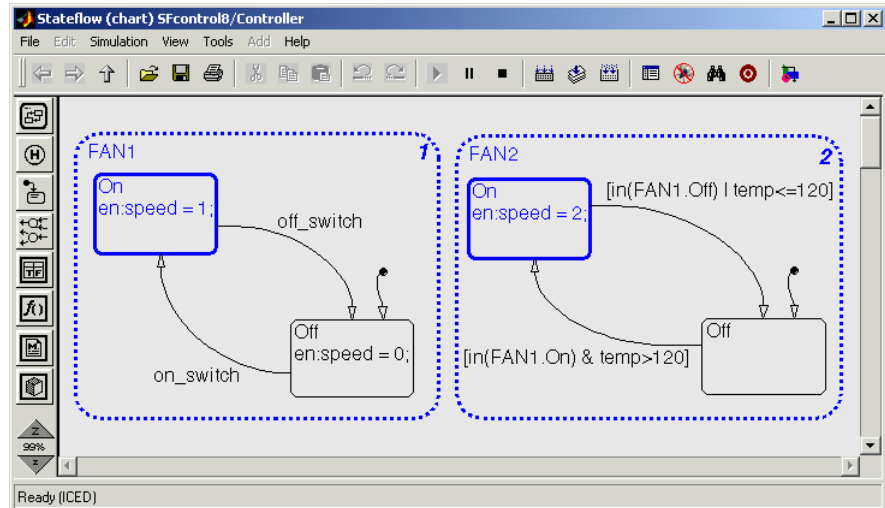


- Use the Manual Switch block to send an `on_switch` event.

FAN1 substate On becomes active.

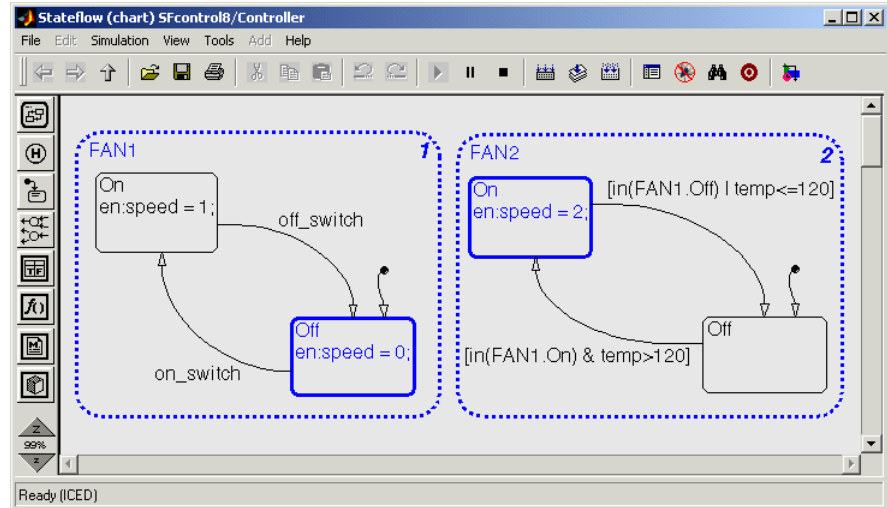


FAN2 substate On becomes active because FAN1 substate On is active and the temperature is greater than 120.

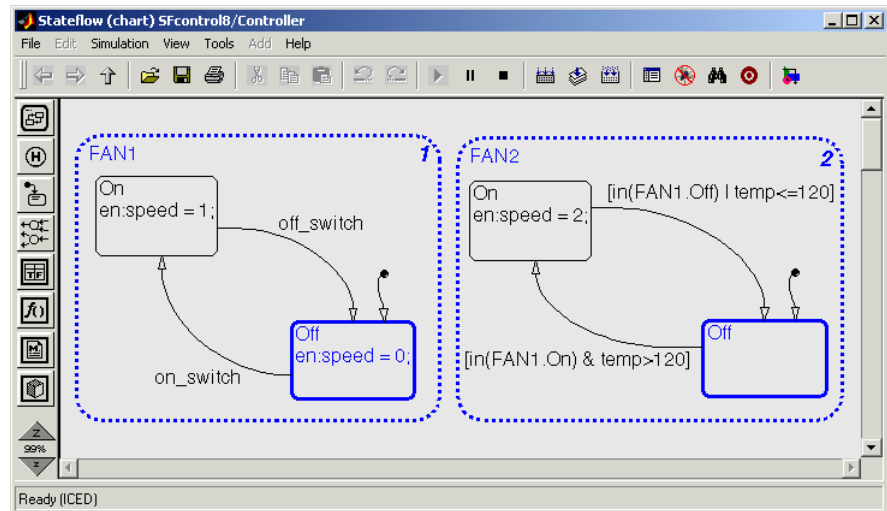


3 Use the Manual Switch block to send an off\_switch event.

FAN1 substate Off becomes active.



FAN2 substate Off becomes active because FAN1 substate Off is active.



**4** Change the input value for temp to 110 and repeat steps 2 and 3.

This time notice that the Off state of FAN2 stays active.



# Stateflow in Simulink

---

Stateflow is the preferred way to create modeling logic in Simulink. This type of logic lends itself easily to use as a control mechanism for a physical plant in Simulink that models a physical process. This chapter shows you how you can use Stateflow to control a physical plant in Simulink.

Controlling a Physical Plant (p. 5-2)

Learn how you can use a Stateflow block to control the simulation of a physical plant.

The Bang-Bang Boiler Demo (p. 5-8)

Use a Stateflow demo model of a Stateflow block controlling a physical plant to further your knowledge of Stateflow.

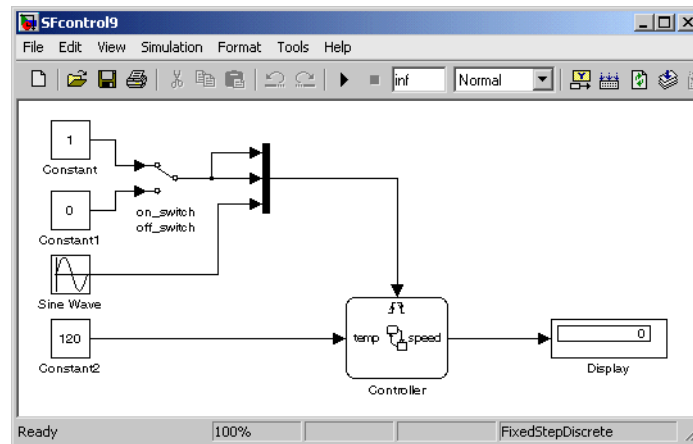
Where to Go from Here (p. 5-13)

Continue your study of Stateflow through other resources at The MathWorks.

## Controlling a Physical Plant

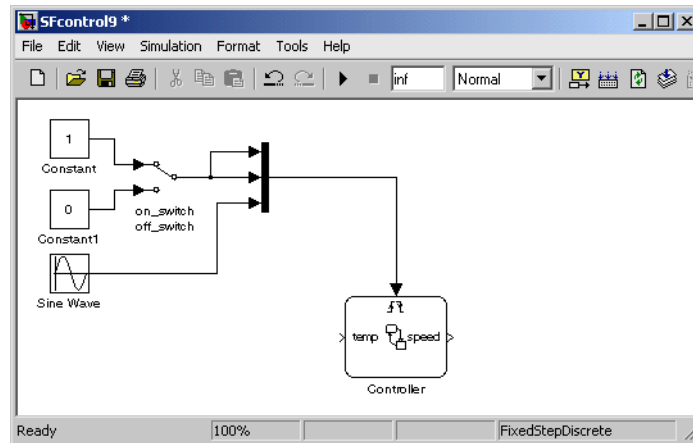
The primary use of Stateflow is to simulate the control of control objects in a physical plant. For example, you can use the Stateflow diagram you developed in “Controlling Objects with Parallel Superstates” on page 4-31, SFcontrol8, to control two fans in an attic space.

- 1 Load the Simulink model SFcontrol8 you save in “Controlling Objects with Parallel Superstates” on page 4-31 and save it as SFcontrol9.

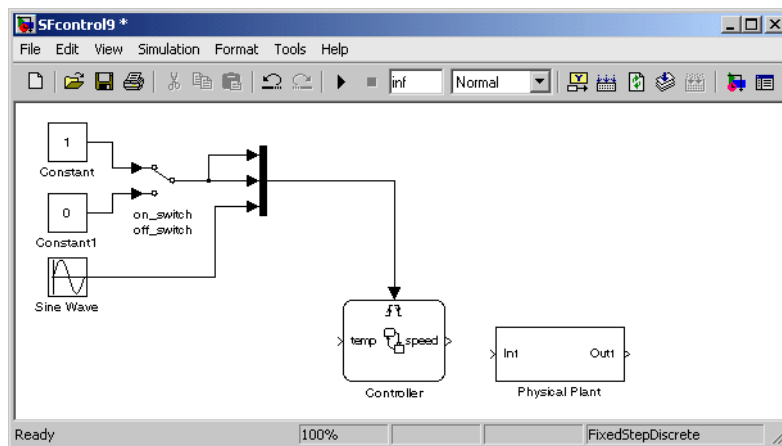




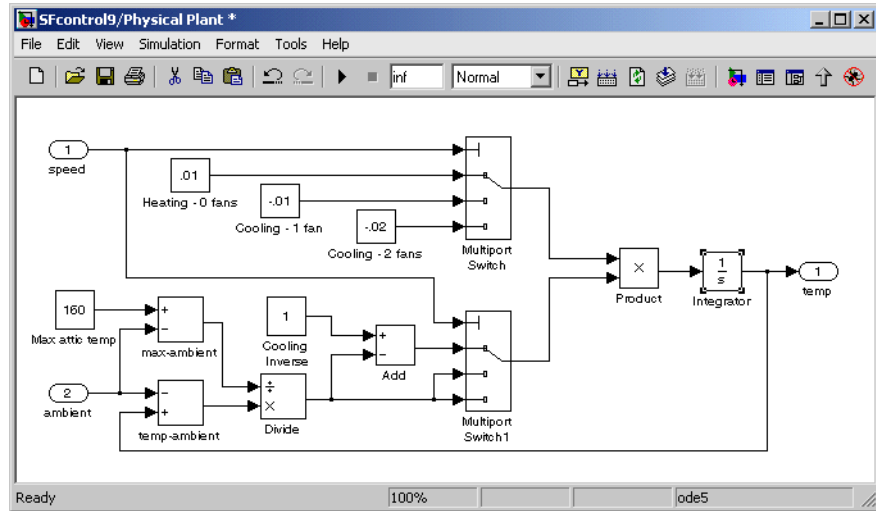
- 2 Delete the Constant and Display blocks and their connections to the Controller Stateflow block.



- 3 Add a subsystem to the right of the Controller block and name it Physical Plant.



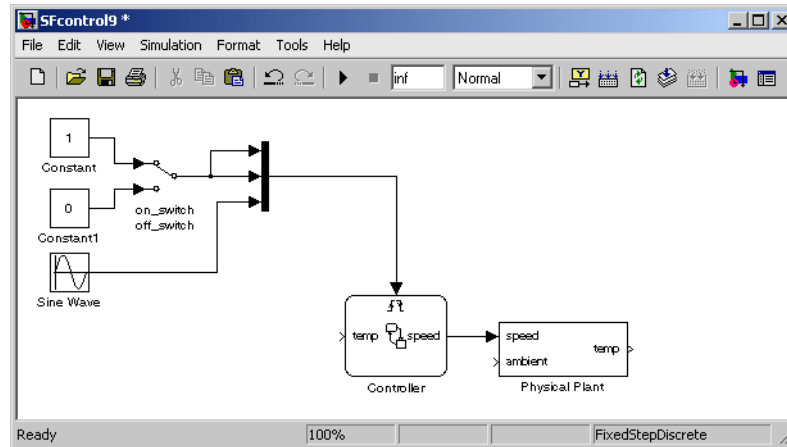
- 4 Double-click the subsystem to open it and program it as follows:



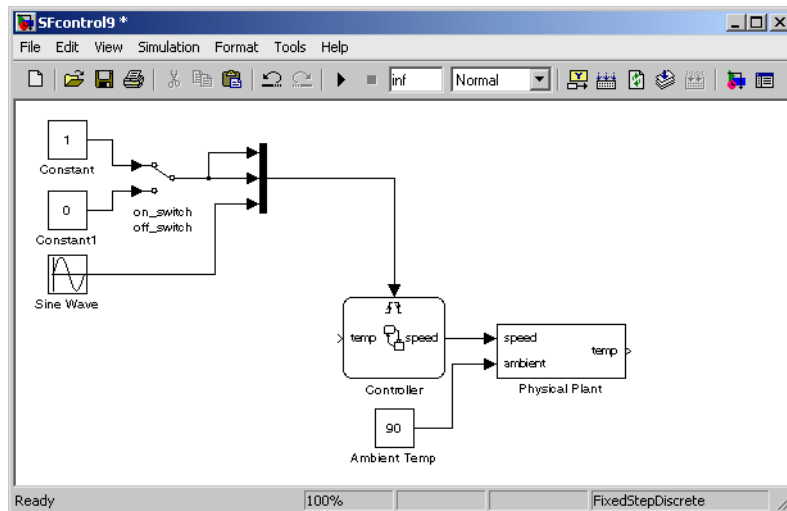
The part of the model that simulates the attic air space is packaged in a single subsystem. Because it is only an example, the model uses a nonanalytical means for computing the air temperature in an attic space. The cooling value is chosen by a multiport switch that takes the value of the data speed output from the Stateflow block as an index. If speed = 2, both fans are running, and the chosen cooling rate is -.015 degrees per sample. If speed = 1, only one fan is running, and the cooling rate is -.01 degrees per sample. If speed = 0, no fans are running, and a cooling rate is .01 degrees per sample (that is, heating) is chosen.

The cooling value is further modified by a factor that depends on how close attic air temperature is to ambient air temperature. Obviously, the closer that ambient air temperature is to the attic temperature, the less cooling takes place. The amount of cooling that is finally calculated is added to a running integral sum that is the attic temperature. This temperature is fed to the output of the physical plant subsystem.

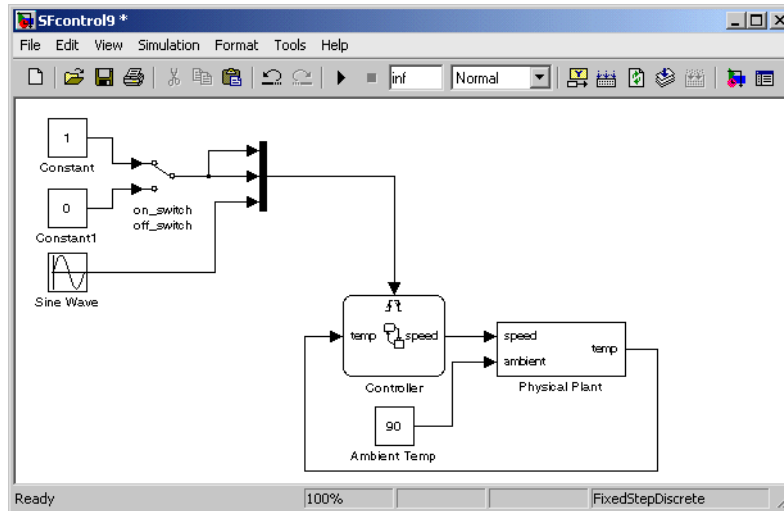
- In the top level Simulink diagram, connect the speed output port of the Controller block to the speed input port of the subsystem.



- Use a Constant block set at a value of 90 to provide input to the ambient input port of the subsystem.

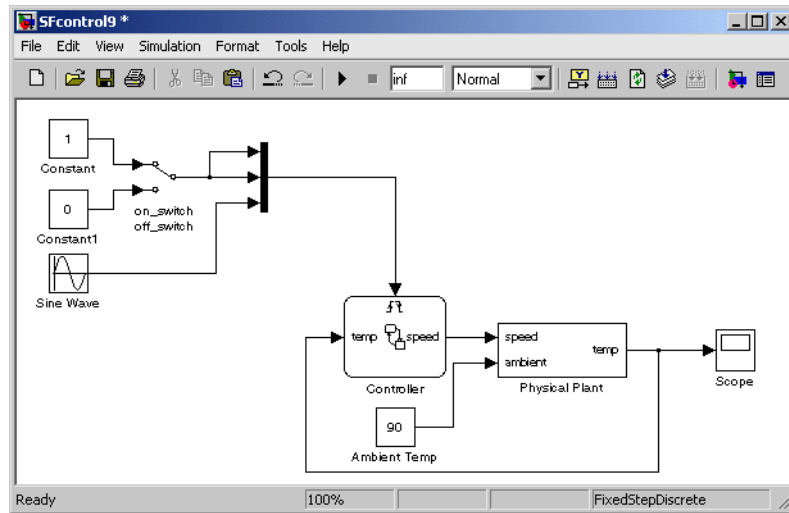


- 7 Connect the temp output port of the subsystem to the temp input port of the Controller Stateflow block.

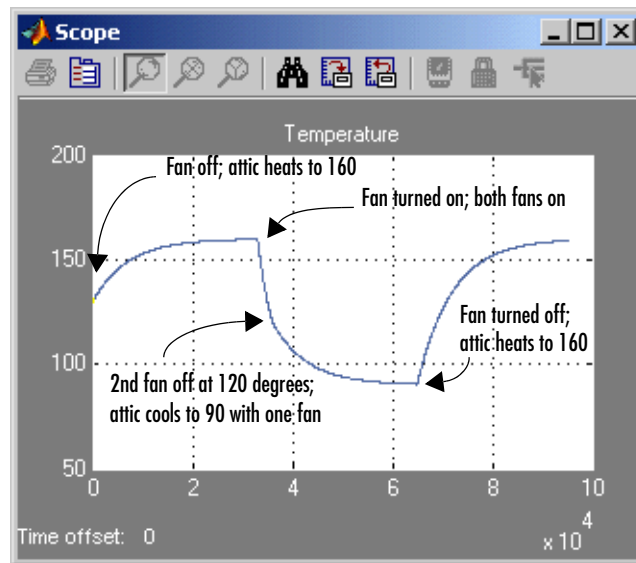


This sends the temperature computed by the physical plant back to the Stateflow block to make a decision on whether the second fan should be running or not.

- 8 Connect the temp output port of the subsystem to a Scope block.



The physical plant model is now complete. Continue by simulating it and observing the temperature in the Scope block display as follows:



# The Bang-Bang Boiler Demo

A good model to use in furthering your study of Stateflow is the Bang-Bang Boiler demo model. This model retains the basic structure of the Stateflow diagrams used in the Getting Started guide, but adds more levels of control with other Stateflow features.

Access the Stateflow demo model Bang-Bang Boiler as follows:

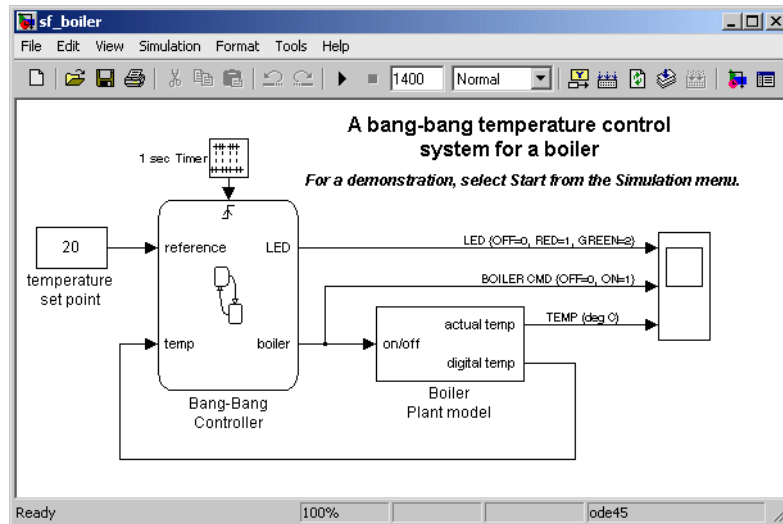
- 1 In the MATLAB window, from the **Help** menu, select **Demos**.

The Help window appears with the Demos topic list displayed in the left pane.

- 2 Expand the **Simulink** node.
- 3 Expand the **Stateflow** node.
- 4 Expand the **Examples** node.

- 5 Double-click the **Bang-Bang Control Using Temporal Logic** node to open the `sf_boiler` model.

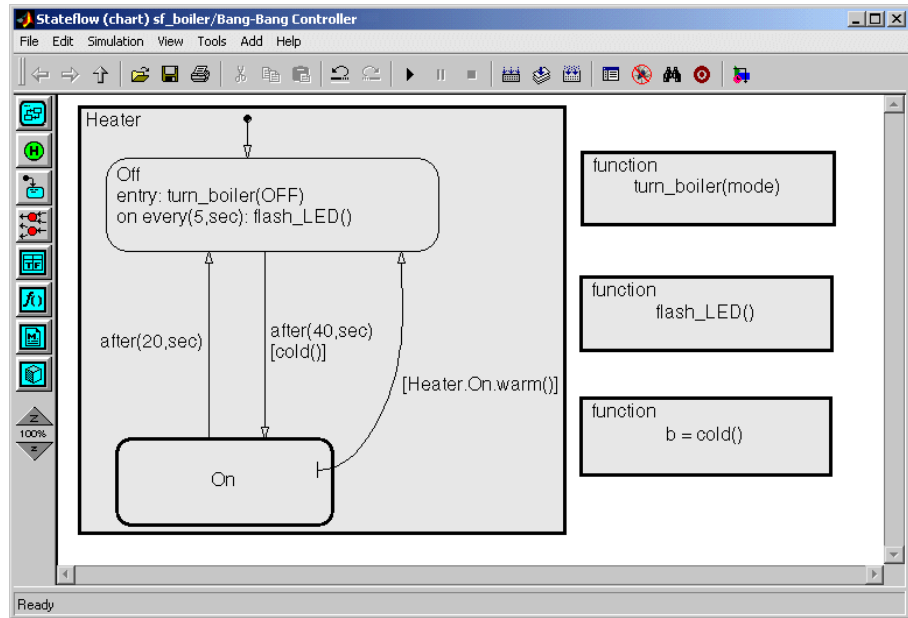
A Simulink model window opens along with a Stateflow diagram and a Scope window. The model has the following appearance:



The structure of this model is similar to the model you constructed in “Controlling a Physical Plant” on page 5-2. Note the following about this model:

- The Stateflow block Bang-Bang Controller receives updates from a Pulse Generator block set to deliver a pulse every second. The Pulse Generator block feeds the rising-edge trigger event `sec` defined for the Stateflow diagram.
- The only other input is from a Constant block, which feeds a value for the setpoint temperature of a boiler to the input data `reference` defined for the Stateflow diagram.
- The Bang-Bang Controller block controls a boiler simulated by the subsystem Boiler Plant Model through a single output data value, `boiler`. For each time sample, the Boiler Plant Model calculates the value of the Simulink signal `digital temp`, which is input to the data `temp` for the Bang-Bang Controller block.

- Double-click to the Bang-Bang Controller block to bring focus to its Stateflow diagram, as shown.



The Stateflow diagram for the Bang-Bang Controller block is very similar to the diagram you built in “Creating Subcharts to Add More Substates” on page 4-20. The diagram has two states at the top level, On and Off. In addition, the state On is a subchart containing other states. Also, the diagram includes the definition of three graphical functions that are called in the Stateflow diagram.



The Bang-Bang Controller Stateflow diagram also introduces you to the following new features:

- The Stateflow diagram is contained by a box named Heater.

Boxes are used to group parts of the model for the sake of design modularity. Boxes add no Stateflow behavior of their own; they just group things. This means that the states On, Off, and any substates of On are designated as part of a heater. Boxes can be grouped (as Heater is) or subcharted as well. They are also part of the containment hierarchy of a diagram, as you will see.

- The transitions between states On and Off contain special event triggers called *temporal events*.

Temporal events count events and become true only when the count reaches the specified level. For example, the temporal event trigger `after(20, sec)` on the transition from On to Off means that if the state On is active, Stateflow waits for 20 sec events before the transition from On to Off is taken.

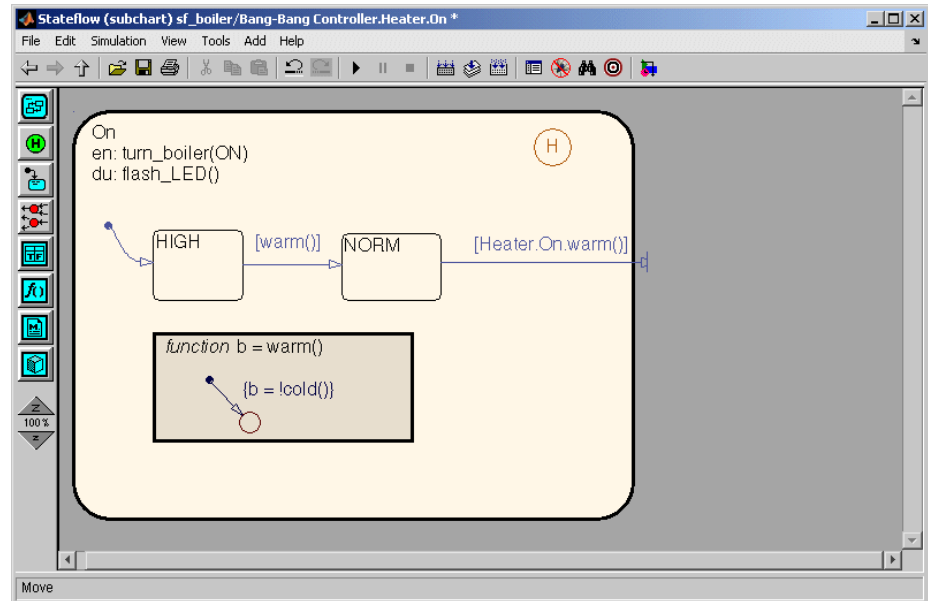
- The state Off contains a special kind of action called an on `<event>` action.

An on `<event>` action executes only if its state is active when it receives the event `<event>`. In this case, if Off becomes active, and stays active for five updates from the trigger event `sec`, the function `flash_LED` is called.

- The transition from the right side of On to Off is a special transition called a *supertransition*.

Supertransitions connect a substate with a peer or superstate of the parent state. This supertransition originates with a substate of On and ends with the state Off.

- 7 Double-click the box Heater to ungroup it and double-click the subchart On to open it, as shown.



- The supertransition to Off originates with the state NORM, a substate of the superstate On. The supertransition also includes a condition on the return value from a call to the function warm, which is defined inside of On. Because a transition is owned by the highest member of the hierarchy that it touches, the supertransition is owned by the chart. That is why the condition is called with [Heater.On.warm()] and not just [warm()].

A supertransition from a substate to a point outside of the parent superstate executes only when the substate and the superstate are active. This means that the supertransition from NORM to Off can execute only when NORM and On are both active.

- The state On has a during action. During actions for a state are executed when the state is active and receives an event such as the trigger event sec. If the event does not cause an outgoing transition from the state to be taken, the during action is executed.
- 8 Be sure to simulate the Bang-Bang Temperature Control Model to watch the preceding features execute.

## Where to Go from Here

In this Getting Started guide, you have learned a great many of the features of Stateflow diagrams, but there is still a lot more to learn. To advance your learning of Stateflow, here are a few places to go:

- **Stateflow Semantics** — This lengthy chapter in your Stateflow documentation is filled with example Stateflow diagrams and detailed point-by-point descriptions of their behavior during execution.
- **Stateflow Demos** — Stateflow has a substantial collection of demonstration models, which you can access as follows:
  - In the MATLAB window, from the **Help** menu, select **Demos**.
  - Expand the Simulink node to display a Stateflow node, among others.
  - Expand the Stateflow node to reveal directories of demo models. Most of these directories are oriented to studying specific Stateflow features. The Examples directories contains some more sophisticated real-life example models.
- **Stateflow Training** — The MathWorks offers regular courses in Stateflow along with other courses in MATLAB and Simulink. To see a list of available Stateflow training courses, visit the MathWorks web site [www.mathworks.com/services/training](http://www.mathworks.com/services/training).



## A

- actions
  - entry for state 2-51
  - modifying output data 2-51
- adding connective junctions 3-13
- adding superstates 4-3
- animation of Stateflow diagrams during simulation 1-12

## C

- C code
  - generating from Stateflow 1-14
- calling a graphical function example 3-24
- compiler
  - for target 1-21
- conditions
  - adding to transitions 3-16
- connective junctions
  - adding 3-13
- converting a state to a subchart 4-20
- copying Stateflow objects 3-8

## D

- data
  - adding output data to Stateflow block 2-53
  - adding sensor data 3-6
  - output data 2-51
  - sending output data to Simulink 2-55
  - using in Stateflow 1-10
- Debugger messages 2-33
- debugging window during simulation 2-32
- decomposition
  - parallel 4-32
- deleting events in Model Explorer 2-44
- deleting Stateflow objects 3-8

- deterministic
  - definition 3-36
- diagram
  - opening Stateflow diagram 2-6
- drawing
  - states 2-7
  - transitions 2-10

## E

- entry actions for state 2-51
- errors
  - reporting Stateflow diagram errors 2-26
- events
  - adding sensor event 3-2
  - adding trigger event to diagram 2-14
  - guarding transitions 2-37
  - implicit event example 4-36
  - multiple trigger events for chart 2-43
  - rising edge trigger event example 3-5
  - sending multiple trigger events 2-47
  - sending trigger event to Stateflow block 2-16
  - Stateflow reaction to 1-8
  - trigger events during simulation 2-56
- examples of Stateflow applications 1-18
- exclusive states 4-31

## F

- flow diagrams 3-37
  - if example 3-38
  - if-else example 3-42
  - in graphical functions 3-23
  - while example 3-42

**functions**

- adding graphical function example 3-19
- calling a graphical function example 3-24

**G**

- generated C code 1-14
- generating code
  - for custom targets 1-16
  - for Real-Time Workshop targets 1-14
  - for simulation targets 1-14
  - Stateflow Coder 1-20
- graphical functions
  - adding 3-19
  - calling example 3-24

**H**

- history junctions
  - simulation example 4-17

**I**

- iced model during simulation 2-26
- if flow diagram example 3-38
- if-else flow diagram example 3-42
- implicit event example 4-36
- installation
  - optional software 1-20
  - prerequisite software 1-20
  - required software 1-19
- introduction to Stateflow 1-1

**J**

- junction behavior example 3-31
- junctions
  - choosing destinations 1-9
  - in flow diagrams 3-37

**K**

- knowledge level required for readers 1-19

**L**

- labeling transitions 2-38
- laptop computer with Stateflow 1-22

**M**

- messages in Debugger 2-33
- model
  - creating in Simulink 2-3
  - iced during simulation 2-26
  - saving 2-5
- Model Explorer
  - deleting events 2-44
- modes of control 1-6
- moving transition source point example 3-33

**N**

- namespaces in parallel states 4-36

**O**

- optional software
  - Real-Time Workshop 1-20
  - Simulink Report Generator 1-20
  - Stateflow Coder 1-20
- ordering transitions example 3-31

- output data
  - adding to Stateflow block 2-53
  - modifying with actions 2-51
  - sending to Simulink 2-55
  
- P**
- parallel decomposition 4-32
- parallel states
  - priority numbers 4-35
  - simulation example 4-37
- parallel states example 4-31
- parsing Stateflow diagram for errors 2-26
- physical plant control example 5-2
- priority numbers in parallel states 4-35
  
- R**
- Real-Time Workshop 1-20
- Real-Time Workshop targets 1-15
- renaming Stateflow objects 3-8
- rising edge trigger example 3-5
  
- S**
- saving Simulink model 2-5
- semantics
  - definition 3-36
- simulation
  - debugging window 2-32
  - event triggers 2-56
  - setting up for Stateflow diagrams 2-22
  - starting simulation of a Stateflow diagram 2-25
  - Stateflow animation 1-12
  - superstate example 4-9
  - simulation of a subchart example 4-28
  - simulation of parallel states example 4-37
- Simulink
  - and Stateflow 1-3
  - creating a model 2-3
  - sending output data to 2-55
- Simulink Report Generator 1-20
- starting simulation of a Stateflow diagram 2-25
- state entry actions 2-51
- Stateflow
  - about 1-1
  - and Simulink 1-3
  - block in model 2-3
  - choosing destinations with junctions 1-9
  - controlling a physical plant example 5-2
  - defined 1-2
  - examples of applications 1-18
  - generated C code 1-14
  - installation prerequisites 1-20
  - on laptop computer 1-22
  - opening diagram 2-6
  - optional software 1-19, 1-20
  - parsing diagram for errors 2-26
  - reacting to events 1-8
  - using data 1-10
  - ways to use it 1-17
  - Web site 1-20
- Stateflow Coder for generating code 1-20
- states
  - as control modes 1-6
  - drawing 2-7
  - exclusive decomposition 4-31
  - parallel decomposition 4-32
- subcharts
  - converting state to 4-20
  - example 4-20
  - simulation example 4-28
- superstate simulation example 4-9

superstates

adding 4-3

parallel states example 4-31

**T**

target compiler

setting up 1-21

targets

generating code for custom target 1-16

generating code for Real-Time Workshop

targets 1-14

generating code for simulation target 1-14

Real-Time Workshop 1-15

transitions

adding conditions 3-16

adding trigger events 2-37

changing active states 1-7

drawing 2-10

guarding with events 2-37

labeling 2-38

moving source point example 3-33

ordering example 3-31

trigger events

adding to Stateflow block 2-14

during simulation 2-56

for transitions 2-37

multiple for chart 2-43

rising edge example 3-5

sending multiple trigger 2-47

sending to Stateflow block 2-16

**W**

ways to use Stateflow 1-17

Web site for Stateflow 1-20

while flow diagram example 3-42