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
`www.mathworks.com/contact_TS.html`	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.
3 Apple Hill Drive
Natick, MA 01760-2098

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

Getting Started with Stateflow

Revision History

June 2004	First printing	New for Version 6.0 (Release 14)
October 2004	Online only	Revised for Version 6.1 (Release 14SP1)
March 2005	Online only	Revised for Version 6.2 (Release 14SP2)
September 2005	Online only	Revised for Version 6.3 (Release 14SP3)
October 2005	Reprint	Version 6.0
March 2006	Second printing	Revised for Version 6.4 (Release R2006a)

Contents

1 Introduction to Stateflow

2 The Stateflow Chart You Will Build

3 Defining the Interface to Simulink

4 Defining the States for Modeling Each Mode of Operation

5 Defining State Actions and Variables

6 Defining Transitions Between States

7 Triggering a Stateflow Chart

8 Simulating the Chart

9 Debugging the Chart

Index

1

Introduction to Stateflow

This chapter describes Stateflow® and its components.

What Is Stateflow? (p. 1-2)	Presents an overview of Stateflow features
What Does a Stateflow Chart Look Like? (p. 1-4)	Presents an example of a Stateflow chart
How Stateflow Works with Simulink (p. 1-6)	Describes how Stateflow works with Simulink®
Basic Workflow for Building a Stateflow Chart (p. 1-8)	Explains each phase of a basic workflow for building a Stateflow chart
Installing Stateflow (p. 1-9)	Provides information about installing Stateflow and prerequisite products.
Related Products (p. 1-11)	Provides information about products that extend the capabilities of Stateflow

What Is Stateflow?

Stateflow is an interactive graphical design tool that works with Simulink to model and simulate event-driven systems, also called reactive systems. *Event-driven systems* transition from one operating mode to another in response to events and conditions. These systems are often used to model logic for dynamically controlling a physical device such as a fan, motor, or pump. Event-driven systems can be modeled as finite-state machines.

Finite-state machines represent operating modes as states. For example, a house fan can have states such as High, Medium, Low, and Off. To construct finite-state machines, Stateflow provides graphical objects that you can drag and drop from a design palette to create state-transition charts in which a series of transitions directs a flow of logic from one state to another. Stateflow also allows you to add

- Input and output data.
- Events for triggering Stateflow charts
- Actions and conditions, which you can attach to states and transitions to further define the behavior of the Stateflow chart.

You will learn more about these elements later in this guide.

Extends the Capabilities of Traditional State Charts

Stateflow allows you to extend the capabilities of traditional state charts by

- Adding hierarchy to charts
- Modeling parallel states
- Defining functions graphically, using flow diagrams; procedurally, using the MATLAB® language; and in tabular form, with truth tables
- Using temporal logic to schedule events
- Defining vector, matrix, and fixed-point data types

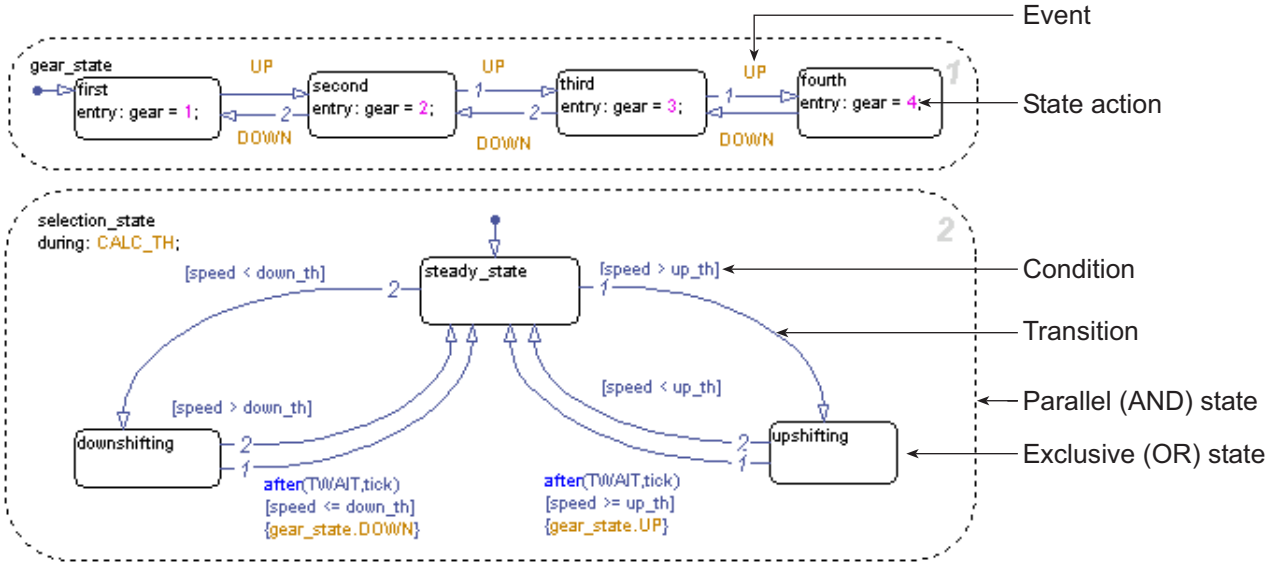
Generates C Code

Stateflow performs simulation by generating a C code implementation of the Stateflow chart. The simulation code is generated from a simulation target. You will learn more about simulation targets later in Chapter 8, “Simulating the Chart”.

You can also generate portable C code from Stateflow charts automatically using Stateflow® Coder (available separately). Stateflow Coder also works with Real-Time Workshop® (available separately) to generate C code for Simulink models that include Stateflow charts.

What Does a Stateflow Chart Look Like?

Here is an example of a Stateflow chart, which models as a finite-state machine the logic required to shift gears in an automatic transmission system of a car:



Notice the following details in this Stateflow chart:

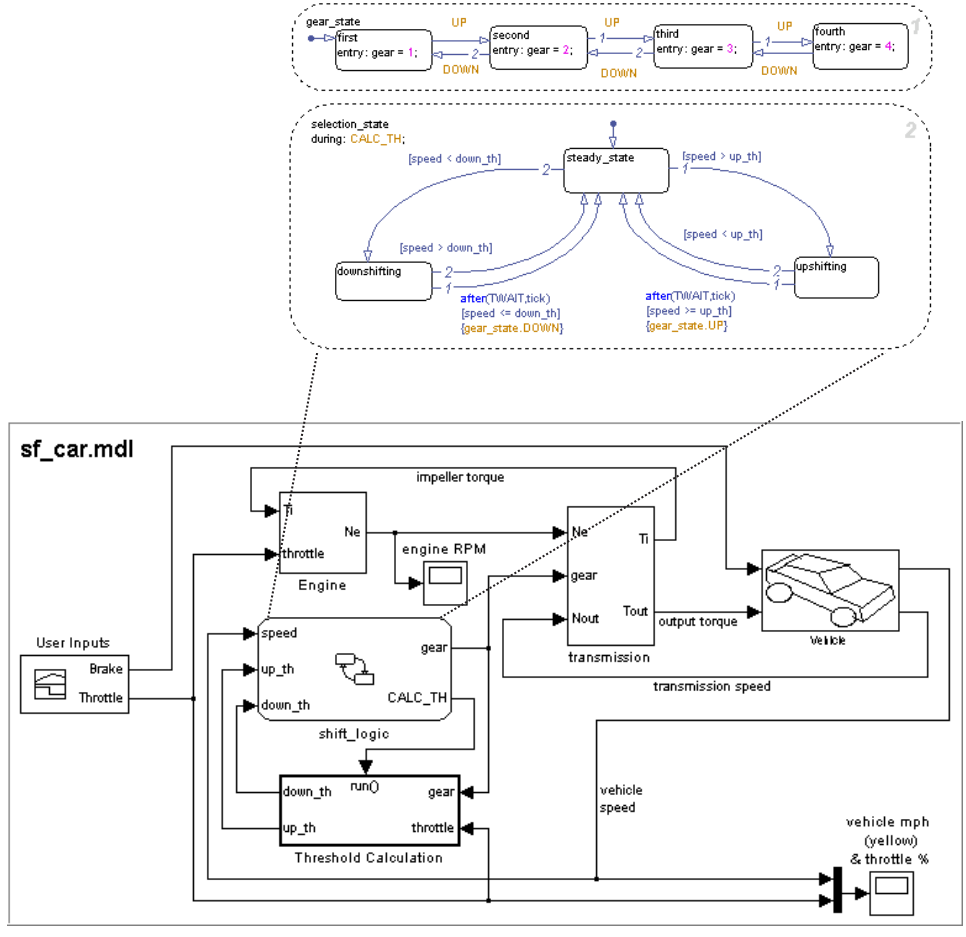
- Each gear and shift position is represented by a state.
- Some states are exclusive (only one can be active at a time) while others are parallel (can be active concurrently).
- Transitions can be triggered by events and conditions.
- States can execute actions while they are active.

You will learn more about these features later in this guide as you build your own Stateflow chart.

This chart is part of a model called `sf_car` that ships with Stateflow. To explore the model further, run it from your MATLAB Command Window, as described in “Running a Demo Model” in the online documentation Getting Started with Simulink.

How Stateflow Works with Simulink

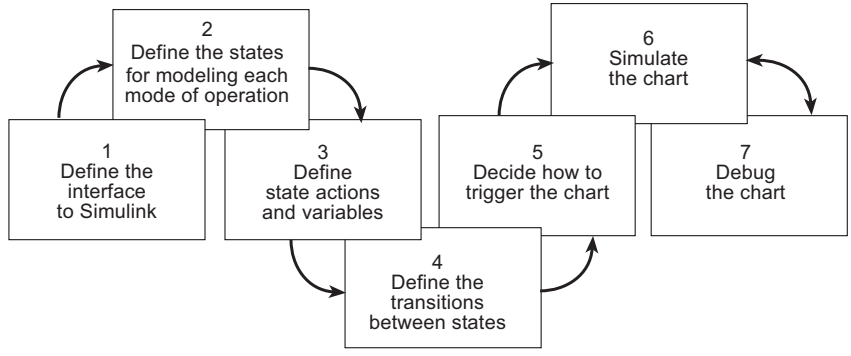
Stateflow charts run as blocks in a Simulink model. The Stateflow block connects to other blocks in the model by input and output signals. Through these connections, Stateflow and Simulink share data and respond to events that are broadcast between model and chart. For example, the Stateflow shift_logic block is integrated with the Simulink `sf_car` model as shown:



You can develop your Stateflow chart before or after the Simulink model in which it will run. Stateflow comes with its own editor and debugger, which allows you to simulate and test the chart logic before you integrate it with a Simulink model. You can test a Stateflow chart independently of its parent model by attaching a Simulink Source block as an input and a Simulink Sink block as an output (see “Sources” and “Sinks” in the online Stateflow Reference documentation). During simulation, you can animate the chart to get visual feedback about its run-time behavior. You will edit, simulate, and debug your chart later in this guide.

Basic Workflow for Building a Stateflow Chart

Here is a basic workflow for building Stateflow charts that accurately model event-driven systems:



To get started quickly, you will participate in hands-on exercises that guide you through each phase of this workflow for building a realistic Stateflow chart and integrating it with a Simulink model. Note that work flows in one direction until you get to phase 6. At this phase, you may need to iterate between simulating and debugging your Stateflow chart as you fix errors and experiment with different parameters until the model produces the desired behavior.

Installing Stateflow

Stateflow runs on Windows and UNIX operating systems. Your platform-specific MATLAB installation documentation provides all the information you need to install Stateflow. Before installing Stateflow, make sure you address the following configuration requirements:

Obtaining a Stateflow License

To install Stateflow, you must obtain a License File or Personal License Password from The MathWorks. These documents identify the products you are permitted to install and use. For information about the license manager, contact The MathWorks Technical Support by using this form: `www.mathworks.com/contact_TS.html`.

Prerequisite Software

Before installing Stateflow, you need the following software:

- MATLAB
- Simulink
- Stateflow Coder
- C or C++ compiler supported by MATLAB

  The compiler is required for compiling code generated by Stateflow for simulation. 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 platforms other than Windows or to install a different compiler, see "Setting Up Your Own Target Compiler" on page 1-10.

Product Dependencies

For information about product dependencies and requirements, see `www.mathworks.com/products/stateflow/requirements.html`.

Setting Up Your Own Target Compiler

If you use the UNIX version of Stateflow or do not wish to use the `lcc` compiler, you must install your own target compiler. You can use any compiler supported by MATLAB, as described in "Building MEX-Files" in the online MATLAB External Interfaces documentation.

To install a compiler for Stateflow, follow these steps:

1. At the MATLAB prompt, type

   ```
   mex -setup
   ```

2. Follow the prompts for entering information about the compiler.

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 may 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 box.
3. Select the **Settings** panel on the **Display Properties** dialog box.
4. Choose a setting that is more than 256 colors and click **OK**.

Related Products

The MathWorks provides several products that extend the capabilities of Stateflow. For information about these related products, see `www.mathworks.com/products/stateflow/related.htm`.

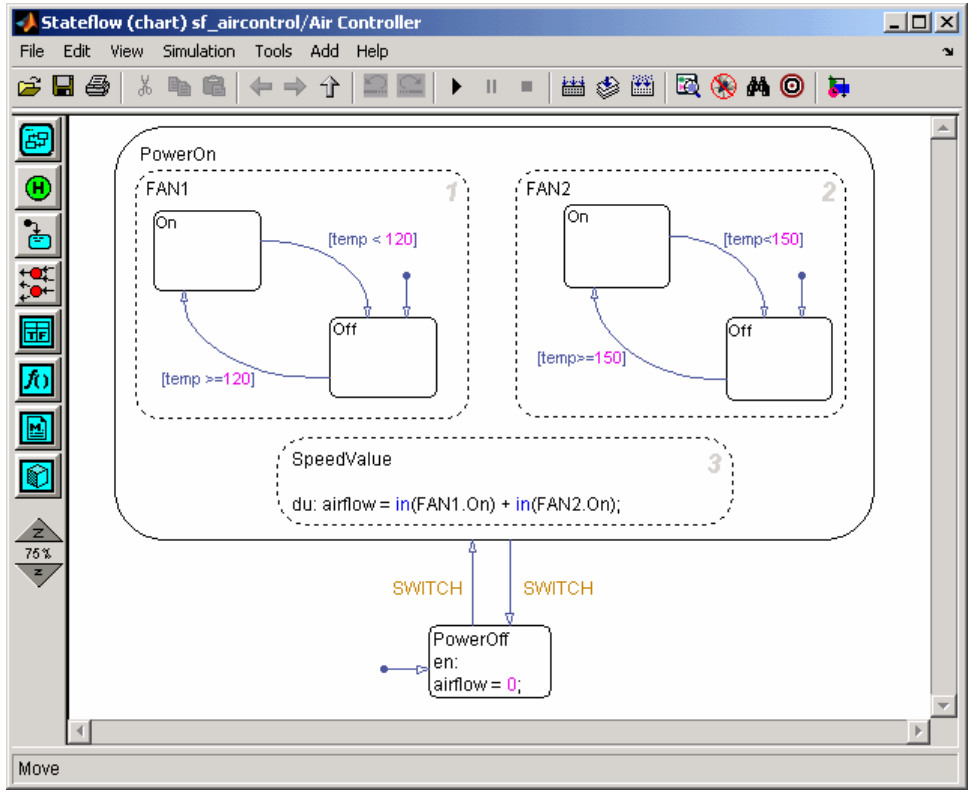
2

The Stateflow Chart You Will Build

To get hands-on experience using Stateflow, you will build a Stateflow chart in incremental steps that follow the basic workflow described in "Basic Workflow for Building a Stateflow Chart" on page 1-8. To give you a context for your development efforts, this chapter describes the purpose and function of the chart you will build and explains how it interfaces with a Simulink model. You will also learn how to run a completed version of the model from the MATLAB command line.

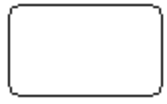
The Stateflow Chart

You will build a Stateflow chart that maintains air temperature at 120 degrees in a physical plant. The Stateflow controller operates two fans. The first fan turns on if the air temperature rises above 120 degrees and the second fan provides additional cooling if the air temperature rises above 150 degrees. When completed, your Stateflow chart should look something like this:



As you can see from the title bar, the Stateflow chart is called Air Controller and is part of a Simulink model called `sf_aircontrol`. When you build this chart, you will learn how to work with the following elements of state-transition charts:

Exclusive (OR) state. State that represents mutually exclusive modes of operation. No two exclusive (OR) states can ever be active or execute at the same time. Exclusive (OR) states are represented graphically by a solid rectangle:



The Air Controller chart contains six exclusive (OR) states:

- `PowerOn`
- `PowerOff`
- `FAN1.On`
- `FAN1.Off`
- `FAN2.On`
- `FAN2.Off`

Parallel (AND) state. State that represents independent modes of operation. Two or more parallel (AND) states at the same hierarchical level can be active concurrently, although they execute in a serial fashion. Parallel (AND) states are represented graphically by a dashed rectangle with a number indicating execution order:

The Air Controller chart contains three parallel (AND) states:

- `FAN1`

- FAN2
- SpeedValue

Transition. Graphical object that links one state to another and specifies a direction of flow. Transitions are represented by unidirectional arrows:

The Air Controller chart contains six transitions, from

- PowerOn to PowerOff
- PowerOff to PowerOn
- FAN1.On to FAN1.Off
- FAN1.Off to FAN1.On
- FAN2.On to FAN2.Off
- FAN2.Off to FAN2.On

Default transition. Graphical object that specifies which exclusive (OR) state is to be active when there is ambiguity between two or more exclusive (OR) states at the same level in the hierarchy. Default transitions are represented by arrows with a closed tail:

The Air Controller chart contains default transitions:

- At the chart level, the default transition indicates that the state PowerOff is activated (wakes up) first when the chart is activated.
- In the FAN1 and FAN2 states, the default transitions specify that the fans be powered off when the states are activated.

State action. Action executed based on the status of a state.

The Air Controller chart contains two types of state actions:

- `entry` (`en`) action in the `PowerOff` state. Entry actions are executed when the state is entered (becomes active).
- `during` (`du`) action in the `SpeedValue` state. During actions are executed for a state while it is active and no valid transition to another state is available.

Note There are other types of state actions besides `entry` and `during`, but they involve concepts that go beyond the scope of this guide. For more information, see "Using Actions in Stateflow" in the online Stateflow User's Guide documentation.

Condition. Boolean expression that allows a transition to occur when the expression is true. Conditions appear as labels for the transition, enclosed in square brackets (`[ ]`).

The Air Controller chart provides conditions on the transitions between `FAN1.On` and `FAN1.Off`, and between `FAN2.On` and `FAN2.Off`, based on the air temperature of the physical plant at each time step.

Event. Object that can trigger a variety of activities, including

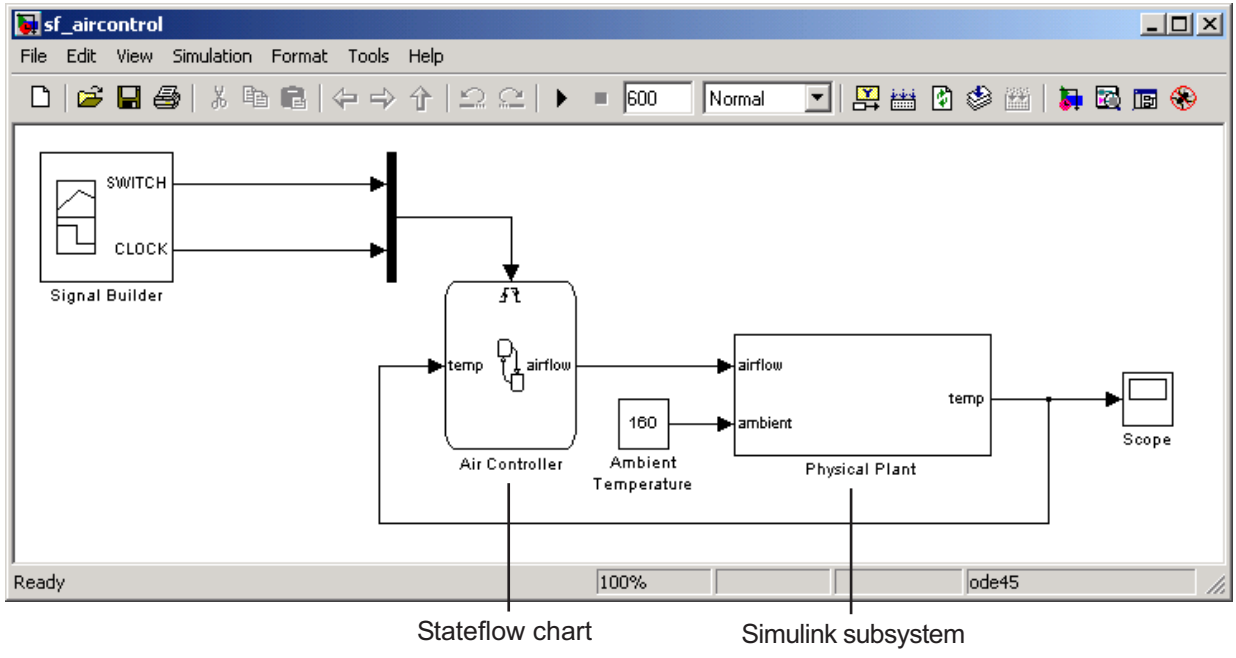
- Waking up a Stateflow chart
- Causing transitions to occur from one state to another (optionally in conjunction with a condition)
- Executing actions

The Air Controller chart contains two edge-triggered events:

- `CLOCK` wakes up the Stateflow chart at each rising or falling edge of a square wave signal.
- `SWITCH` allows transitions to occur between `PowerOff` and `PowerOn` at each rising or falling edge of a pulse signal.

How the Stateflow Chart Works with the Simulink Model

The Stateflow chart you will build appears as a block named Air Controller that is connected to the model of a physical plant in the Simulink `sf_aircontrol` model. Here is the top-level view of the model:

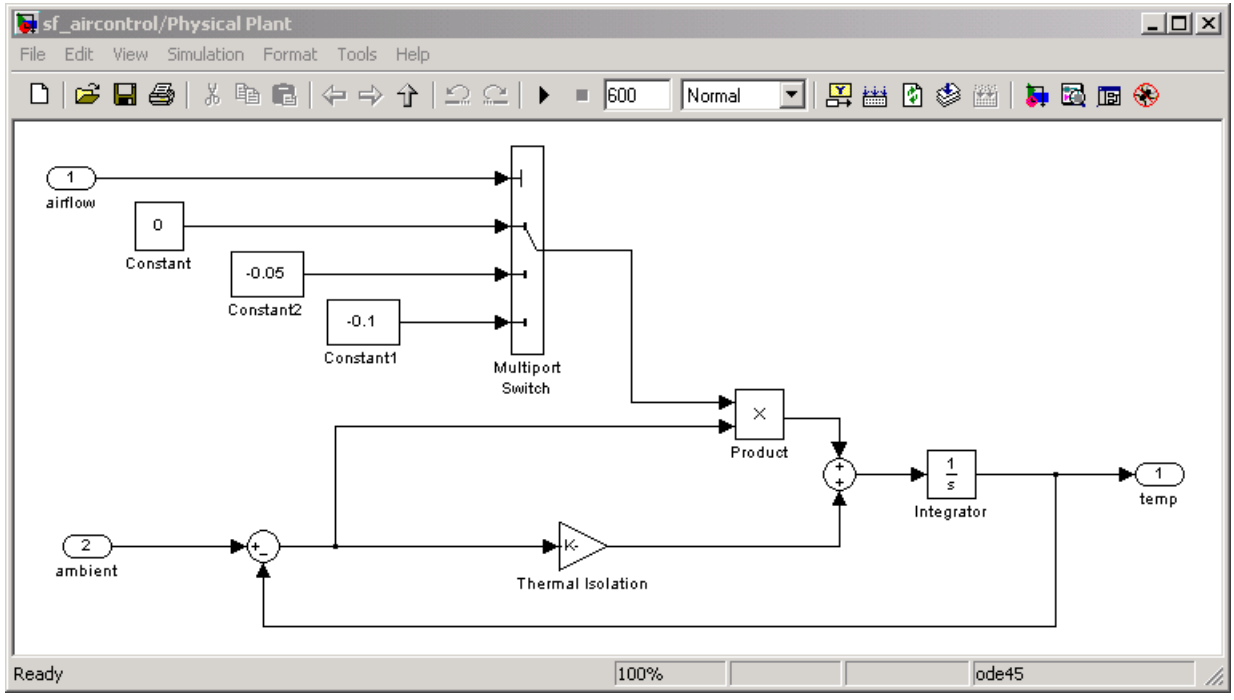


As you can see, Simulink passes the temperature of the plant as an input `temp` to the Stateflow Air Controller block. Based on the temperature of the plant, the controller activates zero, one, or two fans, and passes back to Simulink an output value `airflow` that indicates how fast the air is flowing. The speed of the airflow depends on the amount of cooling activity generated by the fans. As cooling activity increases, air flows faster. Simulink uses the value of `airflow` to simulate the effect of cooling when it computes the air temperature in the plant over time. You will learn more about these design elements in Chapter 3, “Defining the Interface to Simulink”.

The Signal Builder block in the Simulink model sends a square wave signal (`CLOCK`) to wake up the Stateflow chart at regular intervals and a pulse signal (`SWITCH`) to cycle the power on and off for the control system modeled by the Stateflow chart. You will learn more about these design elements in Chapter 7, “Triggering a Stateflow Chart”.

A Look at the Physical Plant

Simulink models the plant using a subsystem called Physical Plant, which contains its own group of Simulink blocks. The subsystem provides a graphical hierarchy for the blocks that define the behavior of the Simulink model. The inputs, airflow speed and ambient temperature, are used to model the effects of the controller activity on plant temperature. Here is a look inside the Physical Plant subsystem:



In this model, the internal temperature of the plant attempts to rise to achieve steady state with the ambient air temperature, set at a constant 160 degrees (as shown in “How the Stateflow Chart Works with the Simulink Model” on page 2-6). The rate at which the internal temperature rises depends in part on the degree of thermal isolation in the plant and the amount of cooling activity.

Thermal isolation measures how much heat flows into a closed structure, based on whether the structure is constructed of materials with insulation or

conduction properties. Here, thermal isolation is represented by a Gain block, labeled `Thermal Isolation`. The Gain block provides a constant multiplier that is used in calculating the temperature in the plant over time.

Cooling activity is modeled using a constant multiplier, derived from the value of `airflow`, an output from the Stateflow chart. Stateflow assigns `airflow` one of three cooling factors, each a value that serves as an index into a multiport switch. Using this index, the multiport switch selects a cooling activity multiplier that is directly proportional to the cooling factor, as follows:

Cooling Factor (Value of Airflow)	What It Means	Cooling Activity
0	No fans are running. The value of `temp` is not lowered.	0
1	One fan is running. The value of `temp` is lowered by the cooling activity multiplier.	-0.05
2	Two fans are running. The value of `temp` is lowered by the cooling activity multiplier.	-0.1

Over time, the subsystem calculates the cooling effect inside the plant, taking into account thermal isolation and cooling activity. The cooling effect is the time-derivative of the temperature and is the input to the Integrator block in the Physical Plant subsystem. Let the variable `temp_change` represent the time derivative of temperature. Note that `temp_change` can be a warming or cooling effect, depending on whether it is positive or negative, based on this equation:

$$\textbf{temp\_change} = ((\textbf{ambient} - \textbf{temp}) * (\textit{thermal isolation multiplier})) + ((\textbf{ambient} - \textbf{temp}) * (\textit{cooling factor}))$$

The Integrator block computes its output `temp` from the input `temp_change`, as follows:

$$temp(t) = \int_{t_0}^{t} temp\_change(t)\,dt + 70$$

Note In this model, the initial condition of the Integrator block is 70 degrees.

`temp` is passed back to the Stateflow Air Controller to determine how much cooling is required to maintain the ideal plant temperature.

Running the Model

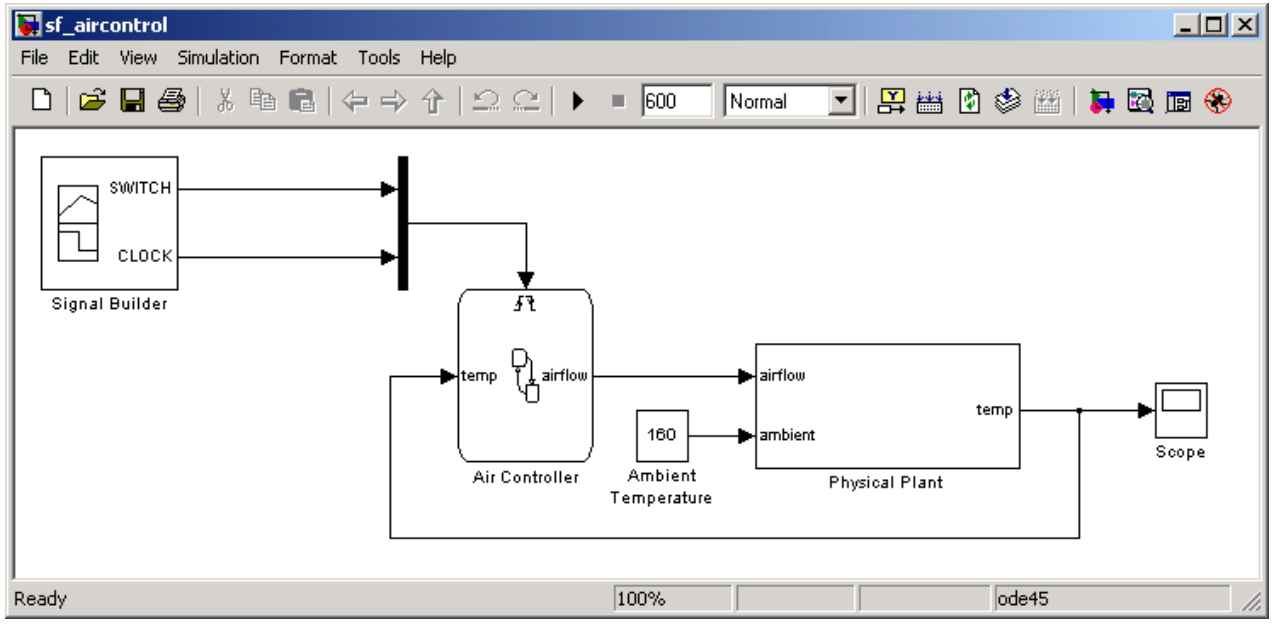
To see how the `sf_aircontrol` model works, you can run a completed, tested version, which includes the Stateflow chart you will build. Here's how to do it:

1. Start MATLAB.

   If you need instructions, consult your MATLAB documentation.

2. Type `sf_aircontrol` at the command line.

   This command starts Simulink and opens the `sf_aircontrol` model:



3. Double-click the Air Controller block to open the Stateflow chart.

4. Double-click the Scope block to display the changes in temperature over time as the model runs.

Tip Position the Air Controller chart and the Scope window so they are both visible on your desktop.

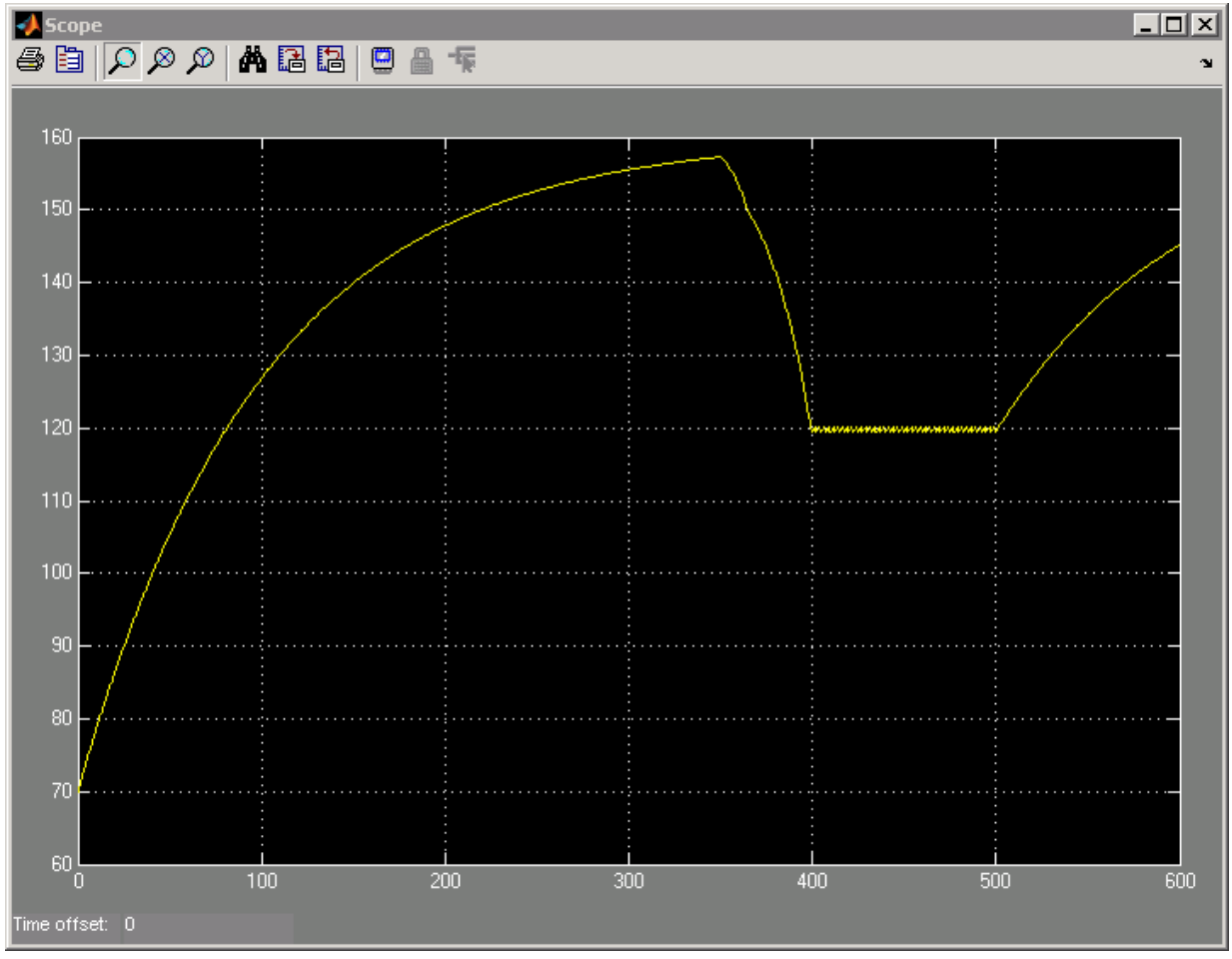
5 Start simulation in the Air Controller chart by selecting **Start** from the **Simulation** menu or clicking the Start Simulation icon:

As the simulation runs, the chart becomes active (wakes up) in the `PowerOff` state. Notice in the Scope that until `PowerOn` becomes active, the temperature rises unchecked. After approximately 350 seconds into the simulation, a rising edge signal switches power on and the fans become active.

Note Simulation time can be faster than elapsed time.

When the temperature rises above 120 degrees, `FAN1` cycles on. When the temperature exceeds 150 degrees, `FAN2` cycles on to provide additional cooling. Ultimately, `FAN1` succeeds in maintaining the temperature at 120 degrees until a falling edge signal switches power off again at 500 seconds. Then, the temperature begins to rise again.

The Scope captures the temperature fluctuations:



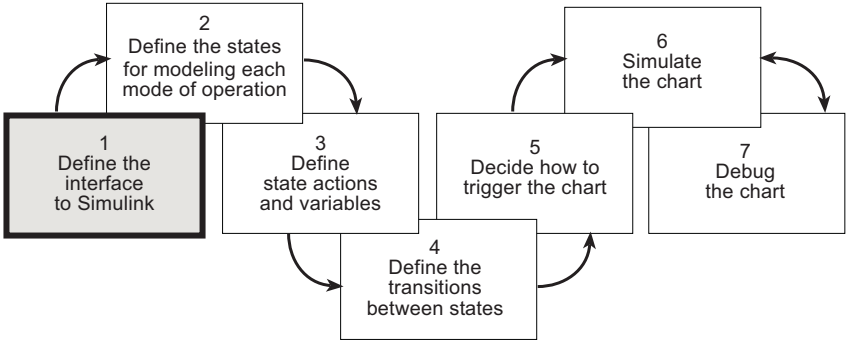
Tip You can stop or pause simulation at any time. To stop simulation, select **Stop** from the **Simulation** menu or click the Stop Simulation icon:

To pause simulation, select **Pause** from the **Simulation** menu or click the Pause Simulation icon:

6 Close the model.

Where to go next. Now you are ready to start building the Stateflow Air Controller chart. Begin at phase 1 of the workflow: Chapter 3, “Defining the Interface to Simulink”.

Defining the Interface to Simulink



You have entered phase 1 of a basic workflow for building a Stateflow chart: *define the interface to Simulink*. This chapter presents the design questions that you must answer and guides you through exercises for implementing the interface, based on your design decisions.

Design Considerations for Defining the Interface (p. 3-2)	Poses the design questions for phase 1 of the workflow and presents the rationale for the solutions
Implementing the Interface (p. 3-4)	Provides hands-on exercises that show you how to implement the interface as designed

Design Considerations for Defining the Interface

You should consider the following design questions when defining the interface between your Stateflow chart and a Simulink model:

- What inputs does the Stateflow chart require from Simulink?
- What outputs does Simulink require from the Stateflow chart?

Inputs Required from Simulink

Type of Input. Temperature of the physical plant

Rationale. The Stateflow chart is designed to control the air temperature in a physical plant. The goal is to maintain an ideal temperature of 120 degrees by activating one or two cooling fans if necessary. Stateflow must check the plant temperature over time to determine the amount of cooling required.

Properties of Input. The properties of the temperature input are as follows:

Property	Value
Name	`temp`
Scope	Input from Simulink
Size	Inherit from Simulink input signal to ensure compatibility
Data type	Inherit from Simulink input signal to ensure compatibility
Port	1
Watch in debugger	Enable

Outputs Required from Stateflow

Type of Output. Speed of airflow, based on how many fans are operating.

Rationale. When the Simulink subsystem determines the temperature of the physical plant over time, it needs to account for the speed of the airflow. Airflow speed is directly related to the amount of cooling activity generated by the fans. As more fans are activated, cooling activity increases and air flows faster. To convey this information, the Stateflow chart outputs a value that indicates whether 0, 1, or 2 fans are running. The Simulink subsystem uses this value as an index into a multiport switch, which outputs a cooling activity value, as described in "A Look at the Physical Plant" on page 2-8.

Properties of Output. The properties of the airflow output are as follows:

Property	Value
Name	`airflow`
Scope	Output from Simulink
Data type	8-bit unsigned integer (The values can be only 0, 1, or 2.)
Port	1
Watch in debugger	Enable

Implementing the Interface

To implement the interface as designed, you need to perform the following tasks:

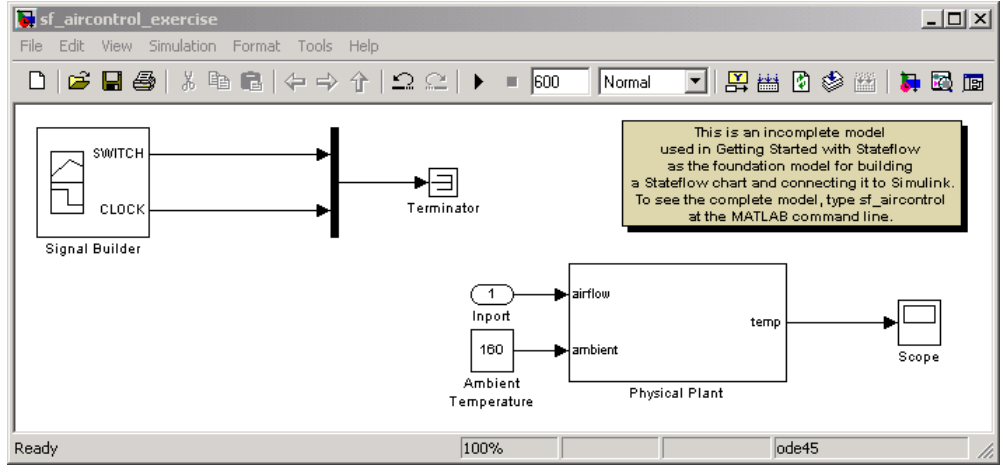
Adding a Stateflow Block to a Simulink Model

To begin building your Stateflow chart, you will add a Stateflow block to a partially built Simulink model called `sf_aircontrol_exercise`, which contains the Physical Plant subsystem, described in "A Look at the Physical Plant" on page 2-8.

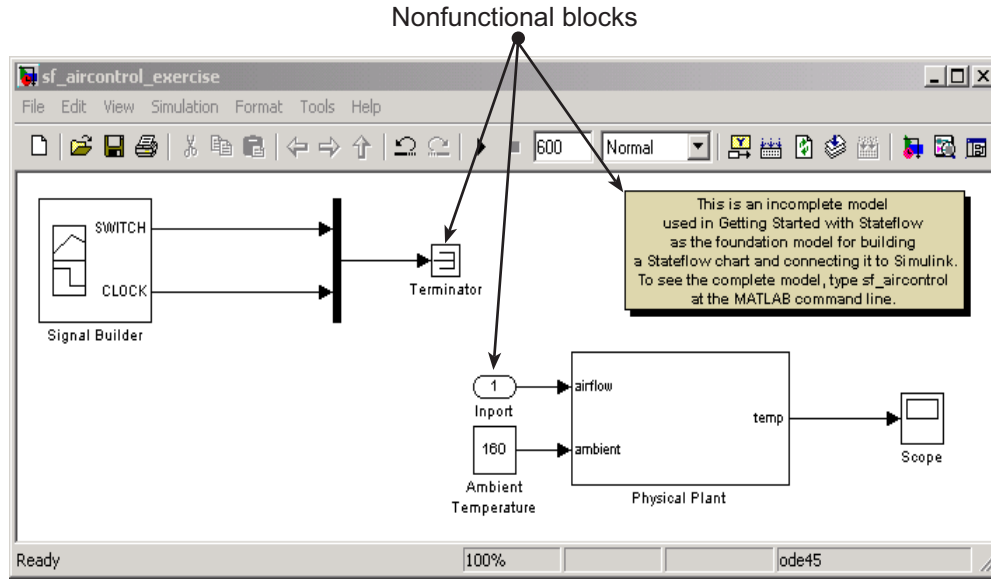
To add a Stateflow block to an existing Simulink model:

1 Open the Simulink model by typing `sf_aircontrol_exercise` at the MATLAB command prompt.

The model opens on your desktop:



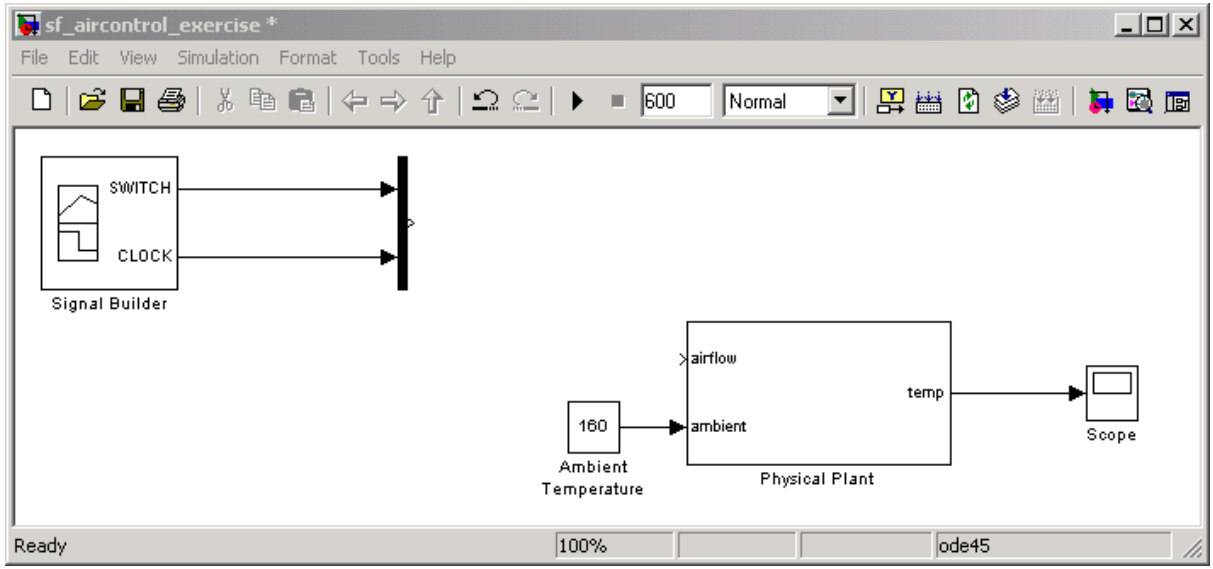
The model is incomplete because it doesn't include the Stateflow chart that you will build as you work through the exercises in this guide. Instead, the model contains several nonfunctional blocks: the Terminator, Inport, and Annotation blocks, as shown:



2 Delete the nonfunctional blocks and their connectors.

Tip Hold down the **Shift** key to select multiple objects, and then press **Delete**.

Your model should now look like this:



3 Save the model as **Stage1Interface** as follows:

   a Create a *new* directory for storing your working model.

   b In the Simulink model window, select **Save As** from the **File** menu.

   c Navigate to the new directory.

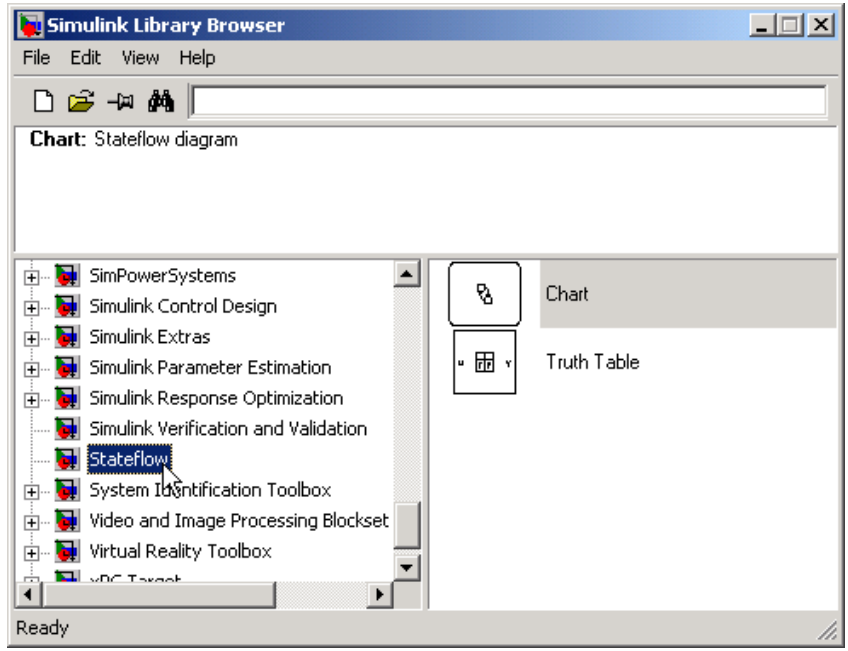
   d Enter `Stage1Interface.mdl` as the file name.

   e Leave the default type as **Simulink Models (*.mdl)**.

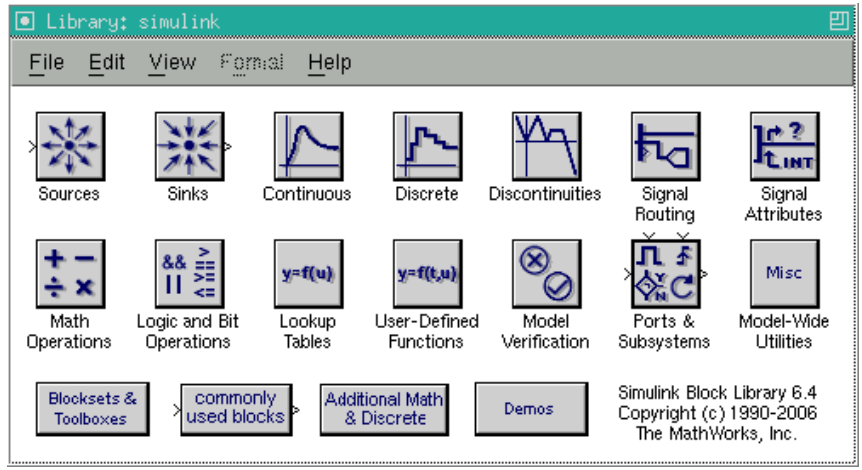
   f Click **Save**.

4 From the toolbar of the Simulink model, select the **Library Browser** icon:

In Windows, the Simulink Library Browser opens on your desktop:



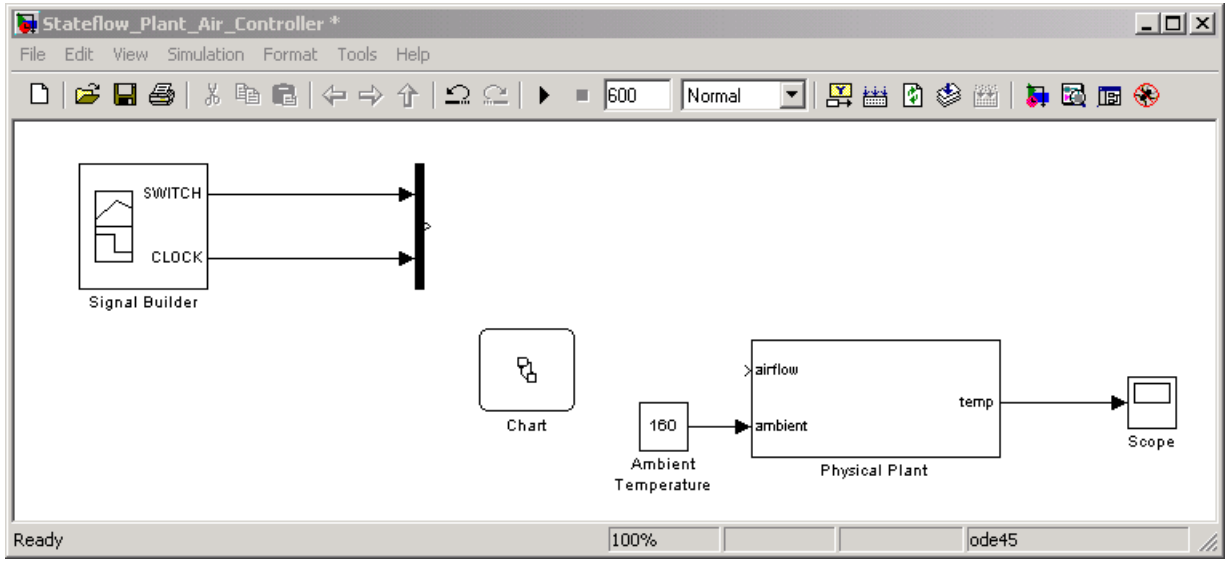
In UNIX, the Simulink library window appears:



5 Add the Stateflow block to the Simulink model by following these steps:

Operating System	Instructions
Windows	Follow these steps: **1** In the left scroll pane of the Library Browser, select **Stateflow**. **2** Drag the first block, called **Chart**, into your Simulink model.
UNIX	Follow these steps: **1** In the Simulink library window, double-click **Blocksets & Toolboxes**. **2** In the bottom row of the Blocksets & Toolboxes window, double-click **Stateflow**. The Stateflow library window `sflib` opens on your desktop. **3** In the `sflib` window, drag the first block, called **Chart**, into your Simulink model.

The Simulink model should now look like this:

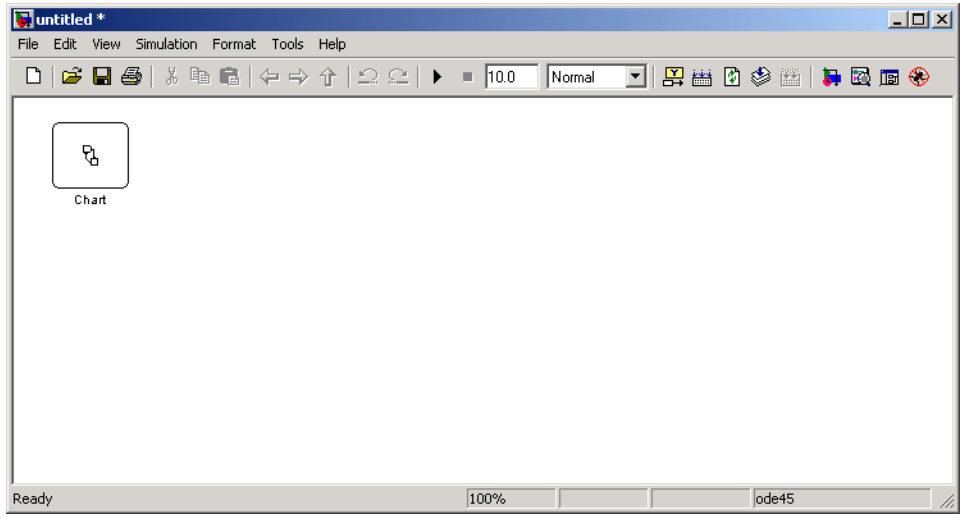


6 Click the label `Chart` under the Stateflow block and rename it `Air Controller`.

Tip There is a shortcut for adding a Stateflow block to a new Simulink model. At the MATLAB command prompt, enter this command:

```
sfnew
```

A new, untitled Simulink model opens on your desktop, automatically configured with a Stateflow block:

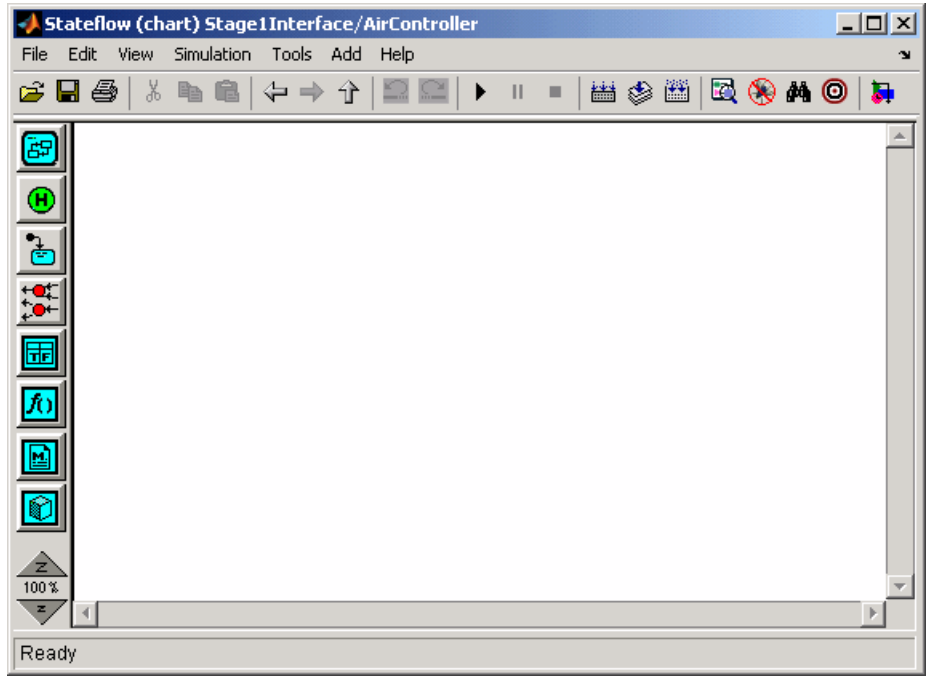


Defining the Inputs and Outputs

Inputs and outputs are data elements in a Stateflow chart that interact with the parent Simulink model. To define inputs and outputs for your Stateflow chart, follow these steps:

1 Double-click the Air Controller block in the Simulink model `Stage1Interface` to open the Stateflow chart.

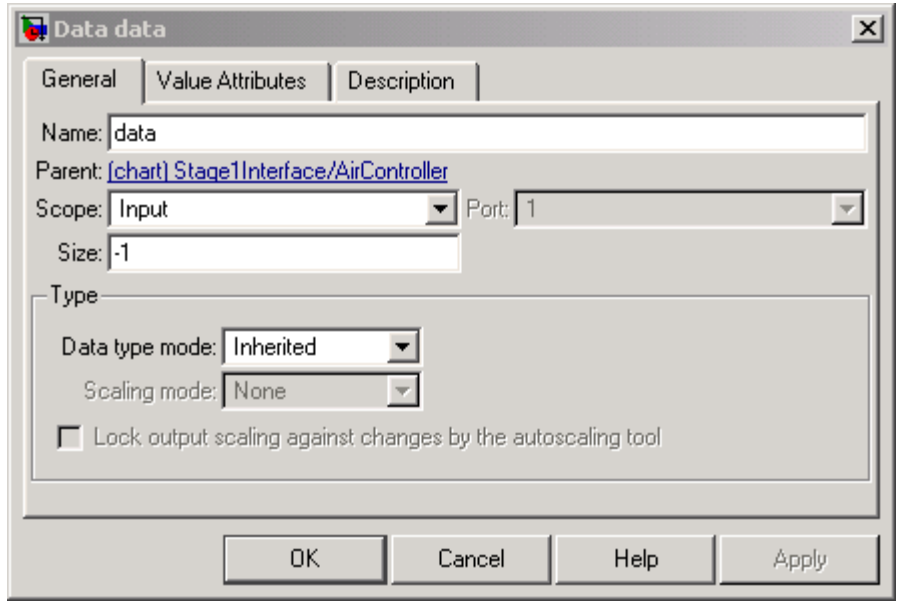
The Stateflow Editor opens on your desktop:



2 Add a data element to hold the value of the temperature input from Simulink by following these steps:

a Select **Data > Input from Simulink** from the **Add** menu.

The Data properties dialog box opens on your desktop with the **General** tab selected:



The default values that appear are based on the scope — in this case, a data input.

b In the **Name** field, change the name of the data element to `temp`.

c Leave the other fields at their default values in the **General** tab because they meet the design requirements, as follows:

Field	Default Value	What It Means
Scope	**Input**	Input from Simulink. The data element gets its value from the Simulink signal on the same input port.
Size	-1	The data element inherits its size from the Simulink signal on the same port.
Data type mode	**Inherited**	The data element inherits its data type from the Simulink signal on the same output port.

Note Ports are assigned to inputs and outputs in the order they are created. Because `temp` is the first *input* you created, it is assigned to *input port* 1.

d Select the **Value Attributes** tab and select the **Watch in debugger** check box.

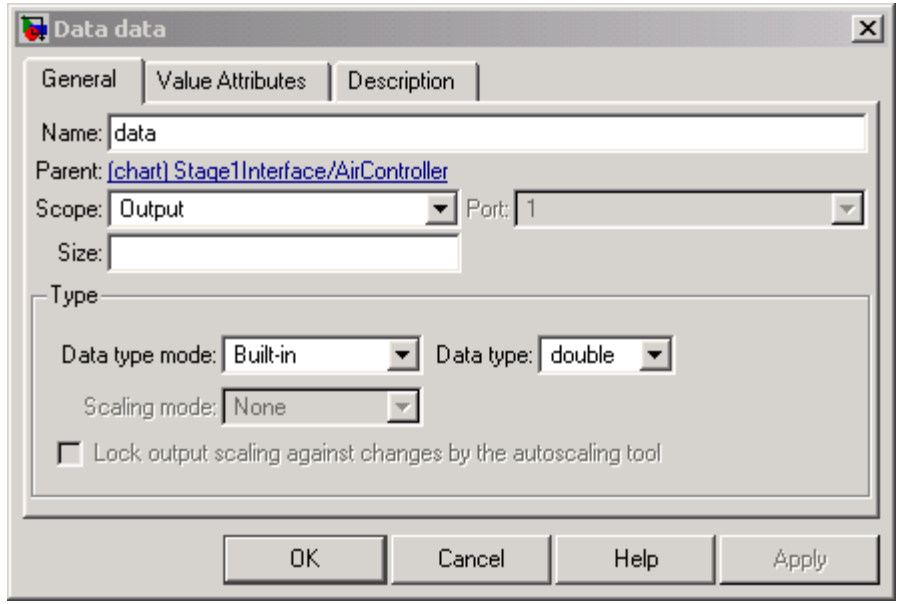
Enabling **Watch in debugger** allows you to examine the value of `temp` during breakpoints in simulation. You will try this later in Chapter 8, “Simulating the Chart”.

e Click **OK** to apply the changes and close the dialog box.

3 Add a data element to hold the value of the airflow output from the Air Controller chart by following these steps:

a Select **Data > Output to Simulink** from the **Add** menu.

The Data properties dialog box opens on your desktop, this time with different default values, associated with the scope **Output**:



b In the **Name** field of the Data properties dialog box, change the name of the data element to `airflow`.

c In the Data type field, select **uint8** (8-bit unsigned integer) from the submenu.

d Leave the other fields at their default values in the **General** tab pane because they meet the design requirements, as follows:

Field	Default Value	What It Means
Scope	**Output**	Output to Simulink.
Data type mode	**Built-in**	You can select **uint8** from a menu in the **Data type** field. The menu lists all data types supported by Stateflow.

Note Because `airflow` is the first *output* you created, it is assigned to *output port* 1.

e Click the **Value Attributes** tab and look at the **Initial value** field.

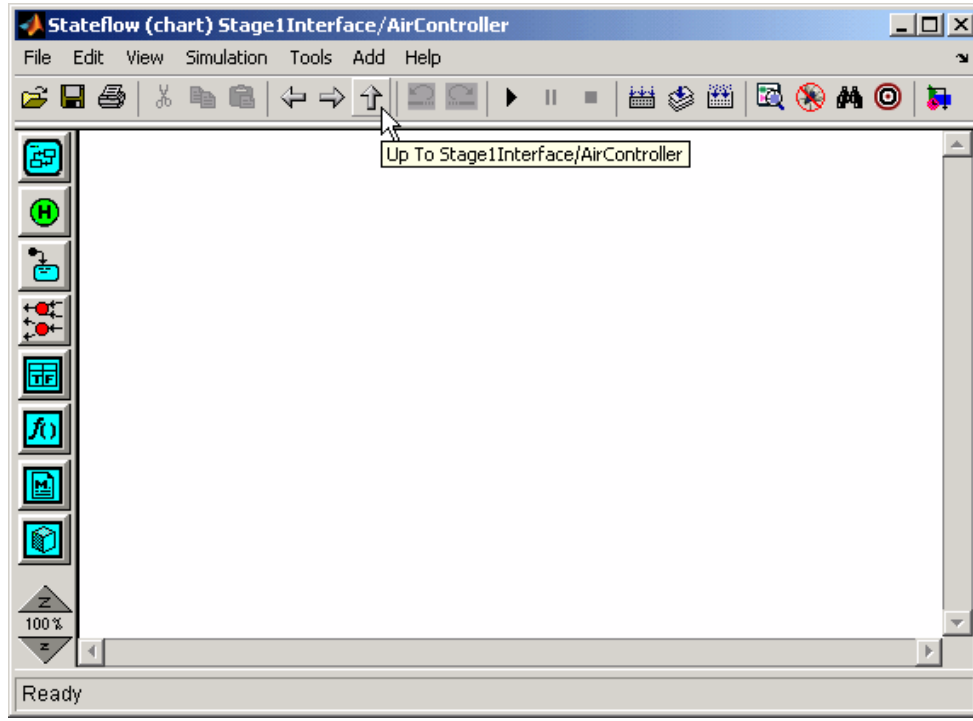
The initial value is a blank expression, which indicates a default value of zero, based on the data type. This value is consistent with the model design, which specifies that no fans are running when the Stateflow chart wakes up for the first time.

f Make the following changes in the Value Attributes tab:

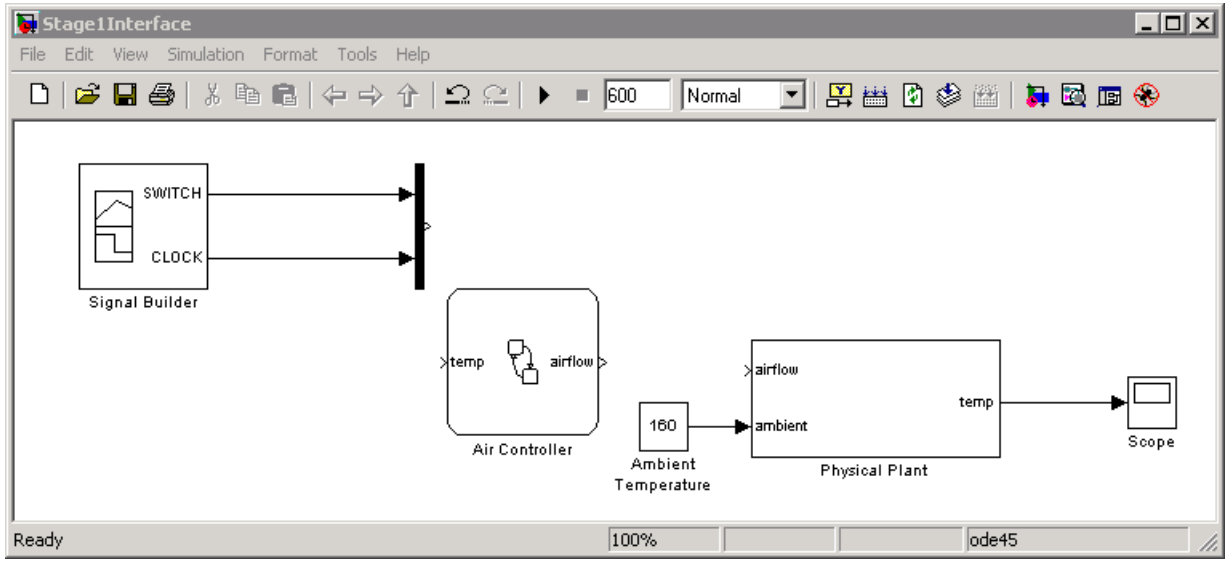
Property	What to Specify
Limit range	Enter **0** for **Minimum** and **2** for **Maximum**.
Watch in debugger	Select the check box to enable this option.

g Click **OK** to apply the changes and close the dialog.

4 Look back at the Simulink model by clicking the up-arrow in the Stateflow Editor toolbar:



Notice that the input `temp` and output `airflow` have been added to the Stateflow block:



Tip You may need to enlarge the Air Controller block to see the input and output clearly. To change the size of the block:

a Select the block and move your cursor over one of the corners until it changes to this shape:

b Hold down the left mouse button and drag the block to the desired size.

5 Save `Stage1Interface`, but leave the Stateflow Editor open for the next exercise.

Tip There are several ways to add data objects to Stateflow charts. You used the Stateflow Editor, which allows you to add data elements to the Stateflow chart that is open and has focus. However, to add data objects not just to a chart, but anywhere in the Stateflow design hierarchy, you can use a tool called the Model Explorer. This tool also lets you view and modify the data objects you have already added to Stateflow. For more information, see "Stateflow Hierarchy of Objects" and "Adding Data Using the Model Explorer" in the online Stateflow User's Guide documentation. You can also add data objects programmatically using the Stateflow API, as described in "Creating Stateflow Objects" in the online Stateflow API documentation.

Connecting the Stateflow Block to the Simulink Subsystem

Now that you have defined the inputs and outputs for the Stateflow Air Controller block, you need to connect them to the corresponding signals of the Simulink Physical Plant subsystem. Follow these steps:

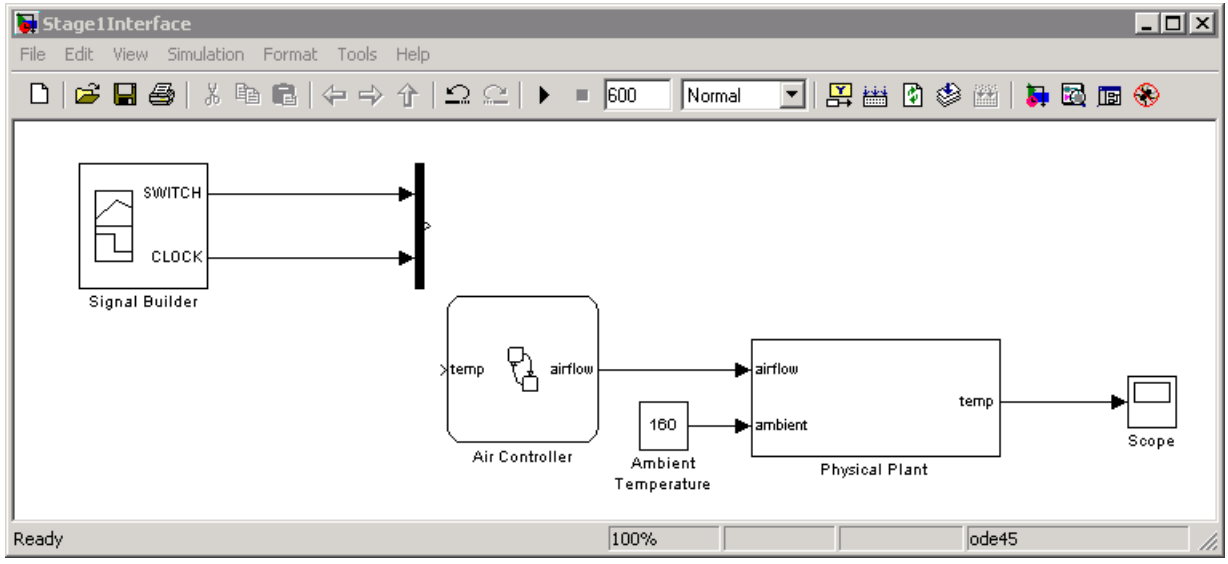
1. In the model `Stage1Interface`, connect the output `airflow` from Air Controller to the corresponding input in Physical Plant:
   a. Position the cursor over the output port for `airflow` on the right side of the Air Controller block.

      Notice that the cursor shape changes to crosshairs.

   b. Hold down the left mouse button and move the cursor to the input port for `airflow` on the left side of the Physical Plant block.

   c. Release the mouse.

The connection should look something like this:

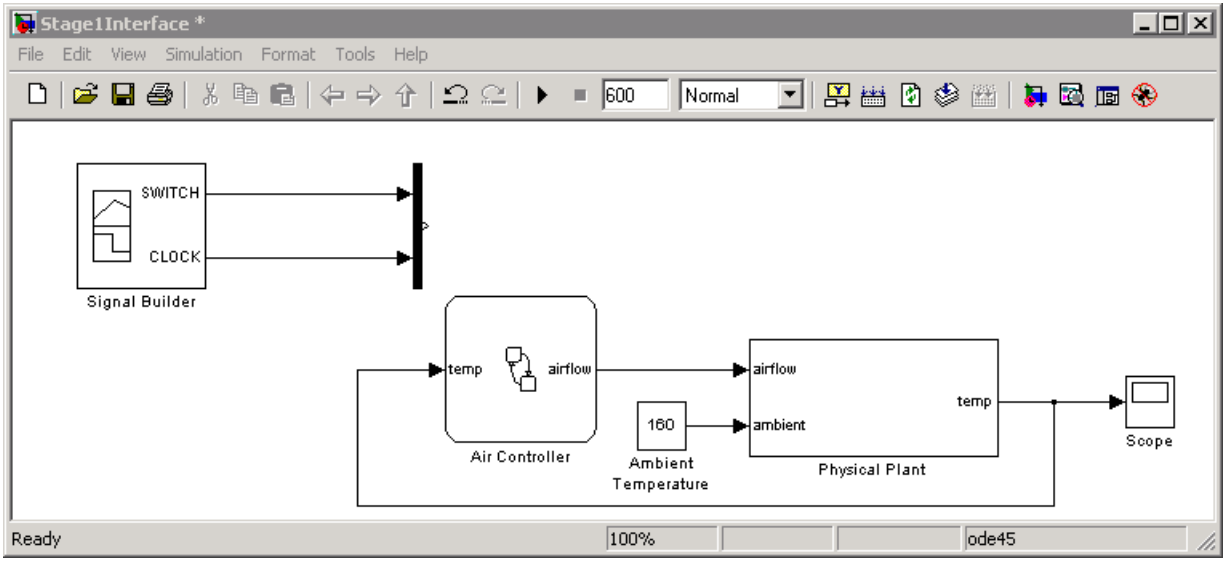


Tip There is a shortcut for automatically connecting blocks. Select the source block, and then hold down the **Ctrl** key and left-click the destination block.

2 Connect the output `temp` from the Physical Plant to the corresponding input in Air Controller by drawing a branch line from the line that connects `temp` to the Scope:

- **a** Position the cursor on the line where you want the branch line to start.
- **b** While holding down the **Ctrl** key, press and hold down the left mouse button.
- **c** Drag the cursor to the input port for `temp` on the left side of the Air Controller block.
- **d** Release the mouse button and the **Ctrl** key.

e Reposition the connection so that it looks like this:



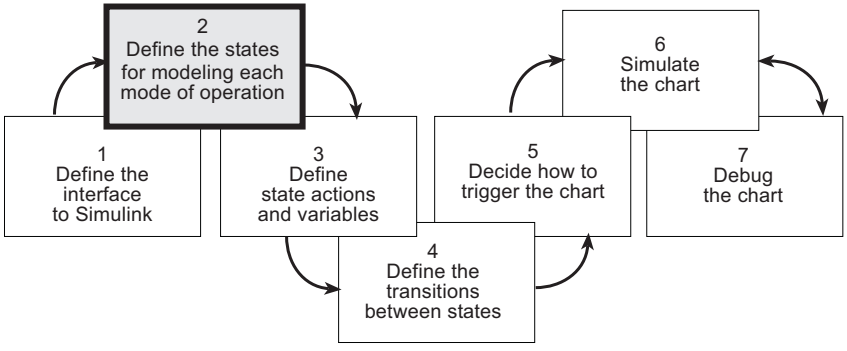
Tip To reposition connections, hold down the left mouse button over any side of the connection line so that the cursor changes to this symbol:

Drag the line to a new location.

3 Save `Stage1Interface`.

Where to go next. Now you are ready to begin phase 2 of the workflow: Chapter 4, “Defining the States for Modeling Each Mode of Operation”.

Defining the States for Modeling Each Mode of Operation



You have entered phase 2 of a basic workflow for building a Stateflow chart: *defining the states for modeling each mode of operation*. This chapter presents the design questions that you must answer and guides you through exercises for implementing the states, based on your design decisions.

Design Considerations for Defining the States

You should consider the following design questions when defining the states for modeling each mode of operation in your control system:

- Do I need states at all?
- What states are required?
- What is the hierarchy of states?
- What is the decomposition of the states?

When to Use States

Whether or not to use states depends on the control logic you want to implement. Stateflow allows you to model two types of control logic: *finite state machines* and *stateless flow charts*. Each is optimized for different applications, as follows:

Control Logic	Optimized for Modeling
Finite state machines	Physical systems that transition between a finite number of operating modes. In Stateflow charts, you represent each mode as a state.
Stateless flow chart	Abstract logic patterns — such as `if`, `if-else`, and `case` statements — and iterative loops — such as `for`, `while`, and `do` loops. These logic constructs are represented by connective junctions and transitions in Stateflow charts. No states are required. See "Flow Diagram Notation with Connective Junctions" in the online Stateflow User's Guide documentation.

The Stateflow Air Controller chart is a system that cools a physical plant by transitioning between several modes of operation and, therefore, can be modeled as a finite state machine. In the following sections, you will design the states that model each mode of operation.

Determining the States to Define

States model modes of operation in a physical system. To determine the number and type of states required for your Air Controller chart, you must identify each mode in which the system can operate. Often, a table or grid is helpful for analyzing each mode and determining dependencies between modes.

Analysis of Operating Modes

For Air Controller, the modes of operation are

Operating Mode	Description	Dependencies
Power Off	Turns off all power in the control system	No fan can operate when power is off.
Power On	Turns on all power in the control system	Zero, one, or two fans can operate when power is on.
Fan 1	Activates Fan 1	Fan 1 can be active at the same time as Fan 2. When activated, Fan 1 can turn on or off.
Fan 1 On	Cycles on Fan 1	Fan 1 On can be active if Fan 1 is active and power is on.
Fan 1 Off	Cycles off Fan 1	Fan 1 Off can be active if Fan 1 is active, and power is on.
Fan 2	Activates Fan 2	Fan 2 can be active at the same time as Fan 1. When activated, Fan 2 can turn on or off.
Fan 2 On	Cycles on Fan 2	Fan 2 On can be active if Fan 2 is active and power is on.

Operating Mode	Description	Dependencies
Fan 2 Off	Cycles off Fan 2	Fan 2 Off can be active if Fan 2 is active and power is on.
Calculate airflow	Calculates a constant value of 0, 1, or 2 to indicate how fast air is flowing. Outputs this value to the Simulink subsystem for selecting a cooling factor.	Calculates the constant value, based on how many fans have cycled on at each time step.

Number of States to Define

The number of states depends on the number of operating modes to be represented. In "Analysis of Operating Modes" on page 4-3, you learned that the Air Controller chart has nine operating modes. Therefore, you need to define nine states to model each mode. Here are the names you will assign to the states that represent each operating mode in "Implementing the States" on page 4-7:

State Name	Operating Mode
`PowerOff`	Power Off
`PowerOn`	Power On
`FAN1`	Fan 1
`FAN2`	Fan 2
`SpeedValue`	Calculate airflow
`FAN1.On`	Fan 1 On
`FAN1.Off`	Fan 1 Off
`FAN2.On`	Fan 2 Off
`FAN2.Off`	Fan 2 Off

Note Notice the use of dot notation to refer to the On and Off states for FAN1 and FAN2. You use namespace dot notation to give objects unique identifiers when they have the same name in different parts of the Stateflow model hierarchy.

Determining the Hierarchy of States

Objects in Stateflow can exist in a hierarchy. For example, states can *contain* other states — referred to as *substates* — and, in turn, can *be contained by* other states — referred to as *superstates*. You need to determine the hierarchical structure of states you will define for the Air Controller chart. Often, dependencies among states imply a hierarchical relationship — such as parent to child — between the states.

Based on the dependencies described in "Analysis of Operating Modes" on page 4-3, here is an analysis of state hierarchy for the Air Controller chart:

Dependent States	Implied Hierarchy
FAN1 and FAN2 depend on PowerOn. No fan can operate unless PowerOn is active.	FAN1 and FAN2 should be substates of a PowerOn state.
FAN1.On and FAN1.Off depend on Fan1 and PowerOn. FAN1 must be active before it can be cycled on or off.	FAN1 should have two substates, On and Off. In this hierarchical relationship, On and Off will inherit from FAN1 the dependency on PowerOn.
FAN2.On and FAN2.Off depend on FAN2 and PowerOn. FAN2 must be active before it can be cycled on or off.	FAN2 should have two substates, On and Off. In this hierarchical relationship, On and Off will inherit from FAN2 the dependency on PowerOn.
The state that calculates airflow needs to know how many fans are running at each time step.	The state that calculates airflow should be a substate of PowerOn so it can check the status of FAN1 and FAN2 at the same level of hierarchy.

Determining the Decomposition of States

The *decomposition* of a state dictates whether its substates execute exclusively of each other — as *exclusive (OR) states* — or can be activated at the same time — as *parallel (AND) states*. No two exclusive (OR) states can ever be active at the same time, while any number of parallel (AND) states can be activated concurrently.

The Air Controller chart requires both types of states. Here is a breakdown of the exclusive (OR) and parallel (AND) states required for the Stateflow chart:

State	Decomposition	Rationale
`PowerOff`, `PowerON`	Exclusive (OR) states	The power can never be on and off at the same time.
`FAN1`, `FAN2`	Parallel (AND) states	Zero, one, or two fans can operate at the same time, depending on how much cooling is required.
`FAN1.On`, `FAN1.Off`	Exclusive (OR) states	Fan 1 can never be on and off at the same time.
`FAN2.On`, `FAN2.Off`	Exclusive (OR) states	Fan 2 can never be on and off at the same time.
`SpeedValue`	Parallel (AND) state	`SpeedValue` is an observer state that monitors the status of Fan 1 and Fan 2, updating its output based on how many fans are operating at each time step. `SpeedValue` must be activated at the same time as Fan 1 and Fan 2, but execute last so it can capture the most current status of the fans.

Implementing the States

When you add states to the Air Controller chart, you will work from the top down in the Stateflow hierarchy, as follows:

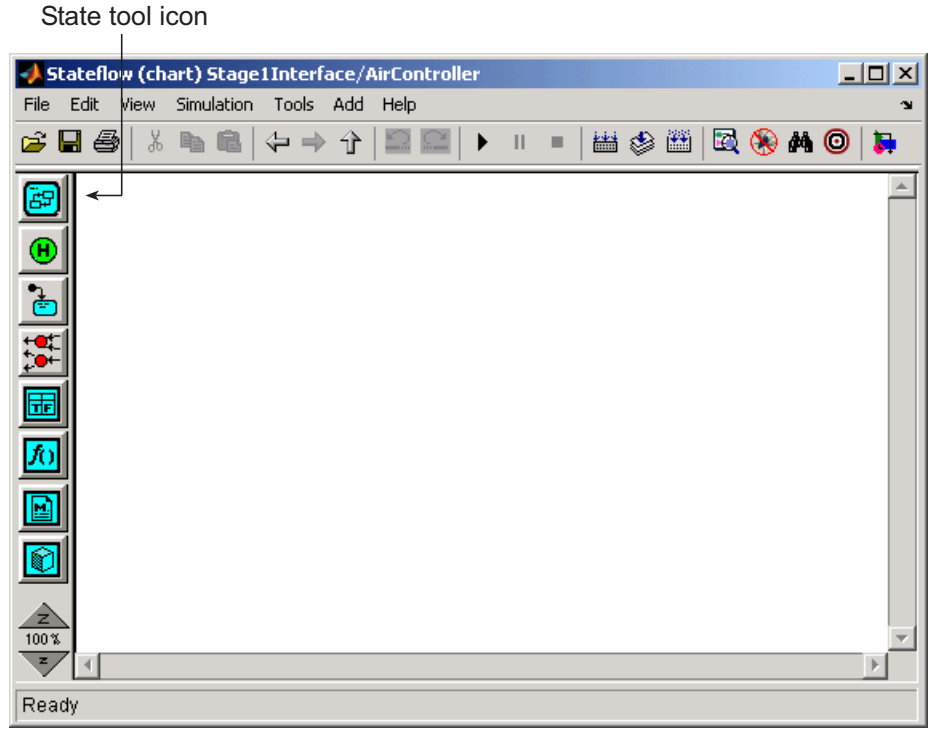
Adding the Power On and Power Off States

As you learned in "Determining the Decomposition of States" on page 4-6, the `PowerOff` and `PowerOn` states are exclusive (OR) states that turn power off and on in the control system. These states are never active at the same time. By default, states are exclusive (OR) states, represented graphically as rectangles with solid borders.

To add `PowerOn` and `PowerOff` to your Stateflow chart, follow these steps:

1. Open the model `Stage1Interface` and save it as `Stage2States` in the same directory.
2. In `Stage2States`, double-click the Air Controller block to open the Stateflow chart.

The Stateflow Editor for Air Controller opens on your desktop. Notice the object palette on the left side of the editor window. This palette displays a set of tools for drawing graphical Stateflow chart objects, including states:



3 Left-click the state tool icon:

4 Move your cursor into the drawing area.

The cursor changes to a rectangle, the graphical representation of a state.

5 Click in the upper-left corner of the drawing area to place the state.

The new state appears with a blinking text cursor in its upper-left corner.

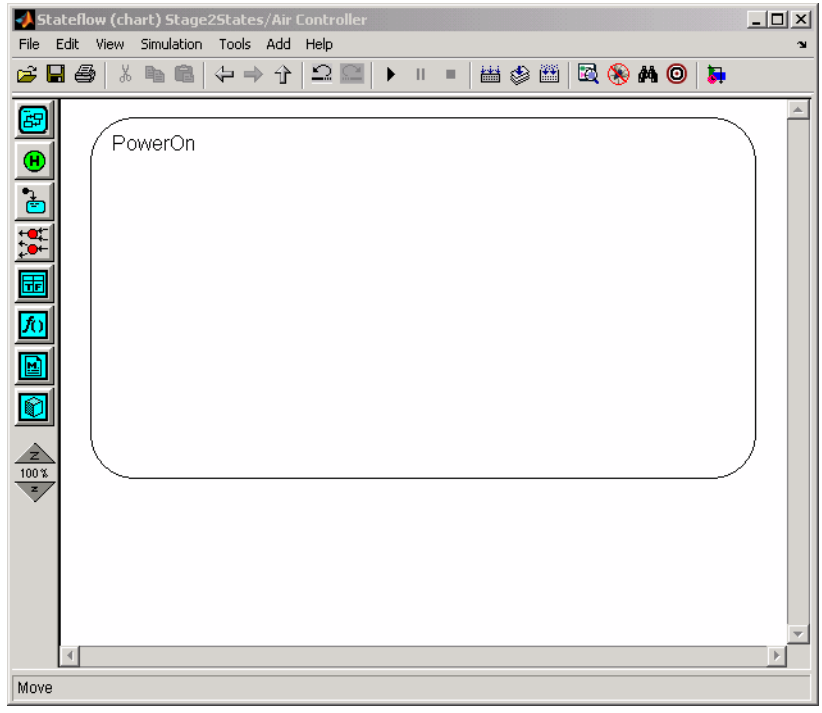
6 At the text cursor, type `PowerOn` to name the state.

Tip If you click away from the text cursor before typing the new name, the cursor changes to a question mark. Click the question mark to restore the text cursor.

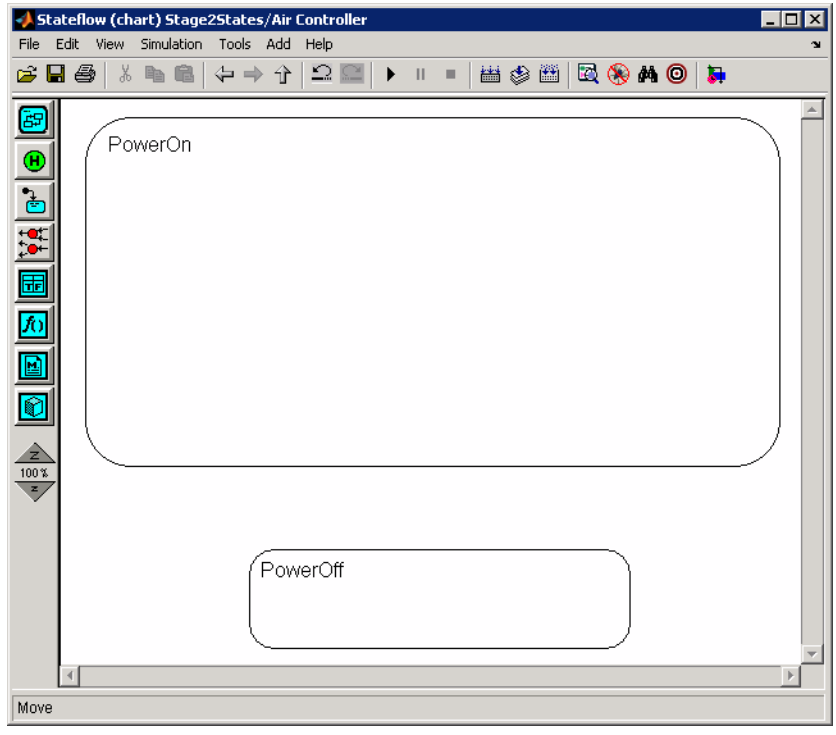
---

7 Move the cursor to the lower-right corner of the rectangle so it changes to this symbol:

8 Drag the lower-right corner to enlarge the state as shown:



9 Click the state tool icon again and draw a smaller state named `PowerOff` at the bottom of the drawing area, like this:



10 Save the chart by clicking **Save Model** in the **File** menu of the Stateflow Editor, but leave the Stateflow Editor open for the next exercise.

Adding and Configuring Parallel States

In "Determining the States to Define" on page 4-3, you learned that `FAN1`, `FAN2`, and `SpeedValue` will be represented by parallel (AND) substates of the `PowerOn` state. Stateflow represents parallel states graphically as rectangles with dashed borders.

In this set of exercises, you will learn how to

- Assign parallel decomposition to `PowerOn` so its substates can be activated concurrently.

  Recall that the decomposition of a state determines whether its substates will be exclusive or parallel.
- Add parallel substates to a state in the chart.
- Set the order of execution for the parallel substates.

  Even though parallel states can be activated concurrently, they execute in a sequential order.

Setting Parallel Decomposition

Follow these steps:

1. In the Stateflow Editor for the chart Air Controller, right-click inside `PowerOn`.

   A submenu opens, presenting tasks you can perform and properties you can set for the selected state.
2. In the submenu, select **Decompostion > Parallel (AND)**.
3. Save the model `Stage2States`, but leave the Stateflow Editor open for the next exercise.

Adding the Fan States

Follow these steps:

1. Left-click the state tool icon in the Stateflow Editor and place two states inside the `PowerOn` state.

Tip Instead of using the state tool icon to add multiple states, you can right-click inside an existing state and drag a copy to a new position in the chart. This shortcut is convenient when you need to create states of the same size and shape, such as the fan states.

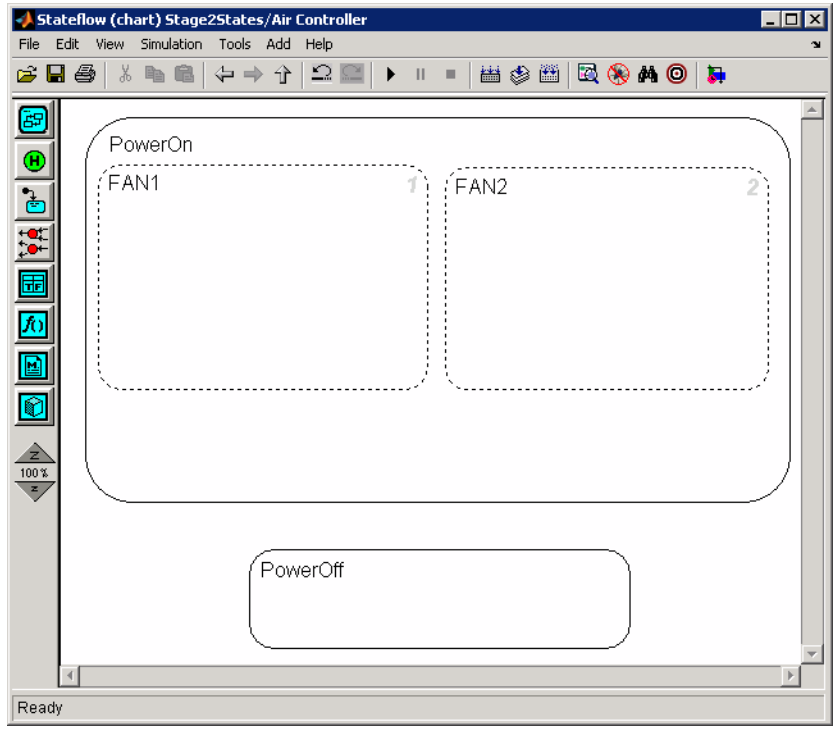
---

2 Notice the appearance of the states you just added.

The borders of the two states appear as dashed lines, indicating that they are parallel states. Note also that the substates display numbers in their upper-right corners. These numbers specify the order of execution. Although multiple parallel (AND) states in the same chart are activated concurrently, Stateflow must determine when to execute each one during simulation.

3 Name the new substates `FAN1` and `FAN2`.

You have created hierarchy in the Air Controller chart. `PowerOn` is now a superstate while `FAN1` and `FAN2` are substates. Your chart should look like something like this:



Note Your chart may not show the same execution order for parallel substates FAN1 and FAN2. The reason is that, by default, Stateflow orders parallel states based on where they are located in the state diagram. Priority goes from top to bottom and then left to right. If FAN2 is higher in position than FAN1 in your diagram, FAN2 moves to the top of the order. You will fine-tune order of activation in a later exercise, "Setting Explicit Ordering of Parallel States" on page 4-14.

Tip If you want to move a state together with its substates — and any other graphical objects it contains — double-click the state. It turns gray, indicating that the state is grouped with the objects inside it and that they can be moved as a unit. To ungroup the objects, double-click again.

4 Save the model Stage2States, but leave the Stateflow Editor open for the next exercise.

Adding the SpeedValue State

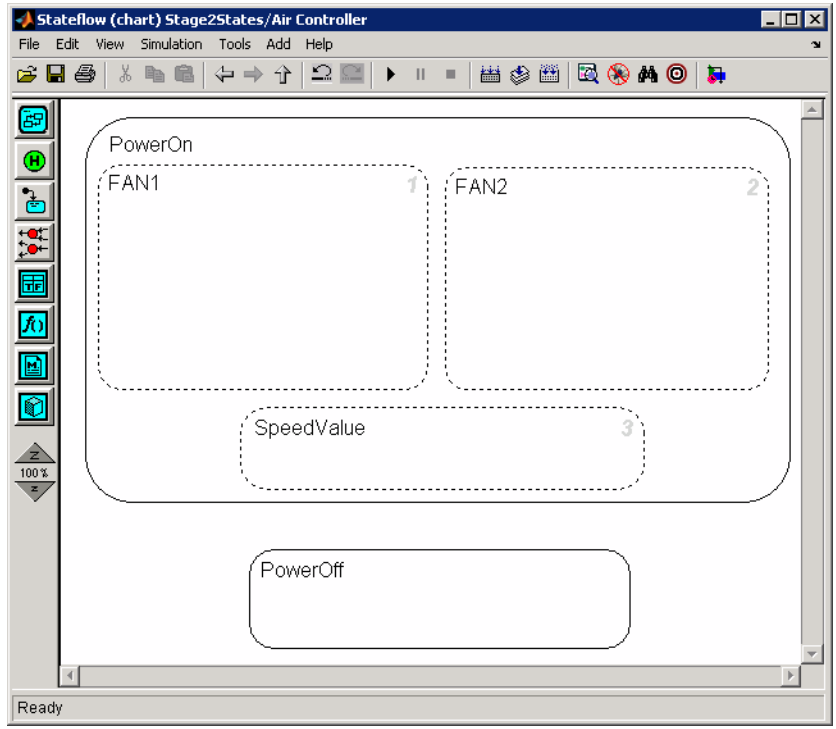
Recall that SpeedValue acts as an observer state, which monitors the status of the FAN1 and FAN2 states. To add the SpeedValue state, follow these steps:

1 Add another substate to PowerOn under FAN1 and FAN2, either by using the state tool icon or copying an existing state in the chart.

You may need to resize the substate so that it doesn't overlap any other substate, but remains within the borders of PowerOn.

2 Name the state `SpeedValue`.

Like `FAN1` and `FAN2`, `SpeedValue` appears as a parallel substate because its parent, the superstate `PowerOn`, has parallel decomposition.



3 Save the model `Stage2States`, but leave the Stateflow Editor open for the next exercise, "Setting Explicit Ordering of Parallel States" on page 4-14.

Setting Explicit Ordering of Parallel States

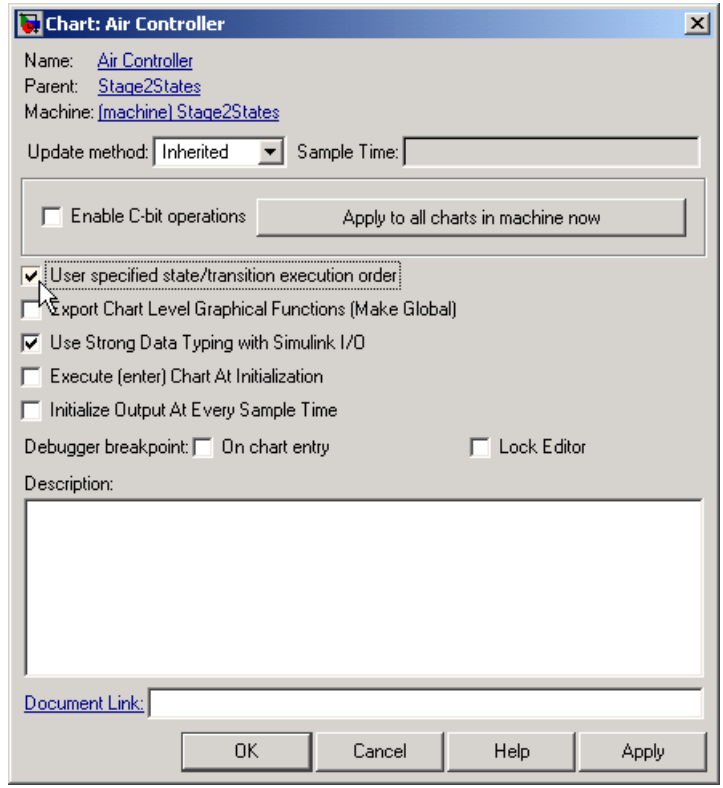
Recall that, by default, Stateflow assigns execution order of parallel states based on their locations in the chart. This is called *implicit ordering*. Implicit ordering creates a dependency between design layout and execution priority. When you rearrange parallel states in your diagram, you may change order of execution and affect simulation results. In this exercise, you will override this default so you can set execution order explicitly for each parallel state in your chart.

Follow these steps:

1 In the Stateflow Editor, select **Chart Properties** from the **File** menu.

The properties dialog box for the Air Controller chart opens on your desktop.

2 Select the check box **User specified state/transition execution order** and click **OK**.



Note Selecting this option also allows you to explicitly specify the order in which transitions are executed when there is a choice of transitions to take from one state to another. This is not an issue for the Air Controller chart because it is deterministic: for each exclusive (OR) state, there is one and only one transition to a next exclusive (OR) state. You will learn more about transitions in "Drawing the Transitions Between States" on page 6-5.

3 Assign order of execution for each parallel state in the Air Controller chart:

   a Right-click inside each parallel state to bring up its state properties submenu.

   b From the submenu, select **Execution Order** and make the following assignments:

For State:	Assign:
FAN1	1
FAN2	2
SpeedValue	3

   Here is the rationale for this order of execution:

   - FAN1 should execute first because it cycles on at a lower temperature than FAN2.
   - SpeedValue should execute last so it can observe the most current status of FAN1 and FAN2.

4 Save the model Stage2States, but leave the Stateflow Editor open for the next exercise, "Adding the On and Off States for the Fans" on page 4-16.

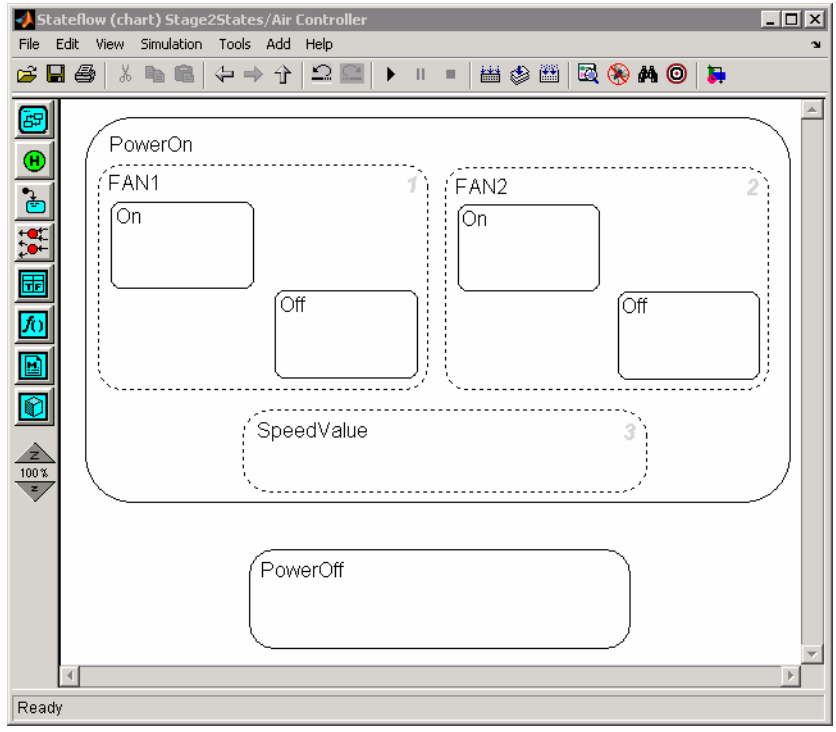
Adding the On and Off States for the Fans

In this exercise, you will enter the on and off substates for each fan. Because fans cannot cycle on and off at the same time, these states must be exclusive, not parallel. Even though FAN1 and FAN2 are parallel states, their *decomposition* is exclusive (OR) by default. As a result, any substate that you add to FAN1 or FAN2 will be an exclusive (OR) state.

Follow these steps:

1. Add two substates inside `FAN1` and `FAN2`.
2. Resize the substates to fit within the borders of `FAN1` and `FAN2`.
3. In each fan state, name one substate `On` and name the other `Off`.

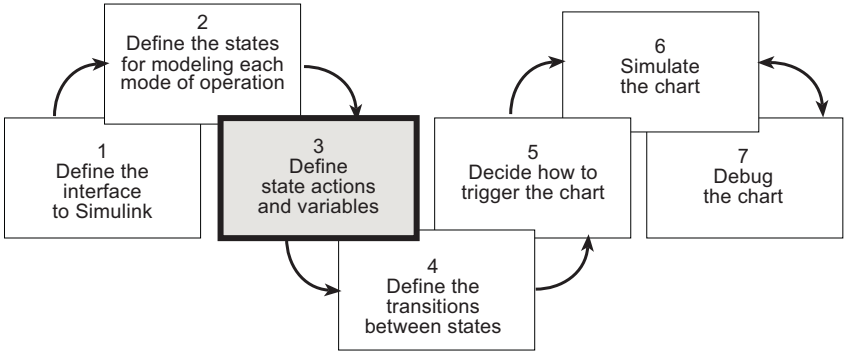
   Your Air Controller chart should now look something like this:



4. Save the model `Stage2States`.

Where to go next. Now you are ready to begin phase 3 of the workflow: Chapter 5, “Defining State Actions and Variables”.

Defining State Actions and Variables



You have entered phase 3 of a basic workflow for building a Stateflow chart: *defining state actions and variables*. This chapter presents the design questions that you must answer and guides you through exercises for defining state actions and variables, based on your design decisions.

Design Considerations for Defining State Actions and Variables (p. 5-2)	Poses the design questions for phase 3 of the workflow and presents the rationale for the solutions
Implementing State Actions (p. 5-5)	Provides hands-on exercises that show you how to implement the state actions as designed

Design Considerations for Defining State Actions and Variables

You should consider the following design questions when defining state actions and variables:

- Do I need to define any local or persistent variables in my Stateflow chart?
- Should any of my states perform actions when they become active?
- What type of state action should I use?

Defining State Variables

At this stage of the workflow for developing Stateflow charts, you need to determine if any of your states require local or persistent variables. If so, you define these data elements using the Stateflow Editor or the Model Explorer, as described in "Adding Data" in the online Stateflow User's Guide documentation.

The states in the Air Controller chart do not require local or persistent data, only the input and output data that you defined in "Defining the Inputs and Outputs" on page 3-10.

Determining Whether to Use State Actions

During simulation of a Stateflow chart, states can perform actions while they are active. Often, actions are used to manipulate data, using a variety of constructs such as binary, bitwise, unary, assignment, pointer, and type cast operators (see "Using Actions in Stateflow" in the online Stateflow User's Guide documentation).

When building the Air Controller chart, you need to determine whether any states should perform actions. Some charts may not use state actions at all, but instead perform actions only during the transitions from state to state. Other charts require both types of state actions.

For the Air Controller chart, think about whether data values need to be initialized or modified during any of its modes of operation. Recall that the chart receives the air temperature of the plant as the input `temp` from Simulink. The chart then uses this value to activate fans if necessary to cool

the air. Based on how many fans are running, the chart sets a value that indicates speed of airflow, which it sends at each time step to Simulink as the output `airflow`. The Air Controller does not modify the value of `temp`, but does need to update the value of `airflow`.

The next consideration is when to update, and for that matter, initialize the value of `airflow`. If the *when* translates to a mode of operation, the action should likely be performed by the state that represents that mode of operation. Here is the analysis for the Air Controller chart:

Action	When	How
Initialize `airflow` to 0.	Before simulation	Set an initial value when you first define `airflow` (as you did in "Defining the Inputs and Outputs" on page 3-10).
Set `airflow` to 0.	Whenever power is off	Add an action in the state `PowerOff`.
Update airflow to 0, 1, or 2, based on how many fans are running.	Whenever power is on	Add an action in the state `SpeedValue`, which becomes active concurrently with `FAN1` and `FAN2` when the state `PowerOn` is active.

Determining the Type of State Action to Use

States perform actions at different phases of their execution cycle from the time they become active to the time they become inactive. Three basic state actions are

Type of Action	When Executed	How Often Executed While State Is Active
Entry	When the state is entered (becomes active)	Once
During	While the state is active and no valid transition to another state is available	At every time step
Exit	Before a transition is taken to another state	Once

For example, you can use `entry` actions to initialize data, `during` actions to update data, and `exit` actions to configure data for the next transition. (There are other types of state actions, but they involve concepts that go beyond the scope of this guide. For more information, see "Using Actions in Stateflow" in the online Stateflow User's Guide documentation.)

Based on the requirements in "Determining Whether to Use State Actions" on page 5-2, you will write the following state actions for the Air Controller chart:

- `Entry` action in state `PowerOff` to set `airflow` to 0
- `During` action in state `SpeedValue` to calculate the value of `airflow` at every time step

Implementing State Actions

To implement the state actions as designed, you need to perform the following tasks:

Writing an Entry Action

The syntax for `entry` actions is

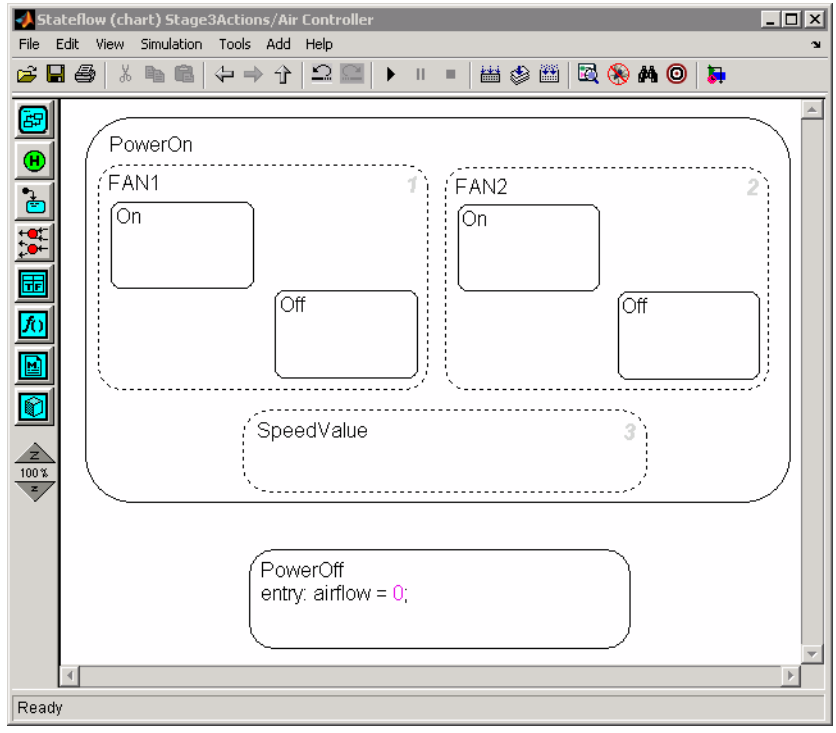
```
entry:one or more actions;
en:one or more actions;
```

To write the `entry` action for `PowerOff`, follow these steps:

1. Open the model `Stage2States` and save it as `Stage3Actions` in the same directory.
2. In `Stage3Actions`, double-click the Air Controller block to open the Stateflow chart.
3. Click inside the `PowerOff` state after the last letter of its name label to get a blinking text cursor.
4. Press the **Enter** key and type

```
entry:  airflow = 0;
```

Your chart should look like this:



5 Save `Stage3Actions`, but leave the Stateflow Editor open for the next exercise.

Writing a During Action

The syntax for `during` actions is

```
during:one or more actions;
du:one or more actions;
```

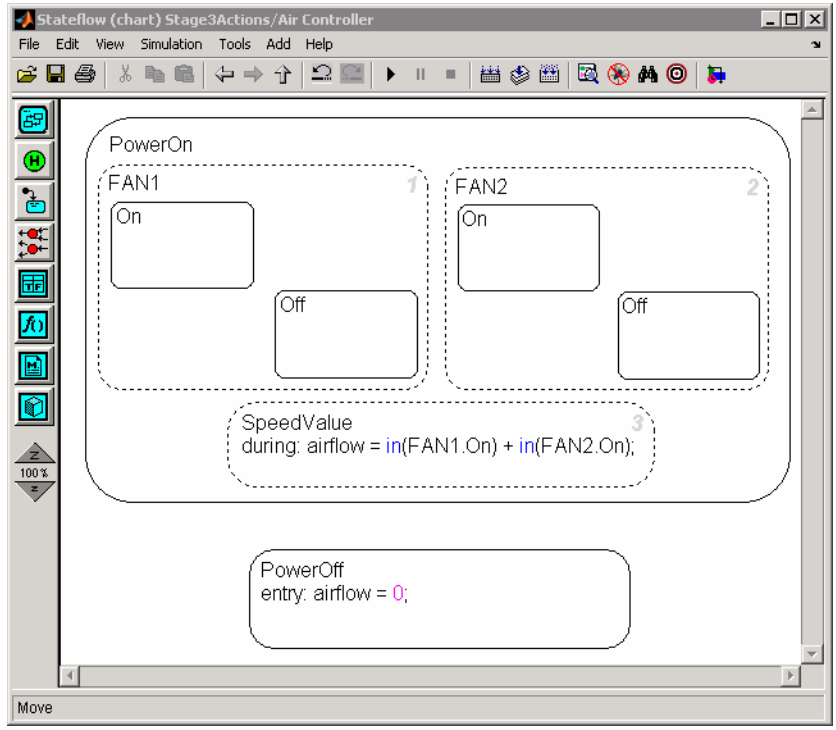
The `during` action for `SpeedValue` uses a Boolean expression to determine whether zero, one, or two fans are running at each time step.

To write the `during` action for `SpeedValue`, follow these steps:

1. Click inside the `SpeedValue` state after the last letter of its name label to get a blinking text cursor.
2. Press the **Enter** key and type

   ```
   during:  airflow = in(FAN1.On) + in(FAN2.On);
   ```

   Your chart should look like this:

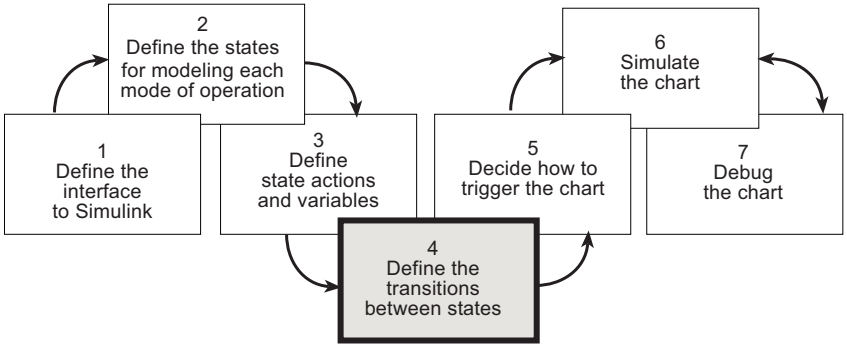


The Boolean expression `in(FAN1.On)` is true — and its value equals 1 — if the `On` state of `FAN1` is active. If `FAN1` is not on — that is, its `Off` state is active or power is off — then `in(FAN1.On)` equals 0. Similarly, the value of `in(FAN2.On)` represents whether `FAN2` is on or off. Therefore, the sum of these Boolean expressions indicates whether 0, 1, or 2 fans are operating during each time step.

3 Save the model `Stage3Actions`.

Where to go next. Now you are ready to begin phase 4 of the workflow: Chapter 6, “Defining Transitions Between States”.

Defining Transitions Between States



You have entered phase 4 of a basic workflow for building a Stateflow chart: *defining the transitions between states*. This chapter presents the design questions you must answer and guides you through exercises for defining transitions, based on your design decisions.

Design Considerations for Defining Transitions Between States (p. 6-2)	Poses the design questions for phase 4 of the workflow and presents the rationale for the solutions
Adding the Transitions (p. 6-5)	Provides hands-on exercises that show you how to implement the transitions as designed

Design Considerations for Defining Transitions Between States

Transitions create paths for the logic flow of a system from one state to another. When a transition is taken from state A to state B, state A becomes inactive and state B becomes active.

Transitions have direction and are represented in a Stateflow chart by lines with arrowheads. Transitions are unidirectional, not bidirectional. You must add a transition for each direction of flow between two states.

You should consider the following design questions when defining transitions between states:

- How does my state machine transition from one operating mode to another?
- Where should I place default transitions?
- How should I guard each transition from one state to another?

Determining How and When to Transition Between Operating Modes

Exclusive (OR) states require transitions. Recall that no two exclusive states can be active at the same time. Therefore, you need to add transitions to specify when and where control flows from one exclusive state to another.

Typically, parallel (AND) states do not require transitions because they execute concurrently.

The Air Controller chart models a system in which power can cycle on and off and, while power is on, fans can cycle on and off. Six exclusive (OR) states represent these operating modes. To model this activity, you need to add the following transitions between exclusive (OR) states:

- `PowerOff` to `PowerOn`
- `PowerOn` to `PowerOff`
- `FAN1.Off` to `FAN1.On`
- `FAN1.On` to `FAN1.Off`

- FAN2.Off to FAN2.On
- FAN2.On to FAN2.Off

Placing Default Transitions

Good design practice in Stateflow requires that you specify default transitions for exclusive (OR) states at each level of hierarchy. *Default transitions* indicate which exclusive (OR) state is to be active when there is ambiguity between two or more exclusive (OR) states at the same level in the Stateflow hierarchy. There are three such areas of ambiguity in the Air Controller chart:

- When the chart wakes up, should power be on or off?
- When FAN1 becomes active, should it be on or off?
- When FAN2 becomes active, should it be on or off?

In each case, the initial state should be off so you will add default transitions to the states PowerOff, FAN1.Off, and FAN2.Off.

Guarding the Transitions

Guarding a transition means specifying a condition, action, or event that allows the transition to be taken from one state to another. Based on the design of the Air Controller chart, here are the requirements for guarding the transitions from one exclusive operating mode to another:

Transition	When Should It Occur?	How to Guard It
PowerOff to PowerOn PowerOn to PowerOff	At regular time intervals	Specify an edge-triggered event

Transition	When Should It Occur?	How to Guard It
`FAN1.Off` to `FAN1.On`	When the temperature of the physical plant rises above 120 degrees	Specify a condition based on temperature value
`FAN1.On` to `FAN1.Off`	When the temperature of the physical plant falls below 120 degrees	
`FAN2.Off` to `FAN2.On`	When the temperature rises above 150 degrees, a threshold indicating that first fan is not providing the required amount of cooling	
`FAN2.On` to `FAN2.Off`	When the temperature falls below 150 degrees	

Adding the Transitions

To implement the transitions as designed, you need to perform the following tasks:

Drawing the Transitions Between States

In "Design Considerations for Defining Transitions Between States" on page 6-2, you learned that the following transitions occur in the Air Controller chart:

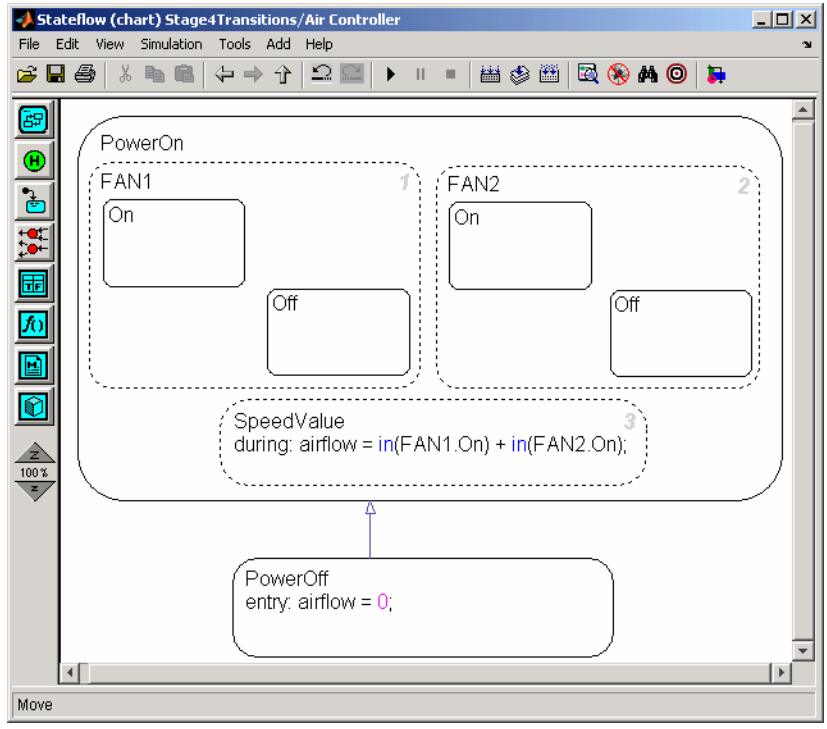
- Power for the control system can cycle on and off.
- Each fan can cycle on and off.

You will model this activity by drawing transitions between the `PowerOn` and `PowerOff` states and between the `On` and `Off` states for each fan. Follow these steps:

1. Open the model `Stage3Actions` and save it as `Stage4Transitions` in the same directory.
2. In `Stage4Transitions`, double-click the Air Controller block to open the Stateflow chart.

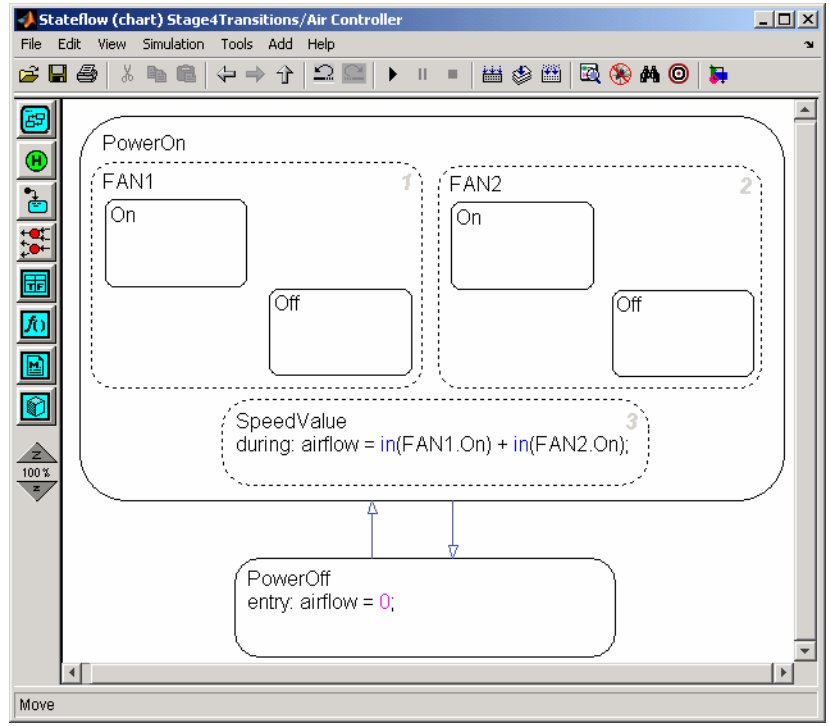
   The Stateflow Editor for Air Controller opens on your desktop.
3. Draw transitions between the `PowerOff` to `PowerOn` states:
   a. Move the cursor over the top edge of `PowerOff` until the cursor shape changes to crosshairs.
   b. Hold down the left mouse button, drag the cursor to the bottom edge of `PowerOn`, and release the mouse.

You should see a transition pointing from `PowerOff` to `PowerOn`:



c Follow the same procedure to draw a transition from `PowerOn` to `PowerOff`.

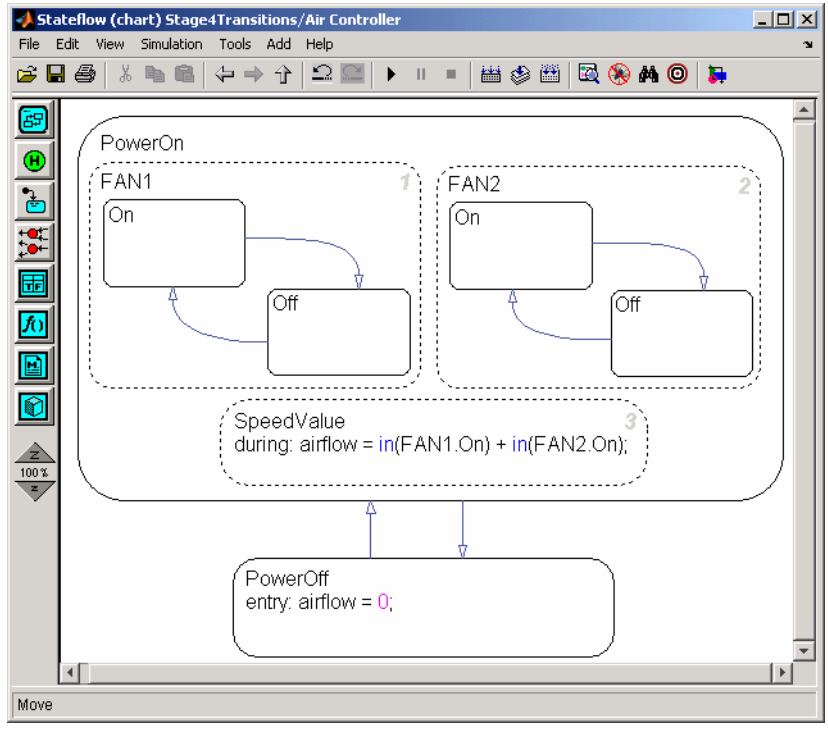
Your chart should now look like this:



4 Follow the procedure described in step 3 to draw the following transitions between the `Off` and `On` states for each fan:

- Transition from `Off` to `On` in `FAN1`
- Transition from `On` to `Off` in `FAN1`
- Transition from `Off` to `On` in `FAN2`
- Transition from `On` to `Off` in `FAN2`

Your chart should now look like this:



5 Save `Stage4Transitions`, but leave the Stateflow Editor open for the next exercise.

Adding Default Transitions

In "Placing Default Transitions" on page 6-3, you learned that you need to add default transitions to `PowerOff`, `FAN1.Off`, and `FAN2.Off`. Follow these steps:

1 In the Stateflow Editor, left-click the default transition icon in the object palette:

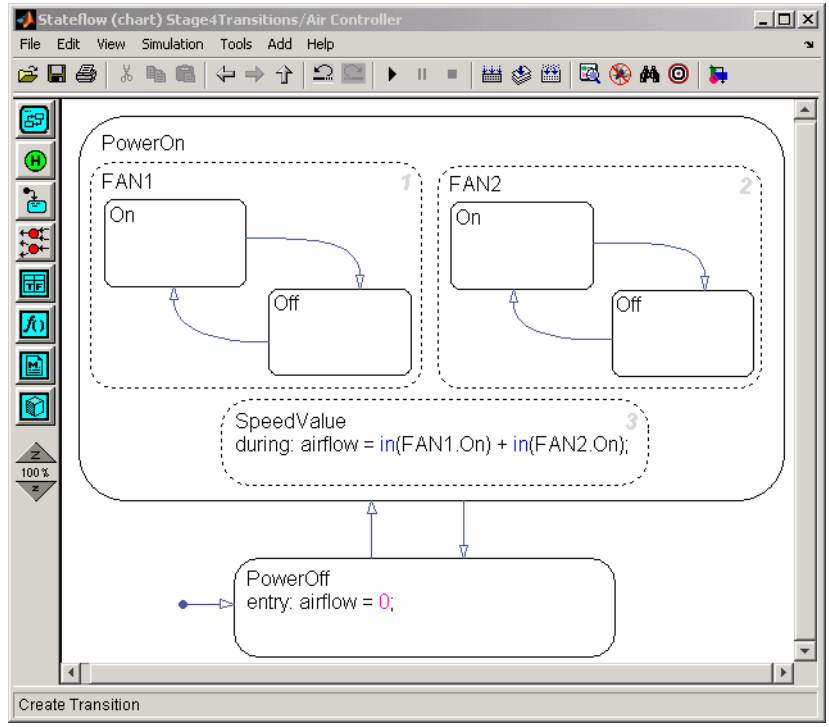
2 Move your cursor into the drawing area.

The cursor changes to a diagonal arrow.

3 Place the cursor at the left edge of the `PowerOff` state.

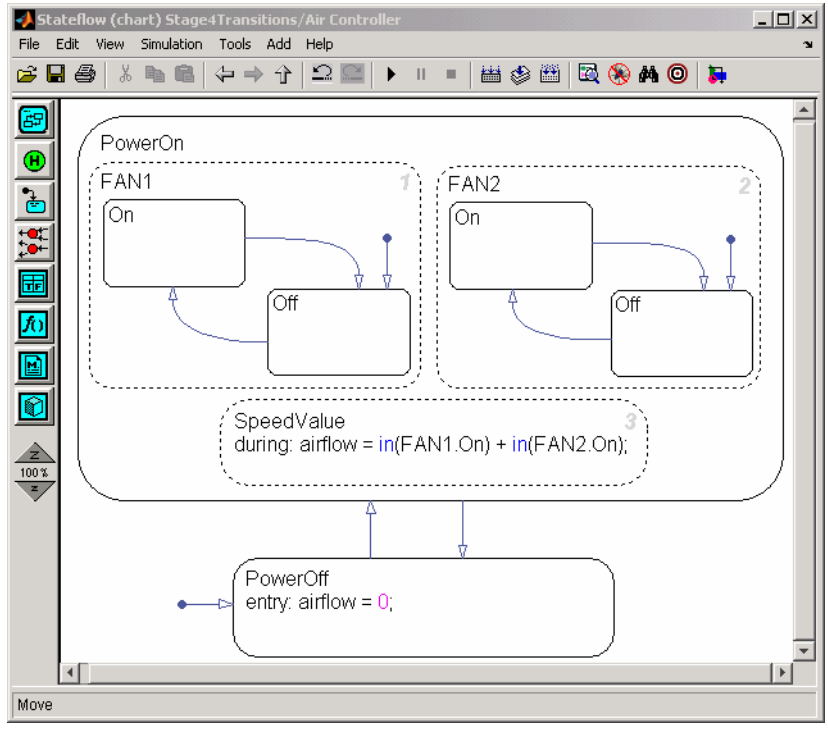
4 When the arrow becomes orthogonal to the edge, release the mouse button.

The default transition attaches to the `PowerOff` state. It appears as a directed line with an arrow at its head and a closed tail:

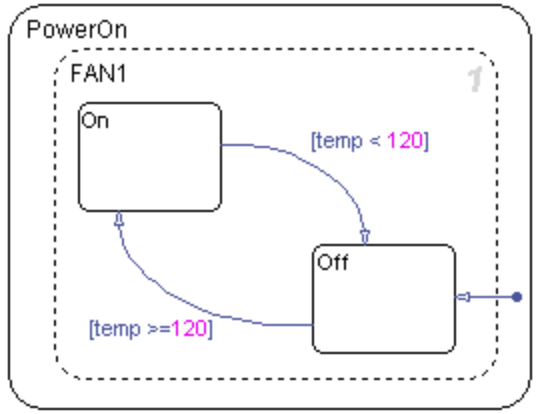


5 Repeat the same procedure to add default transitions at the top edges of `FAN1.Off` and `FAN2.Off`.

Your chart should now look like this:



Tip The location of the tail of a default transition determines the state it activates. Therefore, you must make sure that your default transition fits completely inside the parent of the state that it activates. In the Air Controller chart pictured above, notice that the default transition for `FAN1.Off` correctly resides inside the parent state, `FAN1`. Now consider this chart:



In this example, the tail of the default transition resides in `PowerOn`, not in `FAN1`. Therefore, it will activate `FAN1` instead of `FAN1.Off`.

6 Save `Stage4Transitions`, but leave the Stateflow Editor open for the next exercise.

Adding Conditions to Guard Transitions

Conditions are expressions enclosed in square brackets that evaluate to true or false. When the condition is true, the transition is taken to the destination state; when the condition is false, the transition is not taken and the state of origin remains active.

As you learned in “Guarding the Transitions” on page 6-3, the fans cycle on and off depending on the air temperature. In this exercise, you will add conditions to the transitions in `FAN1` and `FAN2` that model this behavior.

Follow these steps:

1. Click the transition from `FAN1.Off` to `FAN1.On`.

   The transition appears highlighted and displays a question mark (`?`).

2. Click next to the question mark to display a blinking text cursor.

3. Type the following expression:

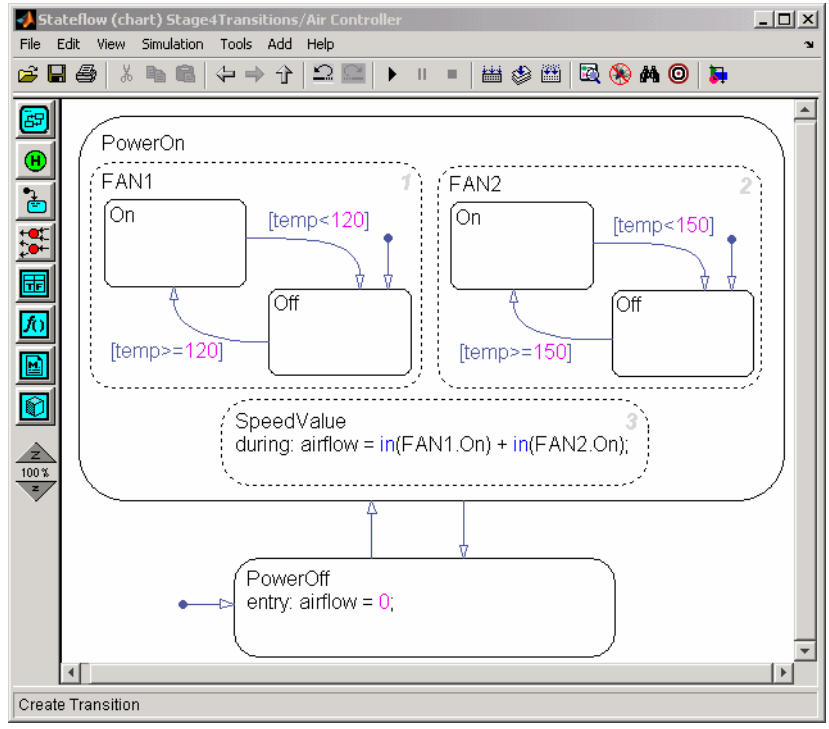
   ```
   [temp >= 120]
   ```

   You may need to reposition the condition for readability. Click outside the condition, then left-click and drag the condition expression to a new location.

4. Repeat these steps to add the following conditions to the other transitions in `FAN1` and `FAN2`:

Transition	Condition
`FAN1.On` to `FAN1.Off`	[temp < 120]
`FAN2.Off` to `FAN2.On`	[temp >= 150]
`FAN2.On` to `FAN2.Off`	[temp < 150]

Your chart should look like this:



5 Save `Stage4Transitions`, but leave the Stateflow Editor open for the next exercise.

Adding Events to Guard Transitions

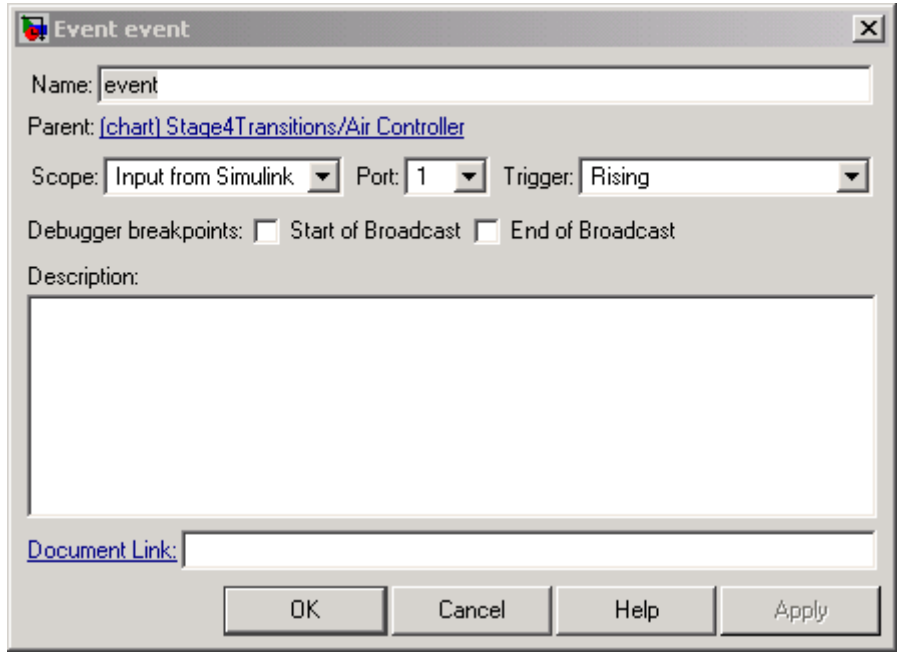
Events are nongraphical objects that trigger activities during the execution of a Stateflow chart. Depending on where and how events are defined, they can trigger a transition to occur, an action to be executed, and state status to be evaluated. In this exercise, you will define an event that triggers transitions.

As you learned in "Guarding the Transitions" on page 6-3, the control system should power on and off at regular intervals. You model this behavior by first defining an event that occurs at the rising or falling edge of an input signal, and then associating that event with the transitions between the `PowerOn` and `PowerOff` states.

Follow these steps to define an edge-triggered event and associate it with the transitions:

1 In the Stateflow Editor, add an input event by selecting **Event > Input from Simulink** from the **Add** menu.

The Event properties dialog box opens on your desktop:



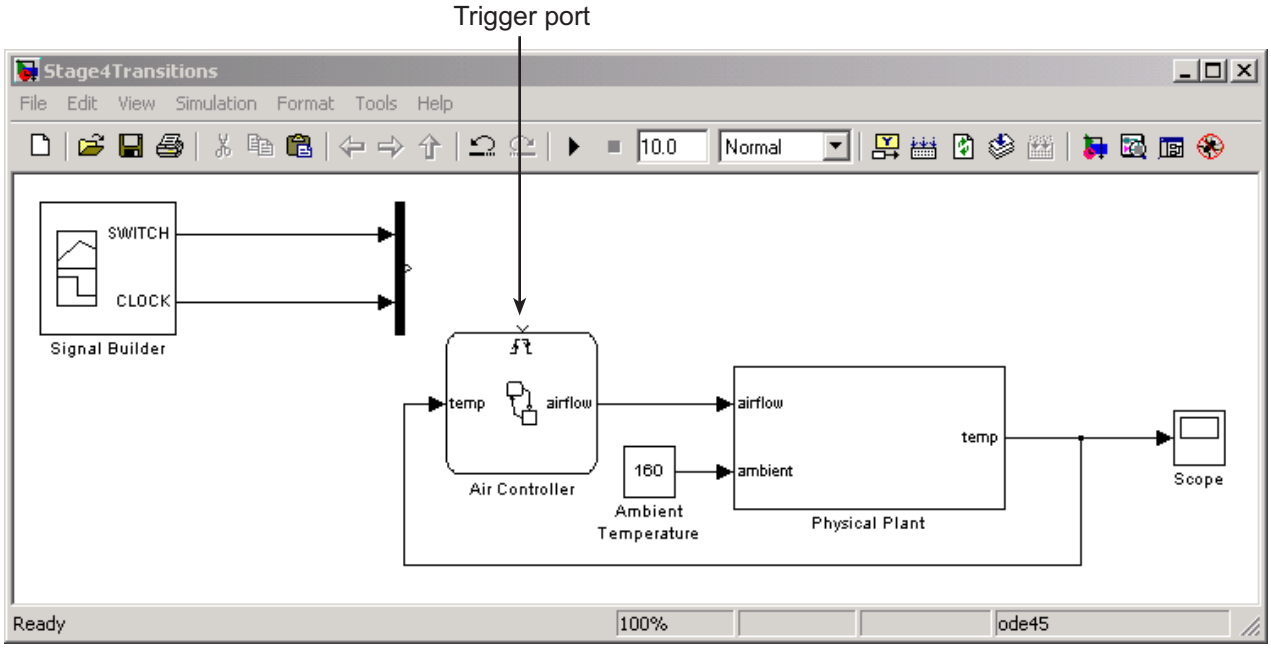
Note that the event is assigned to trigger port 1.

2 Edit the following properties:

Property	What to Specify
Name	Change the name to `SWITCH`.
Trigger	Select `Either` from the drop-down menu so the event can be triggered by either the rising edge or falling edge of a signal.

3 Click **OK** to record the changes and close the dialog box.

4 Look back at the Simulink model and notice that a trigger port appears at the top of the Stateflow block:



When you define one or more input events for a chart, Stateflow adds a single trigger port to the block. External Simulink blocks can trigger the input events via a signal or vector of signals connected to the trigger port.

5 Back in the Stateflow Editor, associate the input event `SWITCH` with the transitions:

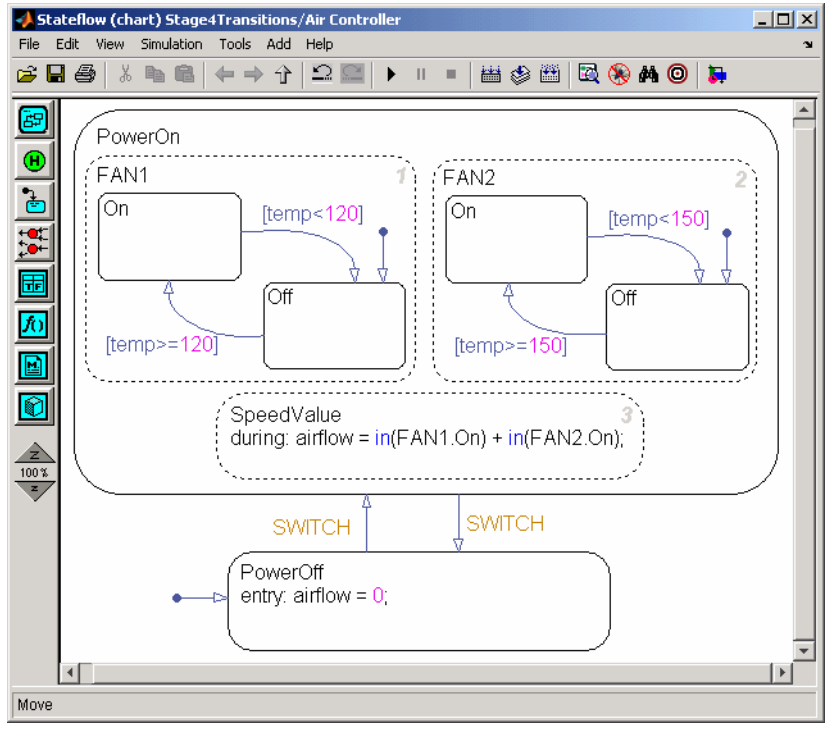
a Select the transition from `PowerOff` to `PowerOn` and click the question mark to get a text cursor.

b Type the name of the event you just defined, `SWITCH`.

You may need to reposition the event text for readability. If so, click outside the text string, left-click the text, and drag it to the desired location.

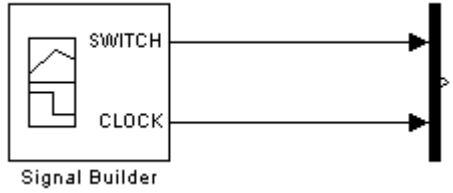
c Repeat these steps to add the same event, `SWITCH`, to the transition from `PowerOn` to `PowerOff`.

Your chart should now look something like this:



Now that you have associated these transitions with the event `SWITCH`, the control system will alternately power on and off every time `SWITCH` occurs — that is, every time the chart detects a rising or falling signal edge.

Note that the Simulink `sf_aircontrol` model has already defined the pulse signal `SWITCH` in the Signal Builder block at the top level of the model hierarchy:

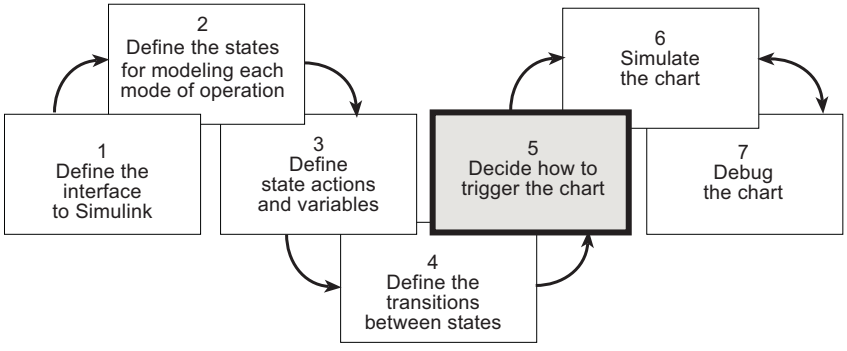


In the next phase of the workflow, you will connect your Stateflow chart to the `SWITCH` signal to trigger the transitions between power on and power off.

6 Save `Stage4Transitions`.

Where to go next. Now you are ready to begin phase 5 of the workflow: Chapter 7, “Triggering a Stateflow Chart”.

Triggering a Stateflow Chart



You have entered phase 5 of a basic workflow for building a Stateflow chart: *deciding how to trigger the chart*. This chapter presents the design questions that you must answer and guides you through exercises for adding triggers, based on your design decisions.

Design Considerations for Triggering Stateflow Charts (p. 7-2)	Examines the design considerations for phase 5 of the workflow and presents the rationale for the solutions
Implementing the Triggers (p. 7-3)	Provides hands-on exercises that show you how to specify the triggers as designed

Design Considerations for Triggering Stateflow Charts

Simulink can wake up a Stateflow chart by

- Sampling the chart at a specified or inherited rate
- Using a signal as a trigger
- Using one Stateflow chart to drive the activity of another

A signal trigger works best for the Air Controller chart because it needs to monitor the temperature of the physical plant at regular intervals. To meet this requirement, you will use a periodic signal to trigger the chart. The source is a square wave signal called `CLOCK`, provided by a Signal Builder block in the Simulink model, described in "How the Stateflow Chart Works with the Simulink Model" on page 2-6. To harness the signal, you will set up an edge trigger event in Stateflow that wakes the chart at the rising or falling edge of `CLOCK`.

The rationale for using an edge trigger in this case is that it uses the regularity and frequency of the signal to wake up the chart. When using edge triggers, it is important to note that there can be a delay from the time the trigger occurs to the time the chart begins executing. This is because an edge trigger causes the chart to execute at the beginning of the next simulation time step, regardless of when the edge trigger actually occurred during the previous time step. The Air Controller can tolerate this delay, as long as the edge occurs frequently enough. (For more information about triggering Stateflow charts, see "Implementing Simulink Update Interfaces" in the online Stateflow User's Guide documentation.)

Recall that you already defined one edge-triggered event, `SWITCH`, to guard the transitions between `PowerOff` and `PowerOn`. You will now define a second edge-triggered event, `CLOCK`, to wake up the chart.

Implementing the Triggers

To implement the trigger as defined, you need to perform the following tasks:

Defining the CLOCK Event

To define the `CLOCK` event, follow these steps:

1. Open the model `Stage4Transitions` and save it as `Stage5Trigger` in the same directory.
2. In `Stage5Trigger`, double-click the Air Controller block to open the Stateflow chart.

   The Stateflow Editor for Air Controller opens on your desktop.
3. In the Stateflow Editor, add an input event by selecting **Event > Input from Simulink** from the **Add** menu.
4. In the Event properties dialog box, edit the following fields:

Property	What to Specify
Name	Change the name to `CLOCK`.
Trigger	Select `Either` from the drop-down menu so the event can be triggered by either the rising edge or falling edge of a signal.

   Because the `SWITCH` event you created in "Adding Events to Guard Transitions" on page 6-13 was assigned to trigger port 1, the `CLOCK` event is assigned to trigger port 2. Nevertheless, only one trigger port appears at the top of the Air Controller block to receive trigger signals. This means that each signal must be indexed into an array, as described in "Connecting the Edge-Triggered Events to the Input Signals" on page 7-4.
5. Click **OK** to record the changes and close the dialog box.

6 Save `Stage5Trigger`, but leave it open for the next exercise.

Connecting the Edge-Triggered Events to the Input Signals

You need to connect the edge-triggered events to the Simulink input signals in a way that

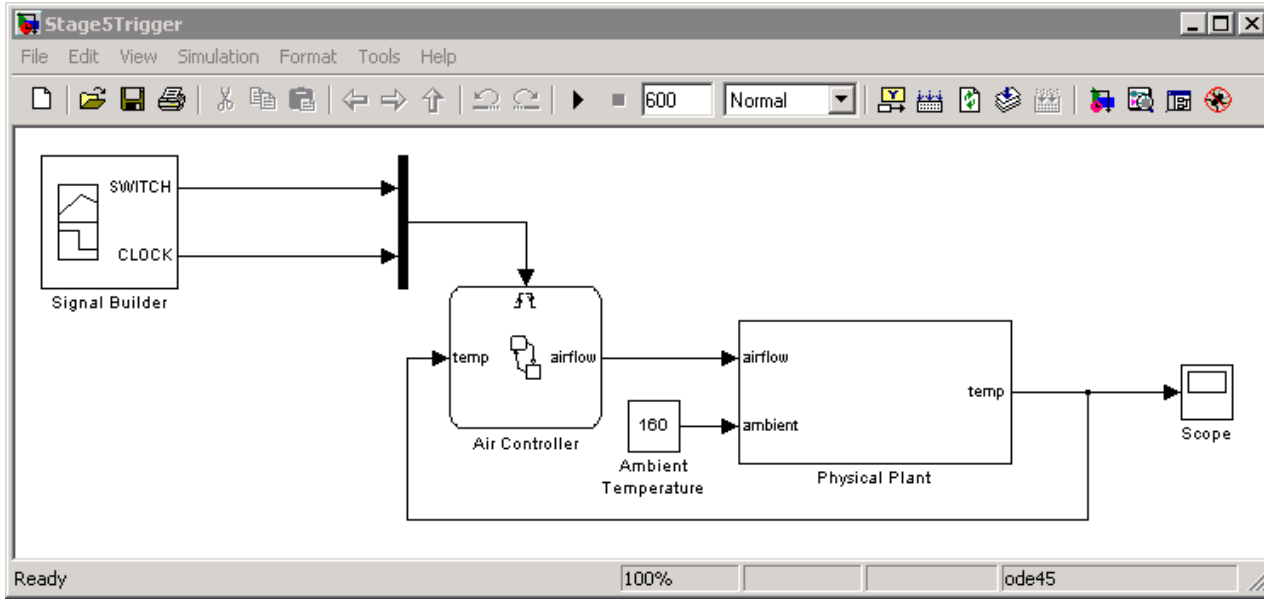
- Associates each event with the correct signal
- Indexes each signal into an array that can be received by the Air Controller trigger port

In `Stage5Trigger`, notice that the two input signals `SWITCH` and `CLOCK` feed into a Mux block where they are joined in an array to a single output. `SWITCH` is a pulse signal and `CLOCK` is a square wave. When you connect the Mux to the trigger port, the index of the signals in the array are associated with the like-numbered ports. Therefore, the `SWITCH` signal at the top input port of the Mux triggers the event `SWITCH` on trigger port 1. Likewise, the `CLOCK` signal at the second input port of the Mux triggers the event `CLOCK` on trigger port 2.

To connect the Mux to the trigger port, follow these steps:

1 Click the Mux block, hold down the **Ctrl** key, and click the Air Controller block.

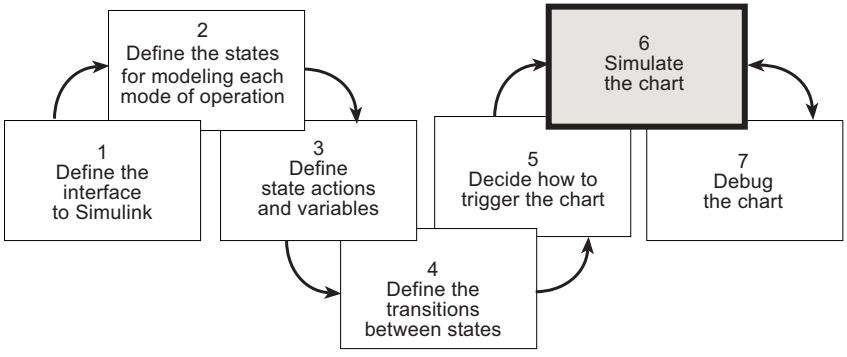
The output signal of the Mux block connects to the input trigger port of the Stateflow block. Your chart should look like this:



2 Save `Stage5Trigger`.

Where to go next. Now you are ready to begin phase 6 of the workflow: Chapter 8, “Simulating the Chart”.

Simulating the Chart



You have entered phase 6 of a basic workflow for building a Stateflow chart: *simulating the chart*. By the time you reach this phase, you have finished building your chart and integrating it with the Simulink model. Now it is time to test the chart by simulating its run-time behavior. During simulation, you can *animate* Stateflow charts to highlight states and transitions as they execute. This chapter guides you through the tasks required for simulating the chart with animation.

Preparing Charts for Simulation (p. 8-3)	Describes common design problems to check for in your Stateflow chart to minimize errors during simulation
Setting Simulation Parameters (p. 8-4)	Shows how to set parameters to control simulation
Animating Stateflow Charts (p. 8-6)	Describes how to animate Stateflow charts during simulation

Setting Breakpoints (p. 8-9)	Shows how to set breakpoints and examine data in the debugger so you can observe key run-time behaviors of your chart during simulation
Simulating the Air Controller Chart (p. 8-11)	Describes how to simulate the Air Controller chart and examines the run-time behavior

Preparing Charts for Simulation

Before starting a simulation session, you should examine your chart to ensure that it conforms to recommended design practices:

- There must be a default transition at every level of the Stateflow hierarchy that contains exclusive (OR) states (has exclusive [OR] decomposition). (See "Placing Default Transitions" on page 6-3.)
- Whenever possible, input data objects *should* inherit properties from the associated input signal in Simulink to ensure consistency, minimize data entry, and simplify maintenance of your model. Recall that in "Defining the Inputs and Outputs" on page 3-10, you defined the input `temp` to inherit its size and type from the Simulink output port `temp`, which provides the input value to the Air Controller chart.
- Output data objects *should not* inherit types and sizes because the values are back propagated from Simulink and may, therefore, be unpredictable. Recall that in "Defining the Inputs and Outputs" on page 3-10, you specified the data type as `uint8` and the size as scalar (the default). (See "Guidelines for Inheriting Data and Event Properties" in the online Stateflow User's Guide documentation.)

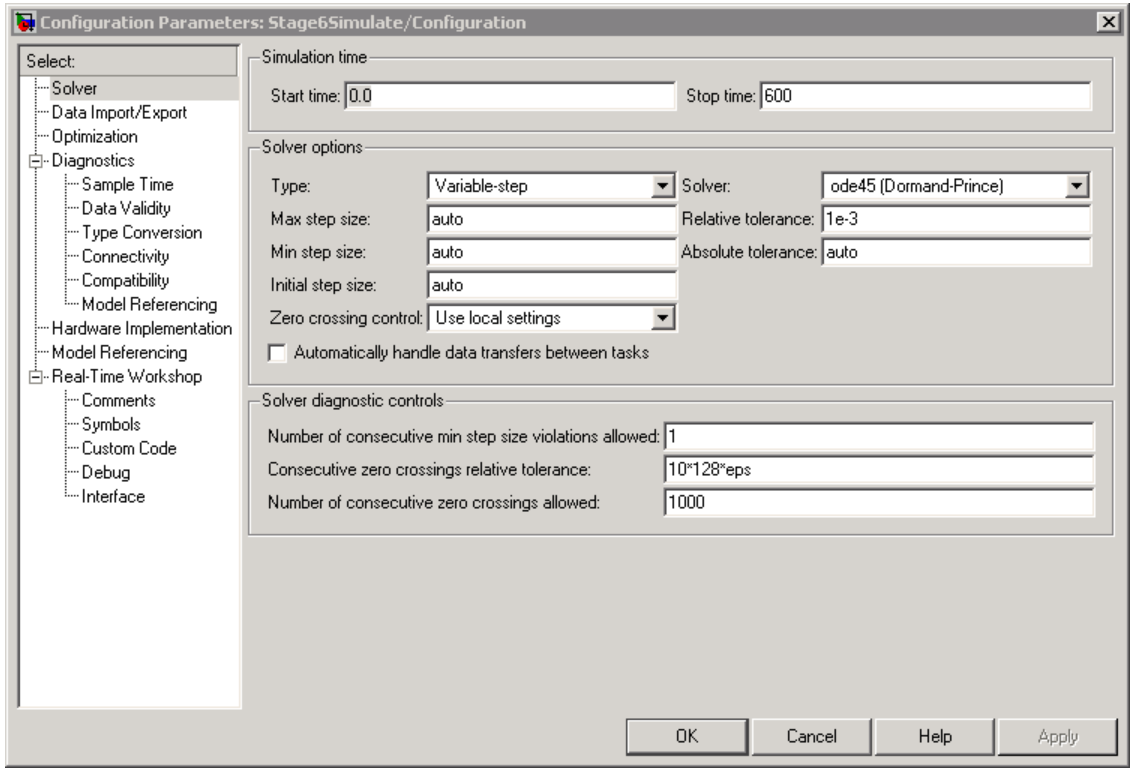
Tip You can specify data types and sizes as expressions in which you call functions that return property values of other variables already defined in Stateflow, MATLAB, or Simulink. Such functions include `size`, `type`, and `fixdt`. For more information, see "Entering Expressions and Parameters for Data Properties" in the online Stateflow User's Guide documentation.

Setting Simulation Parameters

To set simulation parameters, follow these steps:

1. Open the model `Stage5Trigger` and save it as `Stage6Simulate` in the same directory.
2. Double-click Air Controller to open the Stateflow chart.
3. Check the settings for simulation time, as follows:
   a. In the Stateflow Editor, select **Configuration Parameters** from the **Simulation** menu.

      The Configuration Parameters dialog box opens on your desktop:



b Click **Solver** in the left **Select** pane if it not already selected.

Under **Simulation time** on the right, note that the start and stop times have been preset for you. You can adjust these times later as you become more familiar with the run-time behavior of the Stateflow chart.

c Keep the preset values for now and click **OK** to close the dialog box.

4 Leave the Stateflow chart open for the next exercise.

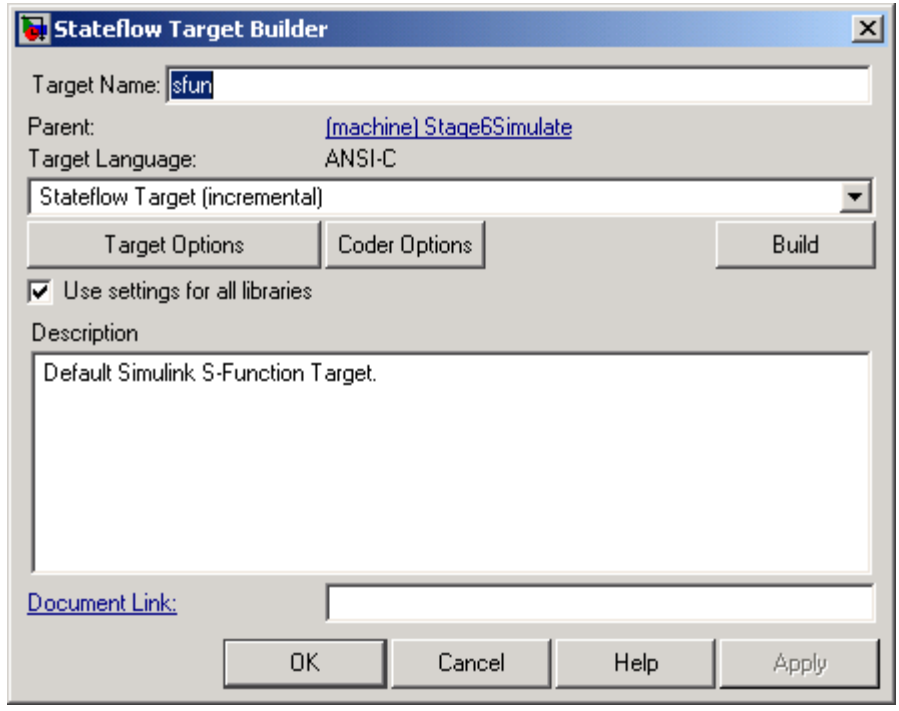
Animating Stateflow Charts

When you simulate a Simulink model, you can animate Stateflow charts to highlight states and transitions as they execute. Animation provides visual verification that your chart behaves as expected. Animation is enabled by default, but you need to set the speed. To configure animation for your simulation session, follow these steps:

1 Make sure animation has been enabled for your chart, as follows:

   a In the Stateflow Editor, select **Open Simulation Target** from the **Tools** menu.

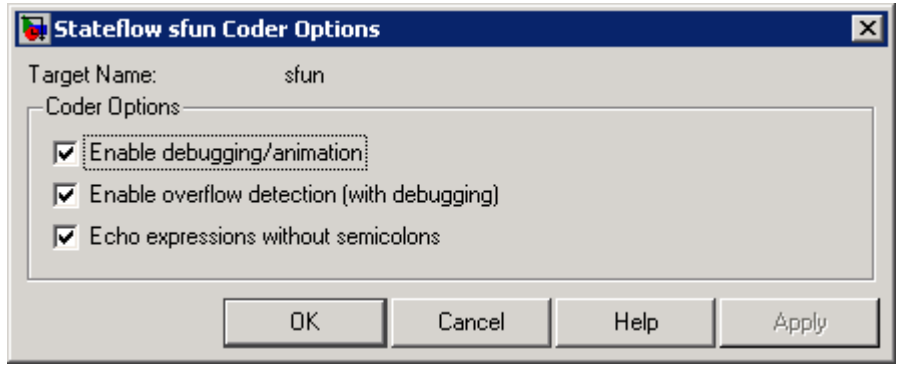
   The Stateflow Target Builder dialog box opens on your desktop.



Note This dialog box is used to configure Stateflow for building targets. A target is a program that executes a Stateflow chart or a Simulink model that contains a Stateflow chart. Stateflow builds a simulation target (`sfun`) that lets you simulate your Stateflow application in Simulink. For more information, see "Building Targets" in the online Stateflow User's Guide documentation.

b Click the **Coder Options** button in the middle of the dialog box.

The Stateflow `sfun` Coder Options dialog box opens on your desktop:



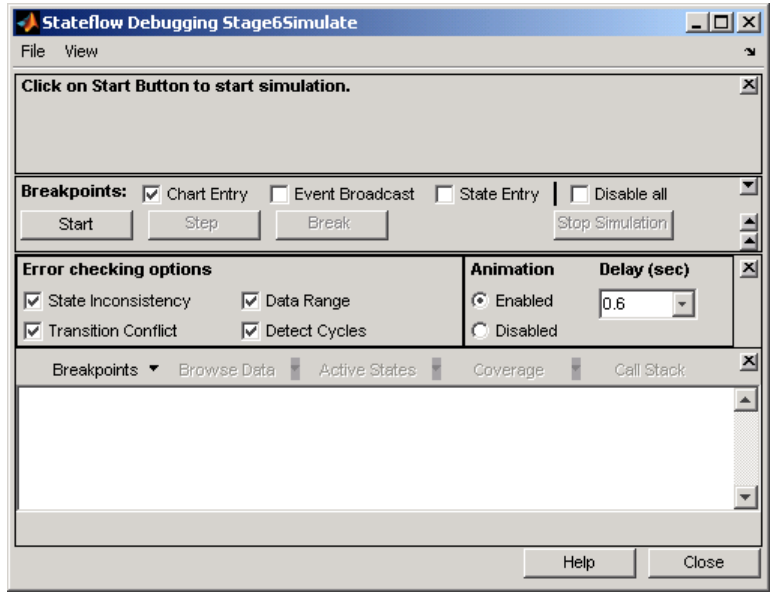
Note that **Enable debugging/animation** is checked.

c Close both dialog boxes.

2 Set the speed of animation, as follows:

a From the Stateflow Editor, open the Stateflow debugger by selecting **Debug** from the **Tools** menu or clicking the Debug icon:

The Stateflow debugger opens on your desktop:



By default, animation is enabled at 0.6-second delay.

b Change the delay to 1 second so the animation will proceed at the slowest speed.

Note You can change the speed of animation at any time during simulation

---

3 Leave the Air Controller chart and the debugger open for the next exercise.

Setting Breakpoints

In this exercise, you will learn how to set breakpoints in the debugger to pause simulation during key run-time activities so you can observe the behavior of your chart in slow motion. You can set the following breakpoints:

Breakpoint	Description
Chart Entry	Simulation halts when the Stateflow chart wakes up.
Event Broadcast	Simulation halts when an event, such as `SWITCH` or `CLOCK`, occurs.
State Entry	Simulation halts when a state becomes active.

You will also learn how to examine data values when simulation pauses.

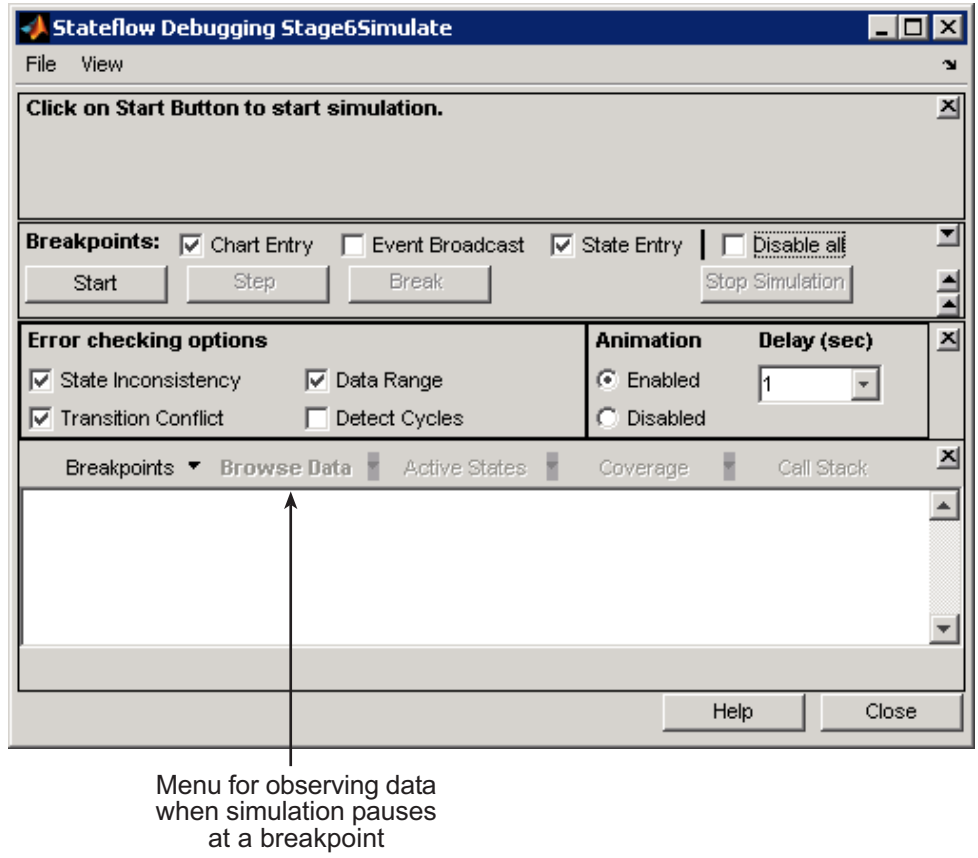
Follow these steps:

1 In the debugger, select **Chart Entry** and **State Entry** as breakpoints.

Note If you also set breakpoints at each event broadcast, simulation would pause at every rising or falling edge of the `SWITCH` and `CLOCK` signals. To keep simulation running at a reasonable pace, leave **Event Broadcast** unchecked.

---

2 Notice the **Browse Data** option in the menu bar just above the output display pane of the debugger:



Note The **Browse Data** option appears grayed out, but becomes active when simulation pauses at a breakpoint. You will use this option in "Simulating the Air Controller Chart" on page 8-11.

3 Leave the debugger open for the next exercise.

Simulating the Air Controller Chart

In this exercise, you will simulate the Air Controller chart. If this is the first time you are simulating the chart in the `Stage6Simulate` model, Stateflow builds the simulation target by performing the following actions before simulation actually starts:

- Parses the chart for state inconsistency errors, like those mentioned in "Preparing Charts for Simulation" on page 8-3.
- Generates C code that represents the behavior of the chart
- Builds the generated code into an executable program for the simulation target, called an `sfun` target
- Creates a directory called `sfprj` in the directory where the chart resides to store the generated files that make up the `sfun` target
- Creates a MEX (MATLAB executable) file that corresponds to the C source file

The MATLAB command line displays status messages during each of these processes. You should see the following messages in your MATLAB Command Window, indicating a successful build:

```
Stateflow parsing for model "Stage6Simulate"...Done
Stateflow code generation for model "Stage6Simulate"....Done
Stateflow compilation for model "Stage6Simulate"...        1 file(s) copied.
Done
>>
```

For more information, see "Building Targets" in the Stateflow User's Guide.

During simulation, you will change breakpoints and observe data values when execution pauses. Follow these steps:

1. In `Stage6Simulate`, open the Scope block. Position the Scope block, Air Controller chart, and debugger so all are visible on your desktop.
2. In the debugger, start simulation by clicking the **Start** button.

After Stateflow finishes building the simulation target, the Stateflow chart appears with a gray background, indicating that simulation has begun. Simulation continues until it reaches the first breakpoint, when the Air Controller chart wakes up. Notice that the status panel at the top of the debugger provides a snapshot of simulation activities at the breakpoint.

Detail	What It Means	What You See at First Breakpoint
Stopped	What executed at the breakpoint	Entry: Chart Air Controller
Executing	Stateflow chart that is executing	Air Controller
Current Event	Event that is processed at this time step	Input event `SWITCH`
Simulink Time	Time at which the simulation paused	0.000000

Note also that the **Browse Data** option is now enabled.

3 Click the down arrow to the right of the **Browse Data** option and select **Watched Data (Current Chart)** from the submenu.

By selecting this option, you will be able to examine the values of the input `temp` and output `airflow`. Recall that you configured these objects as data to be watched in the debugger in the exercise "Defining the Inputs and Outputs" on page 3-10.

Tip You can also view data values from the MATLAB command line at simulation breakpoints. Here's how to do it:

a When simulation pauses at a breakpoint, click in the MATLAB command line and press the **Enter** key.

MATLAB displays a `debug>>` prompt.

b At the prompt, type the name of the data object.

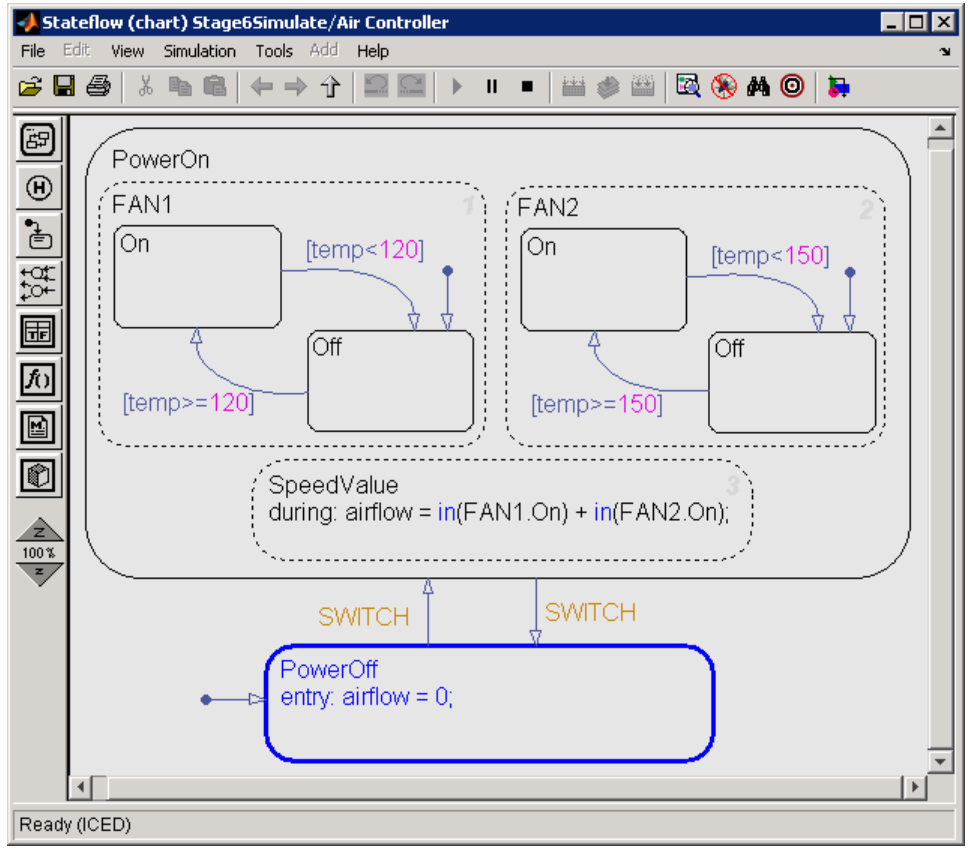
MATLAB displays the value of the data object.

4 Scroll down in the output display pane of the debugger to view the values of `temp` and `airflow`.

Note that `temp` = 70 (below the threshold for turning on `FAN1`) and `airflow` = 0 (indicating that no fans are running).

5 Resume simulation by clicking the **Continue** button.

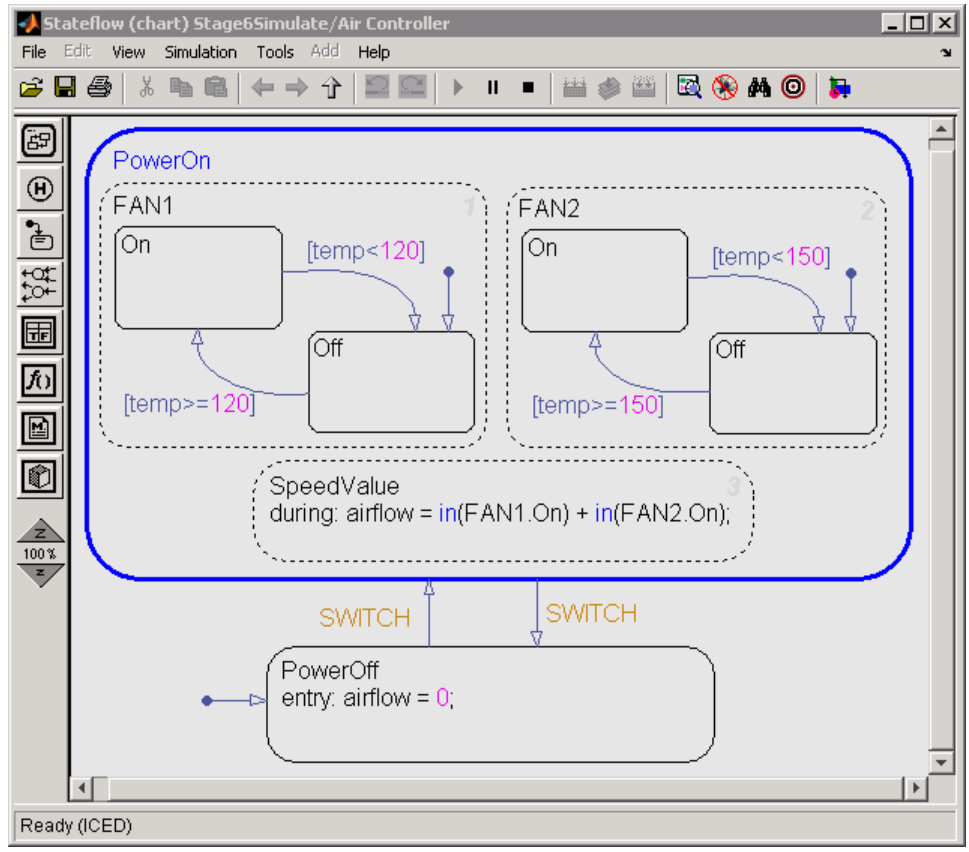
Simulation continues until the next breakpoint, activation of the `PowerOff` state, which appears highlighted in the Stateflow chart (as part of animation):



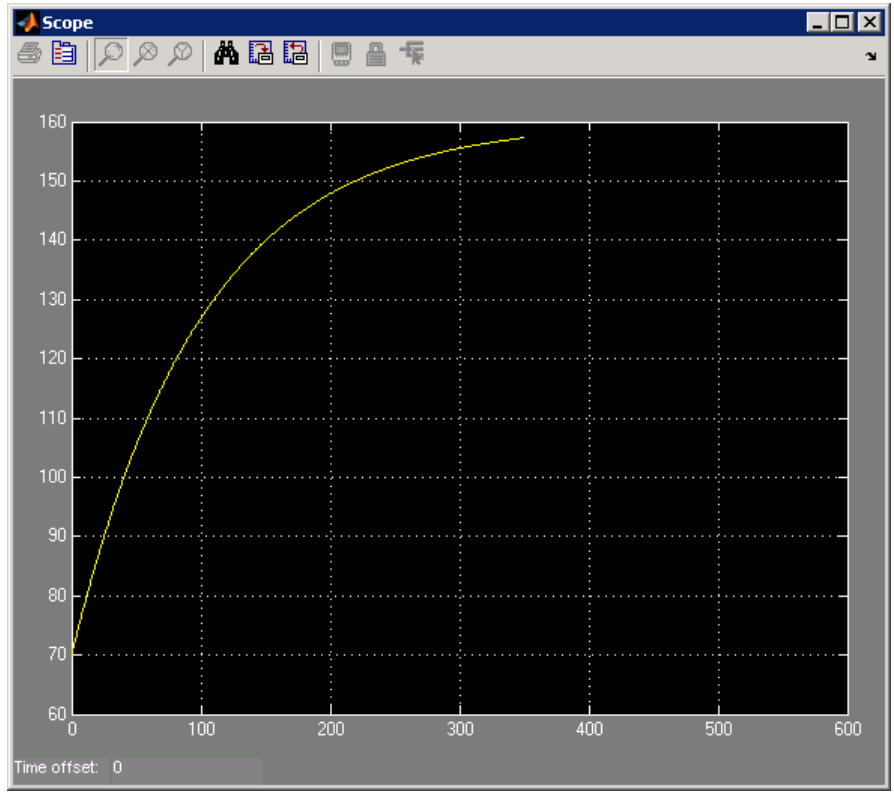
The default transition activates `PowerOff` after the chart wakes up.

6 Uncheck the breakpoint **Chart Entry** and continue simulation.

Simulation continues to the next breakpoint, the activation of the `PowerOn` state:



Note in the output display pane of the debugger that `temp` has risen to over 157 degrees. The Scope displays the temperature pattern:



7 Continue simulation through the following breakpoints, noting chart animation, Scope display, and how data values change:

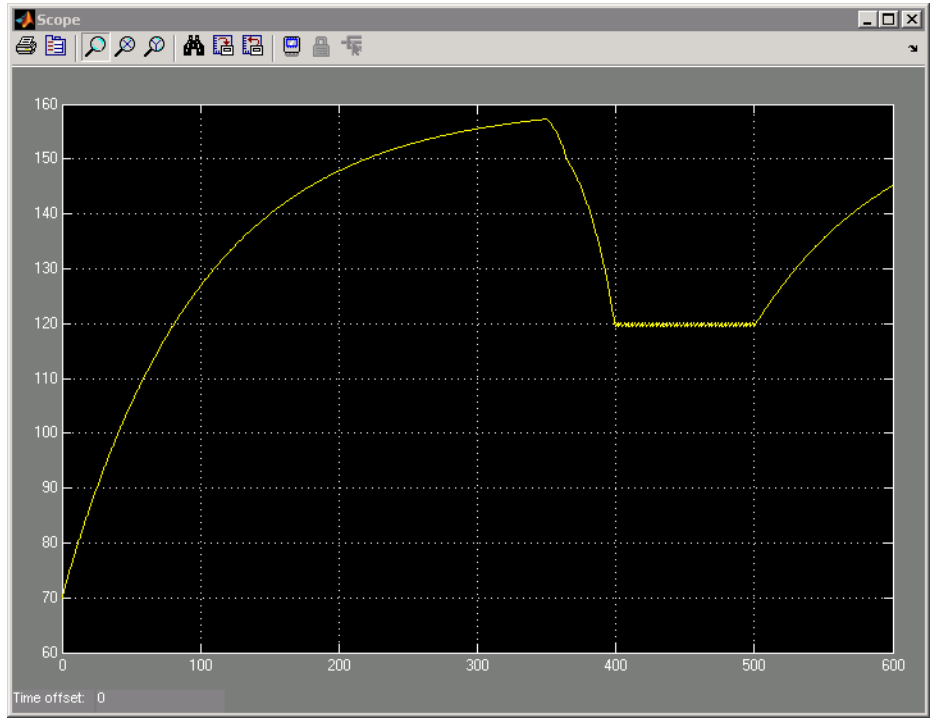
Breakpoint	Value of temp (Degrees)	Value of airflow
Activation of `FAN1`	> 157	0
Default transition to `FAN1.Off`	> 157	0
Activation of `FAN2`	> 157	0

Breakpoint	Value of temp (Degrees)	Value of airflow
Default transition to `FAN2.Off`	> 157	0
Activation of `SpeedValue`	> 157	0
Transition to `FAN1.On` (*because* `temp` >= *120 degrees*)	> 157	0
Transition to `FAN2.On` (*because* `temp` >= *150 degrees*)	> 157	0
Transition to `FAN2.Off` (*because* `temp` < *150 degrees*)	> 149 and < 150	2
Transition to `FAN1.Off` (*because* `temp` < *120 degrees*)	> 119 and < 120	1
Transition to `FAN1.On` (*because* `temp` >= *120 degrees*)	> 120	0
Transition to `FAN1.Off` (*because* `temp` < *120 degrees*)	> 119 and < 120	1

8 To speed through the rest of the simulation, uncheck all breakpoints, change animation delay to **0**, and click **Continue**.

Notice that `FAN1` continues to cycle on and off as `temp` fluctuates between 119 and 120 degrees until power cycles off at 500 seconds. After power cycles off, the fans stop running and `temp` begins to rise unchecked until simulation reaches stop time at 600 seconds.

The Scope captures this activity:

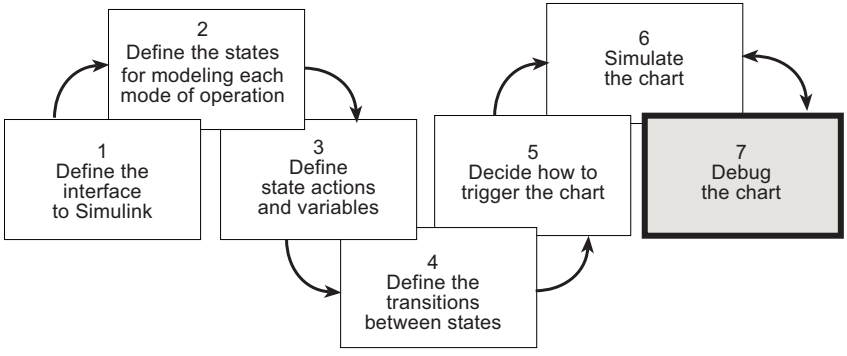


Note This display should look the same as the Scope after running the prebuilt model in "Running the Model" on page 2-11.

9 Save `Stage6Simulate`, and close all other windows and dialogs.

Where to go next. Now you are ready to begin phase 7 of the workflow: Chapter 9, "Debugging the Chart".

Debugging the Chart



You have entered phase 7 of a basic workflow for building a Stateflow chart: *debugging the chart*. In Chapter 8, "Simulating the Chart", you learned how to use the debugger for setting breakpoints and observing data. In this chapter, you will learn how Stateflow detects errors and provides diagnostic assistance.

Debugging State Inconsistencies (p. 9-2)	Shows you how to debug a state inconsistency error
Debugging Data Range Violations (p. 9-6)	Shows you how to debug errors caused by data values going out of range

Debugging State Inconsistencies

In this exercise, you will introduce a state inconsistency error in your Stateflow chart and troubleshoot the problem. Follow these steps:

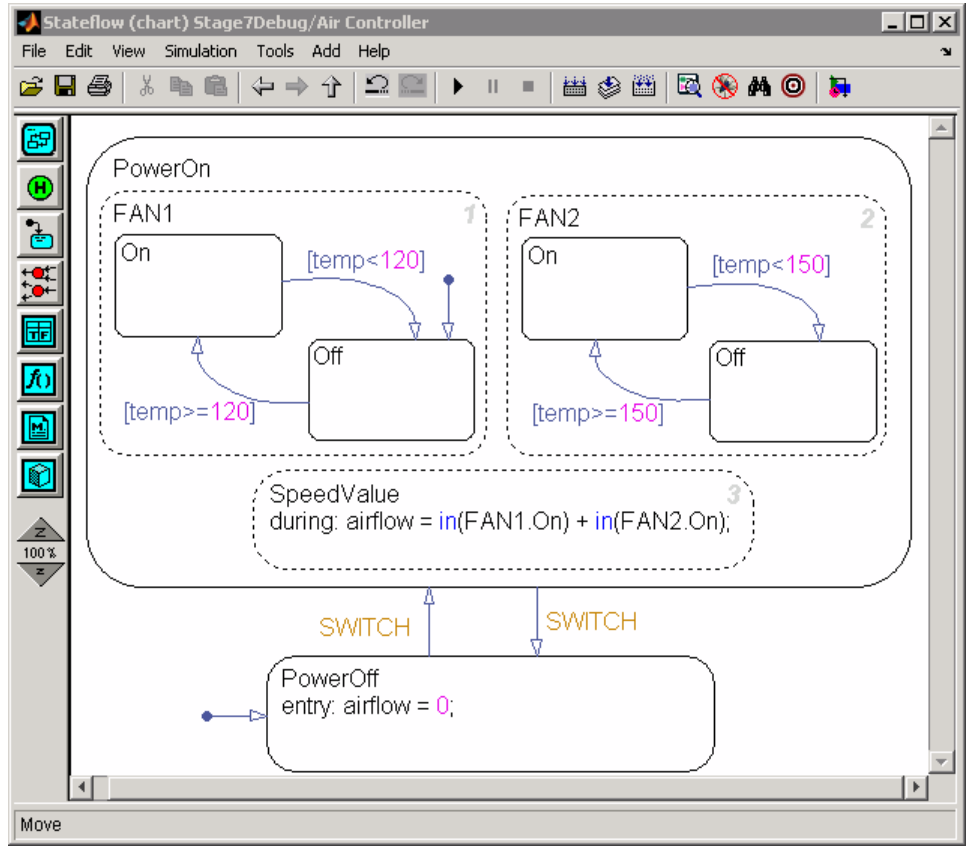
1. Open the model `Stage6Simulate` and save it as `Stage7Debug` in the same directory.

2. Double-click Air Controller to open the Stateflow chart.

3. Delete the default transition to `FAN2.Off` by selecting it and pressing the **Delete** key.

   Removing the default transition will cause a state inconsistency error. (Recall from "Preparing Charts for Simulation" on page 8-3 that there must be a default transition at every level of the Stateflow hierarchy that has exclusive [OR] decomposition.)

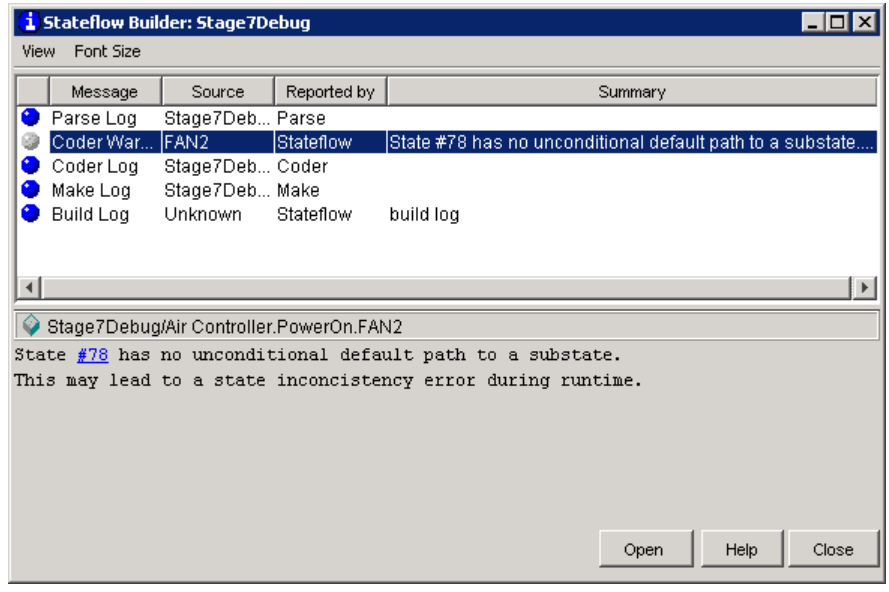
Your Stateflow chart should look like this:



4 Open the Stateflow debugger and make sure **State Inconsistency** is enabled in the **Error checking options** panel.

5 Save the chart, and then build it by clicking the **Build** icon:

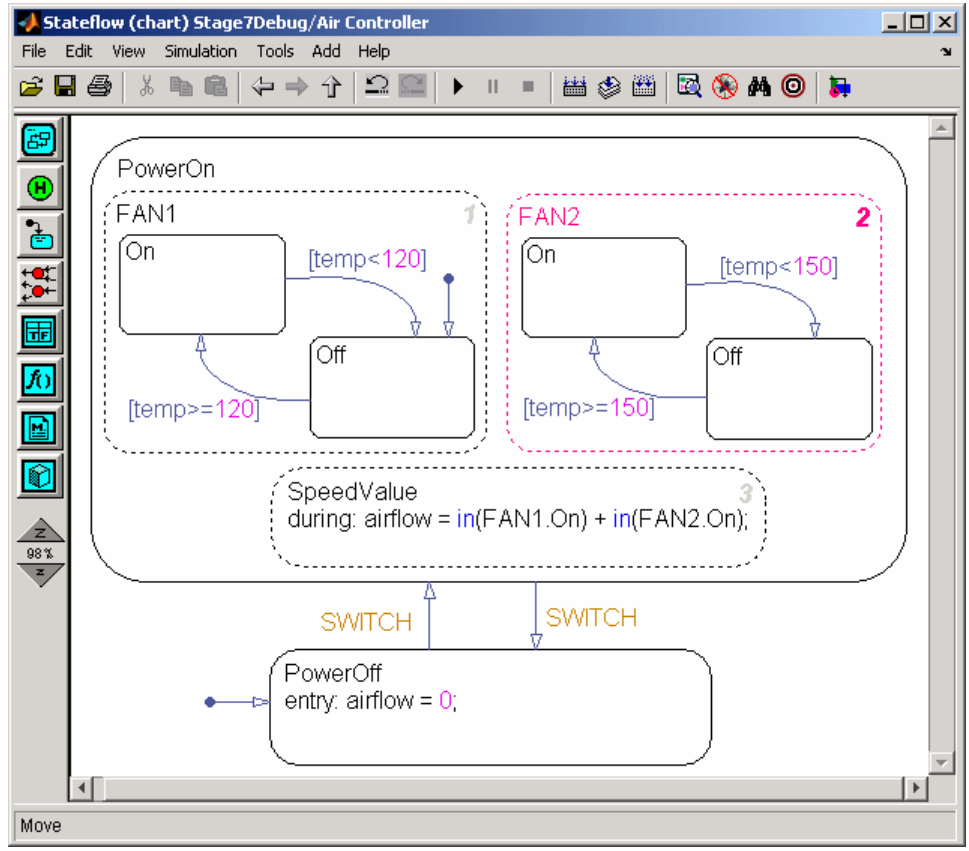
Stateflow generates a coder warning highlighted with a gray bullet. The warning indicates that a state (identified by number) has no unconditional path to a substate and that the source of the problem is FAN2:



Note The state number in your dialog display may differ from the one pictured above.

6 Locate the offending state in the Air Controller chart, either by double-clicking the coder warning text or clicking the link to the state number in the status panel at the bottom of the dialog box.

Stateflow highlights `FAN2` in the chart:



7 Add back the default transition to `FAN2.Off`.

The default transition provides the unconditional path to one of the substates of `FAN2`.

8 Build the chart again.

This time the chart builds successfully without parser or code generation errors.

9 Save `Stage7Debug`, and leave Air Controller open for the next exercise.

Debugging Data Range Violations

In this exercise, you will introduce a data range violation in your Stateflow chart and use the debugger to troubleshoot the problem. Follow these steps:

1. In the Air Controller chart, modify the `during` action in the `SpeedValue` state by adding 1 to the computed value, as follows:

   ```
   during: airflow = in(FAN1.On) + in(FAN2.On) + 1;
   ```

   Recall that in "Defining the Inputs and Outputs" on page 3-10, you set a limit range of 0 to 2 for `airflow`. By adding 1 to the computation, the value of `airflow` will exceed the upper limit of this range when two fans are running.

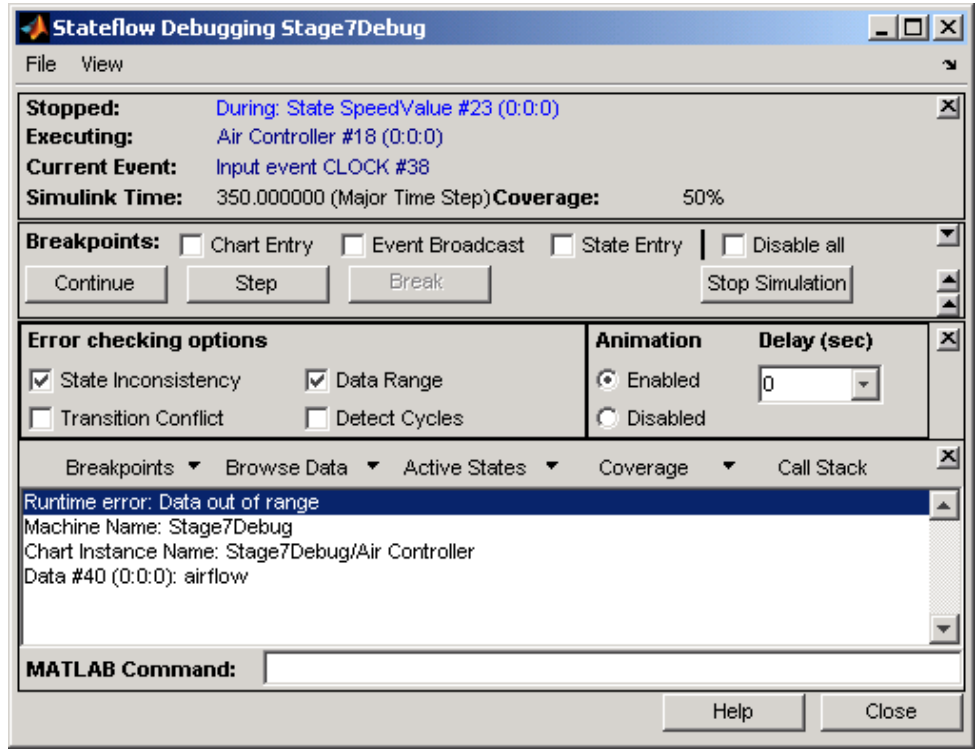
2. Open the Stateflow debugger and make the following changes:
   - Make sure **Data Range** is enabled under **Error checking options**.
   - Uncheck all breakpoints.

3. Save and build the chart.

   The chart should build with no errors or warnings.

4. Start simulation.

Simulation pauses after 350 seconds because Stateflow generates a run-time error, described in the debugger:



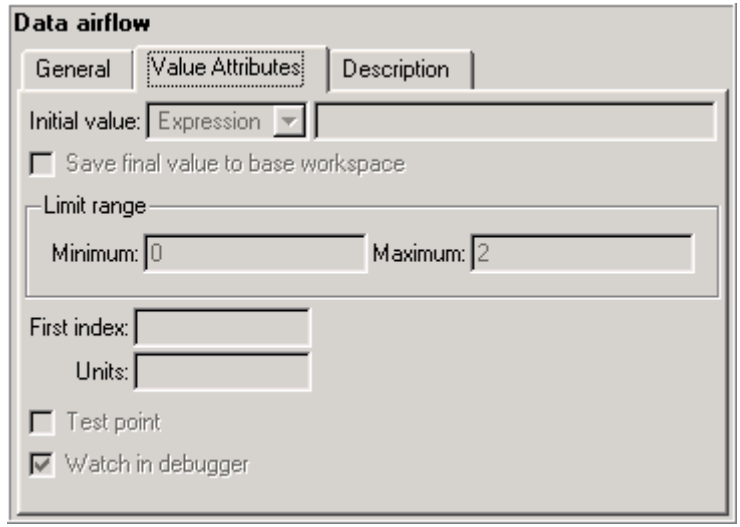
As expected, the error occurs in the `during` action of `SpeedValue` because the value of `airflow` is out of range.

5 To isolate the problem, double-click the last line in the status panel at the bottom of the dialog box:

```
Data #40 (0:0:0):  airflow
```

The Model Explorer opens on your desktop, allowing you to view the properties of airflow in the right, read-only pane (read-only because simulation is running).

6 Click the **Value Attributes** tab and check the limit range for airflow:



7 Back in the debugger, check the value of `airflow` by clicking **Browse Data > Watched Data (Current Chart)**.

```
airflow = 3
```

This value exceeds the upper limit of 2.

8 Stop simulation.

The data range error in Stateflow causes a block error to be generated by Simulink when the model tries to use `airflow` as an index into its multiport switch. At 3, the value of `airflow` exceeds the number of inputs defined for the switch and, therefore, the index is also out of range.

9 Restore the `during` action to its previous code, and then save and rebuild the model.

The model should rebuild with no errors or warnings.

Where to go next. You have completed a basic workflow for building a Stateflow chart, but there is more to learn. To gain further experience with Stateflow, explore these resources:

- **Stateflow documentation** — The MathWorks provides extensive documentation on how to work with Stateflow using the graphical user interface and the API. To access Stateflow documentation, follow these steps:

  a In the MATLAB window, select **Help** from the **Start** menu.

  b In the Help Navigator, select the Contents pane and scroll down to the Stateflow node.

  c Select the Stateflow node and follow the links to documentation resources, including real-world example models.

- **Stateflow demos** — Stateflow provides a collection of demonstration models, which you can access as follows:

  a In the MATLAB window, select **Demos** from the **Start** menu.

  The Help browser opens, displaying information on how to get started using demos.

  b In the Help Navigator, expand the Simulink node to expose the Stateflow node.

  c Expand the Stateflow node to examine demonstration models that illustrate a variety of applications.

- **Stateflow training** — The MathWorks offers classes in Stateflow, MATLAB, and Simulink. To see a list of available Stateflow courses, visit the MathWorks Web site: `www.mathworks.com/services/training`.

Index

S

T