Getting Started With CoCentric® System Studio

Version U-2003.03, March 2003

Comments?
E-mail your comments about Synopsys documentation to doc@synopsys.com

SYNOPSYS®

Printed in the U.S.A.

Document Order Number: 37471-000 PA
Getting Started With CoCentric System Studio, vU-2003.03

Contents

Figures

Preface

This preface includes the following sections:

What’s New in This Release

To see the *CoCentric System Studio Release Notes,*

1. Go to the Synopsys Web page at http://www.synopsys.com and click SolvNet.
2. If prompted, enter your user name and password. (If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.)
3. Click Release Notes in the Main Navigation section, find the U-2003.03 Release Notes, then open the *CoCentric System Studio Release Notes.*

About This Manual

This *Getting Started With CoCentric System Studio* manual is part of the CoCentric System Studio documentation set and is intended for novice-level System Studio users. Intermediate-level users will want to read the *CoCentric System Studio User Guide*. Experienced System Studio users or users who are looking for more detailed technical information should also consult the *CoCentric System Studio Reference Manual*.

The System Studio documentation suite consists of the following manuals:

- *Getting Started With CoCentric System Studio*
- *CoCentric System Studio User Guide*
- *CoCentric System Studio Reference Manual*

- *CoCentric System Studio HDL CoSim User Guide*
- *CoCentric System Studio VirSim User Guide*
- *CoCentric System Studio Model Guide*
- *CoCentric System Studio Developer Kit Guide*
- *CoCentric System Studio DSP Developer Kits User Guide*
- *CoCentric System Studio Filter Design Tool User Guide*

Audience

Getting Started with CoCentric System Studio is written for system designers and electronics engineers designing systems who use System Studio for modeling and simulation.

Related Publications

For additional information about CoCentric System Studio, see

- Synopsys Online Documentation (SOLD), which is included with the software for CD users or is available to download through the Synopsys Electronic Software Transfer (EST) system
- Documentation on the Web, which is available through SolvNet at http://solvnet.synopsys.com
- The Synopsys MediaDocs Shop, from which you can order printed copies of Synopsys documents, at http://mediadocs.synopsys.com

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
`Courier`	Indicates command syntax.
`Courier italic`	Indicates a user-defined value in Synopsys syntax, such as *`object_name`*. (A user-defined value that is not Synopsys syntax, such as a user-defined value in a Verilog or VHDL statement, is indicated by regular text font italic.)
`Courier bold`	Indicates user input—text you type verbatim—in Synopsys syntax and examples. (User input that is not Synopsys syntax, such as a user name or password you enter in a GUI, is indicated by regular text font bold.)
[ ]	Denotes optional parameters, such as *`pin1`* `[`*`pin2 ... pinN`*`]`
|	Indicates a choice among alternatives, such as `low | medium | high` (This example indicates that you can enter one of three possible values for an option: low, medium, or high.)
_	Connects terms that are read as a single term by the system, such as `set_annotated_delay`
Control-c	Indicates a keyboard combination, such as holding down the Control key and pressing c.
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy.

Customer Support

Customer support is available through SolvNet online customer support and through contacting the Synopsys Technical Support Center.

Accessing SolvNet

SolvNet includes an electronic knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. SolvNet also gives you access to a wide range of Synopsys online services including software downloads, documentation on the Web, and “Enter a Call to the Support Center.”

To access SolvNet,

1. Go to the SolvNet Web page at http://solvnet.synopsys.com.
2. If prompted, enter your user name and password. (If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.)

If you need help using SolvNet, click SolvNet Help in the Support Resources section.

Contacting the Synopsys Technical Support Center

If you have problems, questions, or suggestions, you can contact the Synopsys Technical Support Center in the following ways:

- Open a call to your local support center from the Web by going to http://solvnet.synopsys.com (Synopsys user name and password required), then clicking “Enter a Call to the Support Center.”
- Send an e-mail message to support_center@synopsys.com.
- Telephone your local support center.
  - Call (800) 245-8005 from within the continental United States.
  - Call (650) 584-4200 from Canada.
  - Find other local support center telephone numbers at http://www.synopsys.com/support/support_ctr.

1

Introduction

CoCentric System Studio is a system-level design solution consisting of tools, methodologies, and libraries that facilitates the design and simulation of systems-on-a-chip. Such systems are at the heart of advanced applications such as third-generation digital cell phones with voice, data, and video capabilities; in-car entertainment; communications and navigation systems; digital audio; and video broadcast systems.

System Studio is a member of the Synopsys CoCentric family of tools. It supports

- Design abstraction
- System-level reuse
- Hardware-software co-design

This chapter contains the following sections:

System Studio Models

System Studio offers a wide variety of modeling capabilities, providing you with the means to capture complex systems quickly and efficiently. The modeling paradigms supported can be hierarchically mixed at all levels, making System Studio an extremely versatile system modeling platform.

With System Studio, you can mix control models within dataflow models and vice versa. You can embed a data-flow model (whether it is itself a primitive or a hierarchical model) inside a state machine so that it appears to be just another state. In the same way, you can embed control inside a data-flow graph and from the outside it looks like just another data-flow block.

System Studio's models are divided into two distinct domains, algorithmic and architectural, reflecting the primary design focus that each type of model supports:

- Algorithmic Models

  These models describe the functionality of a system at an untimed level. The design is captured using a mixture of data flow and extended hierarchical state machine models.

Implementation details such as the clock and reset signals are not modeled, which gives you

- a simple and efficient modeling process
- the best possible simulation speed, which facilitates the analysis of computationally intensive signal processing algorithms
- a design space that is not over constrained during early project phases

There are several types of algorithmic models:

- DFG (Data Flow Graph) models: These are data-flow models in which the instances they contain communicate by means of FIFO (first in, first out) queues of data traveling on nets.
- OR models: These are hierarchical control models that specify a state transition diagram in which the instances they contain behave like states. Only one instance is active at a time.
- AND models: These are hierarchical control models with multiple sub-instances, called pages, that execute in parallel.
- GATED models: These are hierarchical control models consisting of one or two sub-instances, called pages, and a gating condition that controls which page is executed and which page is suspended.
- PRIM models: These are primitive (non-hierarchical) models, specified as source text. You can use PRIM models inside both data-flow and control models.

- SDS models: these are a primitive (COSSAP) Stream Driven Simulator models. You can create an SDS model in System Studio, or you can import and convert existing COSSAP models.

- Architectural Models

  Architectural (SystemC) models capture the architecture of a system at various levels of granularity and abstraction. These models allow you to describe the overall platform architecture in terms of its buses, memories, processors, and ASIC content, as well as making it possible for you to describe the internal architecture of the individual components.

  The models span a broad range from high-level abstractions suitable for early architectural analysis down to cycle-accurate and pin-accurate synthesizable hardware models.

System Studio integrates both model domains seamlessly. The same design capture and model management functionality is available for all model types. Algorithmic and architectural models can also be nested to give you the unique ability to verify your designs at all stages of the design process within a single environment.

Model Views

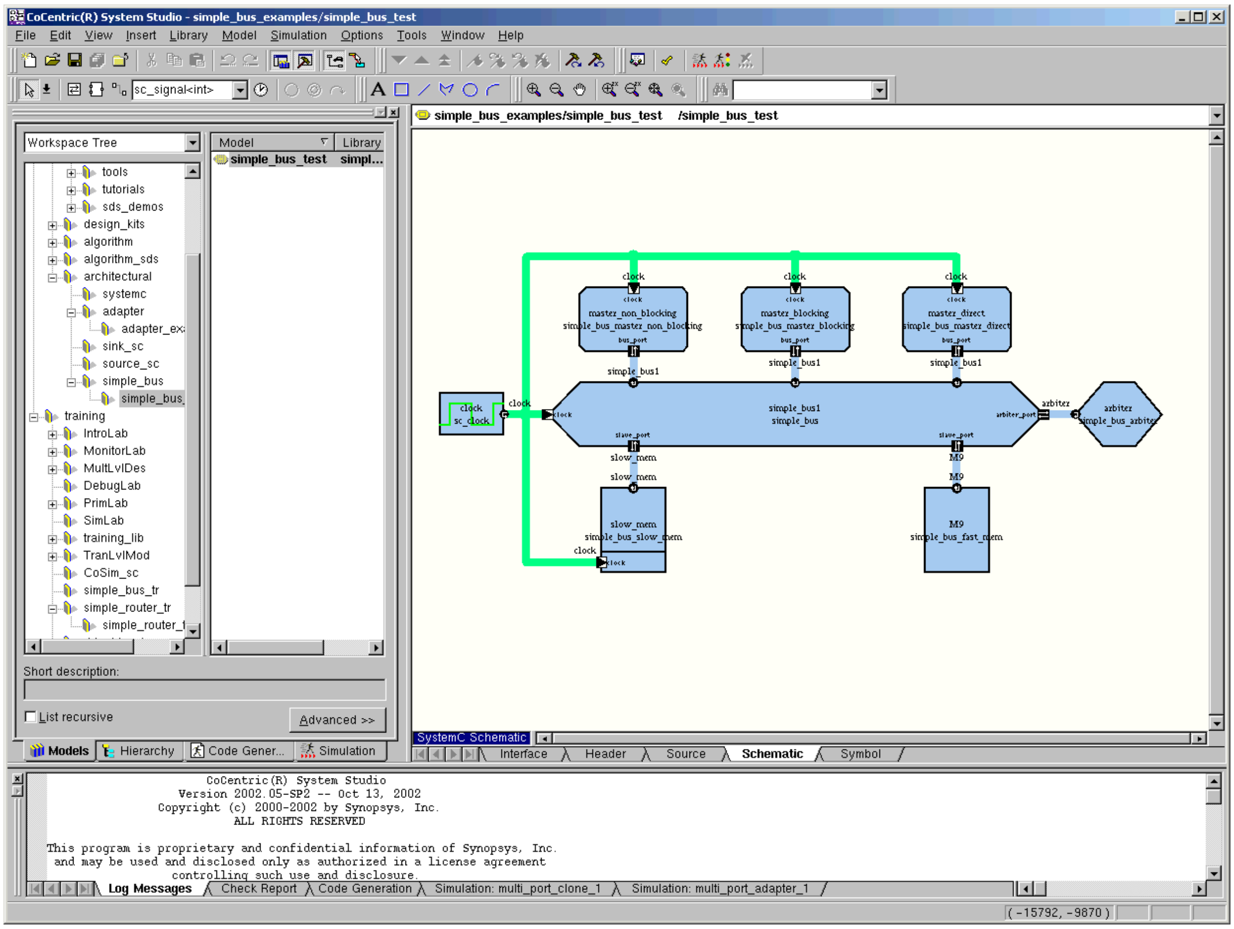
In System Studio, the implementation of the model and its interface are separate. You edit and view the implementation details, the interface, and the model symbol in separate views of the model. The views that you see depend on the type of the model:

- An Interface View gives you access to the model’s ports and parameters. You can define port directions and assign port types. Additionally, you can add a set of parameters, define types and other characteristics of a parameter, and even assign default values. A default value can be a literal or an expression.
- An Implementation View gives you access to a graphical representation of the model design (schematic) for hierarchical models, or the model source code for primitive models. System Studio allows you to browse through the complete hierarchy until you end at either a control flow graph, a primitive model, or a source code view at the leaf level. For SystemC hierarchical models this view is called the Schematic view.
- A Symbol View gives you access to the graphical representation of the model itself. The Interface and Symbol views of a model are interrelated. You create the ports in the Interface View and then you position and configure the pins in the Symbol View.
- For SystemC architectural models, a Source View gives you access to the model’s source code.
- For SystemC architectural models, a Header View gives you access to the model’s header file.

The model views are logically connected so that each change in one view causes all the other views to be updated automatically. This means that you can simply choose the editing scheme that you prefer.

You access each view individually through the System Studio Design Center (see Figure 1-1), which is a multiple-pane easy-to-use graphical user interface. You can not only view and edit the various aspects of your models, but you can also manage your design by organizing the library and model files that your System Studio workspace contains.

Figure 1-1 The System Studio Design Center



```
                CoCentric(R) System Studio
              Version 2002.05-SP2 -- Oct 13, 2002
           Copyright (c) 2000-2002 by Synopsys, Inc.
                     ALL RIGHTS RESERVED

This program is proprietary and confidential information of Synopsys, Inc.
 and may be used and disclosed only as authorized in a license agreement
                controlling such use and disclosure.
```

Model Ports

Model ports provide the external interface for a model. Ports have a data type and a direction (in, out, or inout).

Model Parameters

Model parameters have a data type and an access mode. System Studio supports the following types of access modes:

- Structural

A structural parameter resolves to a constant value during compilation and code generation.

- Read-on-reset

  A read-on-reset parameter value is refreshed (read from memory) every time the corresponding model is reset. This type of parameter can be bound to expressions that contain signals at the boundary between a control model and a dataflow model, thus enabling a control model to control the parameters of a dataflow model. This is the default.

- Dynamic

  A dynamic parameter value is refreshed (read from memory) every time it is used, and hence it is possible to poke its value from the surrounding environment. If a dynamic parameter is used in an expression that is bound to a parameter that is not marked dynamic, it will behave like a read-on-reset parameter and the values will be sampled only when the corresponding model is reset.

- Hidden

  Hidden parameters are static constants. These parameters are not visible from the outside and must have a specified value. These are equivalent to `const` variables in C++.

- Generic parameters

  To make model creation easier, System Studio permits the use of generic data types. Generic data types are parameters to a model. These parameters are denoted by the keyword `type_param` and can assume values of any of the permitted data types. An object of type `type_param` can appear wherever the corresponding specialized types (that is, the resulting type when an actual type is bound to a `type_param`) can legally appear.

Using type parameters greatly enhances the reusability of a model because they allow just one model to be used for integer, fixed-point, single-precision or double-precision floating-point variants.

Supported Data Types

System Studio supports the following rich set of native types for ports, signals, parameters, and variables:

- `bool`
- `int`
- `unsigned`
- `float`
- `double`
- `fixed`
- `sub_range`
- `bit_vector`
- `complex_double`
- `complex_float`
- `complex_fixed`
- `arrays of any type`

Two-dimensional arrays of scalar types (`bool`, `int`, `float`, `double`, `complex_float`, and `complex_double`) are also permitted. They behave like C arrays with valid indexes in the range 0 to *n*-1, where *n* is the size of the array; for example, `int[3][4]`.

There are special one-dimensional data types for `sub_range`, `fixed`, `complex_fixed`, and `bit_vector`. These data types have the suffix “_array”; for example, `bit_vector_array`.

Some data types require further specification; for example, `sub_range(3,7)`. This specifies that it can hold only the values 3, 4, 5, 6, and 7.

For full details of the data types supported in System Studio, and details of how to use your own user-defined types, see the *CoCentric System Studio Reference Manual*.

Specification Language

At the primitive model level, the description is code written in the System Studio native specification language. This feature enables you to specify the model in a C-like language, with some extensions to capture the model structure. There is full editor support for generating these extensions, which means that you need only to enter C code as text, or you can easily learn the extensions and enter the full native language directly if you prefer. In this way you can concentrate on the behavior of your algorithm.

The System Studio language is intended for the specification of untimed system behavior. This is in contrast to the SystemC modeling language, which captures timed behavior (typically at a more detailed level).

System-Level Simulation

System Studio provides a compiled simulation kernel that optimizes parts of your system model. The compiled simulation can be statically scheduled and linked with the dynamically scheduled parts of your system. System Studio contains a full COSSAP-compliant stream driven simulation engine for the accurate execution and reuse of COSSAP Stream Driven Simulator models.

System Studio has a debug mode that produces an efficient, yet fully debuggable simulation. In debug mode, you can pause or single-step the simulation, set breakpoints, and examine the state of the simulation. You can also use the DAVIS data visualization tool to monitor any stream of data.

System Studio also has an optimize mode that produces the fastest possible code.

Finally, to trade off speed against observability, you can specify that certain portions of a System Studio model will be observable for debugging purposes, while other portions will be compiled for maximum possible speed.

2

Exploring System Studio

This chapter gives a brief introductory overview of System Studio. It contains the following sections:

For more complete information on using System Studio, see the *CoCentric System Studio User Guide*.

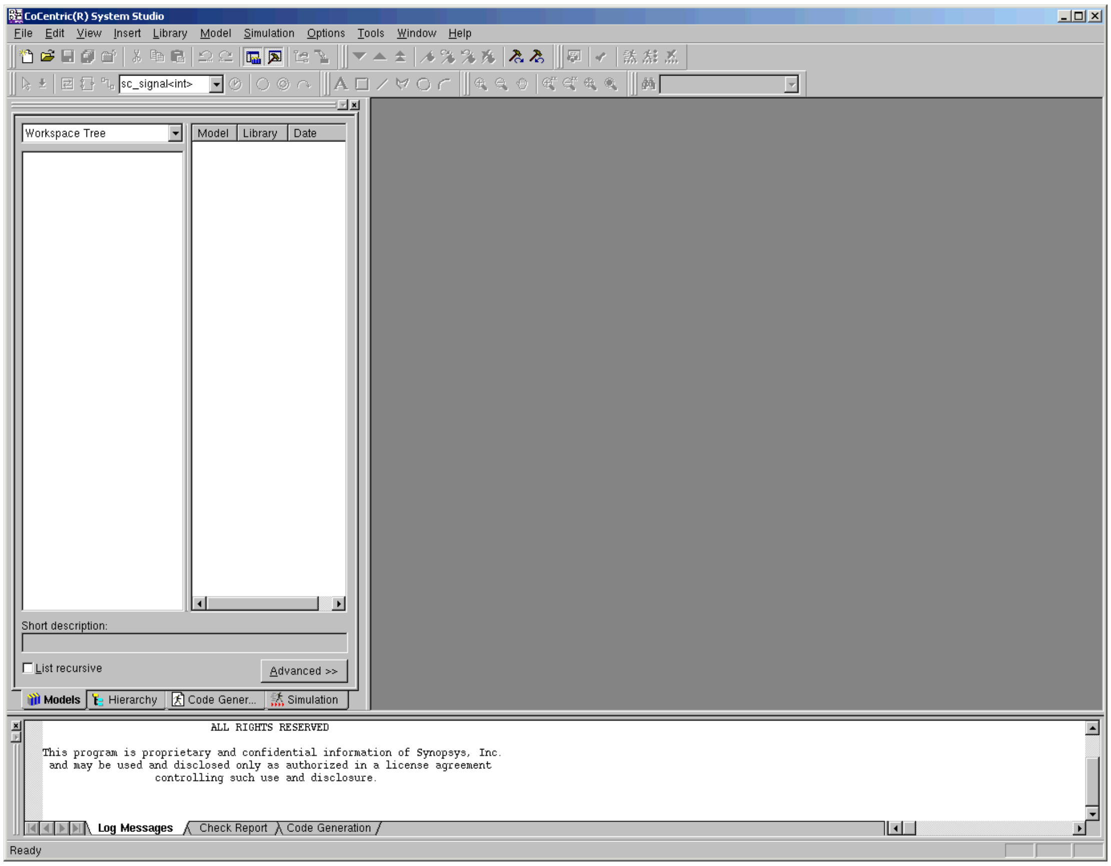
Starting System Studio

To start System Studio, enter

```
% ccss &
```

When System Studio first opens, the design center is displayed as shown in Figure 2-1. However, you can configure System Studio to reload the last workspace that you used with the Options › Preferences › General menu item.

Figure 2-1 System Studio Design Center


CoCentric(R) System Studio
File Edit View Insert Library Model Simulation Options Tools Window Help
sc_signal<int>
Workspace Tree
Model Library Date
Short description:
List recursive
Advanced >>
Models Hierarchy Code Gener... Simulation
Log Messages Check Report Code Generation
Ready

```
                         ALL RIGHTS RESERVED

 This program is proprietary and confidential information of Synopsys, Inc.
  and may be used and disclosed only as authorized in a license agreement
                 controlling such use and disclosure.
```

Opening a Workspace

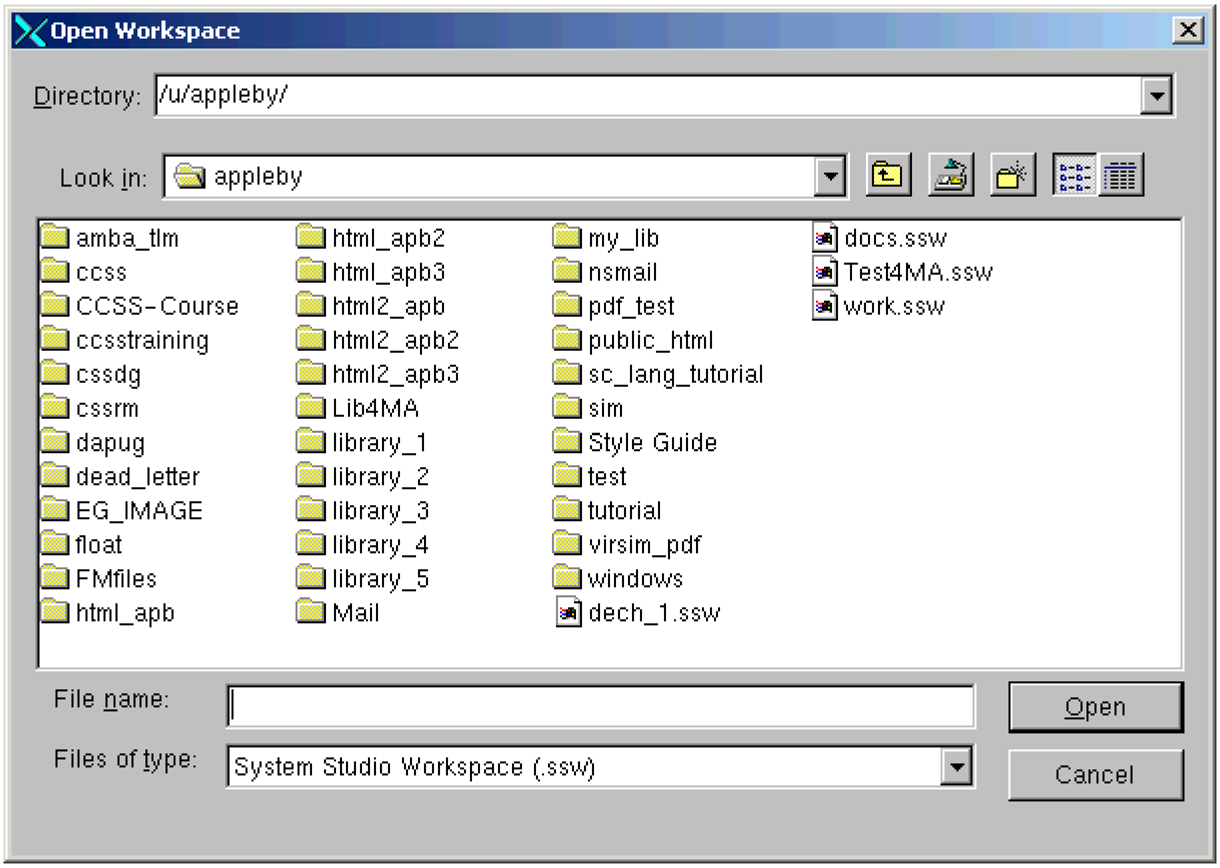
A workspace is your main working environment in System Studio. A workspace typically contains all the libraries and models for a particular project.

The four most recently opened workspaces are available on the File menu.

To open an existing workspace,

- Click the Open toolbar button or choose File › Open.
- The Open Workspace dialog appears, similar to the one shown in Figure 2-2.
- Navigate through the directory and file listing until you find the workspace you want. Select the workspace you want to open and then click Open or press Return.

Figure 2-2 Open Workspace Dialog Box



Directory: /u/appleby/

Look in: appleby

amba_tlm	html_apb2	my_lib	docs.ssw
ccss	html_apb3	nsmail	Test4MA.ssw
CCSS-Course	html2_apb	pdf_test	work.ssw
ccsstraining	html2_apb2	public_html	
cssdg	html2_apb3	sc_lang_tutorial	
cssrm	Lib4MA	sim	
dapug	library_1	Style Guide	
dead_letter	library_2	test	
EG_IMAGE	library_3	tutorial	
float	library_4	virsim_pdf	
FMfiles	library_5	windows	
html_apb	Mail	dech_1.ssw	

Files of type: System Studio Workspace (.ssw)

System Studio provides a quick-type feature as one method of selecting an item in a selection list. To use this feature,

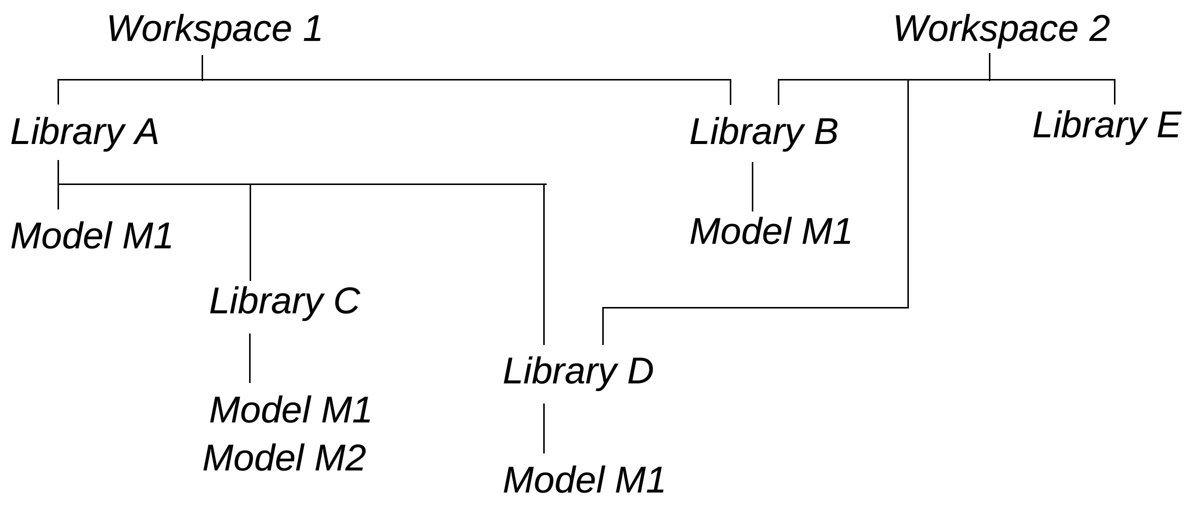
- Click in the selection list.
- Enter the first character of the list item you want displayed.

The selection list item that starts with the letter you entered is scrolled into the visible part of the list and becomes the selected item.

If more than one item starts with the letter you enter, the first matching item is selected. You can then type the letter again or use the arrow keys to move to the next matching item.

Note that a System Studio workspace is not a file system, and a System Studio library is not a directory. A workspace is simply a mapping of libraries into your own environment. A System Studio library is specified by a name and the path to the directory containing its definition; all models in the library's directory belong to the library. This organization allows models to be shared transparently across libraries. Models in different libraries can have the same name, but the name of a library must be unique within a workspace, as shown in Figure 2-3.

Figure 2-3 A System Studio Workspace Structure



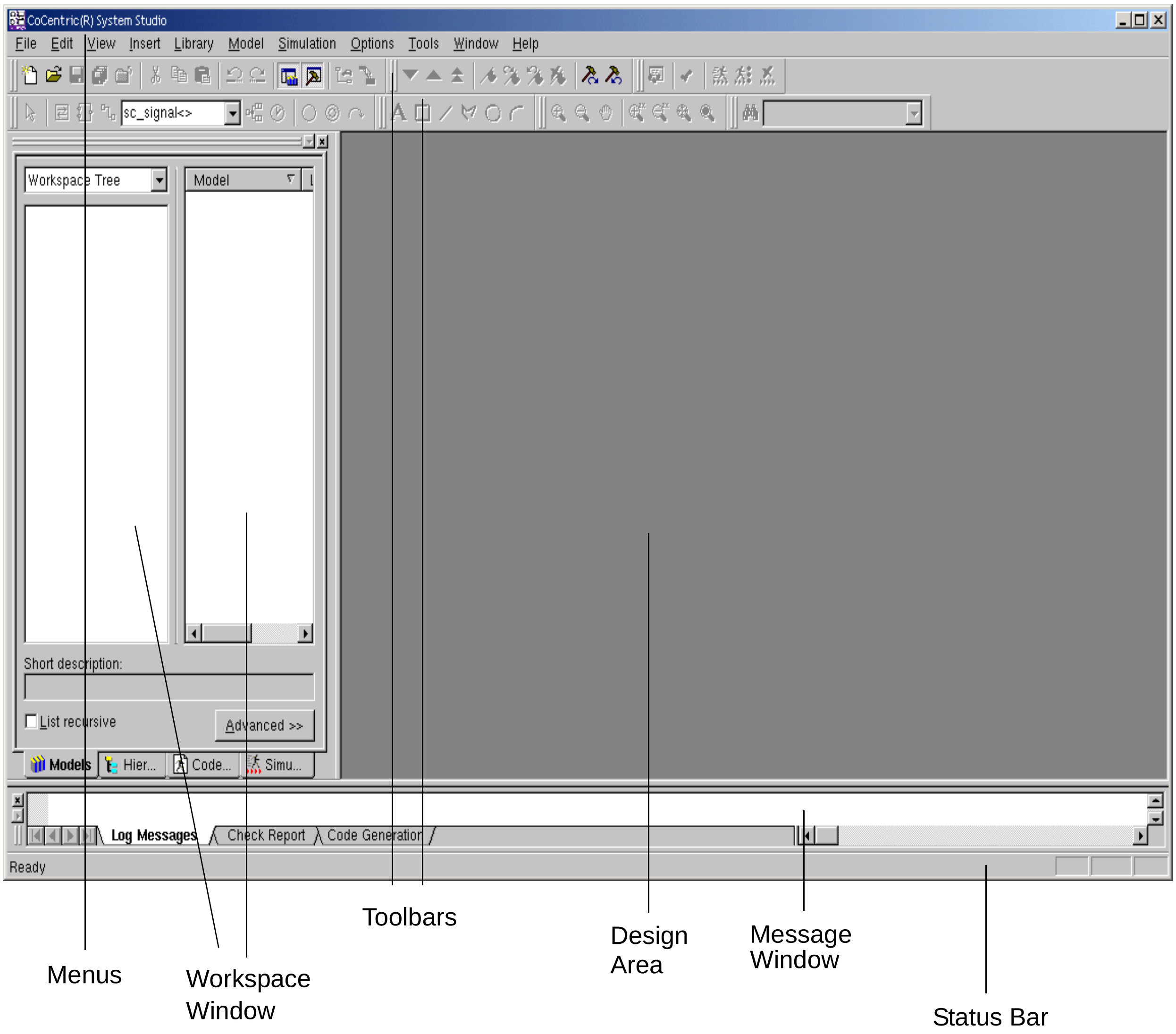
Libraries can be shared between workspaces, but the contents of libraries are restricted to one physical directory. To see whether a model is shared, open the properties for the model in question and check the file path that is shown.

Using the System Studio Design Center Windows

The System Studio design center, shown in Figure 2-4, contains the following elements:

- Menus
- Toolbars
- Workspace window
- Design area
- Message window
- Status bar

Figure 2-4   The System Studio Design Center Window



Menus

You use the System Studio menus to perform operations such as opening and closing model libraries, editing the attributes of a design, and invoking other tools.

Choosing a menu item performs one of the following actions:

- Opens a dialog box. Menu items that open dialog boxes have “...” to the right of the menu item on the menu. Menu items that open submenus have “›” to the right of the menu item.
- Performs a single action, for example, deleting a selected object.
- Enables or disables a mode, such as auto-route.

Some of the menu items are contained in convenience menus that you can access by pressing the right mouse button.

Toolbars

Toolbar buttons enable you to use a single mouse click to perform actions available in the menus.

You can individually display or hide a toolbar from view by choosing the Options › Customize Toolbars menu item and then changing the toolbar appearance and content, as described later in this chapter.

A toolbar can be “docked” or “floating.” A docked toolbar is attached to an edge of the System Studio window. A floating toolbar is not in a fixed position; you can move it anywhere on the screen.

To move a toolbar, click the “move” handle on a docked toolbar, or click the title bar on a floating toolbar and drag it to a new location.

To dock a toolbar, drag it to the edge of the System Studio window. You can dock a toolbar below the System Studio title bar or to the left, right, or bottom edge of the System Studio window. When you drag a toolbar to the edge of the System Studio window, the toolbar outline snaps into place along the System Studio window edge.

To prevent a floating toolbar from docking, press and hold the Control key while you move the toolbar.

There are seven toolbars, each of which can be toggled on and off independently of the rest:

- Standard toolbar
- Navigation toolbar
- Design toolbar
- Schematic toolbar
- Graphics toolbar
- Zooming toolbar
- Find toolbar

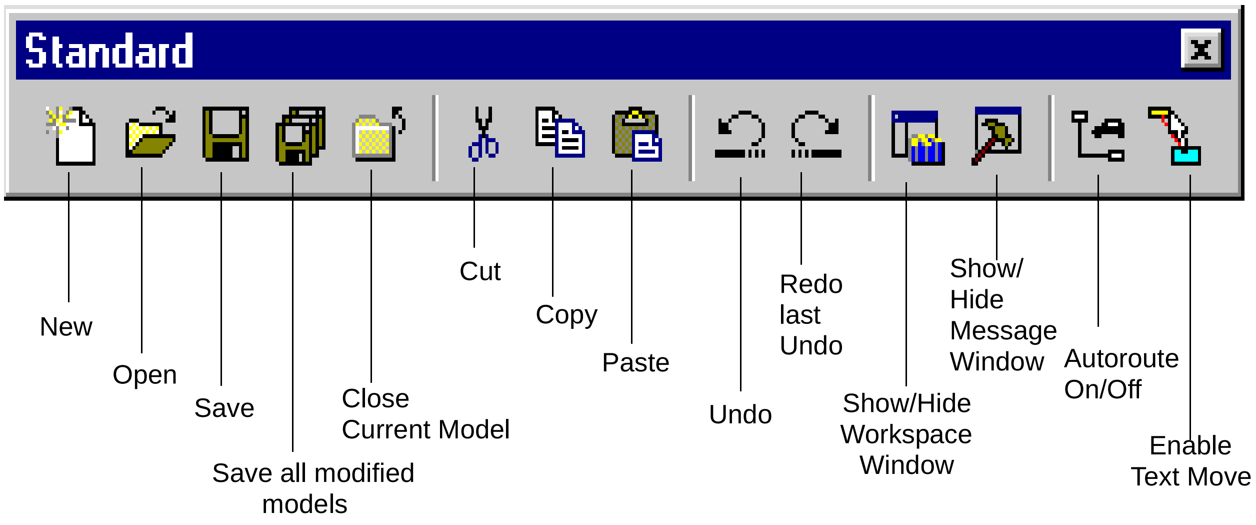
You can add buttons to these toolbars, or remove buttons from them to suit your personal preferences. You can also create new toolbars containing buttons taken from other toolbars, or using buttons that are not part of the default toolbar contents. For details of how to change the contents of the toolbars and how to create your own toolbars, see the *CoCentric System Studio User Guide*.

The default contents of these toolbars are described in the following sections.

Standard Toolbar

The Standard toolbar, shown in Figure 2-5, contains buttons for standard operations on currently selected objects.

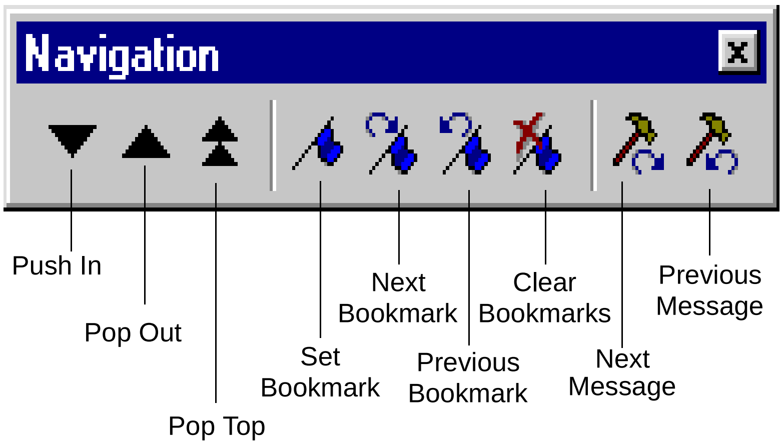
Figure 2-5 The Standard Toolbar



Navigation Toolbar

The Navigation toolbar, shown in Figure 2-6, contains buttons for moving up and down a hierarchical model design.

Figure 2-6 The Navigation Toolbar



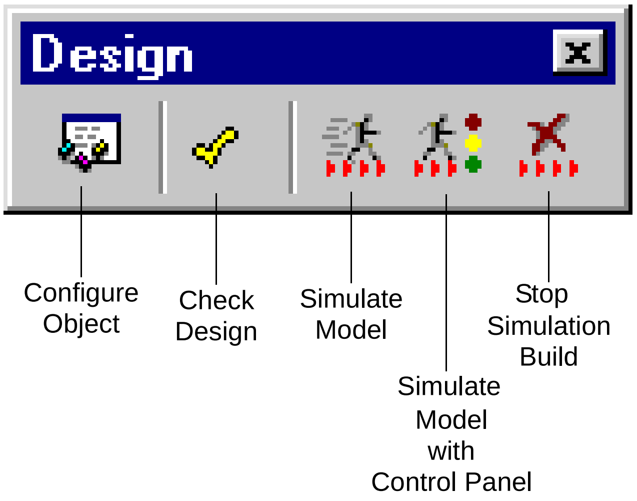
- Use the Push In toolbar button to push into a specific implementation of a model. If the selected model is a hierarchical model, the Push In button moves to a lower level of hierarchy and displays the corresponding design. If the selected instance is a primitive model, using the Push In button opens the source code in the System Studio text editor.

- Use the Pop Out toolbar button to move to the next higher level of hierarchy.
- Use the Pop Top toolbar button to move directly to the top level of the hierarchy.
- Use the bookmark buttons to navigate through source code. When the text editor is open (for example, after you push into a model to display the code), the bookmark buttons are activated. You can use bookmark buttons to find and clear bookmarks in the file for this editing session. Using bookmarks makes moving around the file much easier.
- Use the message buttons to navigate through the contents of the message windows. When a message window is open and it contains messages, you can use the Show Next Message and Show Previous Message buttons to move quickly through the messages.

Design Toolbar

You can use the Design toolbar, shown in Figure 2-7, to check your design and run a simulation.

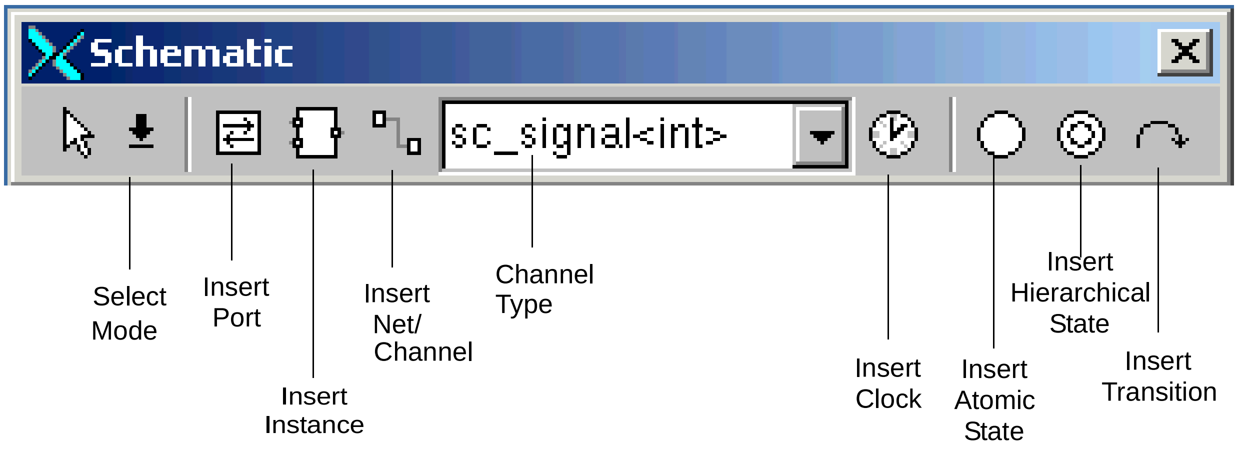
Figure 2-7   The Design Toolbar



Schematic Toolbar

You can use the Schematic toolbar, shown in Figure 2-8, to choose the type of model object to add to a model. Buttons will be dimmed if they are not applicable for the current type of model.

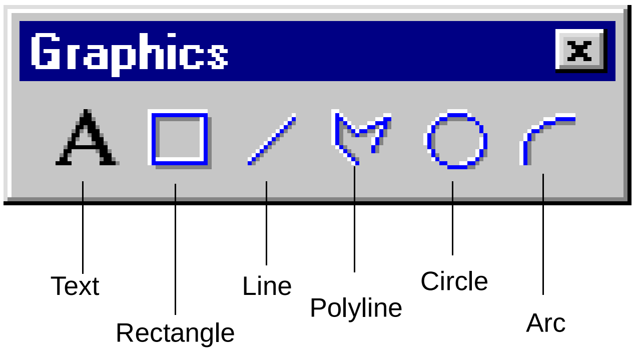
Figure 2-8   The Schematic Toolbar



Graphics Toolbar

You can use the Graphics toolbar, shown in Figure 2-9, to choose the type of simple graphics object to add to a model schematic or a model symbol.

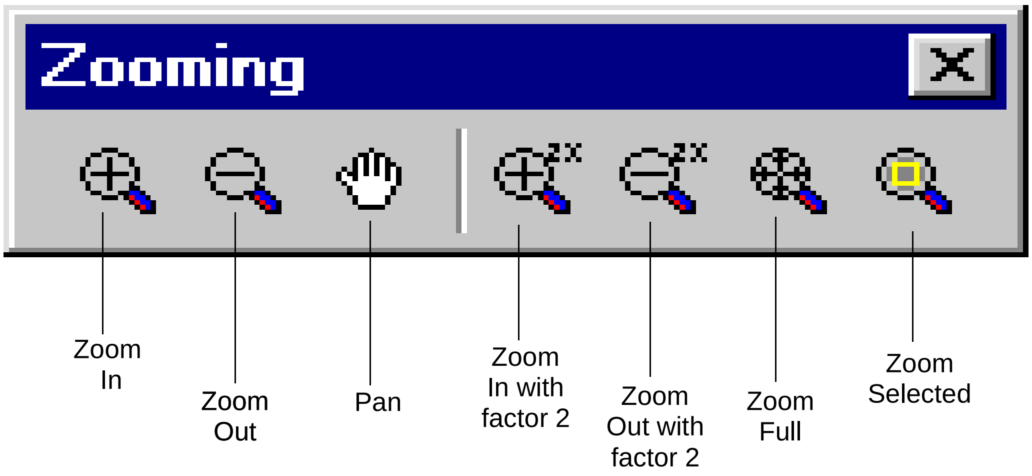
Figure 2-9 The Graphics Toolbar



Zooming Toolbar

You can use the Zooming toolbar, shown in Figure 2-10, to zoom in and out of a design in a variety of ways.

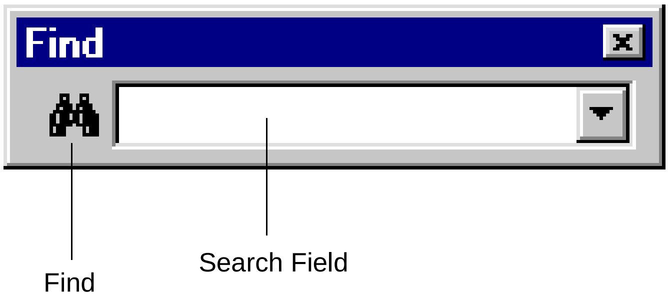
Figure 2-10 The Zooming Toolbar



Find Toolbar

The Find toolbar, shown in Figure 2-11, consists of a pop-up list from which you can search for a named object or objects.

Figure 2-11 The Find Toolbar



Workspace Window

Your main working area is called a workspace. A workspace can contain one or more libraries, each containing a collection of models. In general, your workspace contains all the libraries (standard libraries and user libraries) you have used for a project together with your specific design. All of these objects are organized in a collapsible hierarchical tree display that makes it easier for you to navigate around your working environment. In a design team, the workspace might easily contain a complete system-on-a-chip design.

The Workspace window, shown in Figure 2-4 on page 2-7, contains the four distinct pages:

- Models page
- Hierarchy browser
- Code Generation page

- Simulation control panel

You access these pages by a tab at the bottom of the window. The pages are described in the following sections.

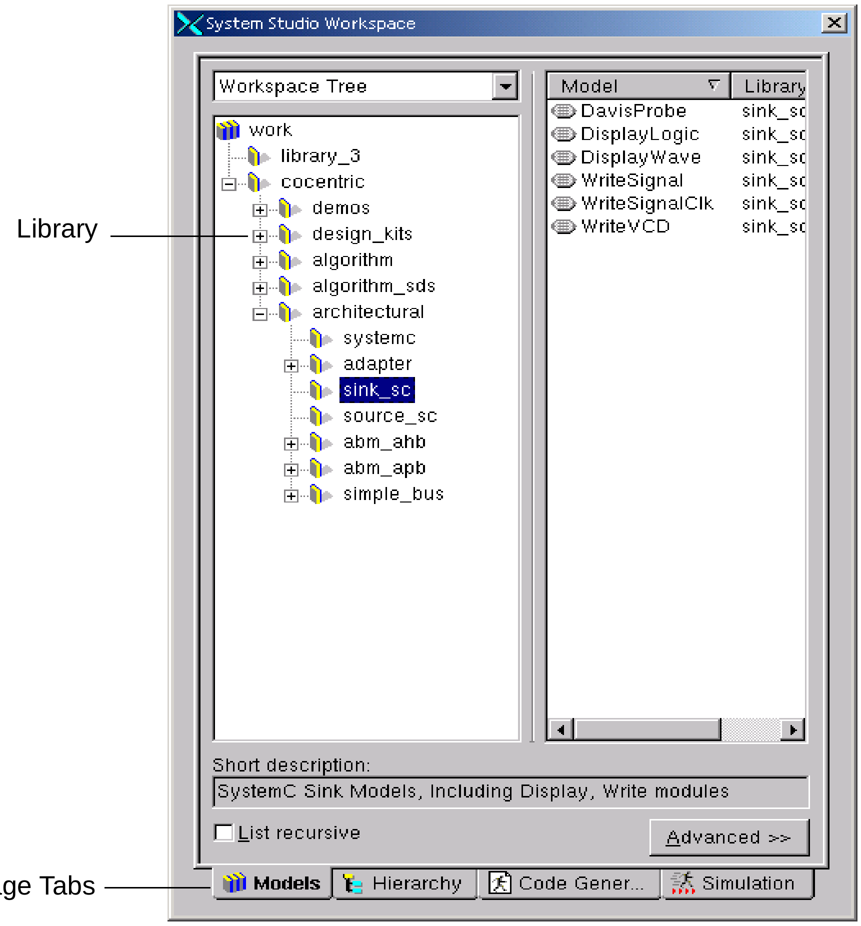
Models Page

The Models page is the default page, which is displayed when System Studio is started.

The Models page shows a tree-like display of your current workspace: the libraries and sublibraries, and the models contained in each library, as shown in Figure 2-12. The libraries and models you see in this page will, of course, depend on the contents of your workspace.

You can expand and collapse parts of the tree display by clicking on the + and - symbols, respectively.

Figure 2-12 Models Page With Workspace Tree View



Model	Library
DavisProbe	sink_sc
DisplayLogic	sink_sc
DisplayWave	sink_sc
WriteSignal	sink_sc
WriteSignalClk	sink_sc
WriteVCD	sink_sc

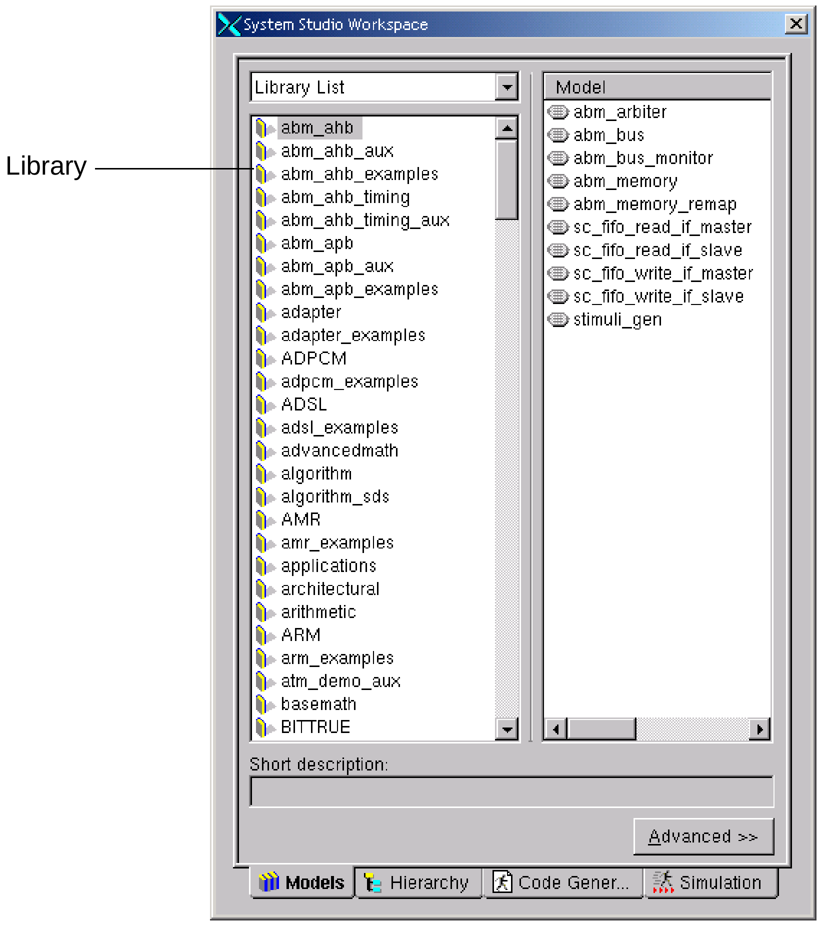
Short description:
SystemC Sink Models, Including Display, Write modules

The Models page contains the following three views of your workspace. You select a view from the list at the top of the window.

- Workspace Tree view (this is the default view, shown in Figure 2-12)
- Library List view (shown in Figure 2-13)
- Model Category Tree view (shown in Figure 2-14)

The Models page view is designed to give you the maximum in model navigation capabilities for large systems, including extensive collections of existing COSSAP legacy models.

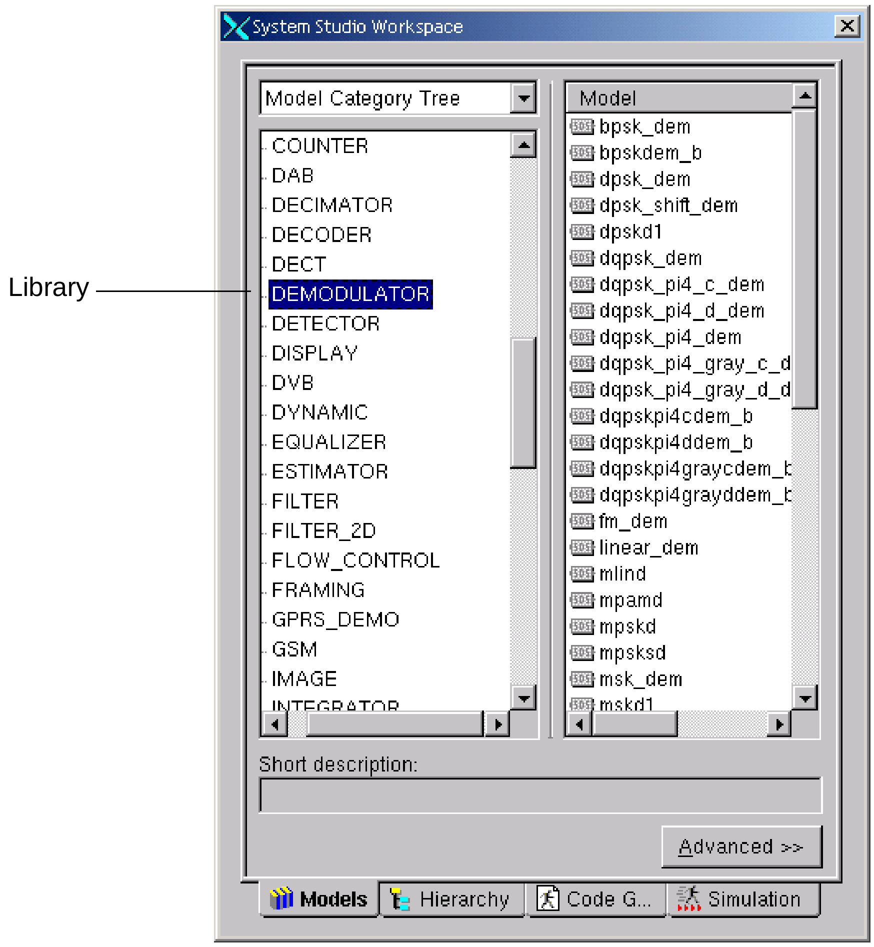
Figure 2-13   Library List View



abm_ahb
abm_ahb_aux
abm_ahb_examples
abm_ahb_timing
abm_ahb_timing_aux
abm_apb
abm_apb_aux
abm_apb_examples
adapter
adapter_examples
ADPCM
adpcm_examples
ADSL
adsl_examples
advancedmath
algorithm
algorithm_sds
AMR
amr_examples
applications
architectural
arithmetic
ARM
arm_examples
atm_demo_aux
basemath
BITTRUE

abm_arbiter
abm_bus
abm_bus_monitor
abm_memory
abm_memory_remap
sc_fifo_read_if_master
sc_fifo_read_if_slave
sc_fifo_write_if_master
sc_fifo_write_if_slave
stimuli_gen

Figure 2-14 Model Category Tree View



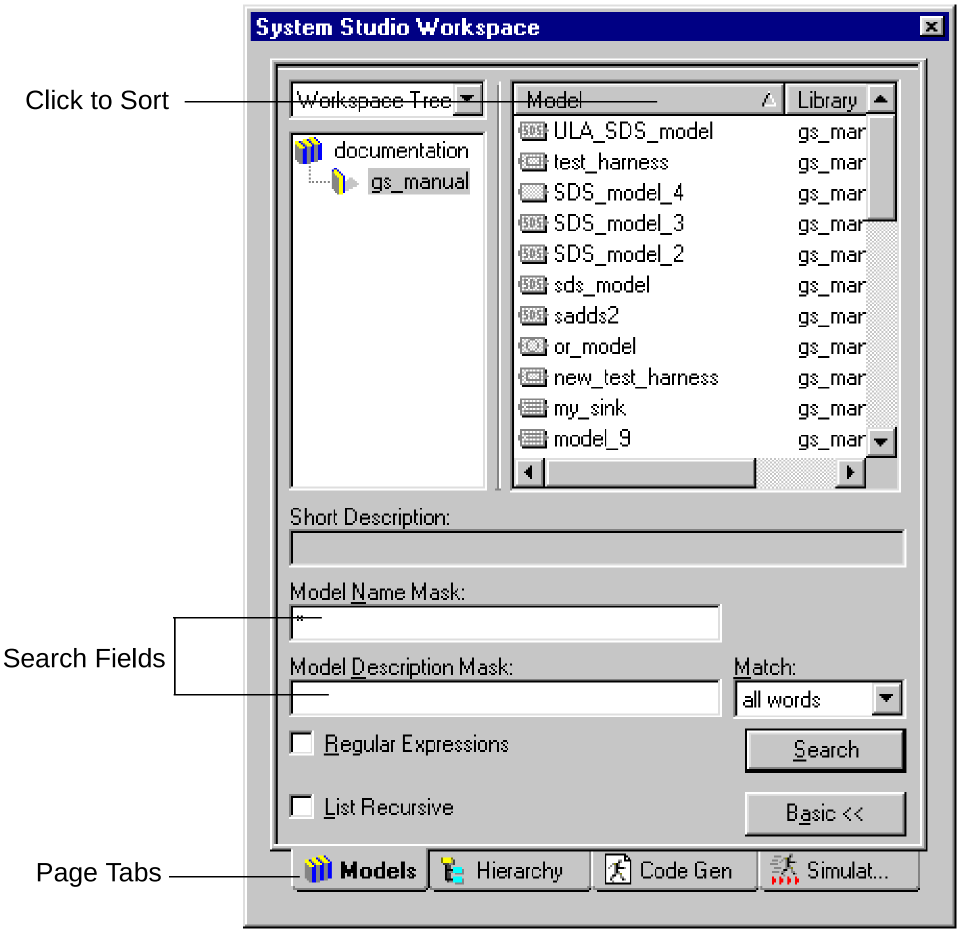
COUNTER
DAB
DECIMATOR
DECODER
DECT
DEMODULATOR
DETECTOR
DISPLAY
DVB
DYNAMIC
EQUALIZER
ESTIMATOR
FILTER
FILTER_2D
FLOW_CONTROL
FRAMING
GPRS_DEMO
GSM
IMAGE
INTEGRATOR

bpsk_dem
bpskdem_b
dpsk_dem
dpsk_shift_dem
dpskd1
dqpsk_dem
dqpsk_pi4_c_dem
dqpsk_pi4_d_dem
dqpsk_pi4_dem
dqpsk_pi4_gray_c_d
dqpsk_pi4_gray_d_d
dqpskpi4cdem_b
dqpskpi4ddem_b
dqpskpi4graycdem_b
dqpskpi4grayddem_b
fm_dem
linear_dem
mlind
mpamd
mpskd
mpsksd
msk_dem
mskd1

Libraries can be nested inside other libraries, and they can be shared with other users.

You can click the Advanced button to display a more sophisticated page layout that includes fields that enable you to select models more quickly, as shown in Figure 2-15.

Figure 2-15 Searching with The Advanced Feature

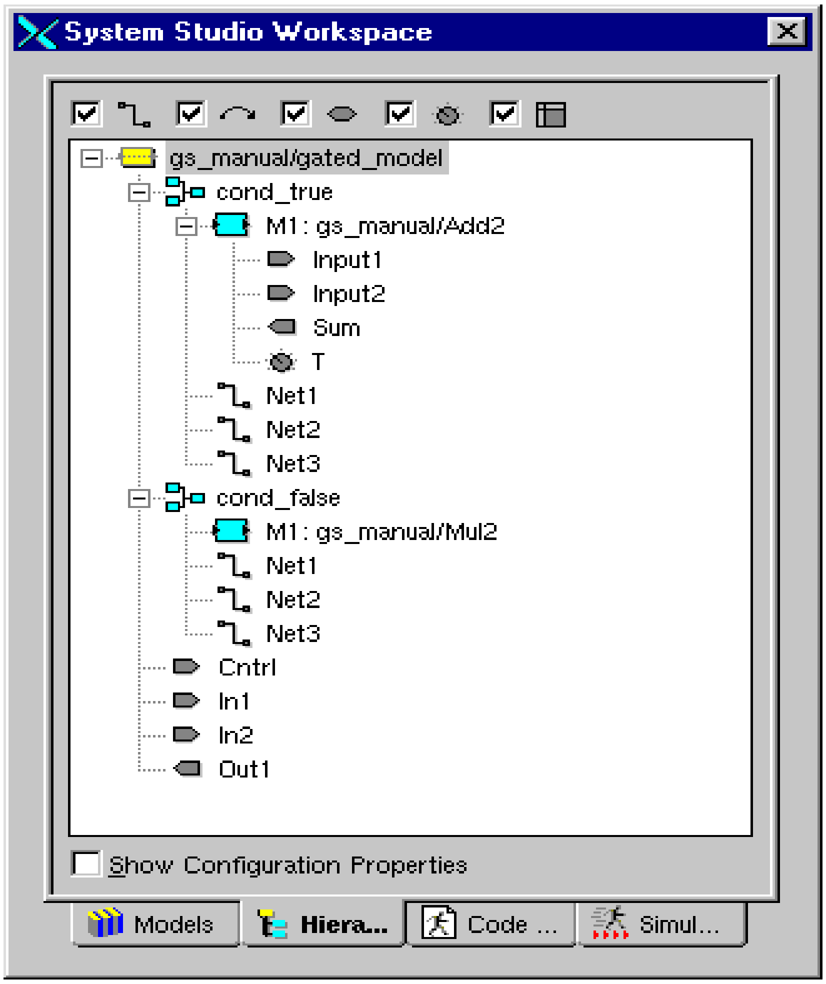


You can search for a specific model by entering part of its name or description in the search fields and then clicking Search. You can use the standard wildcard characters (* and %) as part of your search pattern or you can click the Regular Expressions check box to enable a search using regular expressions (for the syntax of regular expressions, see the UNIX man pages on regular expressions or consult the System Studio online Help).

Hierarchy Browser

The hierarchy browser, shown in Figure 2-16, allows you to browse through the various parts of a model in much the same way as you can browse through the models in the workspace. This browser is especially helpful when you are trying to determine the location for breakpoints and data and level watch points when you are debugging simulations. Click on an object to show where it is used; double-click on an object to show its implementation.

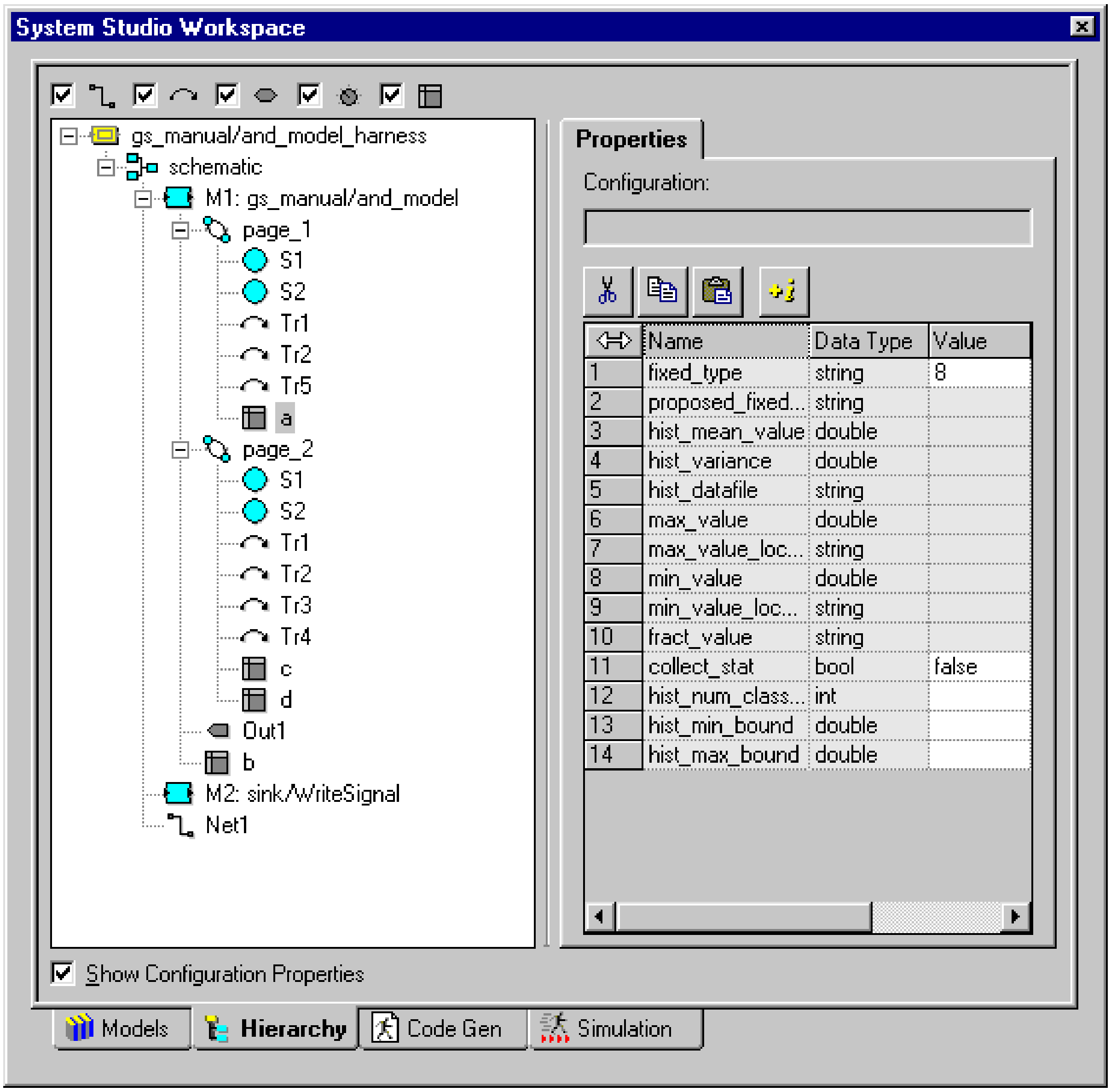
Figure 2-16 The Hierarchy Browser



```
gs_manual/gated_model
  cond_true
    M1 : gs_manual/Add2
      Input1
      Input2
      Sum
      T
    Net1
    Net2
    Net3
  cond_false
    M1 : gs_manual/Mul2
    Net1
    Net2
    Net3
  Cntrl
  In1
  In2
  Out1
```

Selecting "Show Configuration Properties" allows you to view the configurations properties of those parts of a model that have them. Values that can be edited have a white background, as shown in Figure 2-17.

Figure 2-17 The Hierarchy Browser Configuration Properties

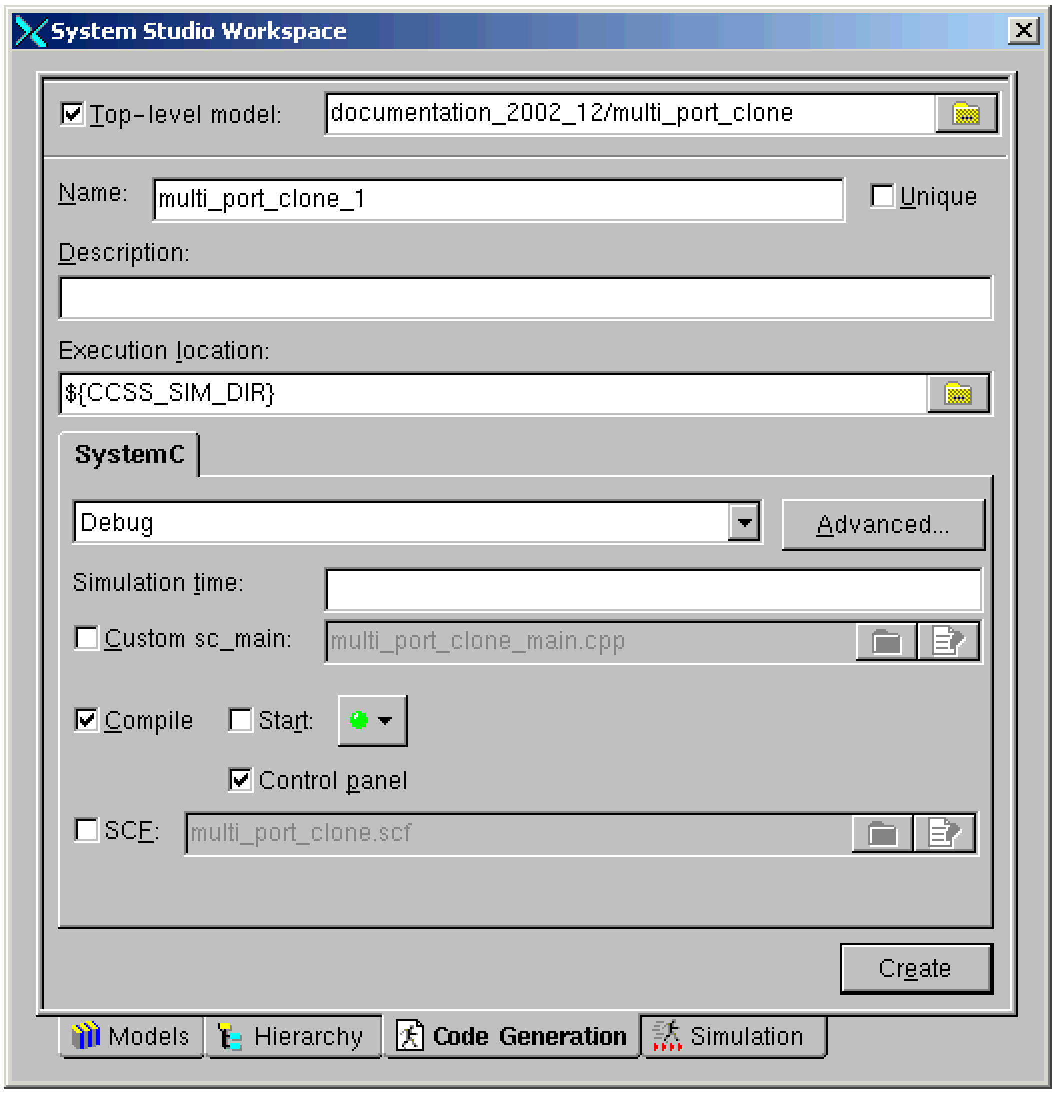


	Name	Data Type	Value
1	fixed_type	string	8
2	proposed_fixed...	string	
3	hist_mean_value	double	
4	hist_variance	double	
5	hist_datafile	string	
6	max_value	double	
7	max_value_loc...	string	
8	min_value	double	
9	min_value_loc...	string	
10	fract_value	string	
11	collect_stat	bool	false
12	hist_num_class...	int	
13	hist_min_bound	double	
14	hist_max_bound	double	

Code Generation Page

You use the Code Generation page, shown in Figure 2-18, to set switches and the values of parameters that govern the simulation code generation. The simulation execution options dialog box is shown in Figure 2-19.

Figure 2-18 Setting the Code Generation Parameters



☑ Top-level model: documentation_2002_12/multi_port_clone

Name: multi_port_clone_1 ☐ Unique

Description:

Execution location: ${CCSS_SIM_DIR}

Debug

Simulation time:

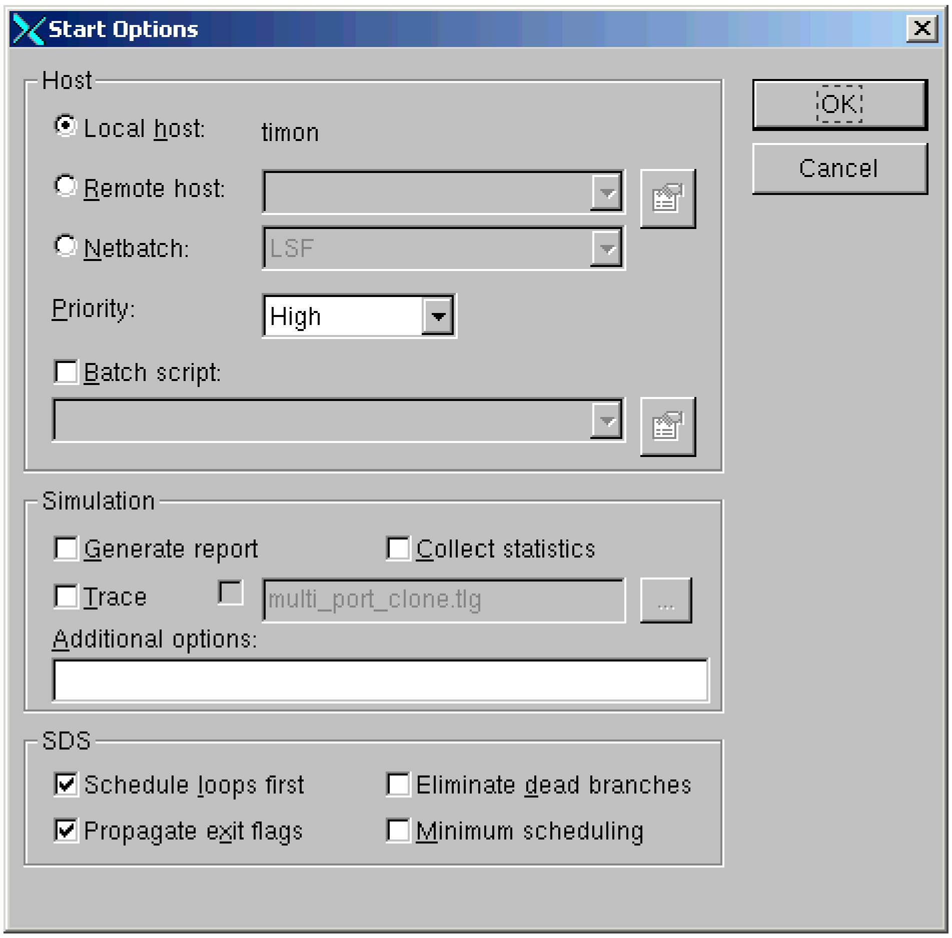
☐ Custom sc_main: multi_port_clone_main.cpp

☑ Compile ☐ Start:

☑ Control panel

☐ SCF: multi_port_clone.scf

Figure 2-19   Setting the Simulation Execution Options


Start Options

Host

● Local host: timon

○ Remote host:

○ Netbatch: LSF

Priority: High

☐ Batch script:

Simulation

☐ Generate report   ☐ Collect statistics

☐ Trace   ☐ multi_port_clone.tlg

Additional options:

SDS

☑ Schedule loops first   ☐ Eliminate dead branches

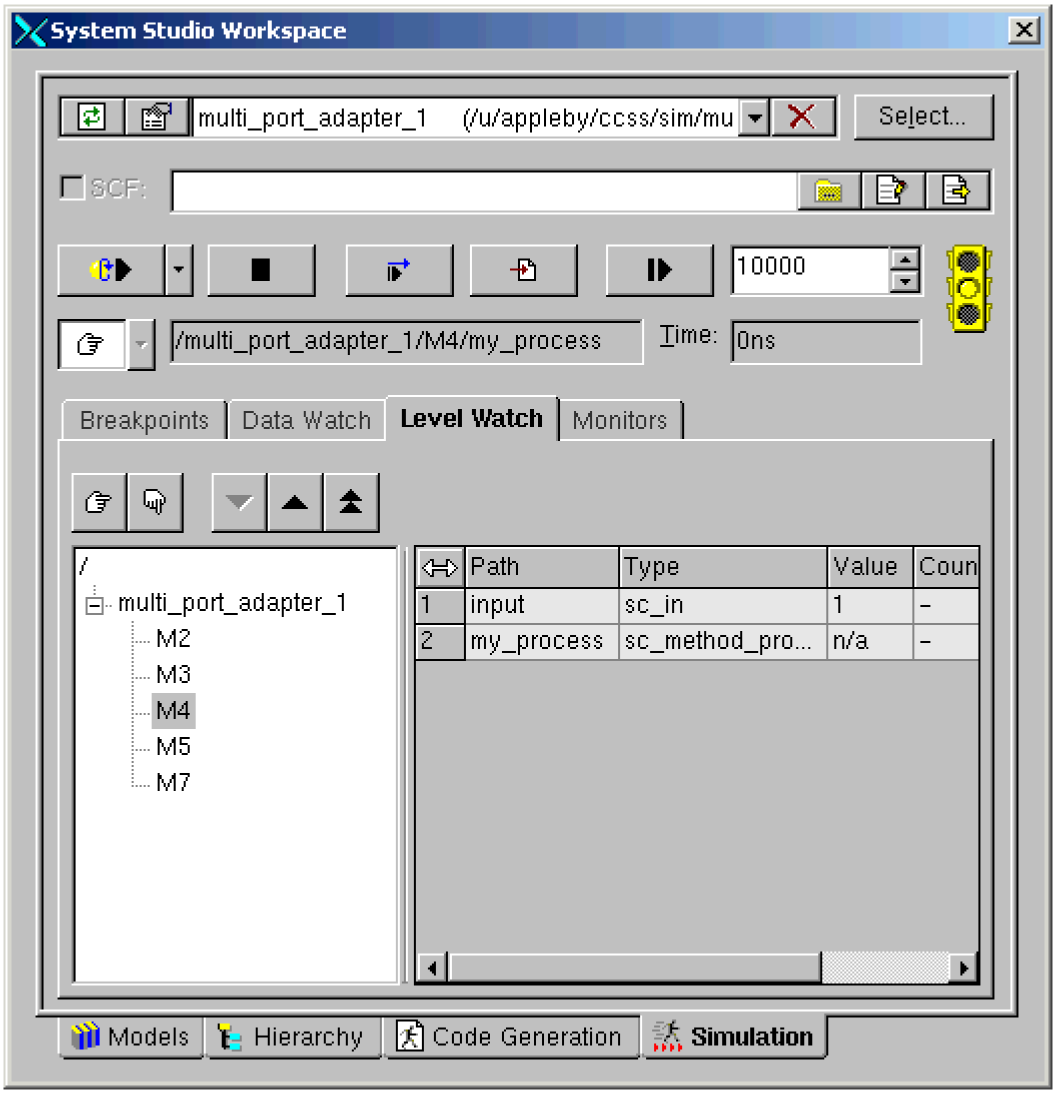
☑ Propagate exit flags   ☐ Minimum scheduling

For details on using the Simulation control panel, see the *CoCentric System Studio User Guide*.

Simulation Control Panel

Use the simulation control panel, shown in Figure 2-20, to control the simulation execution and interact with the simulation to debug it.

Figure 2-20 The Simulation Control Panel



multi_port_adapter_1 (/u/appleby/ccss/sim/mu

10000

/multi_port_adapter_1/M4/my_process

Time: 0ns

/
- multi_port_adapter_1
  - M2
  - M3
  - M4
  - M5
  - M7

	Path	Type	Value	Coun
1	input	sc_in	1	–
2	my_process	sc_method_pro...	n/a	–

For details on using the Simulation control panel, see the *CoCentric System Studio User Guide*.

Design Area

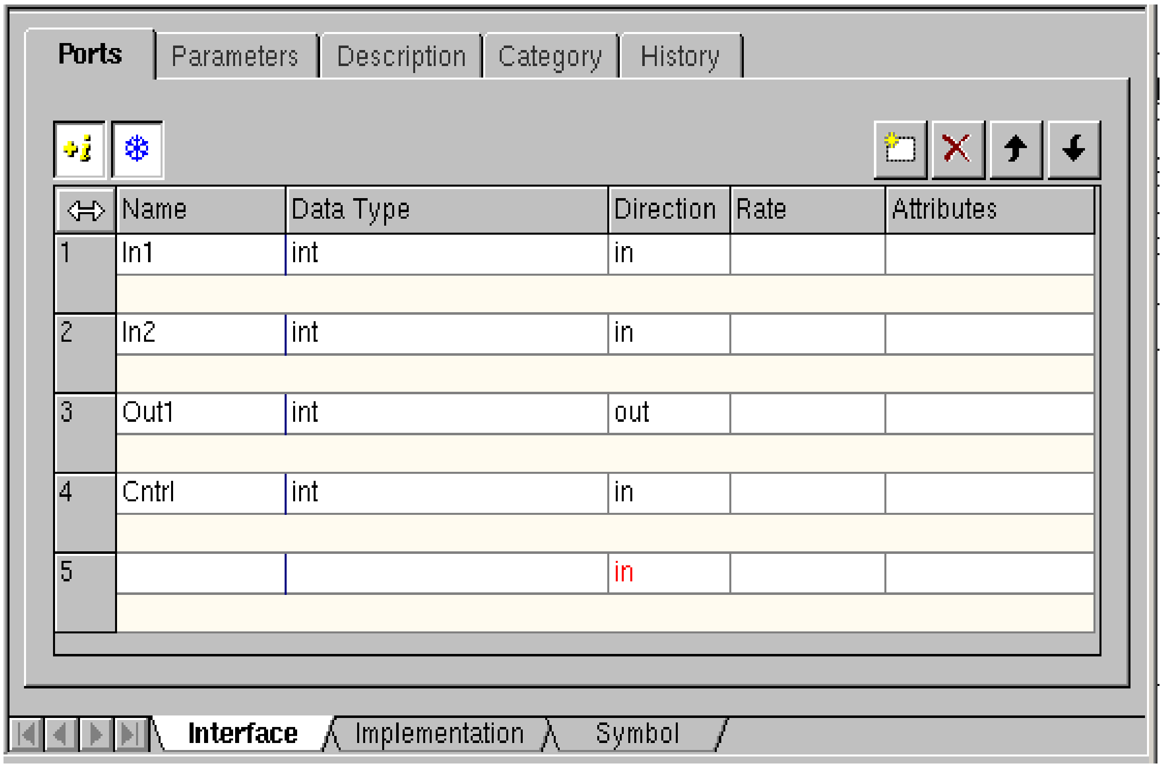
The Design Area occupies the major part of the System Studio display. It is here that your design actually takes shape. The Design Area gives you different views of your design, each accessible via a tab. The number and types of view that are available depend upon whether your model is an algorithmic, architectural primitive or architectural hierarchical design.

For the algorithmic domain, the views are as follows:

- Interface View: through which you set the design’s ports and parameters
- Implementation View: giving you a graphical representation of the design, or a view of the source code, depending on the model’s implementation
- Symbol View: giving you a view of the graphical representation of the model that will be used when it is instantiated into a higher-level model

Interface View. You use the interface view to describe the model’s ports, parameters, description, classification, and history as shown in Figure 2-21.

Figure 2-21 Viewing the Model Interface

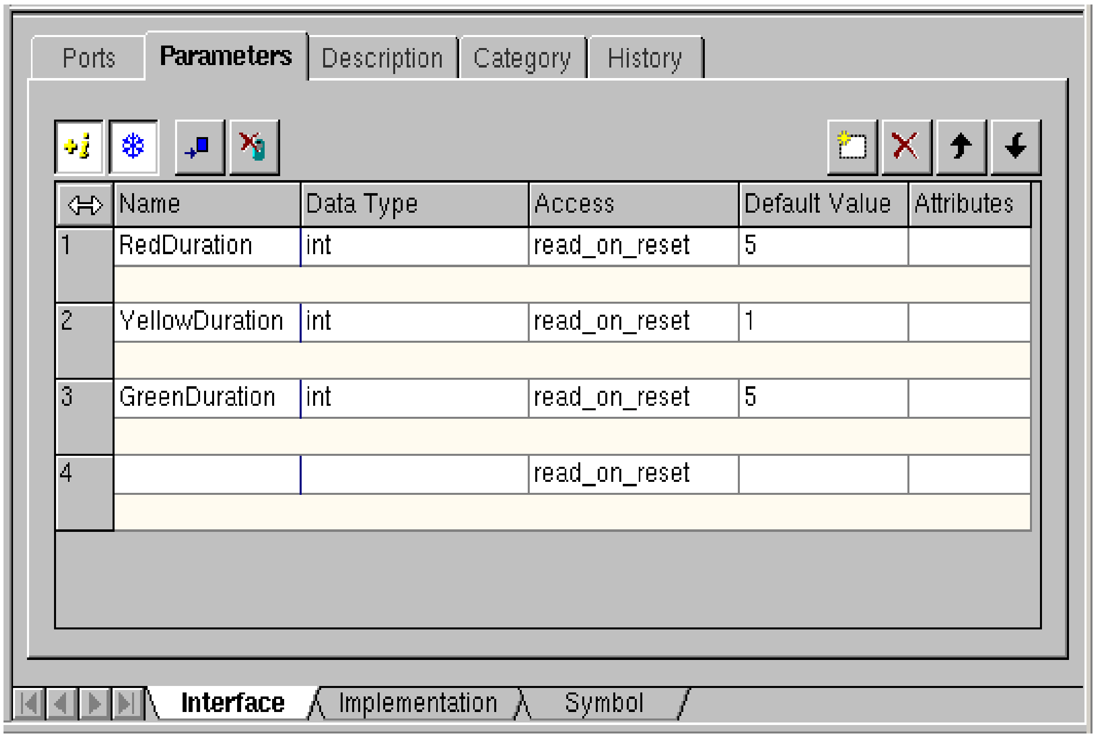


	Name	Data Type	Direction	Rate	Attributes
1	In1	int	in		
2	In2	int	in		
3	Out1	int	out		
4	Cntrl	int	in		
5			in		

To add an entry, delete an entry, or move a row up or down, click the appropriate button.

To enter or change a value, click the entry and a list of possible values for you to choose will be displayed, as shown in Figure 2-22. Unsaved changes are shown in red.

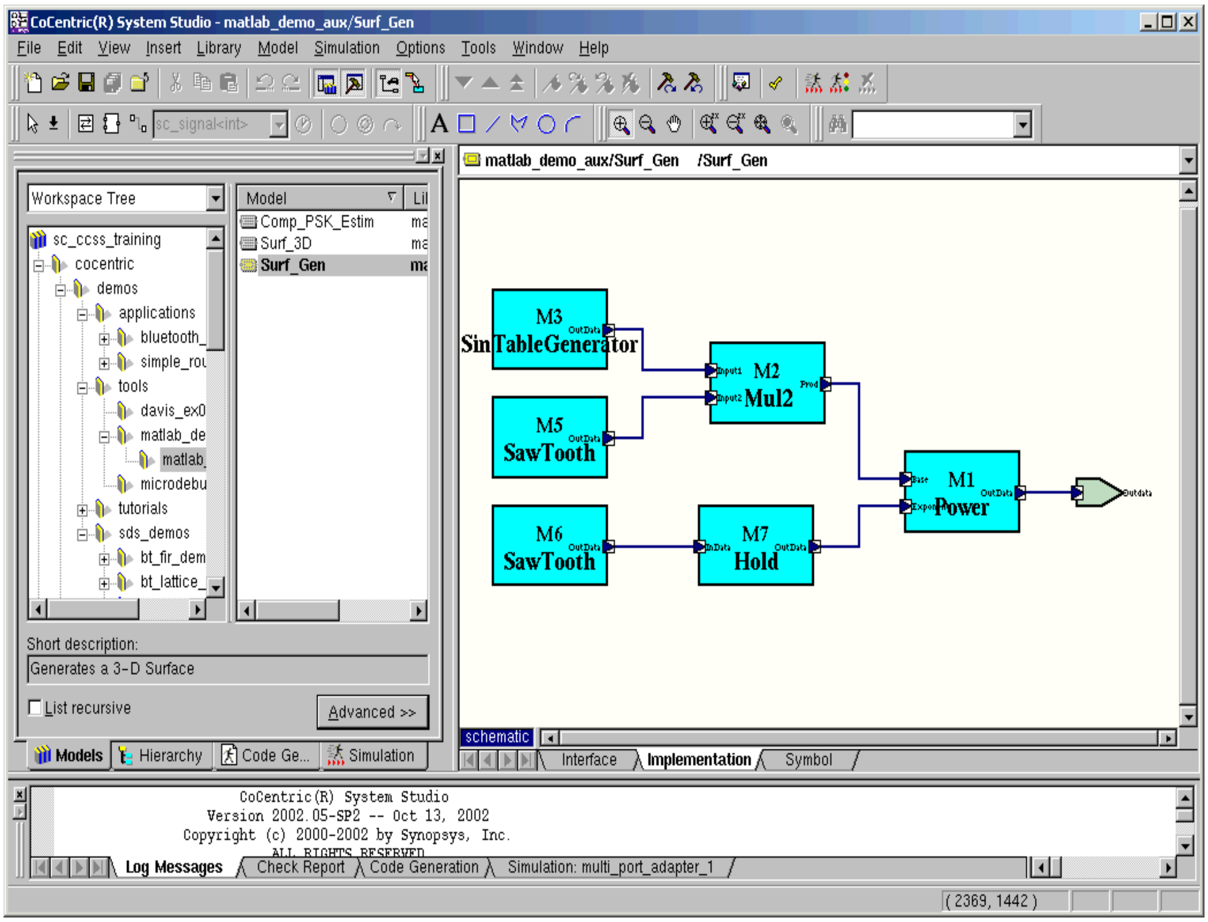
Figure 2-22 Setting a Parameter for the Model Interface



	Name	Data Type	Access	Default Value	Attributes
1	RedDuration	int	read_on_reset	5	
2	YellowDuration	int	read_on_reset	1	
3	GreenDuration	int	read_on_reset	5	
4			read_on_reset		

Implementation View. You use the implementation view to create the model behavior. See Figure 2-23 for an example of a dataflow graph implementation of a model.

Figure 2-23   Viewing the Model Implementation

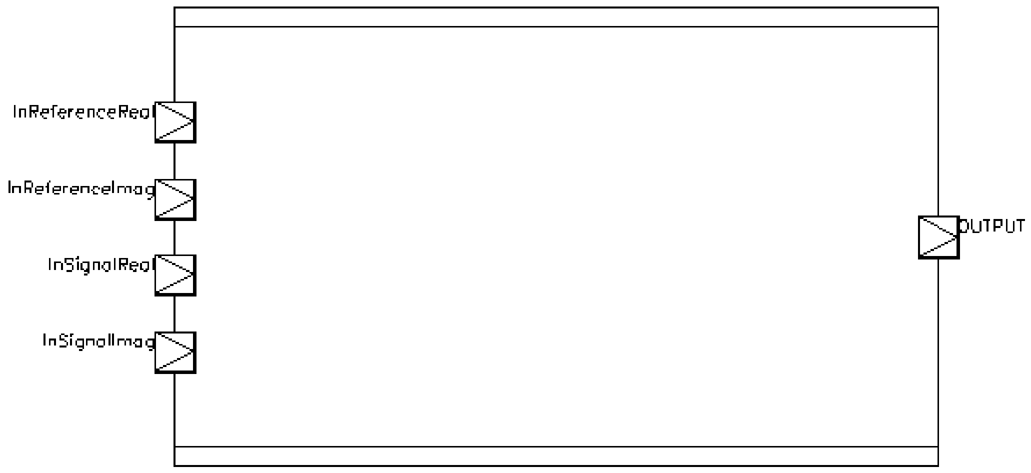


```
CoCentric(R) System Studio
Version 2002.05-SP2 -- Oct 13, 2002
Copyright (c) 2000-2002 by Synopsys, Inc.
ALL RIGHTS RESERVED
```

The implementation view is explained in more detail in the modeling tutorials later in this manual.

Symbol View. You use the Symbol view, as shown in Figure 2-24, to edit the appearance of model symbols.

Figure 2-24 A Typical Model Symbol



You can change the shape of a symbol, or add comment text and lines for documentation purposes. You can add any of the graphics from the Graphics toolbar. You cannot, however, change the structural or behavioral attributes of the design object that the symbol represents.

Pressing the right mouse button in the Symbol view opens a context-sensitive menu that gives you quick access to commands for working with the graphic objects.

Graphics shapes are ordered into “layers.” When shapes are stacked on top of each other, the top-most shape is said to be at the front and the bottom-most shape is said to be at the back. Using the Simple Graphics option you can change the order (level) of the selected shape(s) in one of four ways:

- Bring Forward - Moves the shape(s) one level to the front
- Send Backward - Moves the shape(s) one level to the back

- Bring To Front - Moves the shape(s) in front of all others
- Send To Back - Moves the shape(s) behind all others

Working in the architectural domain is similar to working in the algorithmic domain with the exception that two more views are available. The additional views are:

- The Header View
- The Source View

Header View. The header view is an empty template into which you can add data members, method and processes for your SystemC model. You may also read details of the port and parameter information that is entered in other views.

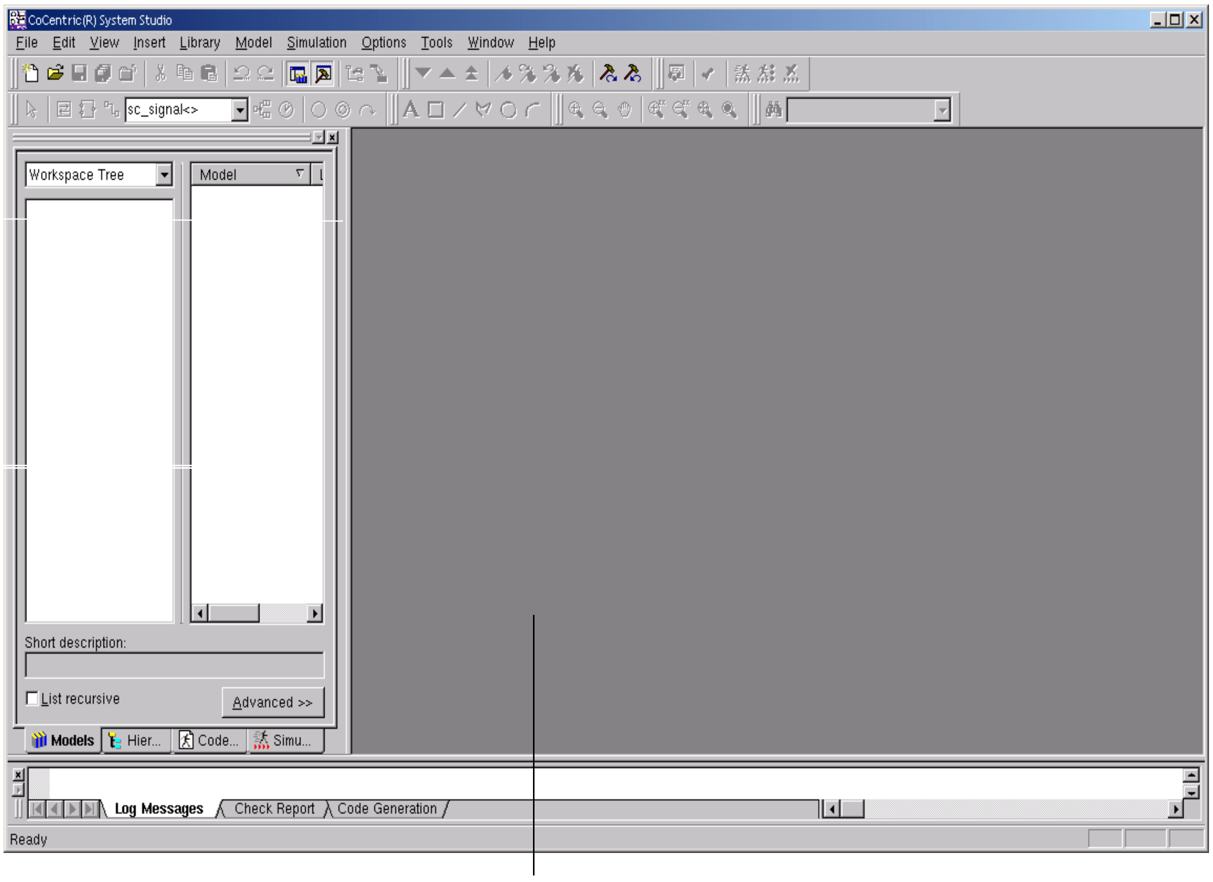
Source View. In the Source view, you enter the model source code.

Working in the Design Area

The Design Area is where you create, edit, and view the implementations of a model, set the implementation parameters, and edit the model symbol. You can think of the Design Area as your main working area.

When you first start System Studio, the Design Area is empty, as shown in Figure 2-25. After you load or create a workspace, the Design Area shows the implementation for the currently selected object.

Figure 2-25   The Basic Design Area



Design area

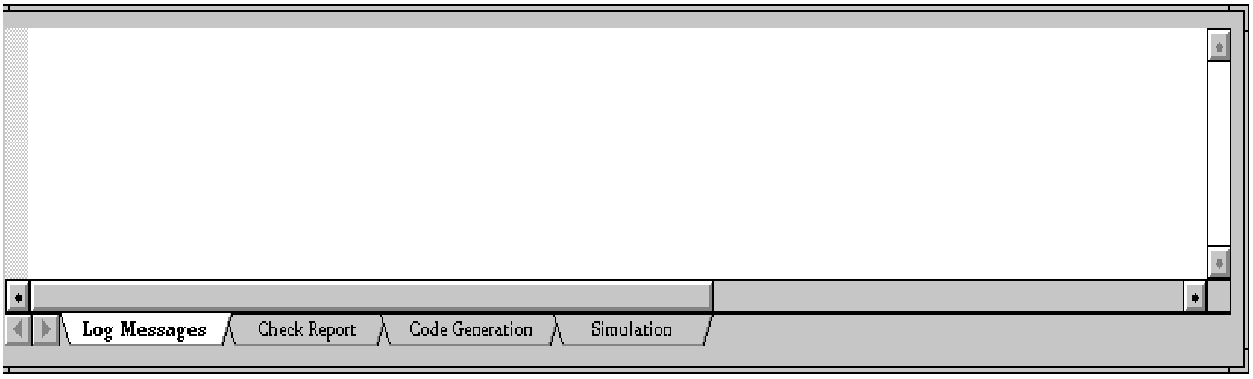
A context-sensitive convenience menu provides access to some of the more commonly used menu items from the Edit, Zoom, Model, and Show menus.

Open a model, position your mouse pointer in the design area (you must have a model open), and press the right mouse button to view the convenience menu. Choose a menu item or a submenu item to execute the corresponding command. The convenience menu that will be displayed depends on whether you have selected anything or not and, if you have selected an object, what that object is.

Message Window

The message window, shown in Figure 2-26, displays messages returned by the action that is being performed by System Studio. These messages (errors, warnings, progress reports, and so on) are grouped into separate pages according to the activity.

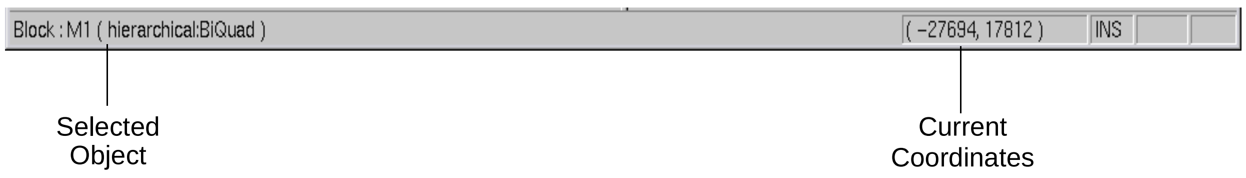
Figure 2-26 The Message Window



Status Bar

The status bar, shown in Figure 2-27, displays information about the current state of the tool.

Figure 2-27 The Status Bar



Selecting Design Objects

You can select objects after you have instantiated them. A combination of keyboard and mouse actions allows easy selection of objects. You can select the following objects:

- An individual object - Move the cursor over an object, then click.
- Multiple objects - Select the first object; Shift-click or Control-click the left mouse button and select the next object.
- A group of objects - Press and hold the left mouse button and drag the cursor until the target objects are surrounded by a selection box, then release the left mouse button.
- Nets - Click once to select a net segment; click twice to select the complete net.

Selected objects appear highlighted. The appearance of selected items is user-configurable. For information about the editable display attributes, see the *CoCentric System Studio User Guide*.

Editing Design Objects

You can perform the following kinds of edits on design objects:

- Move
- Copy
- Cut
- Delete
- Paste

Moving

Use the following procedure to move one or more objects from one location to another.

- Select the object or objects to move.

  The editor highlights the object.

- Move the cursor over the selected object. If you have selected multiple objects, move the cursor over any of the selected objects.

  The cursor turns into a cross hair.

- Click and hold the left mouse button, then move the cursor to the location where you want to place the objects.

  An outline of the selected objects attaches to the cursor.

- Release the left mouse button.

  The selected objects appear in the new location. If autorouting is active, the connections are rerouted as necessary.

Copying

The copy function puts a copy of the objects you have selected on the clipboard.

When a copied instance is pasted back into a design, the copy retains its properties such as parameter and port values.

To copy design objects,

- Select the object or objects to copy.

The objects are highlighted.

- Click the Copy toolbar button, or press Control-C.

  A copy of the selected objects is placed on the clipboard.

  Note:

  The models contained within the libraries supplied with System Studio are write protected. There are special methods for copying protected models; these methods are dealt with in the relevant sections of the tutorial chapters.

Cutting

The cut function deletes selected objects and puts a copy of them on the clipboard. If you cut an instance or a pin and the autorouting feature is active, System Studio reroutes any nets attached to that block or pin. Typically, rerouted nets get shortened to small segments on other connections.

Note that when a cut instance is pasted back into the design, the copy retains its parameter and data-set values.

To cut design objects,

- Select the object or objects to cut.

  The objects are highlighted.

- Click the Cut toolbar button, or press Control-X.

  The selected objects are removed from the schematic and a copy is placed on the clipboard.

Deleting

The delete function cuts selected objects but does not put a copy of them on the clipboard.

To delete design objects,

- Select the object to delete.

  The objects are highlighted.

- Choose Edit › Delete or press the Delete key on your keyboard.

  The selected objects are removed from the design.

Pasting

The paste function copies the contents of the clipboard to the cursor, which you can position anywhere in the design before placing the pasted objects. You can paste the same object repeatedly until you put another object on the clipboard. Because deleted objects are never put on the clipboard, you cannot paste them back into a design.

When a cut or copied instance is pasted back into a design, the copy retains its parameter values.

To paste design objects,

- Click the Paste toolbar button and move the cursor to the design.

  The cursor turns into a cross-hair, with the outline of the object on the clipboard attached.

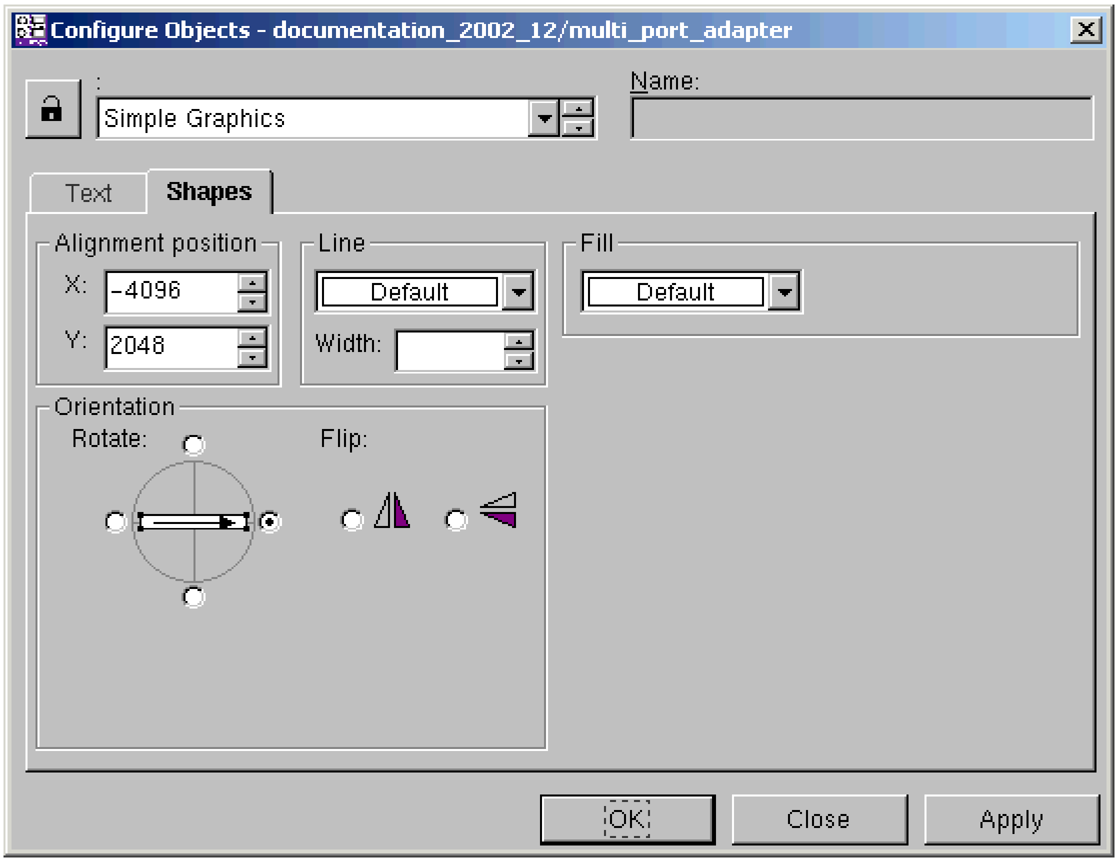
- Move the cursor to the location where you want to put the pasted object.
- Click the left mouse button or press Control-v to place the object.

  The object is placed and the cursor changes to the select symbol.

Setting the Properties

Select a graphic object (rectangle, line, polyline, circle or arc) and press the right mouse button. This displays a menu. Choose Configure Object and the Graphics Properties dialog box opens as shown in Figure 2-28.

Figure 2-28 Setting the Graphic Object Properties


Configure Objects - documentation_2002_12/multi_port_adapter

Simple Graphics

Name:

Text | Shapes

Alignment position
X: -4096
Y: 2048

Line: Default
Width:

Fill: Default

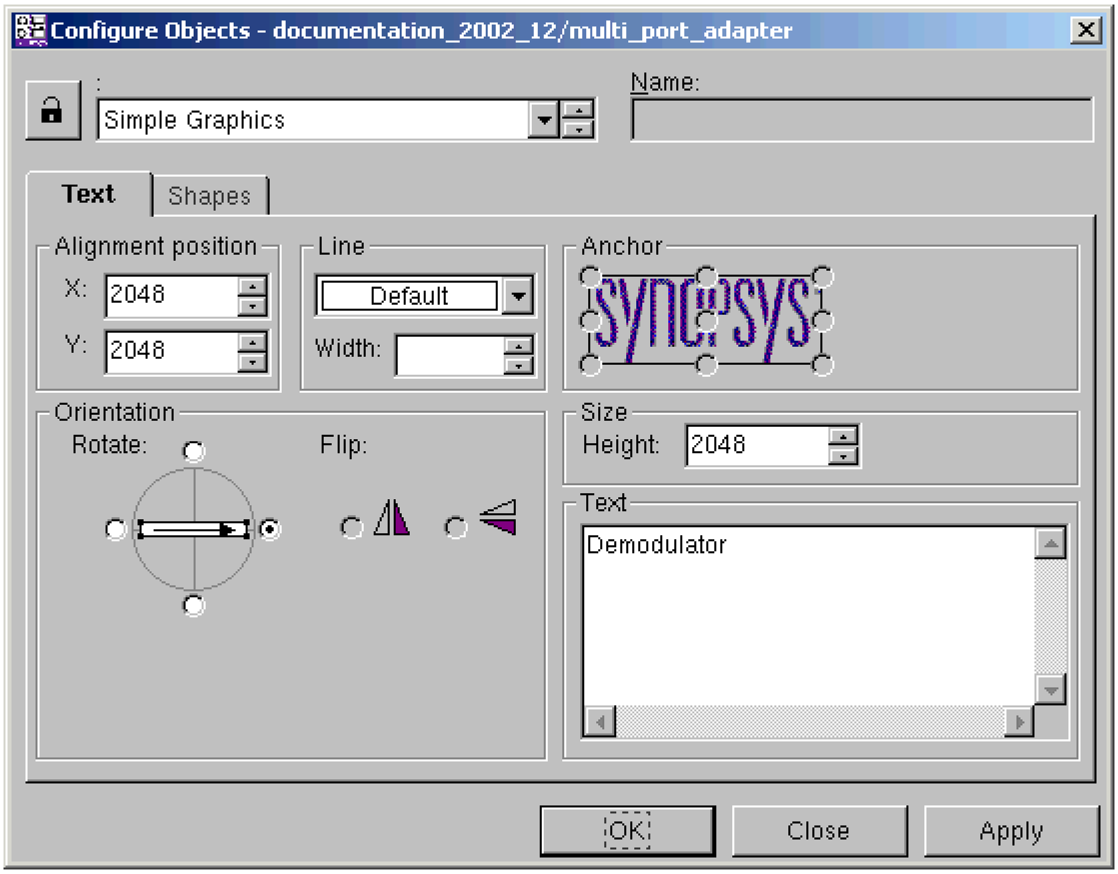
Orientation
Rotate:
Flip:

OK | Close | Apply

You can change the alignment of the graphics object, its orientation, the color used to draw its outline (line) and the color and pattern (if any) used to make the object solid (Fill). By default, graphic objects have no fill color or pattern so that they are transparent.

Select a text object (a text caption) and press the right mouse button. This displays the text object properties dialog box shown in Figure 2-29.

Figure 2-29 Setting the Text Object Properties



Configure Objects - documentation_2002_12/multi_port_adapter

: Simple Graphics

Name:

Text | Shapes

Alignment position
X: 2048
Y: 2048

Line
Default
Width:

Anchor

Orientation
Rotate:
Flip:

Size
Height: 2048

Text
Demodulator

OK | Close | Apply

You can change the alignment of the text caption, its orientation, and the color used to draw it (line). You can edit the text itself, you can change the size in pixels of the text, and you can specify which part of the caption (anchor) is to be used as the alignment point when the text snaps to a grid position.

3

Data Flow Model Tutorial

This tutorial describes the steps required to create a data flow model. It contains the following sections:

Data Flow Graph Models

In System Studio, you can create and configure block diagram schematics. You also create and edit block diagrams that combine both primitive models and hierarchical models to create hierarchical models. The hierarchical models that you create can then be placed as instances along with other primitive or hierarchical models in a data flow graph (DFG) model.

You can assign fixed or variable values to parameters that define the characteristics of individual model instances and of the model as a whole.

System Studio provides a wide selection of standard models that you can instantiate in your design. You can also add instances of user-written models. Each model, whether provided by System Studio or user-written, has one symbol associated with it.

For information on creating your own model and its symbol file or files, refer to the *CoCentric System Studio User Guide*.

Configuring model instances includes setting parameters and assigning I/O data sets. System Studio supports a wide range of parameter data types; these are detailed in the *CoCentric System Studio User Guide*. Model parameters can be

- Structural, which cannot be changed after code generation
- Read-on-reset, which can be changed whenever a model is reset
- Dynamic, which can be changed at any time
- Const, which cannot be changed after code generation and are not visible from outside the model

In addition to model parameters, you can declare type parameters. Type parameters give you the functionality of generic data types, but they can greatly enhance the reusability of a model because they allow a single model to be used for integer, fixed-point, single-precision, or double-precision floating-point variants.

In general, a model has input ports, output ports, or inout (input/output) ports. A data flow model cannot have inout ports.

The following connection rules apply to a complete system configuration:

- A net has exactly one source. A source can be either an outside source (an input port symbol) or an output port of a model instance.
- A net has one or more destinations. A destination can be either a port outside the schematic (an output port symbol) or an input port of a model instance.

In the following sections of this chapter you will learn how to create a very basic System Studio data flow model, simulate it, and display the data using the CoCentric DAVIS data visualization tool.

Creating the Model Structure

Before you can start a model design, you must create

- A workspace
- A library within the workspace
- A model within the library

Starting System Studio

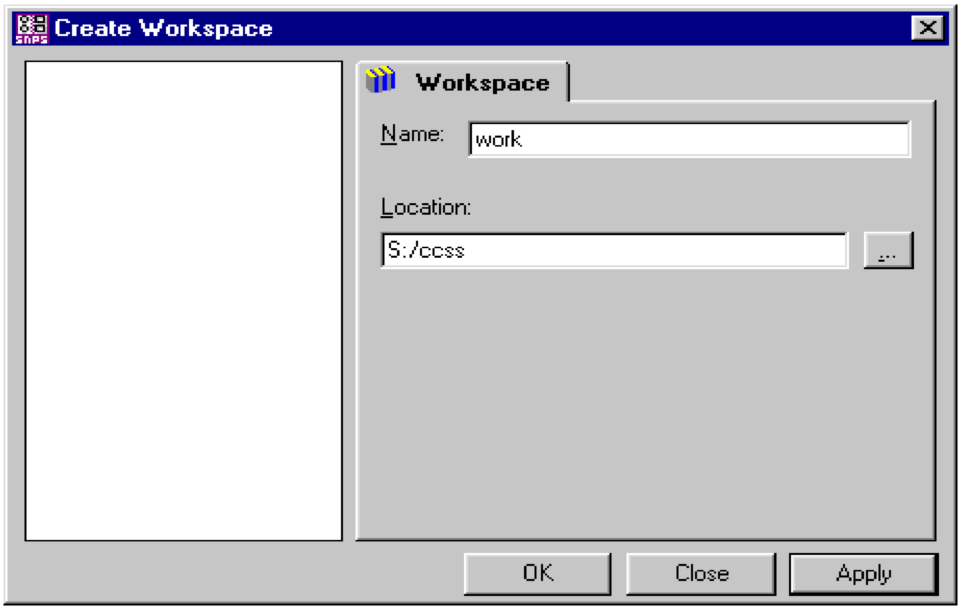
To start System Studio, enter the following command:

```
% ccss &
```

Creating a Workspace

Select File › New from the System Studio main menu bar. A dialog box appears. The Create Workspace dialog box, shown in Figure 3-1, is displayed first.

Figure 3-1 Creating a Workspace



Name: work

Location: S:/ccss

The Name field contains the workspace name “work.” Change the name if you want. The file that defines your workspace is stored in the path indicated in the Location field.

Click Apply to create the workspace (the Create Workspace dialog box will remain active until you click OK or close).

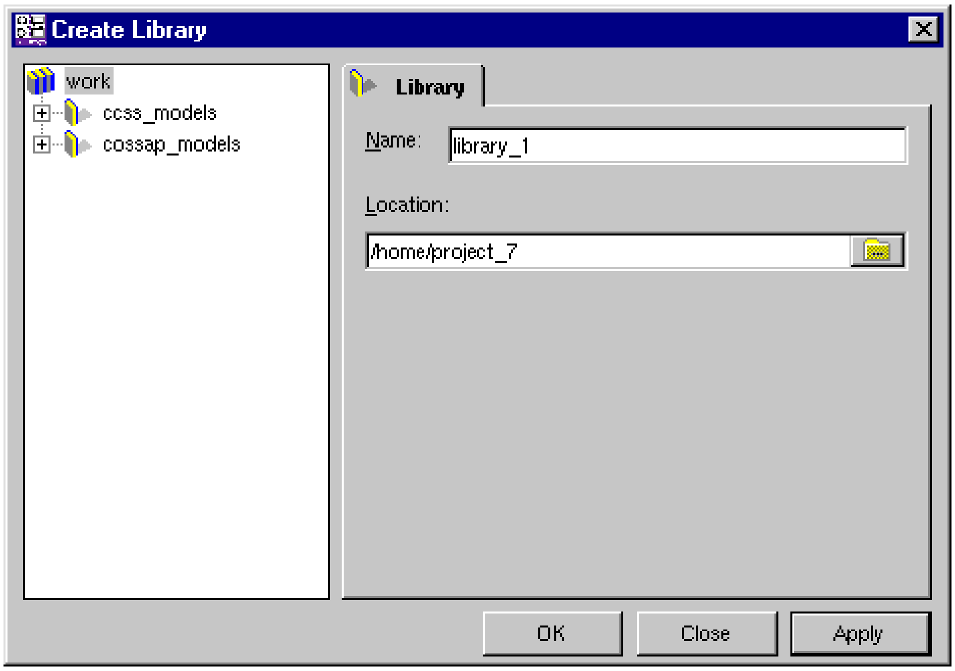
Creating a Library

In the Create Workspace dialog box, with a workspace selected, click the Library tab. The dialog box changes to the Create Library dialog box, as shown in Figure 3-2.

Create a user library under the workspace that you just created. Enter a name in the Name field or use the default name. You can specify a different location for the library directory in the Location field.

After you enter a library name (or accept the default name), click Apply.

Figure 3-2 Creating a Library



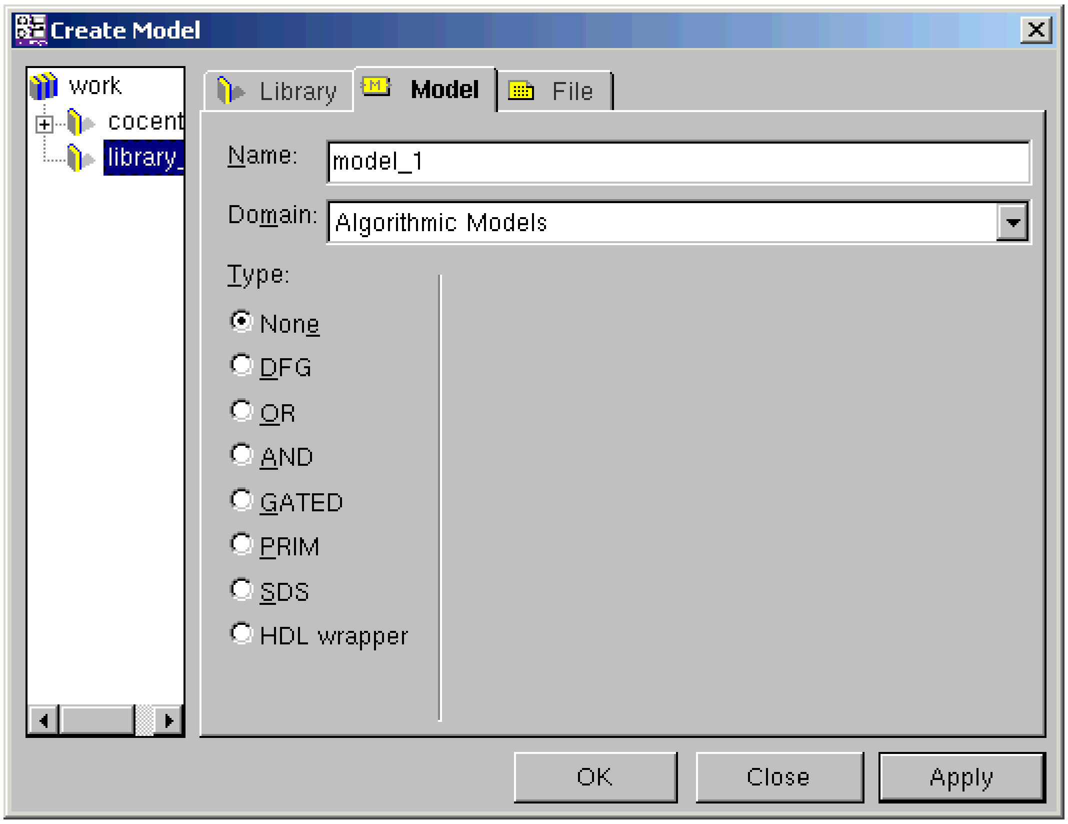
Name: library_1

Location: /home/project_7

Creating a Model

In the Create Library dialog box, select a library and click the Model tab. The dialog box changes to the Create Model dialog box, as shown in Figure 3-3. If you do not select a library, the Model tab will not be visible.

Figure 3-3 Creating a Model



Name: model_1

Domain: Algorithmic Models

Type:
[✓ None]
DFG
OR
AND
GATED
PRIM
SDS
HDL wrapper

Enter a name and location for your model or use the defaults.

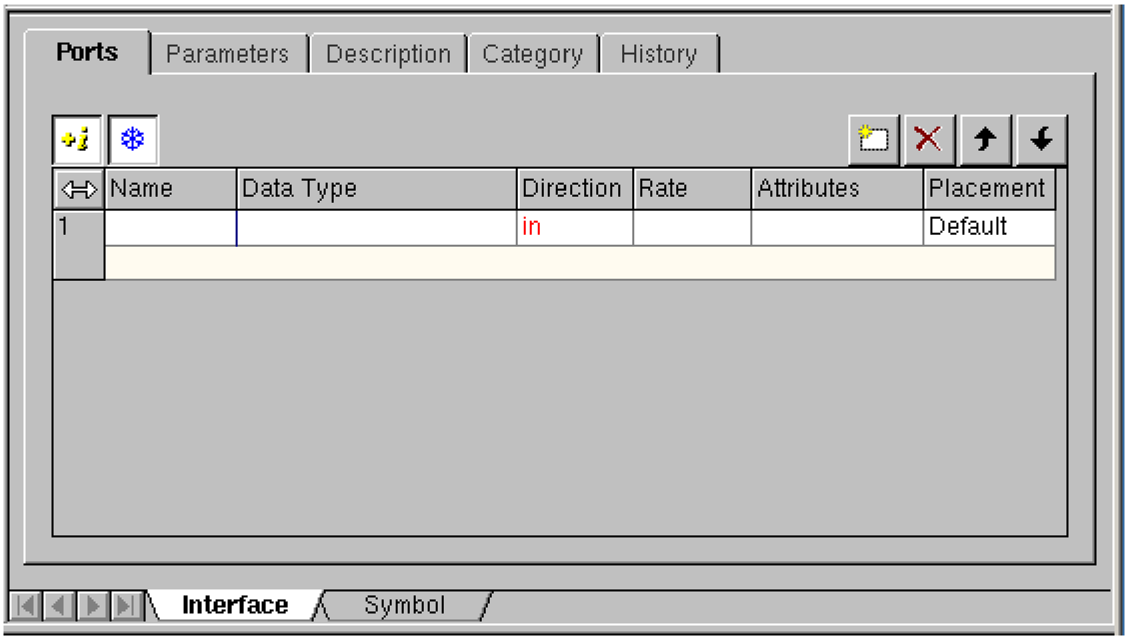
Choose the Algorithmic domain.

If you want to define the implementation, choose the relevant implementation by selecting the appropriate Type radio button, otherwise select None. When you are ready, click Apply.

If you select an implementation, the Implementation view opens with the appropriate type of implementation loaded (for example, for a primitive model a skeleton source code file is opened).

If you select None as the model type, the Interface view is displayed (the Interface tab is raised) and there is no Implementation view, as shown in Figure 3-4.

Figure 3-4 Viewing the Model Interface



	Name	Data Type	Direction	Rate	Attributes	Placement
1			in			Default

Notice that the workspace window displays the complete hierarchy of the model. When the model has an implementation, the model branch is composed of three components: an Interface view, an Implementation view, and a Symbol view. If you left the Type as none in the create model dialog box, there will be no Implementation view yet.

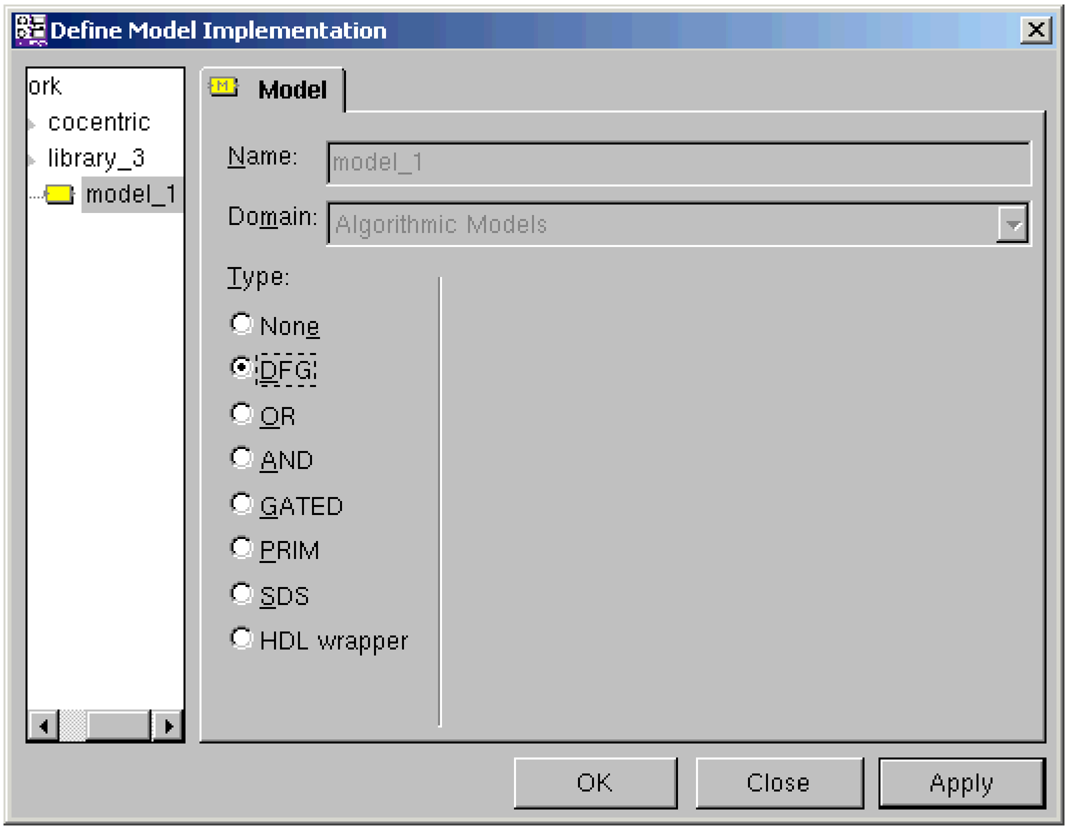
Defining the Model Behavior

You define the model behavior by using the Define Implementation dialog box.

- Choose Model › Redefine Implementation from the main menu. This method allows you to redefine a previously set implementation.

In the Define Model Implementation dialog box, select DFG, as shown in Figure 3-5, and then click OK.

Figure 3-5 Defining the Model Implementation



Name: model_1
Domain: Algorithmic Models
Type:
None [✓ DFG] OR AND GATED PRIM SDS HDL wrapper

The schematic editor is opened in the design window (the Implementation tab is raised). Note that the Schematic toolbar is now active, from which you can choose a block, port or net to be added to the design.

Creating the Model Schematic

The DFG model you are going to enter is a simple model that consists of two blocks: a sine wave generator and a sink. Both are existing models contained in the System Studio libraries.

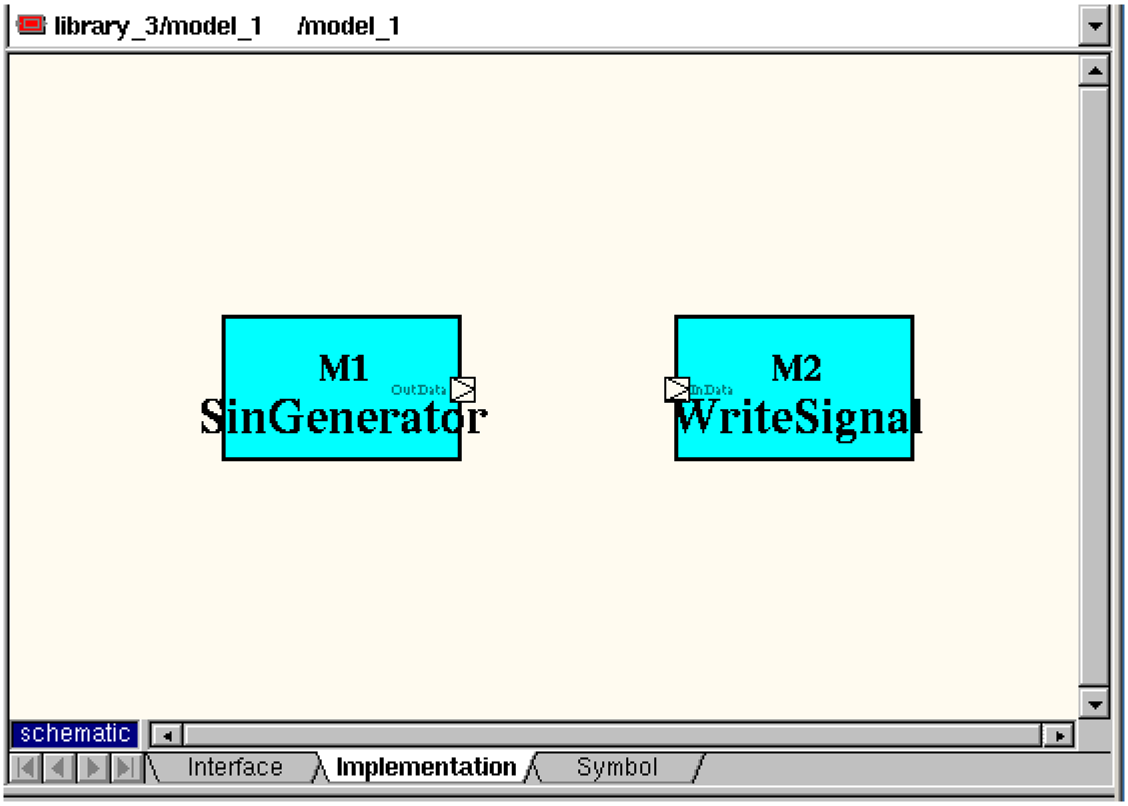
To create the model schematic,

1. If the “cocentric” library is collapsed, expand the library by clicking the plus sign (+) next to it in the Workspace window; expand the “algorithm” library and then select the “source” library. The models in this library are now listed in the Model area.
2. Click the Insert Instance button in the Schematics toolbar.
3. Select the SinGenerator model from the “source” library located in the algorithm library.
4. Move the cursor into the design area. You will see the outline of the block displayed as you move the cursor around.

   Click the left mouse button when the outline of the block is positioned where you want to place the block.
5. Now move the cursor back into the Workspace window and select the WriteSignal block from the System Studio models “sink” library located in the algorithm library.
6. Position the WriteSignal block to the right of the SinGenerator block, as shown in Figure 3-6, and click the left mouse button.

7. Click the middle mouse button to cancel the insert Instance function.

Figure 3-6 Inserting Blocks



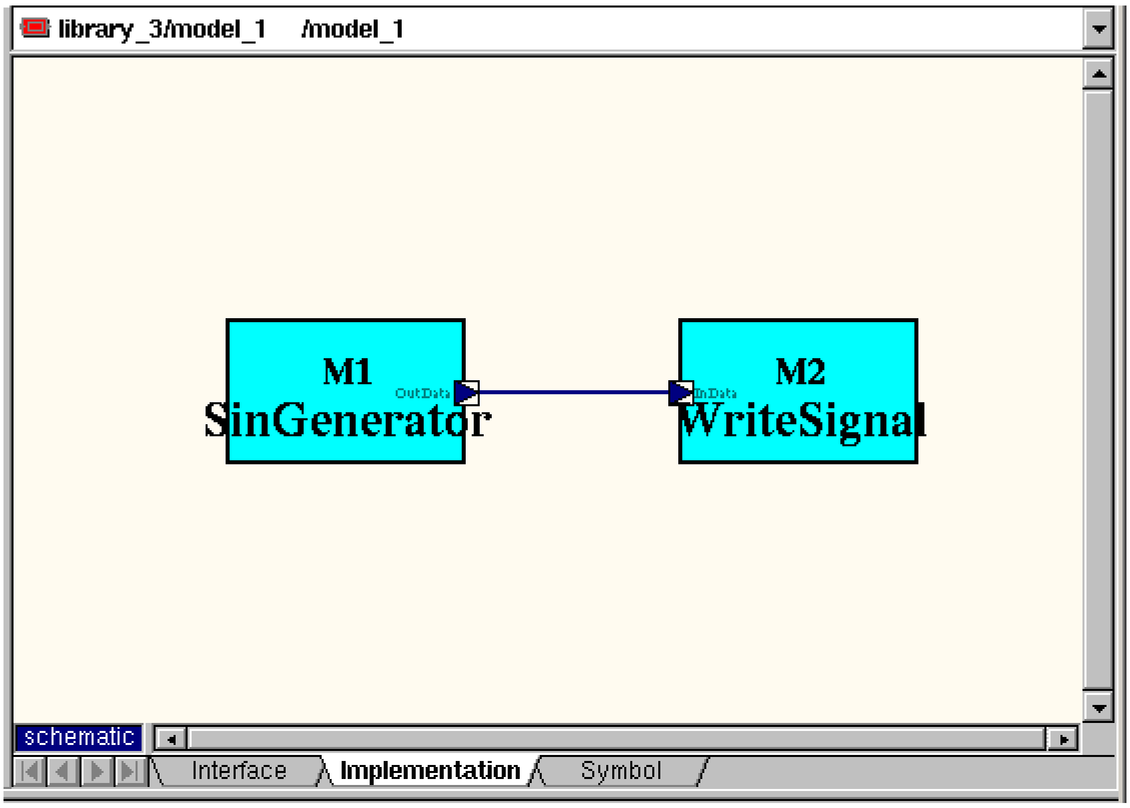
8. The next step is to connect the two blocks together with a net. Click the Insert Net button in the Schematic toolbar.

9. Position the cursor over the output port of the SinGenerator block. Press and hold the left mouse button, and create a net by dragging a line to the input port of the WriteSignal block.

10. Release the mouse button; the two blocks are now connected. The end result should look similar to the example shown in Figure 3-7.

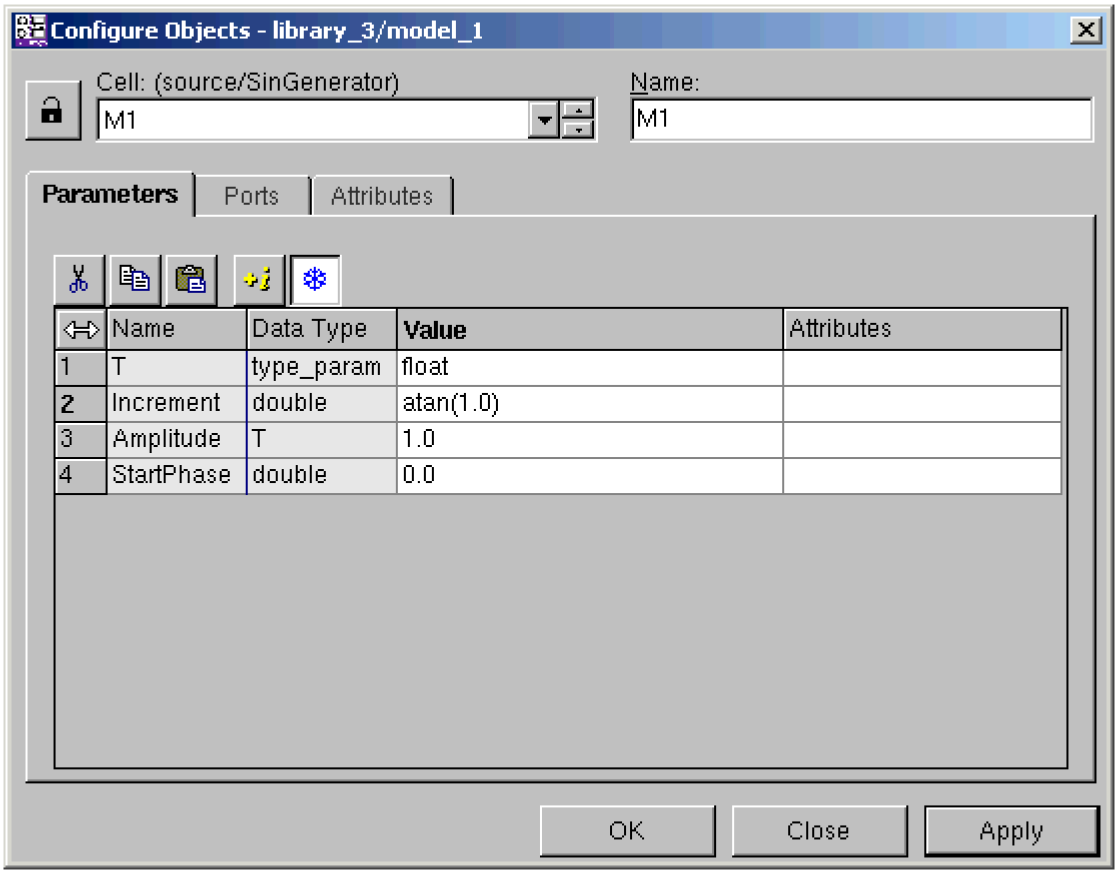
    Click the middle mouse button to cancel the Insert Net function.

Figure 3-7   View After Connecting the Blocks



11. To finish the design, the blocks have to be parameterized. Position the mouse pointer over instance M1 (the SinGenerator model) and click the right mouse button. Choose Configure Object from the context-sensitive menu. The Configure Objects dialog box is displayed as shown in Figure 3-8.

Figure 3-8   Setting the Model Parameters


Configure Objects - library_3/model_1
Cell: (source/SinGenerator)
M1
Name:
M1
Parameters
Ports
Attributes
OK
Close
Apply

	Name	Data Type	Value	Attributes
1	T	type_param	float	
2	Increment	double	atan(1.0)	
3	Amplitude	T	1.0	
4	StartPhase	double	0.0	

For your convenience, some of the parameters have default values (for example, in this model the WriteSignal instance is completely configured by the default parameter values).

12. Assign the expression `atan(1.0)` to the parameter "Increment" for the SinGenerator instance M1, and click OK.

The schematic design is now complete. In the following sections you will learn how to simulate the model and display the output data set.

Simulating the Model

You can create the executable simulation either automatically or manually.

Automatically Building the Simulation

Instead of entering all the steps manually, you can perform the steps required to build and start the simulation automatically.

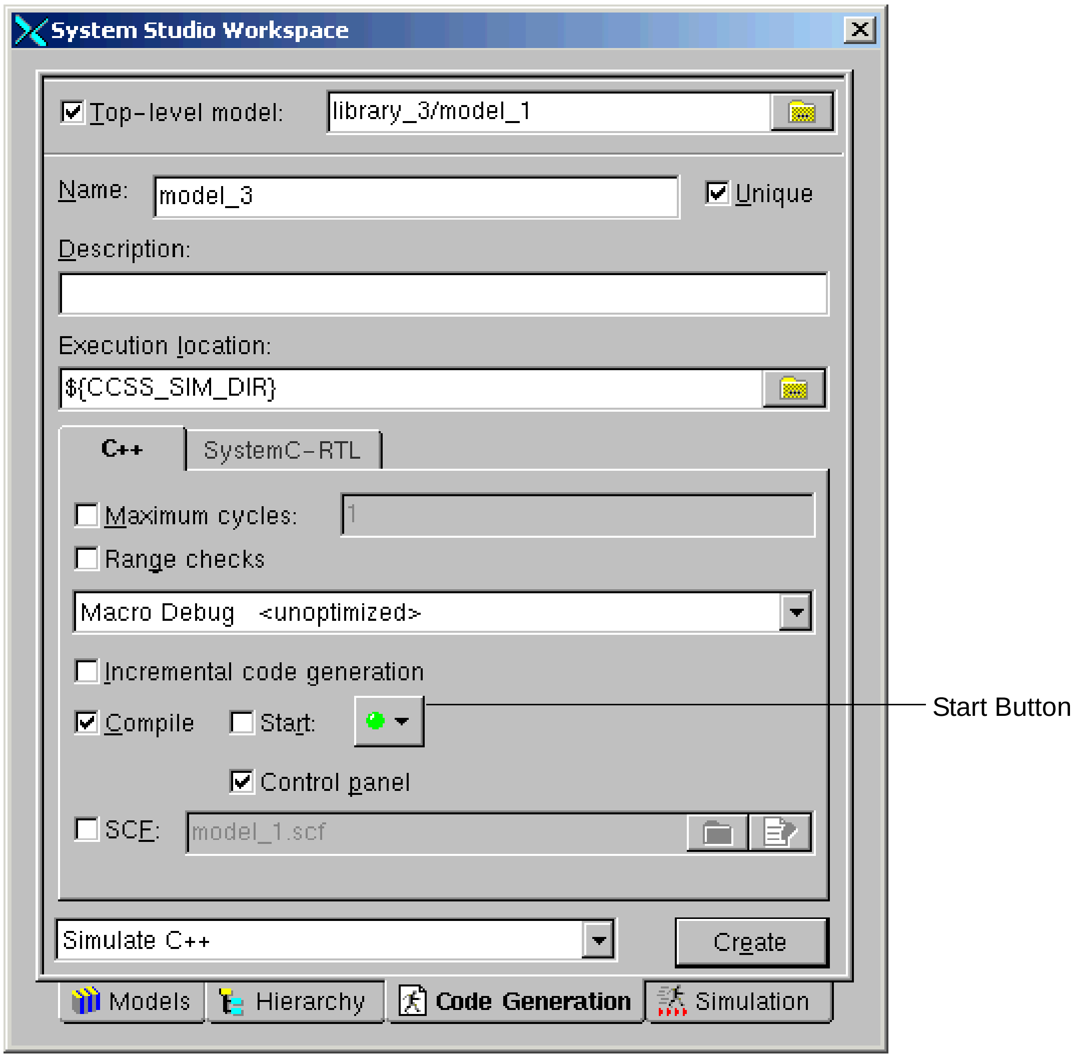
Click the Simulate the Model button or choose Simulation › Simulate Model from the menu bar. All the steps will be performed automatically. These steps include creating the optimized executable code, compiling and linking the model, and starting the simulation in pause mode.

Manually Building the Simulation

To create the simulation manually,

1. Click the Code Generation tab, make your selections, then click Create (see Figure 3-9).

Figure 3-9 Controlling the Code Generation



☑ Top-level model: library_3/model_1

Name: model_3 ☑ Unique

Description:

Execution location: ${CCSS_SIM_DIR}

☐ Maximum cycles: 1

☐ Range checks

Macro Debug <unoptimized>

☐ Incremental code generation

☑ Compile ☐ Start:

☑ Control panel

☐ SCF: model_1.scf

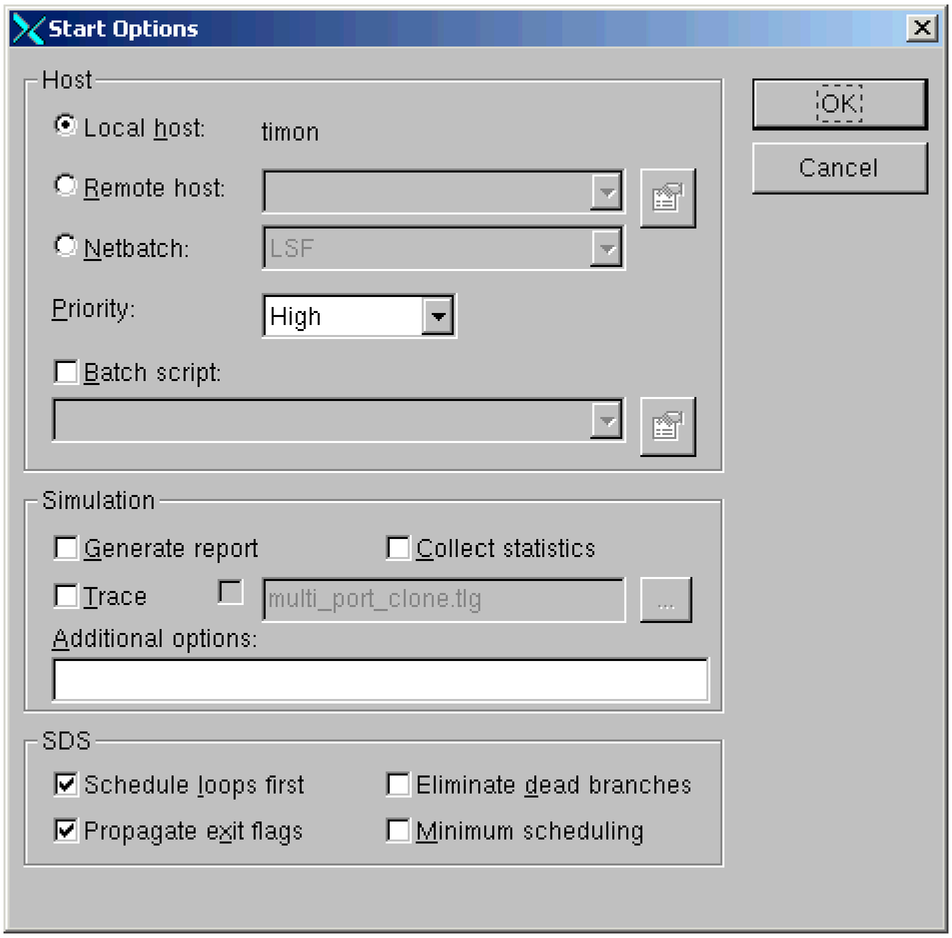
Simulate C++

2. If you want to view some of the simulation execution options, click the Start button, as shown in Figure 3-9, to display the menu.

3. Click Options to open the Start Options dialog box as shown in Figure 3-10.

4. After you have reviewed the options, click OK in the Start Options dialog box, and then click Create in the Code Generation page to execute the simulation.

Figure 3-10 Controlling the Simulation Execution


Start Options

Host
- [✓ Local host:] timon
- Remote host:
- Netbatch: LSF
- Priority: High
- ☐ Batch script:

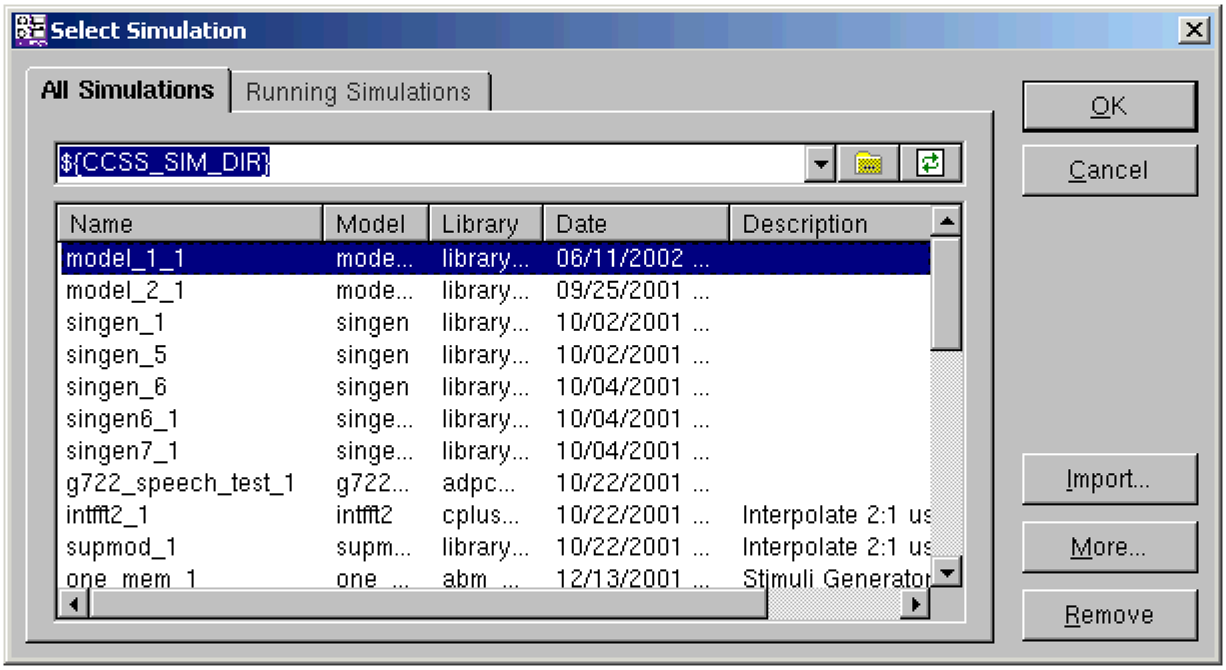
Simulation
- ☐ Generate report
- ☐ Collect statistics
- ☐ Trace
- ☐ multi_port_clone.tlg
- Additional options:

SDS
- ☑ Schedule loops first
- ☐ Eliminate dead branches
- ☑ Propagate exit flags
- ☐ Minimum scheduling

Controlling the Simulation

To control the simulation, choose Simulation › Open Control Panel. The Select Simulation dialog box is displayed as shown in Figure 3-11.

Figure 3-11   Selecting the Simulation



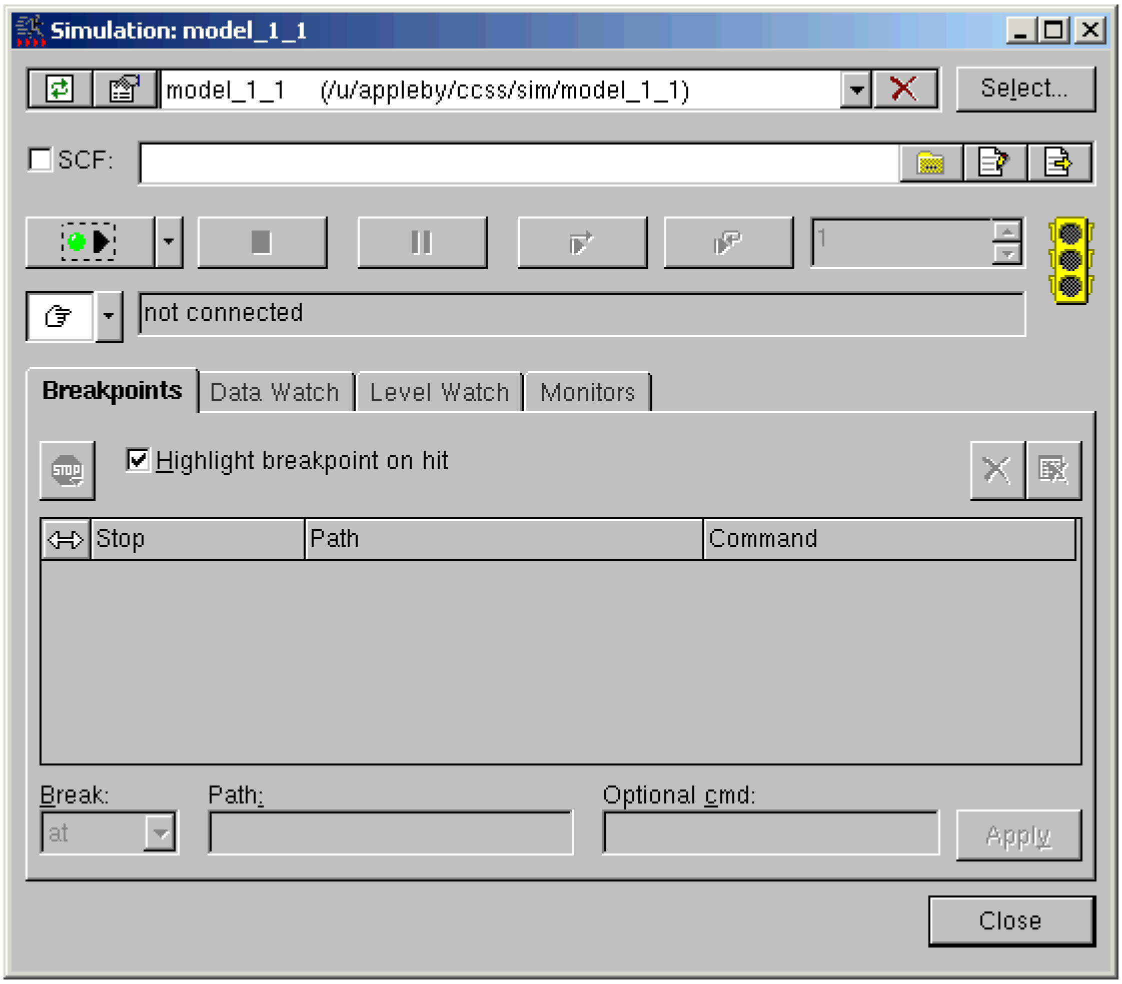
${CCSS_SIM_DIR}

Name	Model	Library	Date	Description
model_1_1	mode...	library...	06/11/2002 ...	
model_2_1	mode...	library...	09/25/2001 ...	
singen_1	singen	library...	10/02/2001 ...	
singen_5	singen	library...	10/02/2001 ...	
singen_6	singen	library...	10/04/2001 ...	
singen6_1	singe...	library...	10/04/2001 ...	
singen7_1	singe...	library...	10/04/2001 ...	
g722_speech_test_1	g722...	adpc...	10/22/2001 ...	
intfft2_1	intfft2	cplus...	10/22/2001 ...	Interpolate 2:1 us
supmod_1	supm...	library...	10/22/2001 ...	Interpolate 2:1 us
one_mem_1	one ...	abm ...	12/13/2001 ...	Stimuli Generator

Select the simulation that you created and click OK.

The simulation control panel is opened as shown in Figure 3-12.

Figure 3-12   The Simulation Control Panel



The Simulation control panel not only gives you full control over the simulation, it allows you to set breakpoints and watch points in the code execution, just as you would expect from any debugging tool.

You can display and modify the values of most objects, such as nets, at any level of the model. Values that you cannot modify are shown dimmed in the list. The fields that you can modify have a white background.

You can examine nets, ports, parameters, and variables anywhere in the hierarchy of the active instance.

You can also, through a command-line interface, enter text commands to directly control the simulation without using any part of the graphical user interface. For more information on the different methods of controlling a simulation, see the *CoCentric System Studio User Guide*.

Click the Start Model Simulation button in the simulation toolbar, or the green start simulation button in the simulation control panel to start the simulation now.

A message will tell you when the simulation is successfully completed.

Displaying System Studio Data

The CoCentric data visualization tool (DAVIS) enables you to visually analyze and graphically display and postprocess data. It also includes a calculator to interact with graphs and data, and gives you the ability to interact with a System Studio simulation.

The data can originate from simulation results, data set information, or other data that you specify. You can add text and labels, and print and save graphs.

You can also use DAVIS to display data generated by other tools as well as for displaying data from System Studio simulations.

Alternatively, you can use the VirSim tool, which allows you to display the data from System Studio and also allows you to control the System Studio simulation.

Graphical Display

You can use any one DAVIS sheet to display multiple, overlaid graphs, where each graph is a single representation of data. For example, you can portray the results of several simulations in one sheet and compare it with a theoretical result that you specified in a function equation. You can choose how to visually portray the data or graph so that it is most intuitive and helpful for analysis. You can choose a line plot, scatter diagram, eye diagram, histogram, or logic signal display.

You can modify many cosmetic aspects of the graph appearance, such as whether and how a histogram is filled, and the type of symbols DAVIS uses as markers.

DAVIS can automatically generate text and labels, or you can create them yourself. You can also specify the color, font, height, and orientation of the text (at an angle, vertical, or horizontal).

You can delete a graph and merge separate line or scatter graphs into a single graph.

Acquiring and Interpreting Data

To retrieve visual results from a simulation, specify the simulation name, the desired data set, and the simulation iteration, if applicable. The System Studio Design Center associates a default plot format with a schematic when you create it; therefore, you do not need to specify a plot type to display an intuitive graph when you use DAVIS. For example, you do not have to specify the file name for the data set, the scaling, the type of diagram, the dimension type (time versus sample), or the axis labels.

The ease of displaying results, however, does not restrict you from plotting the same data in other ways, or from specifying attributes; flexibility is never sacrificed. You can also portray a subset of data-set elements. For example, you can use only every fourth element (instead of every one) to construct a graph.

Displaying the Output Data Set

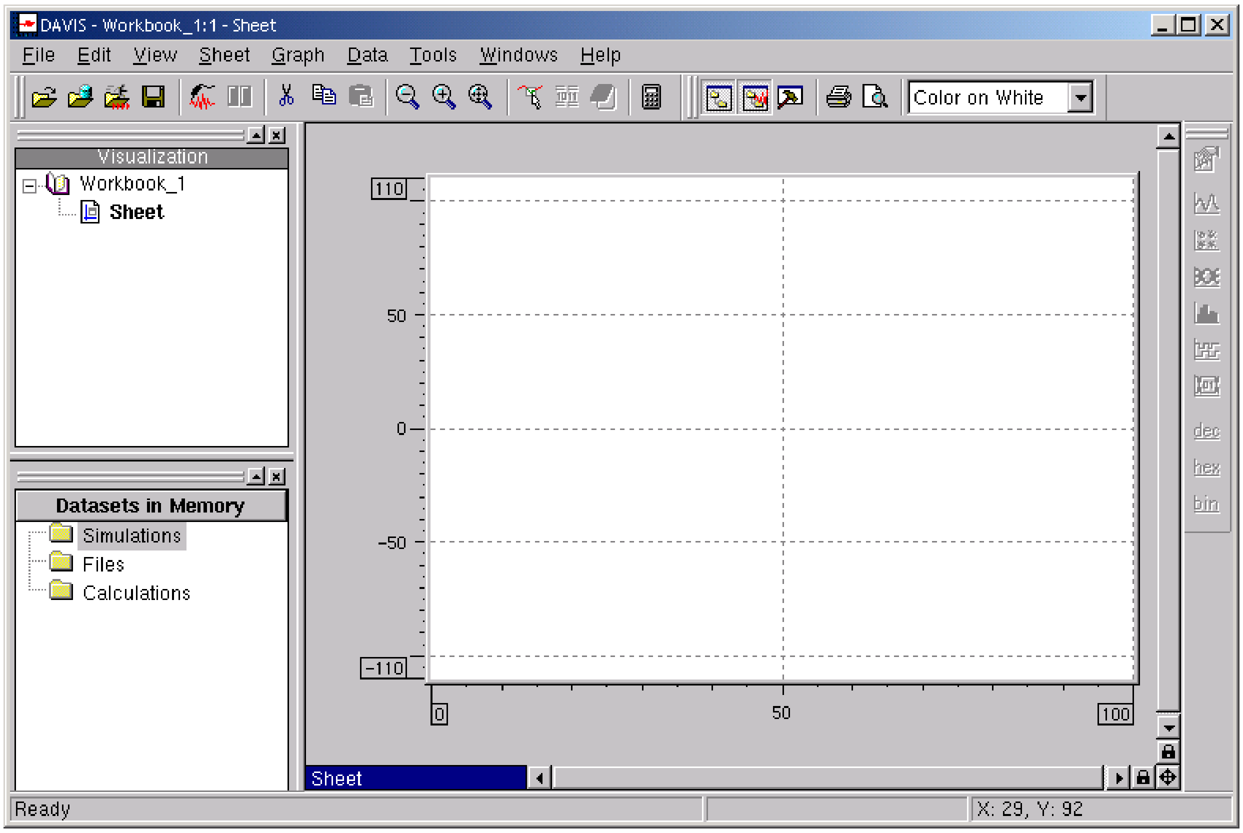
As well as being able to connect to a running simulation and display the data dynamically, DAVIS supports various binary and ASCII file formats. You can plot your own data, and DAVIS can interpret files to contain x, y, or x-y data, because it can also automatically generate values along either the x-axis or y-axis.

In the dataflow model described in this chapter, the WriteSignal model is used to write the data to a simulation data-set file. In this section, you will learn how to use DAVIS to open that file and display the simulation data contained in it.

To display the simulation data set,

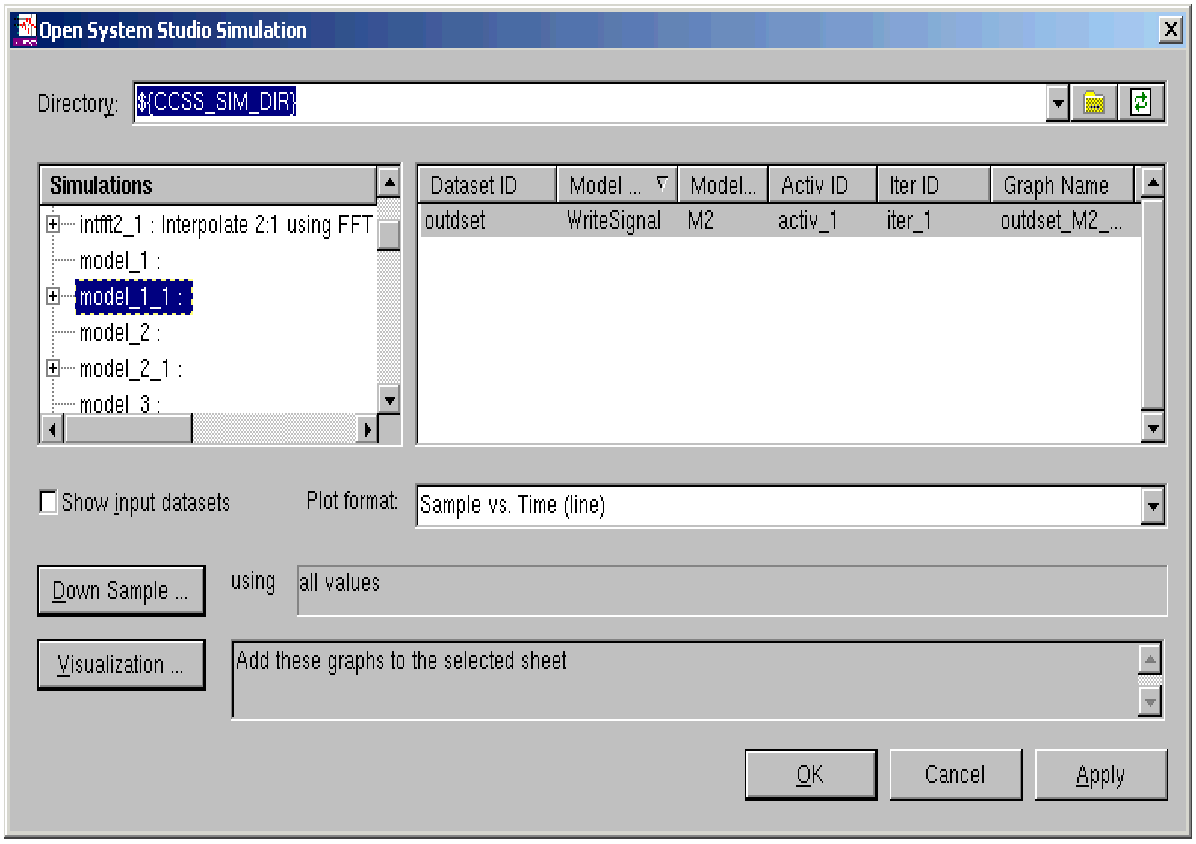
1. Choose Tools › Davis from the Design Center menu bar. The DAVIS tool opens with the default workbook as shown in Figure 3-13.

Figure 3-13 The Default DAVIS Workbook Display



2. In DAVIS, choose File › Open System Studio Simulation from the menu bar. The Open System Studio dialog box, shown in Figure 3-14, is displayed.

Figure 3-14   Opening a System Studio Simulation in DAVIS


Open System Studio Simulation

Directory: ${CCSS_SIM_DIR}

Simulations
- intfft2_1 : Interpolate 2:1 using FFT
- model_1 :
- model_1_1 :
- model_2 :
- model_2_1 :
- model_3 :

Dataset ID	Model ...	Model...	Activ ID	Iter ID	Graph Name
outdset	WriteSignal	M2	activ_1	iter_1	outdset_M2_...

☐ Show input datasets   Plot format: Sample vs. Time (line)

Down Sample ...   using   all values

Visualization ...   Add these graphs to the selected sheet

OK   Cancel   Apply

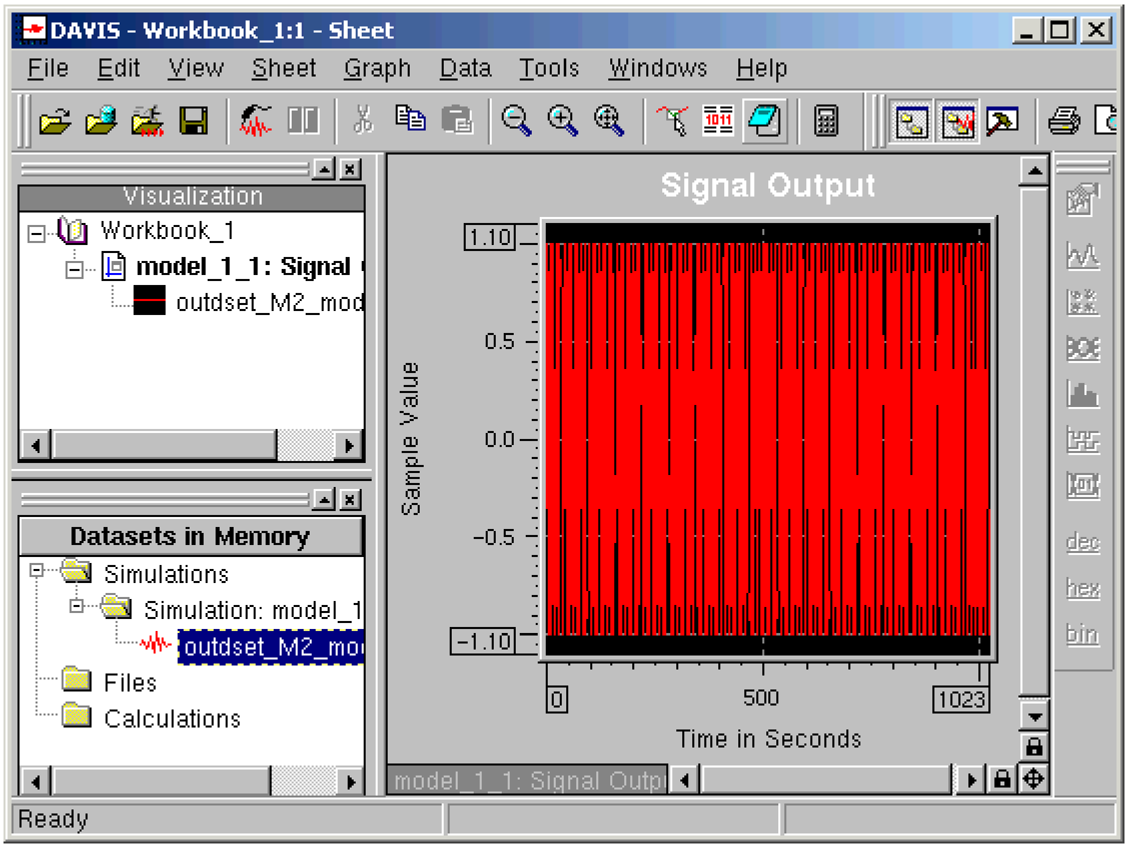
3. Navigate through the directories until you find the simulation data file you want, as shown in Figure 3-14.

   Note:

   If no output dataset (outdset) is available, then it is possible that you did not run your simulation to completion. Return to the simulation control panel and click the Pause/Continue button to complete the simulation. You will get a message to tell you that the simulation has finished. To learn how to connect to a running simulation, see the *CoCentric DAVIS User Guide.*

4. After you select the correct file, click OK and the simulation data will be displayed in a DAVIS workbook, as shown in Figure 3-15.

Figure 3-15 The Data File Displayed in DAVIS



Note:

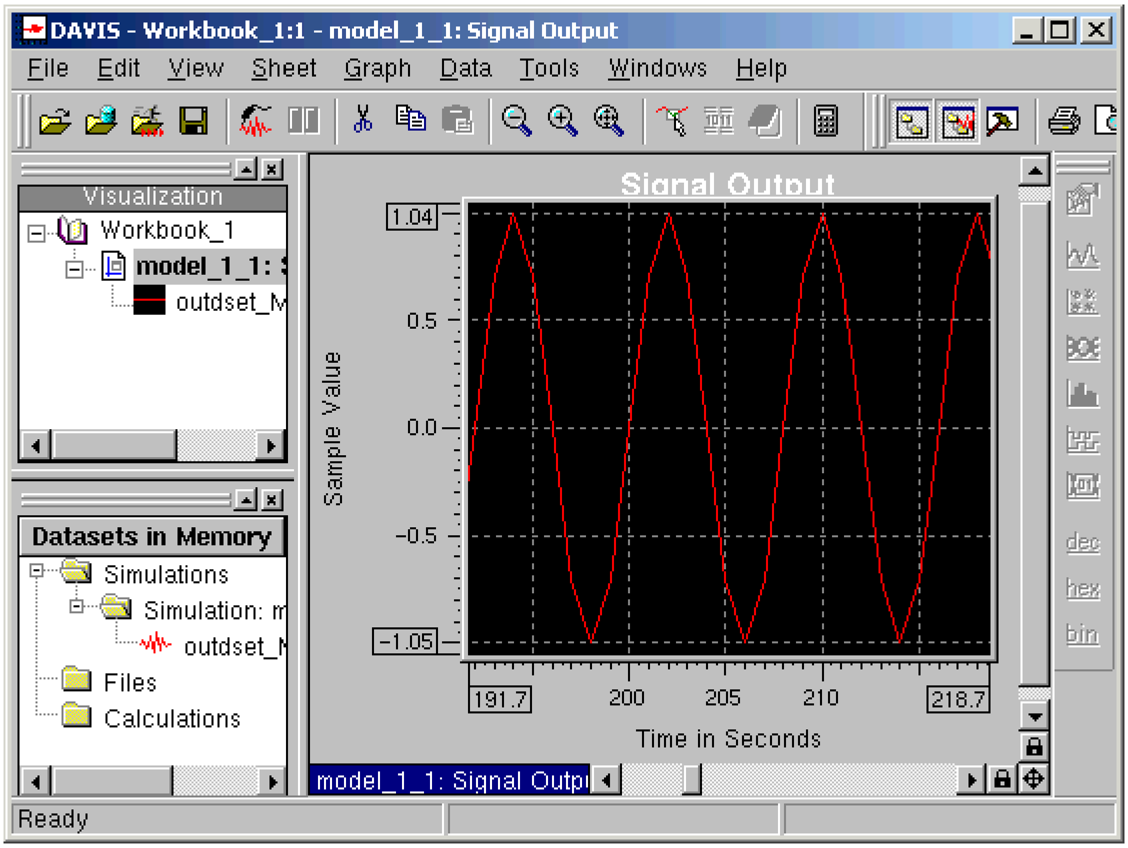
System Studio stores its files in the directory specified by the variable $CCSS_SIM_DIR. The data file for a specific simulation can then be found in

```
$CCSS_SIM_DIR/simulation_name/activ_m/iter_n/*.*
```

where *simulation_name* is the name of the simulation, *m* is the number of the activation (this is incremented by 1 each time you run a simulation), and *n* is the number of the simulation iteration.

You can now manipulate and modify the data-file display in DAVIS, performing sophisticated calculations with the data or simply zooming in to display more detail, as shown in Figure 3-16.

Figure 3-16 Zooming the Simulation Data Graph Display



Creating and Viewing Model Documentation

Creating the documentation for a model can be a tedious and time-consuming task. System Studio enables you to create the documentation for a model with the click of a single button. Using

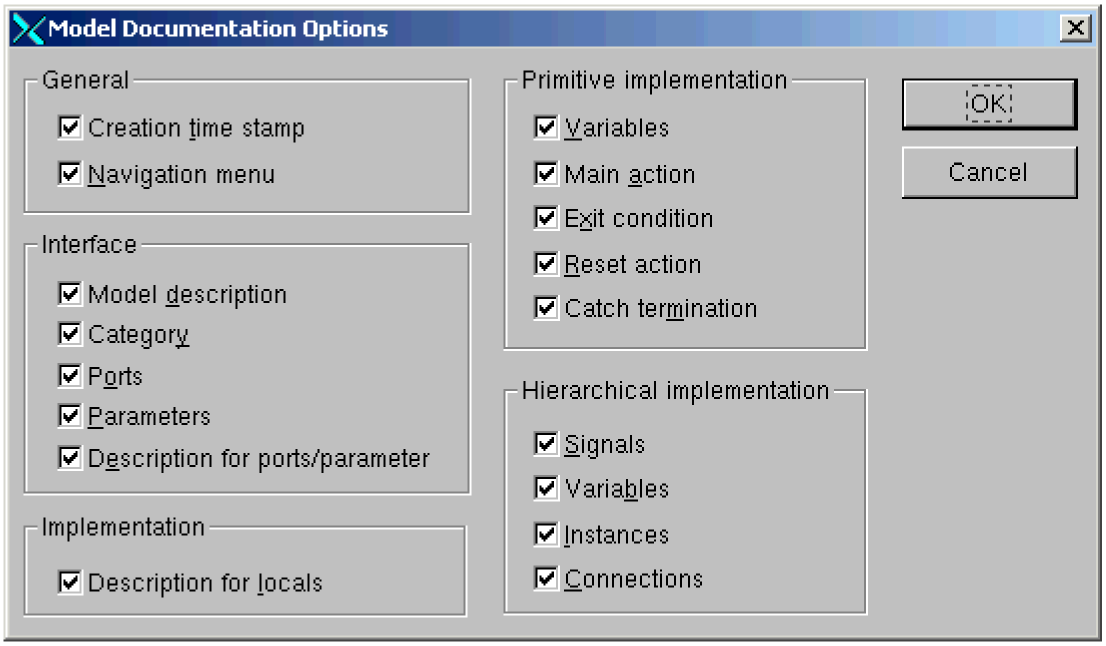
System Studio documentation tools, you can also display complete documentation for an existing System Studio model or a complete library.

Creating Model Documentation

To generate documentation for your model, choose Options › Model Documentation from the System Studio menu bar. The Model Documentation Options dialog box is displayed, as shown in Figure 3-17. This dialog box gives you full control over the contents of the documentation that will be generated.

Select a few options and click OK.

Figure 3-17 Creating the Model Documentation



Model Documentation Options

General
- ☑ Creation time stamp
- ☑ Navigation menu

Interface
- ☑ Model description
- ☑ Category
- ☑ Ports
- ☑ Parameters
- ☑ Description for ports/parameter

Implementation
- ☑ Description for locals

Primitive implementation
- ☑ Variables
- ☑ Main action
- ☑ Exit condition
- ☑ Reset action
- ☑ Catch termination

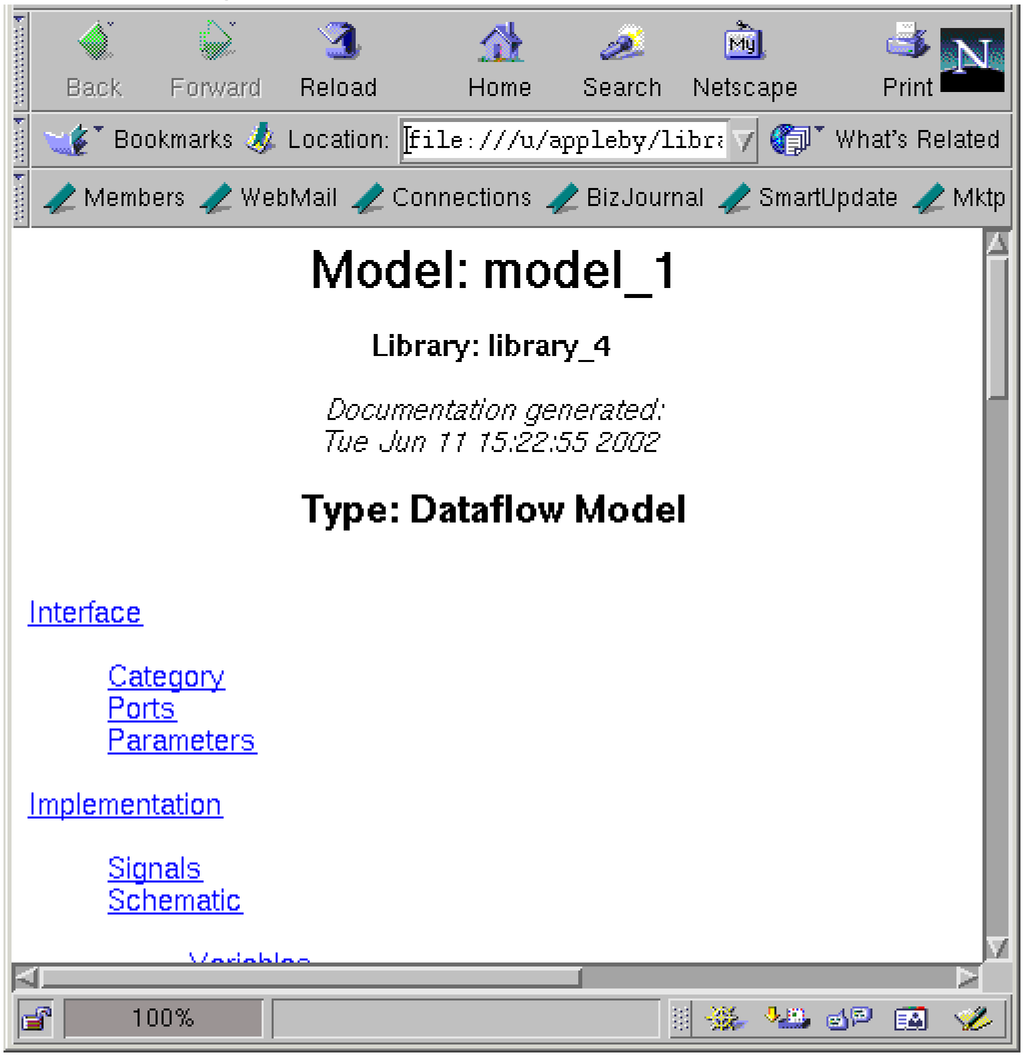
Hierarchical implementation
- ☑ Signals
- ☑ Variables
- ☑ Instances
- ☑ Connections

The model documentation is created in HTML, which you can display and print using an HTML browser such as Netscape Navigator.

Viewing Model Documentation

Choose Model › Show Documentation to display the documentation for the model after you have created it (see Figure 3-18).

Figure 3-18 Viewing the Model Documentation



Model: model_1

Library: library_4

Documentation generated:
Tue Jun 11 15:22:55 2002

Type: Dataflow Model

Interface

Category
Ports
Parameters

Implementation

Signals
Schematic

Click Close to close the HTML browser.

This completes this tutorial. You can now save your workspace and exit System Studio at this point if you want.

4

OR Model Tutorial

This tutorial describes the steps required to create the design for an OR model. It contains the following sections:

The OR Model

An OR model represents a finite state machine (FSM). Atomic states are the leaf-level objects within an FSM. Actions can be associated with both states and transitions.

In System Studio, the start and exit states of an FSM are implied; no special atomic states are required to identify them, and there are no specific start or exit transitions. Instead, a transition from a state that has no other state as its destination is assumed to be the exit transition, and a transition that has no starting state is assumed to be the start transition.

The states of OR models can be either hierarchical or atomic. Hierarchical states can contain other OR models, or any other type of System Studio model. Atomic states can have inlined actions and can manipulate the values of local variables, local signals, or output ports.

A transition can have a condition, an action, and a priority associated with it. Conditions must evaluate to true (nonzero) or false (zero). Actions can modify the values of local variables, local signals, or output ports. The priority of a transition determines its precedence with respect to other transitions (1 is the highest priority, 5 is the lowest priority).

Transitions can be either strong, weak, or exit-handling. The nature of a transition also determines the priority in which the transitions will be evaluated, but it is primarily concerned with how lower-level models in the hierarchy will be evaluated.

- Weak transitions allow the underlying lower-level models to be evaluated before the transition itself is executed. A weak transition can be thought of as a soft interrupt. Use weak transitions for defaults.
- Strong transitions forcibly terminate the underlying lower-level models without allowing any kind of cleanup actions to be performed (hierarchical states do not terminate in a controlled manner). A strong transition can be thought of as a hard interrupt.
- Exit-handling transitions catch exits from underlying hierarchical models (as long as any condition on the transition is satisfied) if they are not passed to the model instance above. The behavior of an exit-handling transition is analogous to exceptions being thrown and caught in languages such as C++ and Java.

In addition, a transition can also be specified as immediate. Immediate transitions allow multiple transitions to be followed within a single execution step. States that are passed through during an immediate transition are reset (this is not relevant for an atomic state) and perform no other actions.

To resolve possible conflicts between transitions, a transition can also have a numeric priority value, explicitly specified for it, ranging from 1, highest priority, to 5, lowest priority.

Creating the Model Structure

Before you can start a model design, you must create

- A workspace
- A library within the workspace
- A model within the library

Starting System Studio

To start System Studio, enter the following command:

```
% ccss &
```

Creating a Workspace

In the Design Center, choose File › New from the menu bar. Type the name of your new workspace in the dialog box and click OK

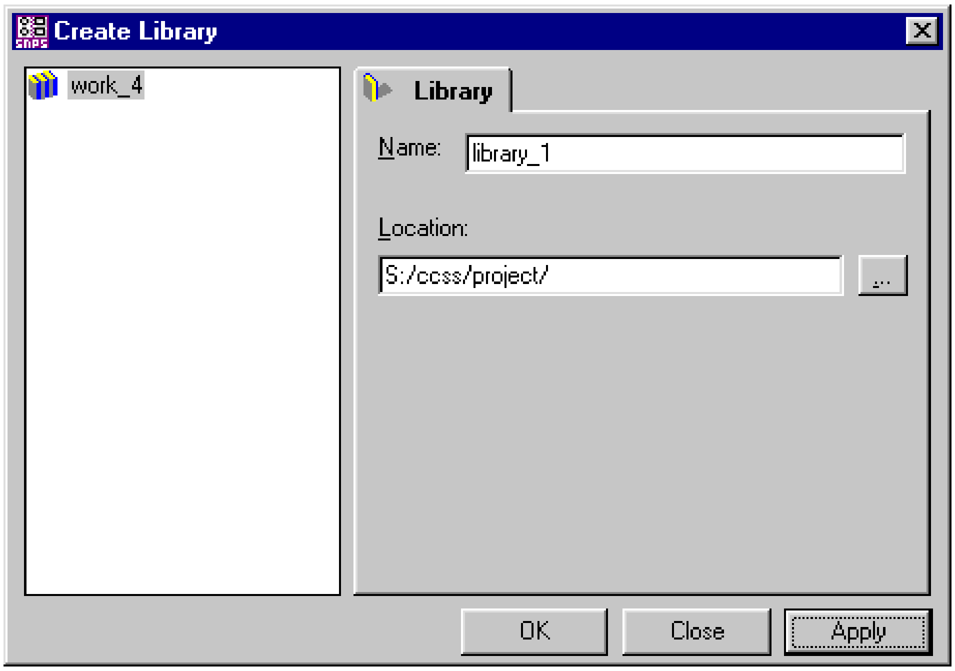
Creating a Library

Choose File › New to access the Create Library dialog box, as shown in Figure 4-1.

Create a user library under the workspace that you just created. Enter a name in the Name field or use the default name. You specify the location of the library directory in the Location field.

After you enter a library name (or accept the default name), click Apply.

Figure 4-1 Creating a Library



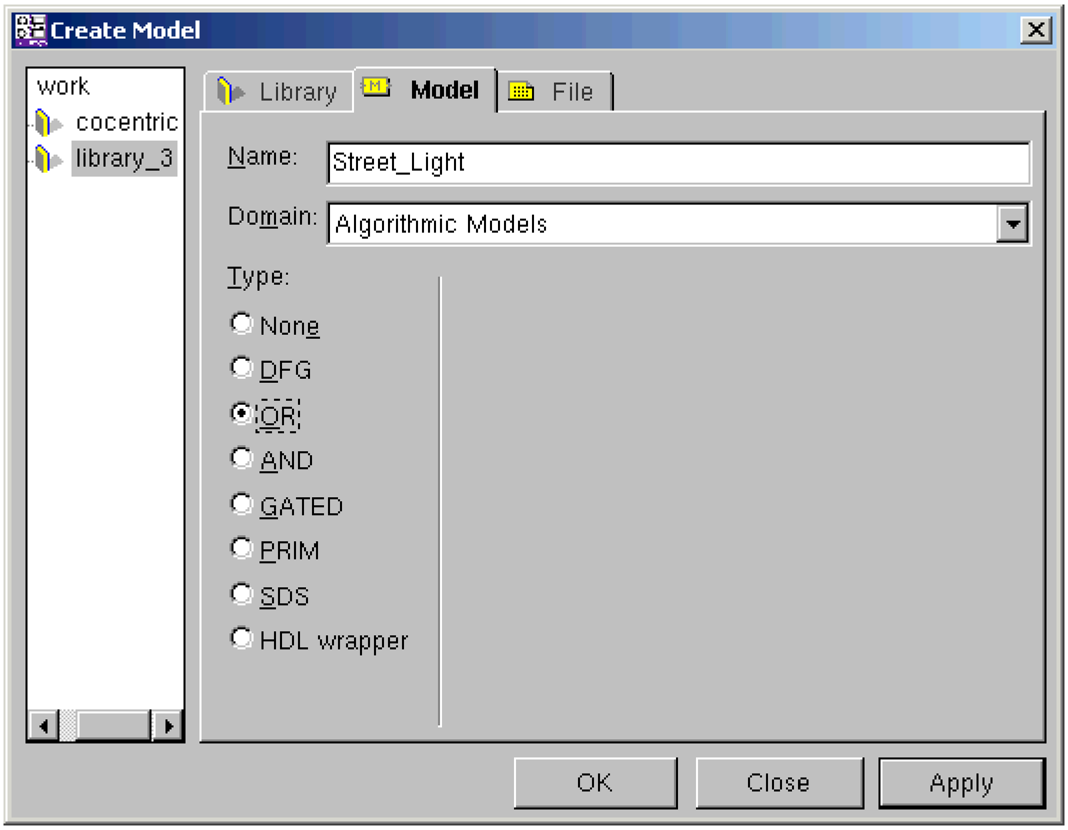
Name: library_1

Location: S:/ccss/project/

Creating a Model

Select the library in which you want to create the model (the Model tab will not be displayed if a library is not selected). Click the Model tab to display the Create Model page, as shown in Figure 4-2.

Figure 4-2   Creating the Model



Name: Street_Light

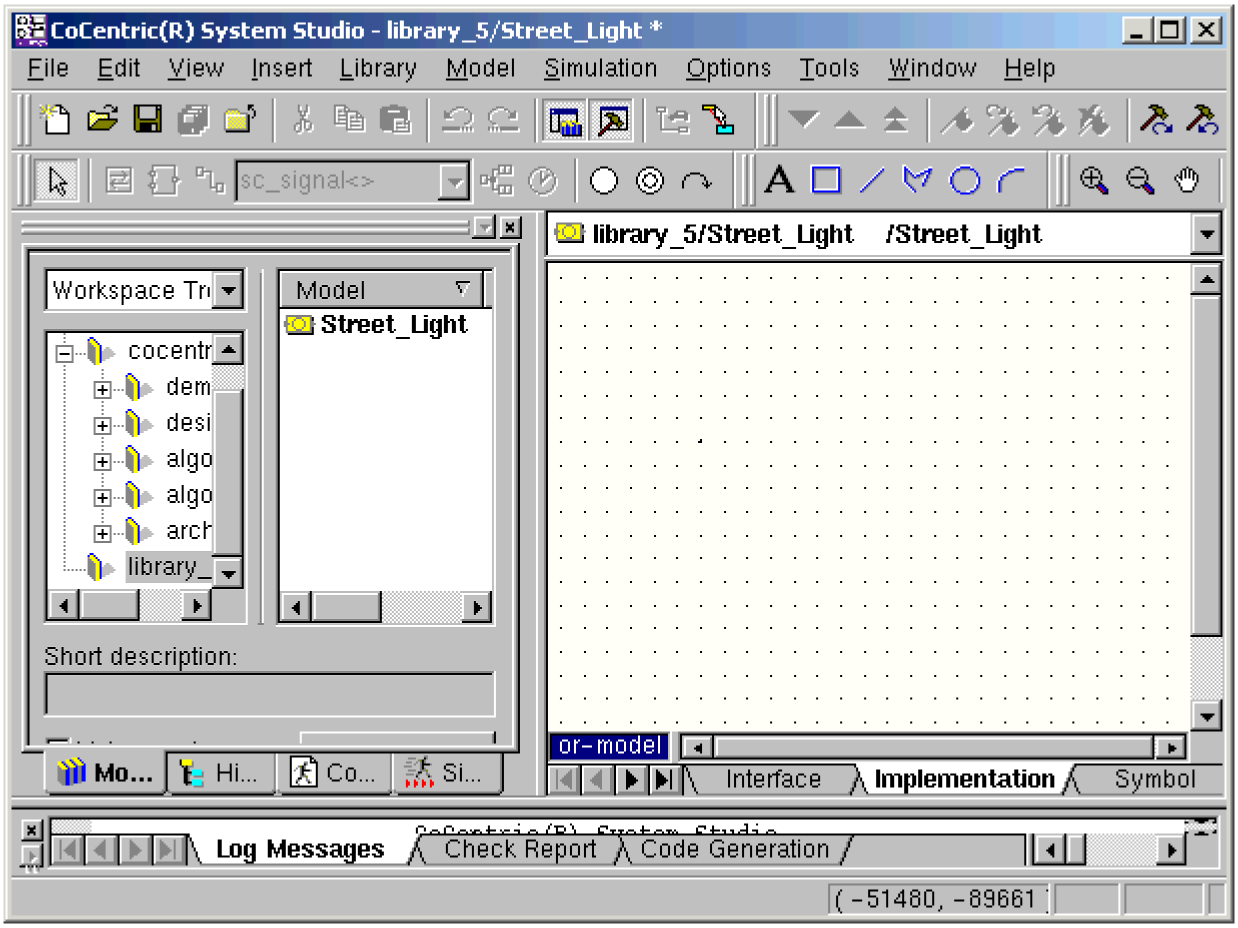
Domain: Algorithmic Models

Type:

- None
- DFG
- [✓ OR]
- AND
- GATED
- PRIM
- SDS
- HDL wrapper

Enter a name for your model or use the default name. Select OR as the model type (you can leave the definition of the model implementation until later, which allows you to approach your design in either a top-down or a bottom-up manner), and then click OK. The Implementation view will be displayed in the design area (the Implementation tab will be highlighted), as shown in Figure 4-3.

Figure 4-3 Viewing the Model Implementation


CoCentric(R) System Studio - library_5/Street_Light *
File Edit View Insert Library Model Simulation Options Tools Window Help
sc_signal<>
library_5/Street_Light /Street_Light
Workspace Tr
Model
Street_Light
cocentr
dem
desi
algo
algo
arch
library_
Short description:
Mo... Hi... Co... Si...
or-model
Interface Implementation Symbol
Log Messages Check Report Code Generation
(-51480, -89661

Notice that the workspace window displays the complete hierarchy of the model, and the model contains three components: the Interface view, the Implementation view, and the Symbol view.

Creating the Model Schematic

The OR model you are going to enter is a simple OR model that imitates a street light. It consists of three atomic states, one for the red lamp, one for the yellow lamp, and one for the green lamp.

One of the states will be specified as being the start state by virtue of being the destination of a start transition. You do not have to specify start and exit transitions explicitly; any transition without a source is automatically interpreted as being a start transition, and any transition without a destination is interpreted as being an exit transition. The model does not have an explicit exit state.

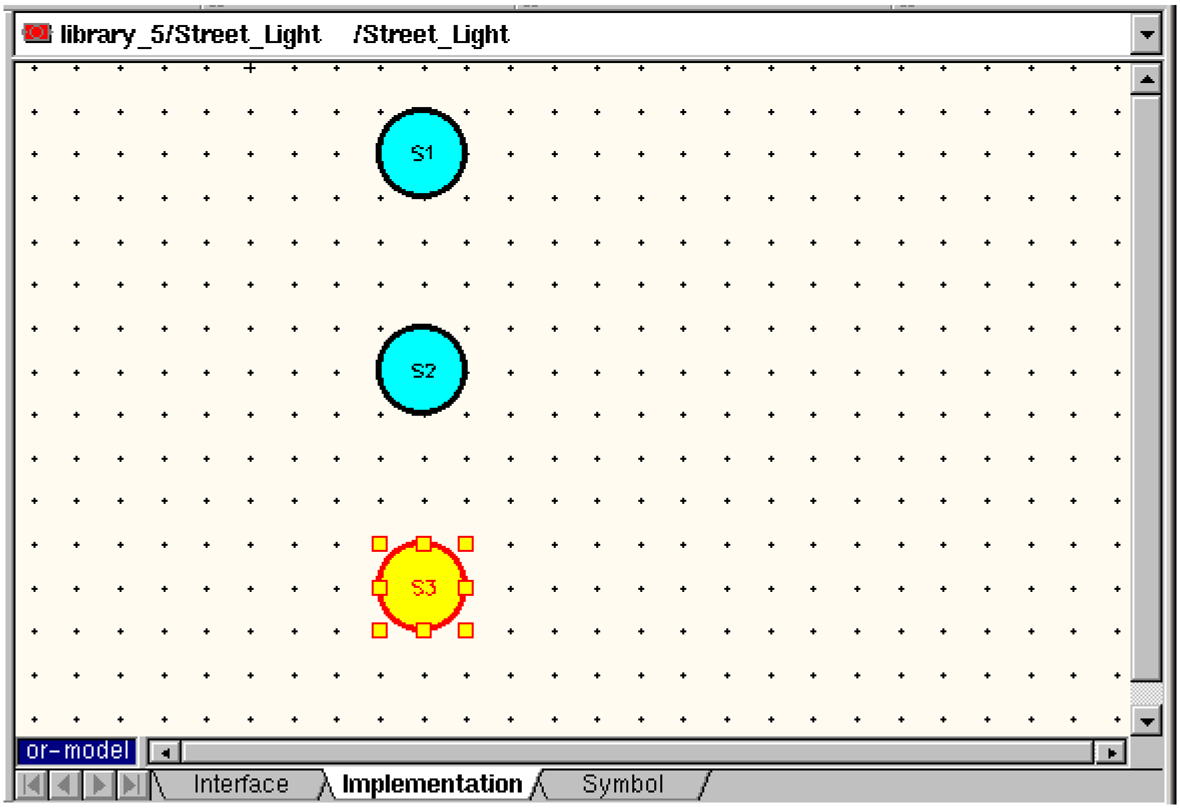
Creating and Configuring the States

The first stage of creating the model is to create the states:

1. Click the Add atomic state button in the System Studio FSM objects toolbar and add three atomic states, s1, s2 and s3, to the FSM design. Click the middle mouse button to cancel the Insert function.

   The end result should look something like Figure 4-4. (Note that atomic state S3 is selected in the figure so it is highlighted.)

Figure 4-4 Inserting the Atomic States

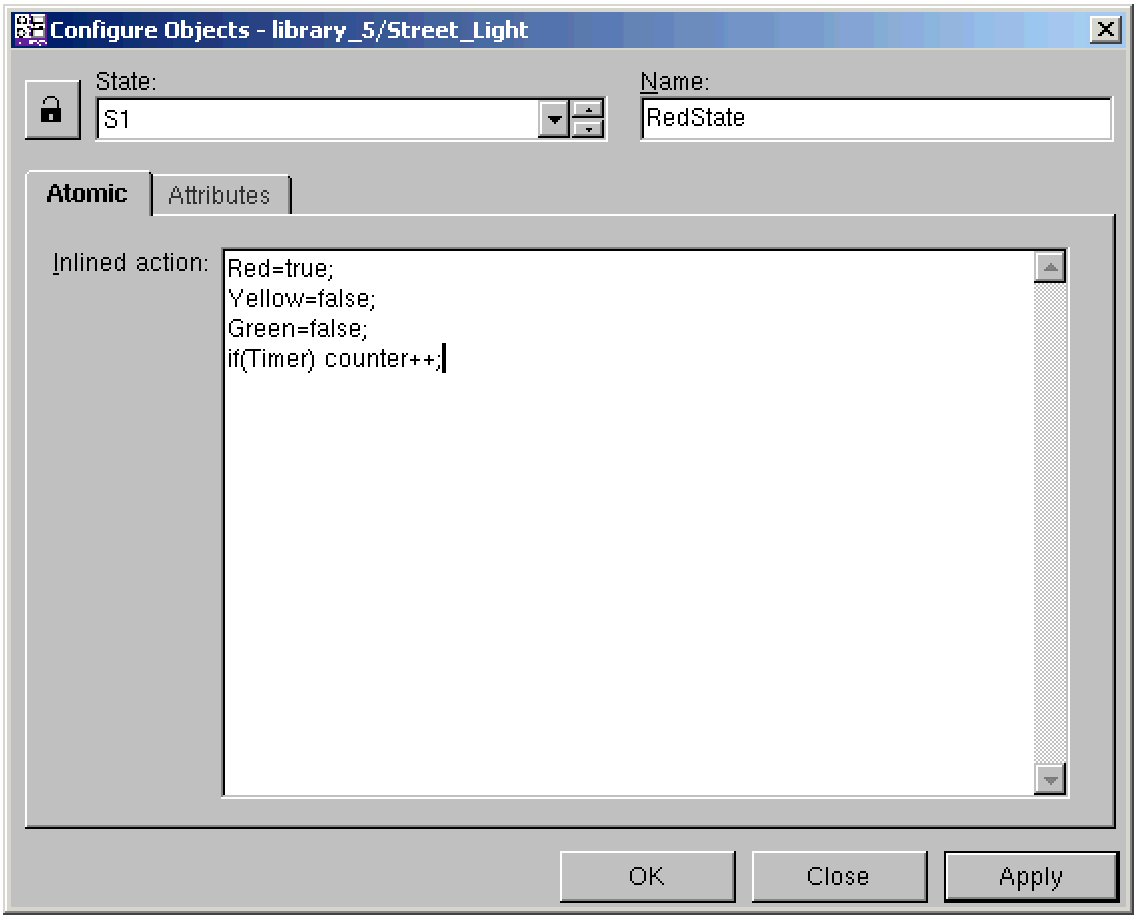


2. Now you need to define the Inlined actions for atomic states. Select the atomic state S1 in the design.

3. Make sure that the state is highlighted, double-click the atomic state, or click the right mouse button when the cursor is positioned over the S1 state and choose Configure Object from the pop-up menu. Alternatively, select the object and choose Configure Object from the System Studio main menu.

   The Configure Objects dialog box is displayed, as shown in Figure 4-5.

Figure 4-5   Configuring the Atomic States



State: S1

Name: RedState

```
Red=true;
Yellow=false;
Green=false;
if(Timer) counter++;
```

4. Define the following for atomic state S1:

   - Name: RedState
   - Inlined action: Red=true;
   - Inlined action: Yellow=false;
   - Inlined action: Green=false;
   - Inlined action: if(Timer) counter++;

   Click Apply.

Note:

Notice that the name of the atomic state in the design area changes from S1 to RedState, and the actions are displayed next to the symbol for the state.

5. Define the following for atomic state S2:

   - Name: YellowState
   - Inlined action: Red=false;
   - Inlined action: Yellow=true;
   - Inlined action: Green=false;
   - Inlined action: if(Timer) counter++;

   Click Apply.

6. Define the following for atomic state S3:

   - Name: GreenState
   - Inlined action: Red=false;
   - Inlined action: Yellow=false;
   - Inlined action: Green=true;
   - Inlined action: if(Timer) counter++;

   Click Apply.

Entering and Configuring the Transitions

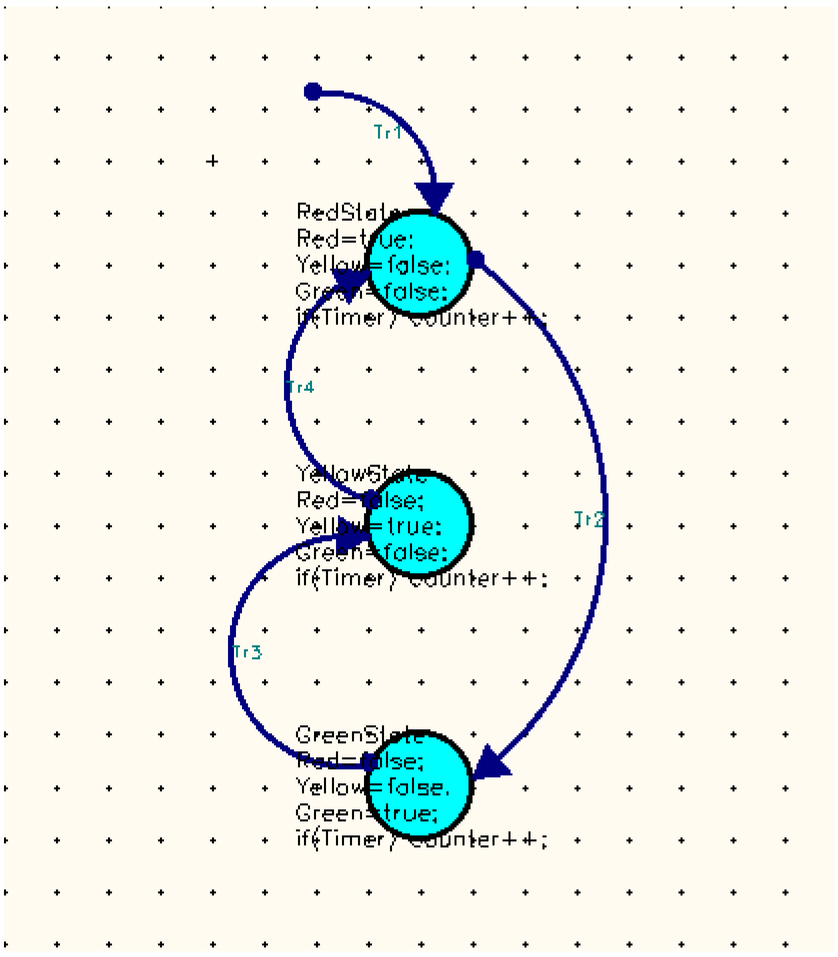
The next steps are to enter the transitions between the states and then configure them.

1. Click the Insert Transition button on the FSM toolbar and draw the following transitions as in Figure 4-6:

   - Tr1: (start) to RedState (S1)
   - Tr2: RedState (S1) to GreenState (S3); the street light goes from red to green
   - Tr3: GreenState (S3) to YellowState (S2); the street light goes from green to yellow
   - Tr4: YellowState (S2) to RedState (S1); the street light goes from yellow to red

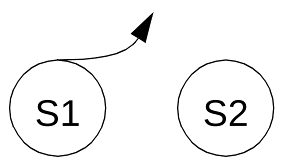
   Transitions that do not have a source are automatically interpreted as start transitions, and transitions that do not have a sink are automatically interpreted as exit transitions. Your schematic should look like Figure 4-6.

Figure 4-6 Drawing the Transitions

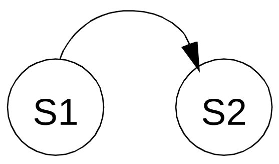


Note that a transition will not always take the path that you might want it to. If you want a transition to curve in a particular direction, it helps to initially draw the transition to some intermediate point along the curve that you want it to follow first. Then select the end of the transition and drag it to its final destination, as shown in Figure 4-7.

Figure 4-7   Drawing Transitions



a) Draw the transition to an intermediate point



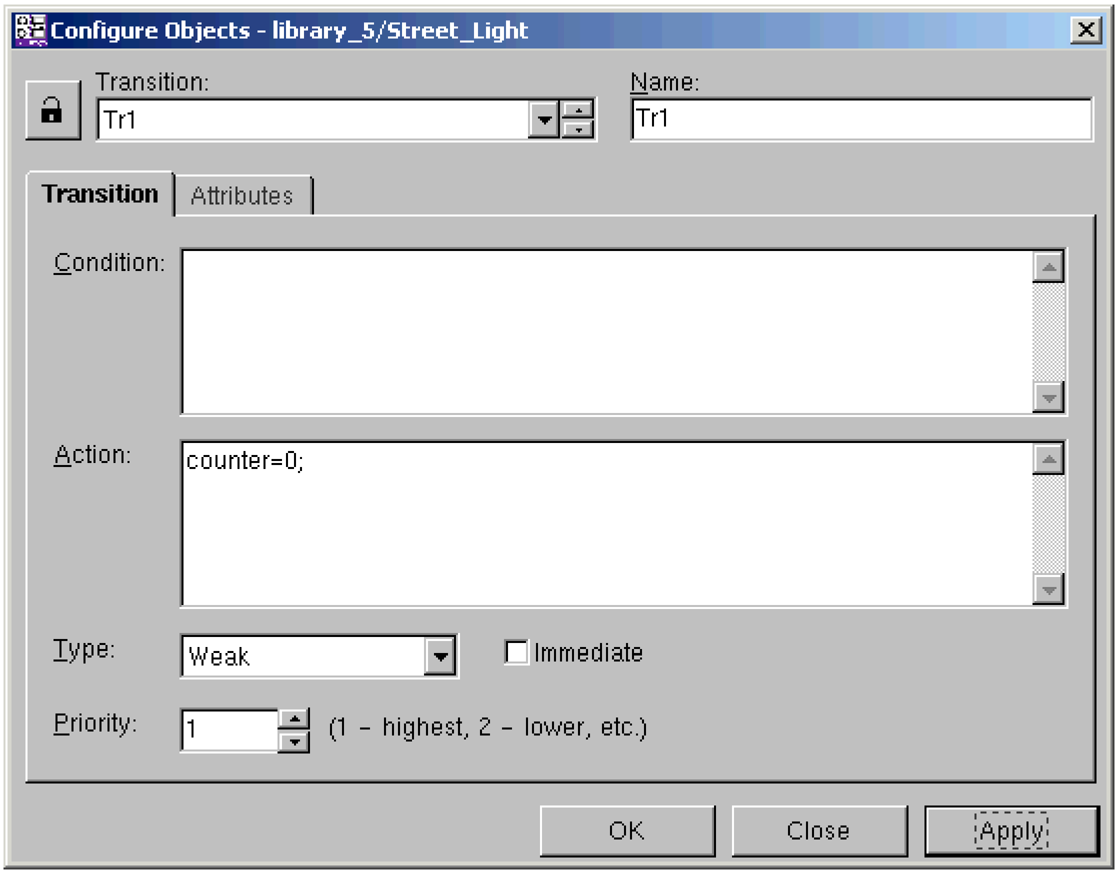
b) Then drag the end point to the final destination

2. Now configure the transitions. Double-click a transition to display the Configure Objects dialog box for that transition. Enter the detail required and then click OK.

Configure the transitions as follows:

- Transition Tr1: (start) to RedState (S1) (see Figure 4-8)
    - Condition: true (defaults to true)
    - Action: counter=0;
    - Type: weak
    - Priority: 1

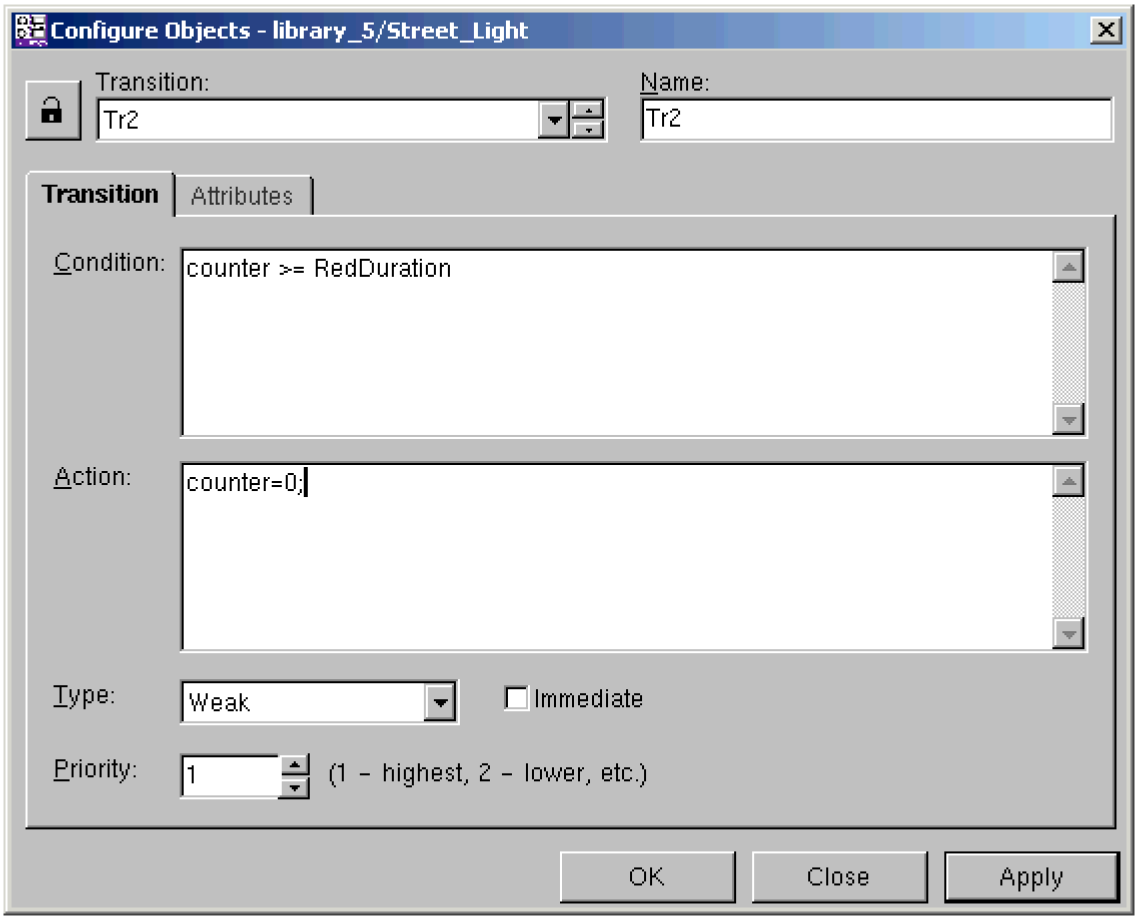
Figure 4-8 Defining Transition Tr1



Transition: Tr1
Name: Tr1
Condition:
Action: counter=0;
Type: Weak
☐ Immediate
Priority: 1 (1 – highest, 2 – lower, etc.)

- Transition Tr2: RedState (S1) to GreenState (S3) (see Figure 4-9)
  - Condition: counter >= RedDuration
  - Action: counter=0;
  - Type: weak
  - Priority: 1

Figure 4-9 Defining Transition Tr2



Transition: Tr2

Name: Tr2

Condition: counter >= RedDuration

Action: counter=0;

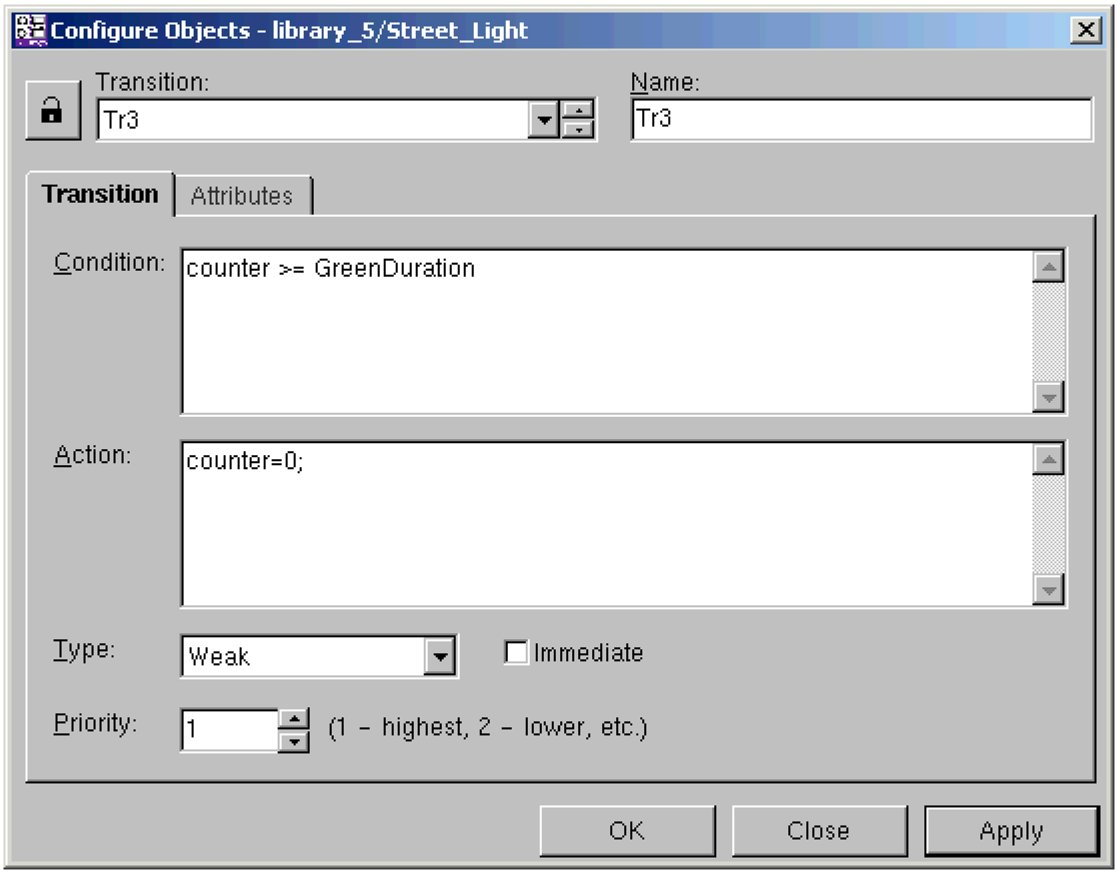
Type: Weak

☐ Immediate

Priority: 1 (1 – highest, 2 – lower, etc.)

- Transition Tr3: GreenState (S3) to YellowState (S2) (see Figure 4-10)
    - Condition: counter >= GreenDuration
    - Action: counter=0;
    - Type: weak
    - Priority: 1

Figure 4-10 Defining Transition Tr3



Transition: Tr3

Name: Tr3

Condition: counter >= GreenDuration

Action: counter=0;

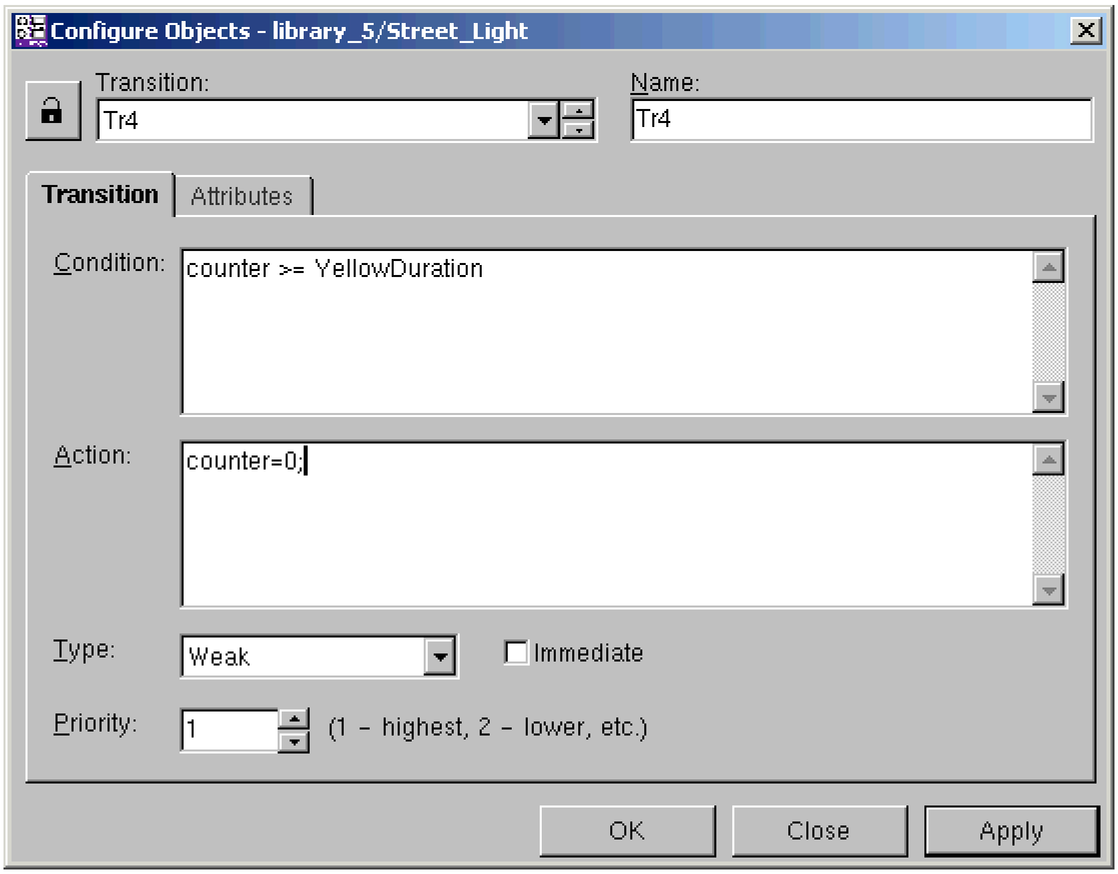
Type: Weak

☐ Immediate

Priority: 1 (1 – highest, 2 – lower, etc.)

- Transition Tr4: YellowState (S2) to RedState (S1) (see Figure 4-11)
  - Condition: counter >= YellowDuration
  - Action: counter=0;
  - Type: weak
  - Priority: 1

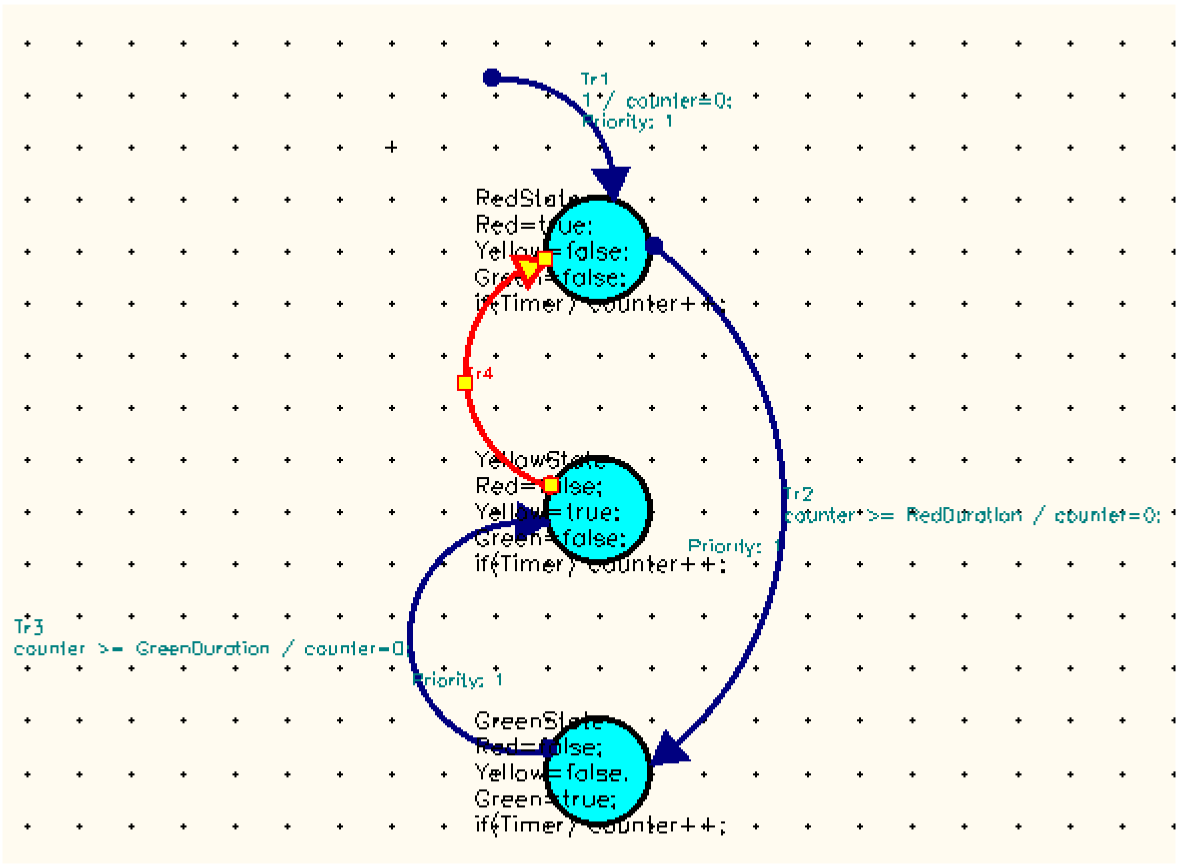
Figure 4-11   Defining Transition Tr4



Transition: Tr4
Name: Tr4
Condition: counter >= YellowDuration
Action: counter=0;
Type: Weak
Immediate: ☐
Priority: 1 (1 – highest, 2 – lower, etc.)

The design entry phase is complete; it should now look like Figure 4-12.

Figure 4-12 The Complete OR Model



This completes the model schematic. You can now define the model's interface.

Defining the Interface

The next step in defining the model is to define the ports. This OR model will have one input port (a timer) and three output ports (the red, yellow and green light signals).

In the design area, click the Interface tab to display the Interface view, shown in Figure 4-13. If the Ports page is not displayed, click the Ports tab.

To add a port, click the Create New Port button. The Port page appears, in which you enter port characteristics.

The data type and direction fields provide selection menus from which you select values for the fields. Pressing the Tab key moves the cursor to the next field, except when the cursor is in the last field.

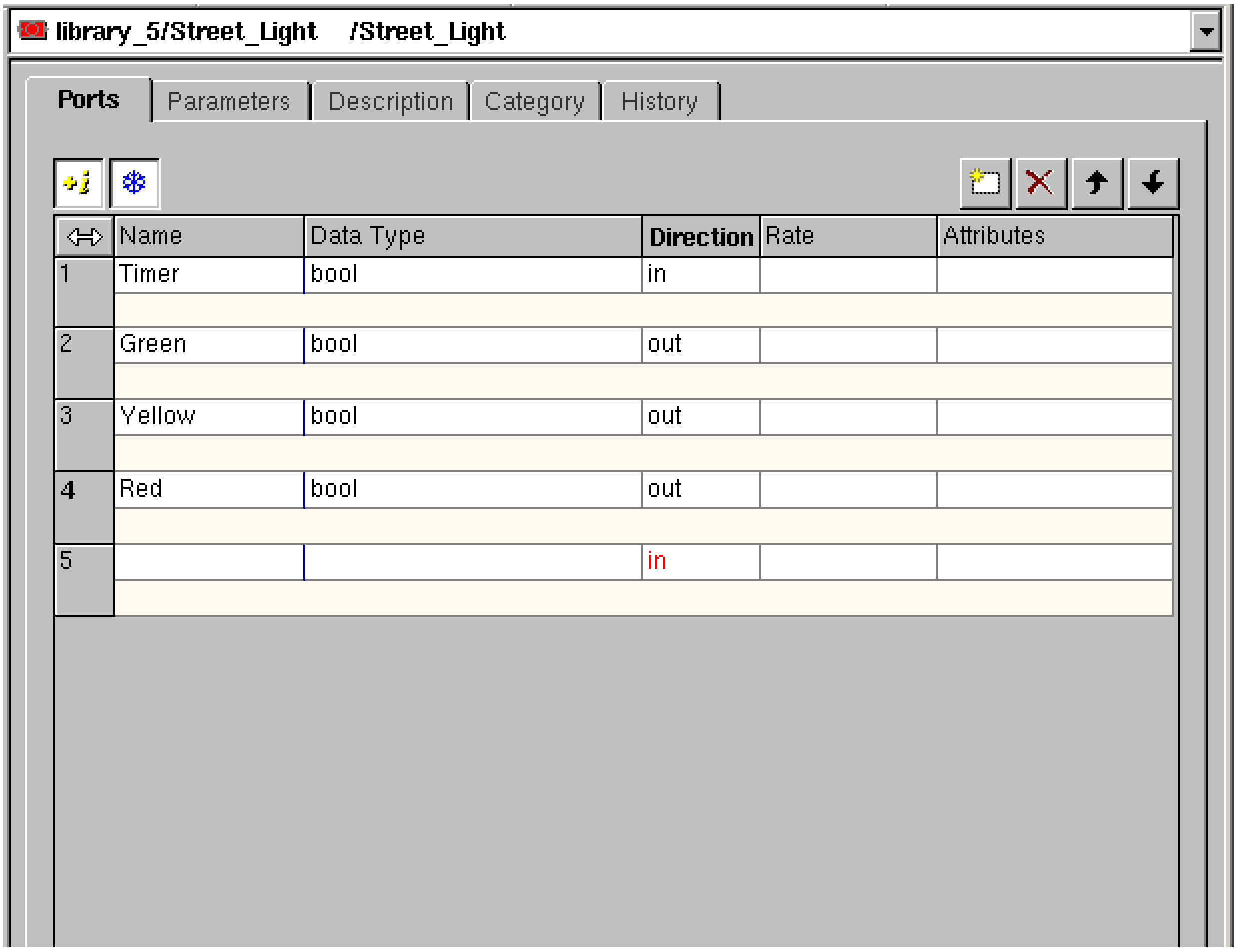
Create four ports with the following characteristics:

- Input port 1
  - Name: Timer
  - Data Type: bool
  - Direction: in
- Output port 1
  - Name: Green
  - Data Type: bool
  - Direction: out
- Output port 2
  - Name: Yellow
  - Data Type: bool
  - Direction: out
- Output port 3

- Name: Red
- Data Type: bool
- Direction: out

The end result should look like Figure 4-13.

Figure 4-13 Defining the Ports



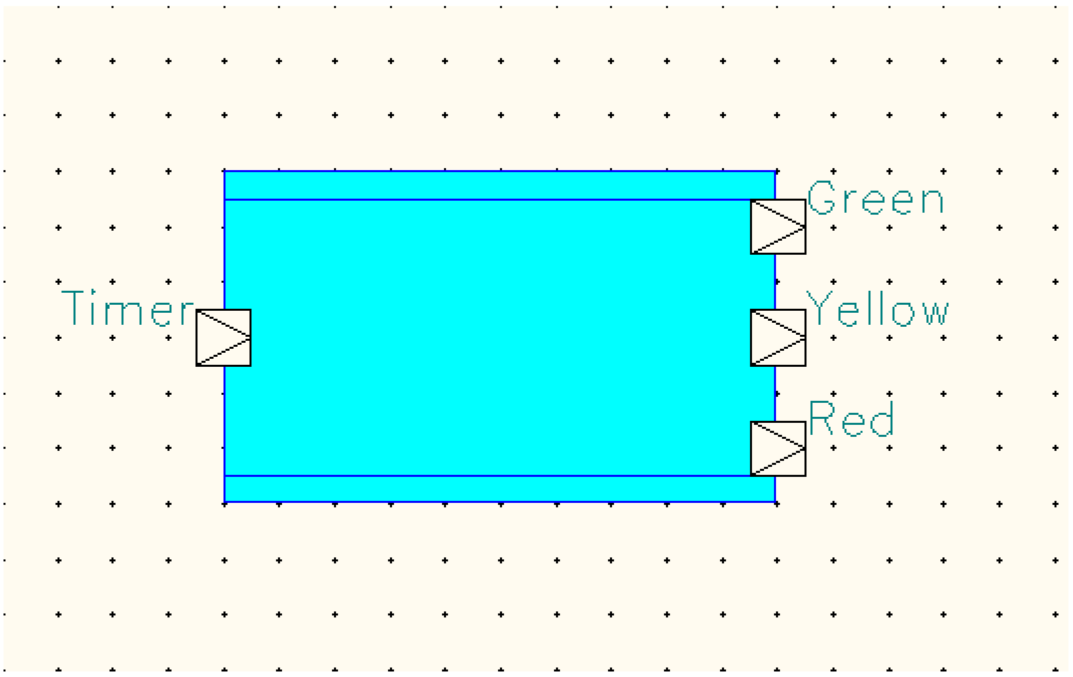
library_5/Street_Light /Street_Light

	Name	Data Type	Direction	Rate	Attributes
1	Timer	bool	in		
2	Green	bool	out		
3	Yellow	bool	out		
4	Red	bool	out		
5			in		

After you enter the ports, click the Symbol tab in the design area and look at the symbol that has been created for the model.

You will see a symbol with one input port and three output ports defined, as shown in Figure 4-14.

Figure 4-14   The Model Symbol



The interface to the model and the symbol for the model are now complete.

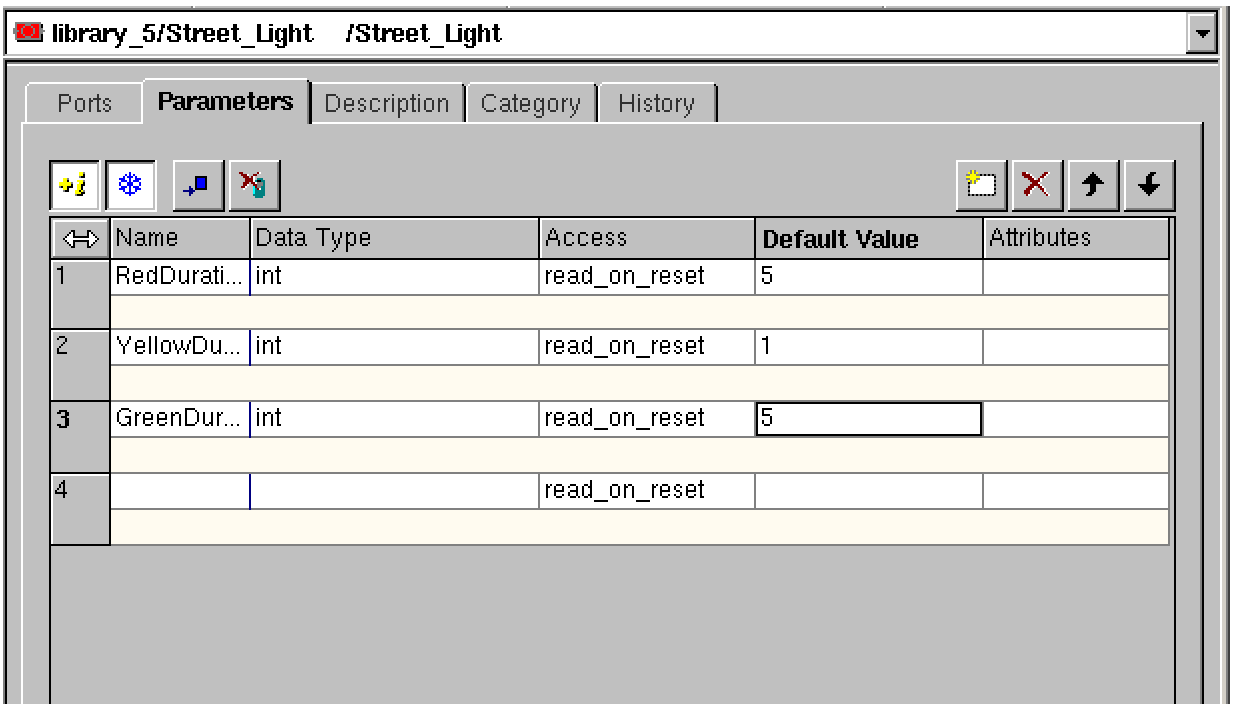
Defining the Model Parameters

While you were specifying the transitions, you set the conditions in terms of RedDuration, YellowDuration and GreenDuration. These are model parameters and you must specify types and values for these parameters.

- In the Interface view, click the Parameters tab
- Specify the parameters as follows:
  - RedDuration:
    Data Type: int
    Access: read_on_reset
    Default Value: 5
  - YellowDuration:
    Data Type: int
    Access: read_on_reset
    Default Value: 1
  - GreenDuration:
    Data Type: int
    Access: read_on_reset
    Default Value: 5

The completed parameters definition page should look like Figure 4-15.

Figure 4-15 Specifying the Model Parameters



	Name	Data Type	Access	Default Value	Attributes
1	RedDurati...	int	read_on_reset	5	
2	YellowDu...	int	read_on_reset	1	
3	GreenDur...	int	read_on_reset	5	
4			read_on_reset		

Defining the Local Variables

Now you define the local variables for the model. Use one of the following two methods to open the Declare Locals dialog box:

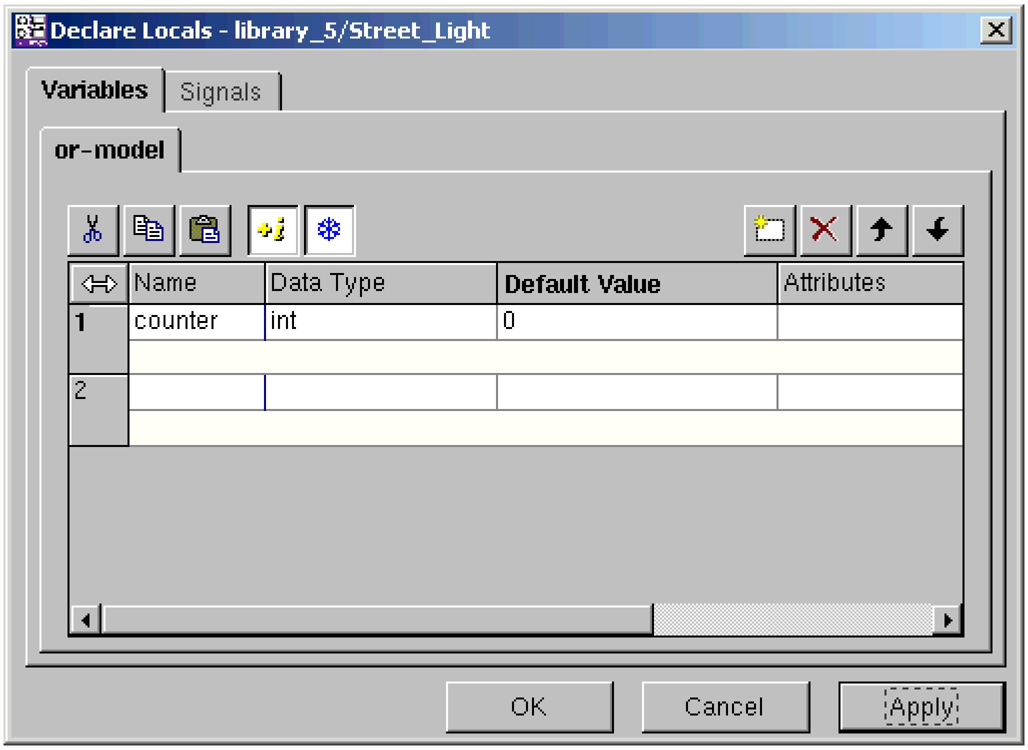
- Choose Model › Declare Locals from the System Studio main menu.
- From the Interface page, position the cursor in the design, press the right mouse button, and choose Model › Declare Locals from the pop-up menu.

The Declare Locals dialog box is displayed, as shown in Figure 4-16.

Enter the following variable, then click OK.

- Variable 1:
  - Name: counter
  - Type: int
  - Default Value: 0
- Now check your design for errors by clicking the Check Design button on the System Studio toolbar (or press Shift-F5).

Figure 4-16 Declaring the Local Variables



	Name	Data Type	Default Value	Attributes
1	counter	int	0	
2				

The output of the design check is displayed in the output window under the Check Report tab. If there are any errors, correct the errors and run Check again.

Even after the model is complete and you ensure that there are no errors, the model still cannot be used because it does not have any source for input data. The next step is, therefore, to include this model inside a verification environment.

Creating the Testbench

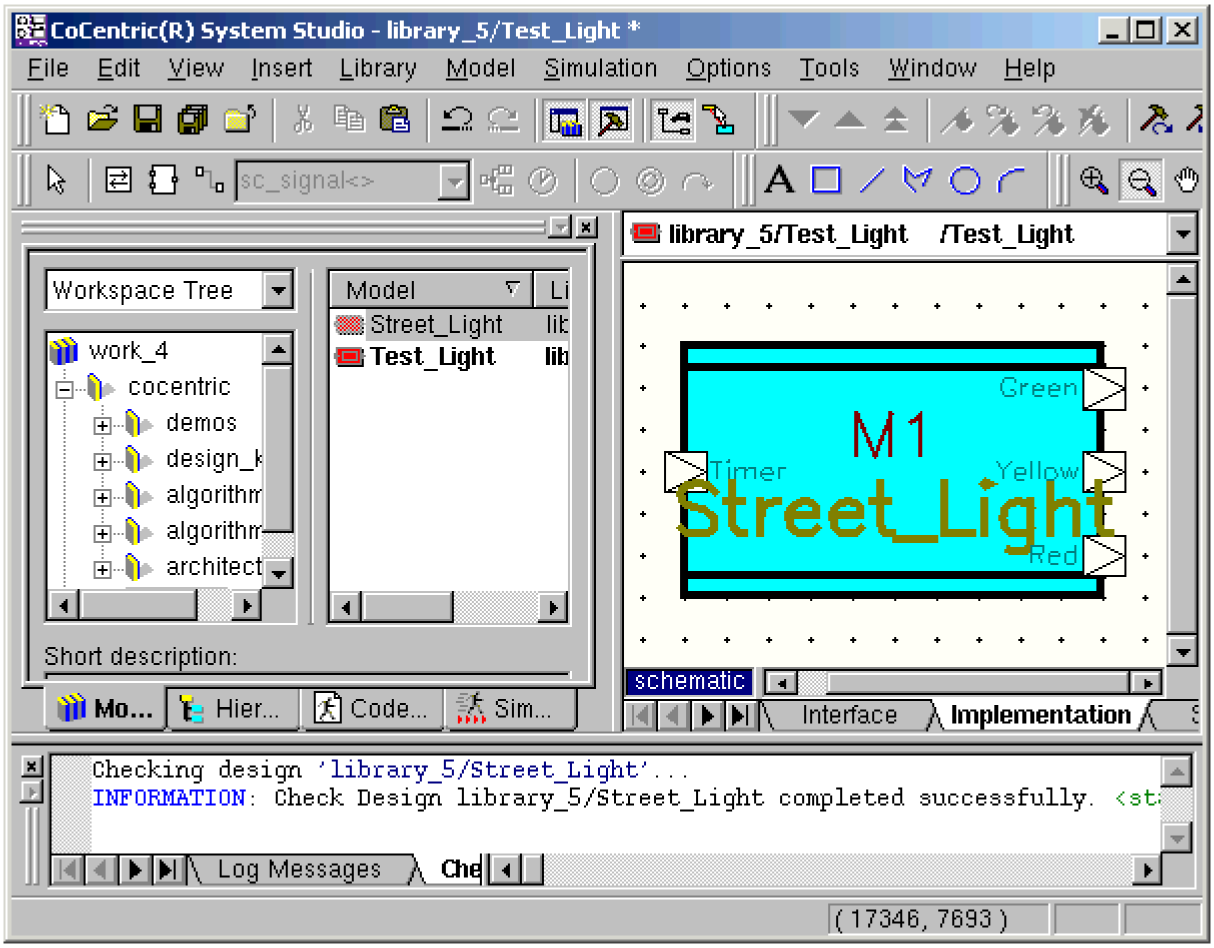
The next steps are to connect the completed OR model to a testbench and run the model in a simulation.

1. Choose File › New from the System Studio main menu to create a new model. This new model will instantiate the completed OR model.
2. Create the new model (in our example Test_Light) in the same library as the OR model. (For instructions, see "Creating a Model" on page 4-5.)
3. Select DFG as the model type and click OK.

   The schematic editor window appears. Note that the Dataflow Objects toolbar is now active.
4. Click the insert Instance button in the Dataflow Objects toolbar. In the Workspace window, select the name of the OR model you created and position it in the schematic design.

   Alternatively, you can select the OR model you created, drag it (keeping the left mouse button pressed) into the workspace and drop it (release the left mouse button) where you want it to be.
5. Verify that the names of the ports are the same as the ones you created (see Figure 4-17).

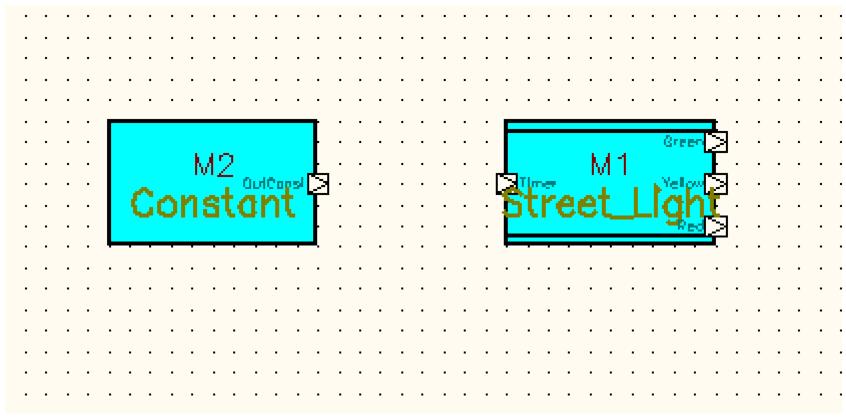
Figure 4-17   The OR Model Implementation



```
Checking design 'library_5/Street_Light'...
INFORMATION: Check Design library_5/Street_Light completed successfully. <st
```

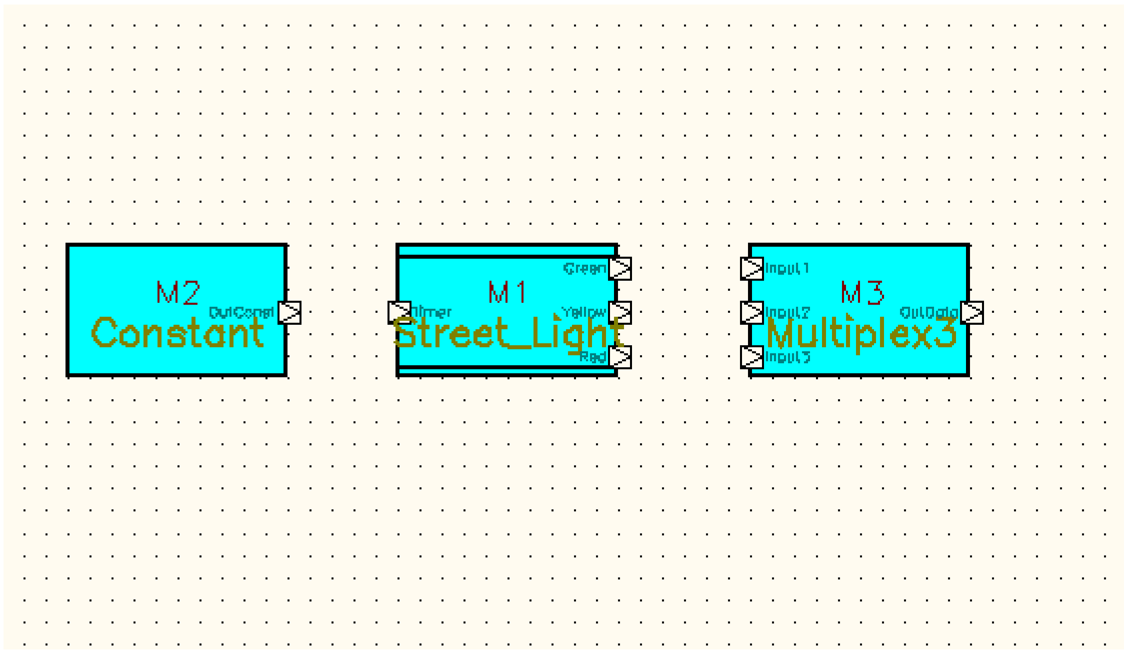
6. In the workspace window, navigate to the cocentric/algorithm/ source library. Select the model Constant, and position it in the design to the left of (before) the OR model, as shown in Figure 4-18.

Figure 4-18 Adding the Constant Model



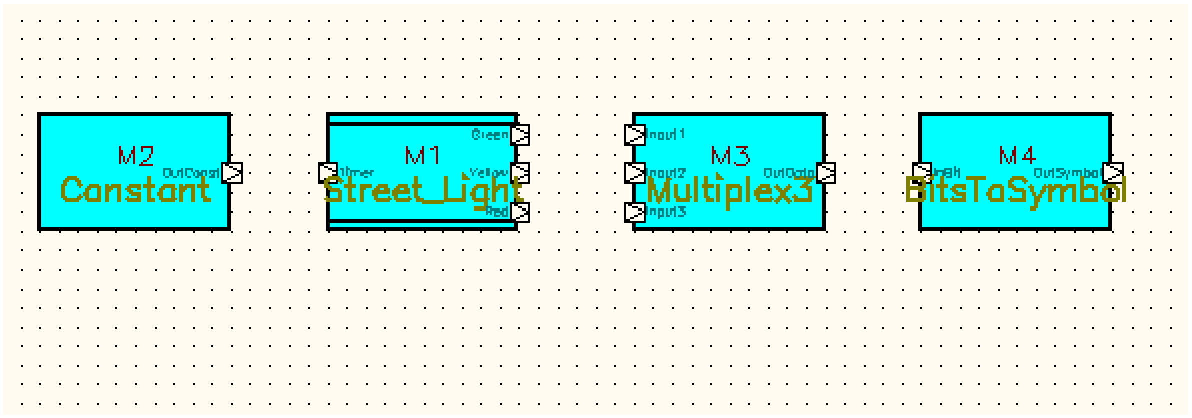
7. Select the model named Multiplex3 from the cocentric/algorithm/ flowcontrol library and position it in the design to the right of your OR model, as shown in Figure 4-19.

Figure 4-19 Adding the Multiplexer Model



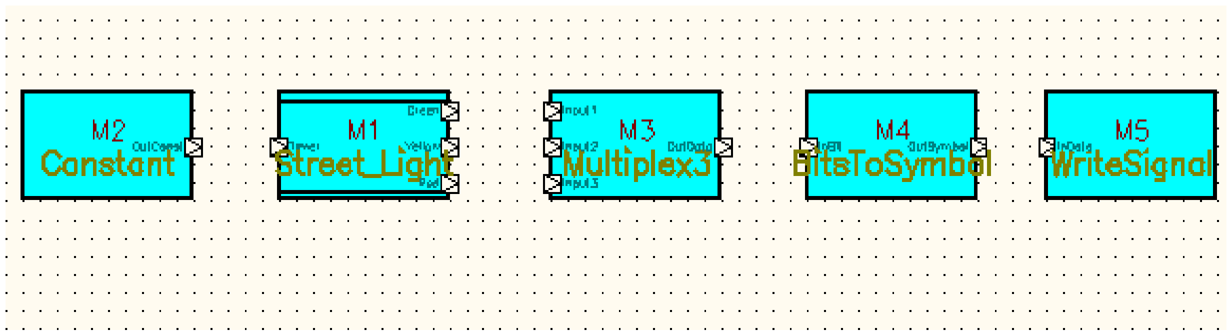
8. Select the model named BitsToSymbol from the cocentric/ algorithm/conversion library and position it in the design to the right of the Multiplex3 model, as shown in Figure 4-20.

Figure 4-20 Adding the BitstoSymbol Model



9. Select the model named WriteSignal from the cocentric/algorithm/sink library and position it in the design to the right of the BitsToSymbol model, as shown in Figure 4-21.

Figure 4-21 Adding the WriteSignal Model

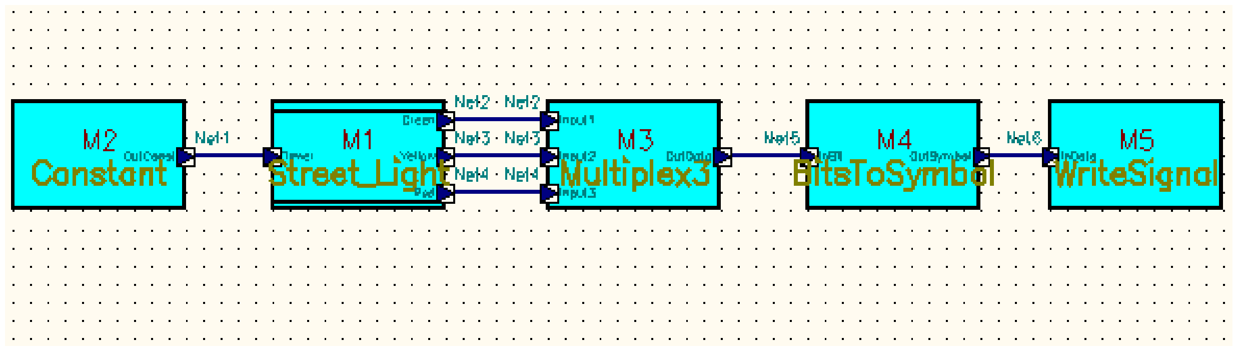


10. After you have positioned all the blocks, click the Net button and connect the components as follows:

    - Connect the output of the Constant block to the input of the OR model.
    - Connect the outputs of the OR model to the inputs of the Multiplex3 block.
    - Connect the output of the Multiplex3 block to the input of the BitsToSymbol block.
    - Connect the output of the BitsToSymbol block to the input of the WriteSignal block.

    The end result should look like the display shown in Figure 4-22.

Figure 4-22 The Completed Test Harness Schematic

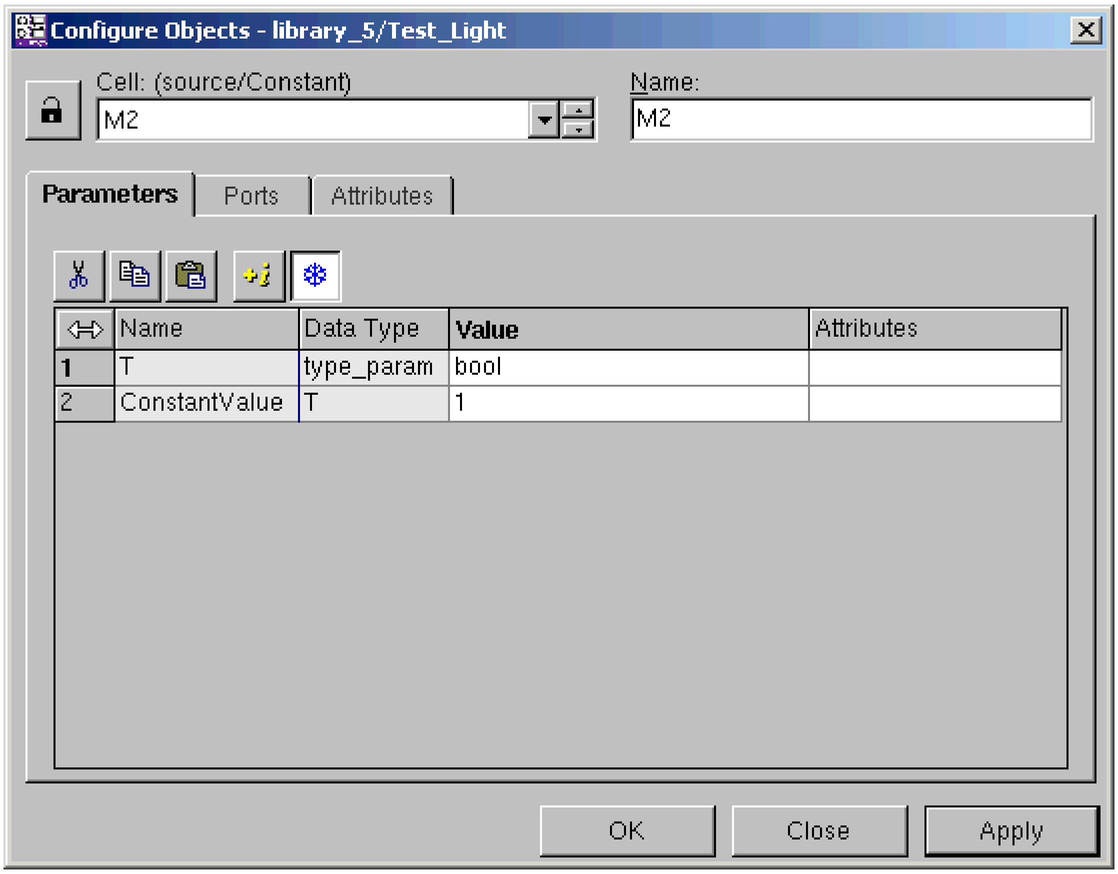


Configuring the Model Instances

The data-flow models used in this design were developed for use in a wide range of schematics. To make this possible, the data type is defined by means of a type parameter. Within this schematic, it is not possible to determine the type of parameter, so you must explicitly set the type.

- Select the Constant block (M2). Click the right mouse button and choose Configure Object from the context-sensitive menu. In the Configure Objects dialog box, set the value of parameter T to bool and the value of ConstantValue to 1, as shown in Figure 4-23. Click OK.

Figure 4-23 Defining the Constant Model Data Type Parameter



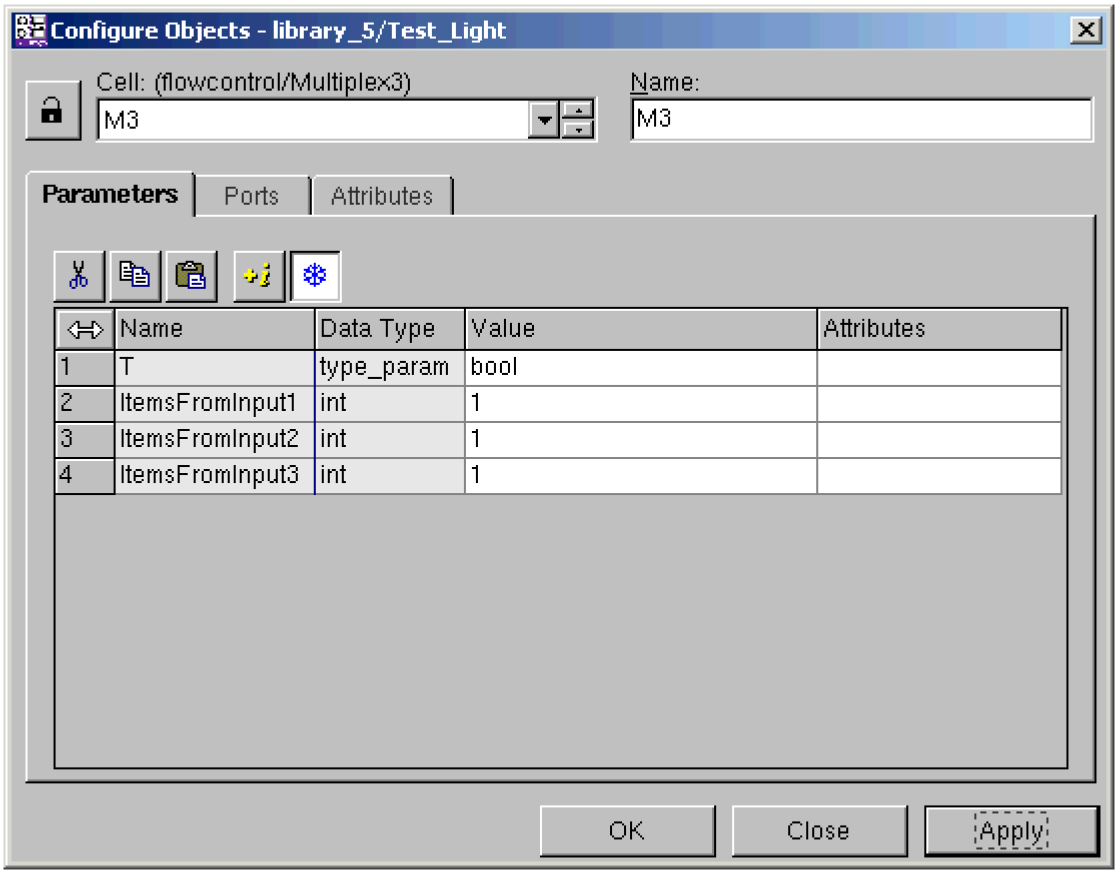
Cell: (source/Constant) M2

Name: M2

	Name	Data Type	Value	Attributes
1	T	type_param	bool	
2	ConstantValue	T	1	

- Select the Multiplex3 block (M3). Click the right mouse button and choose Configure Object from the context-sensitive menu. In the Configure Objects dialog box, set the value of parameter T to bool and check that the ItemsFromInput values are set to 1, as shown in Figure 4-24. Click OK.

Figure 4-24 Defining the Multiplex3 Model Parameters

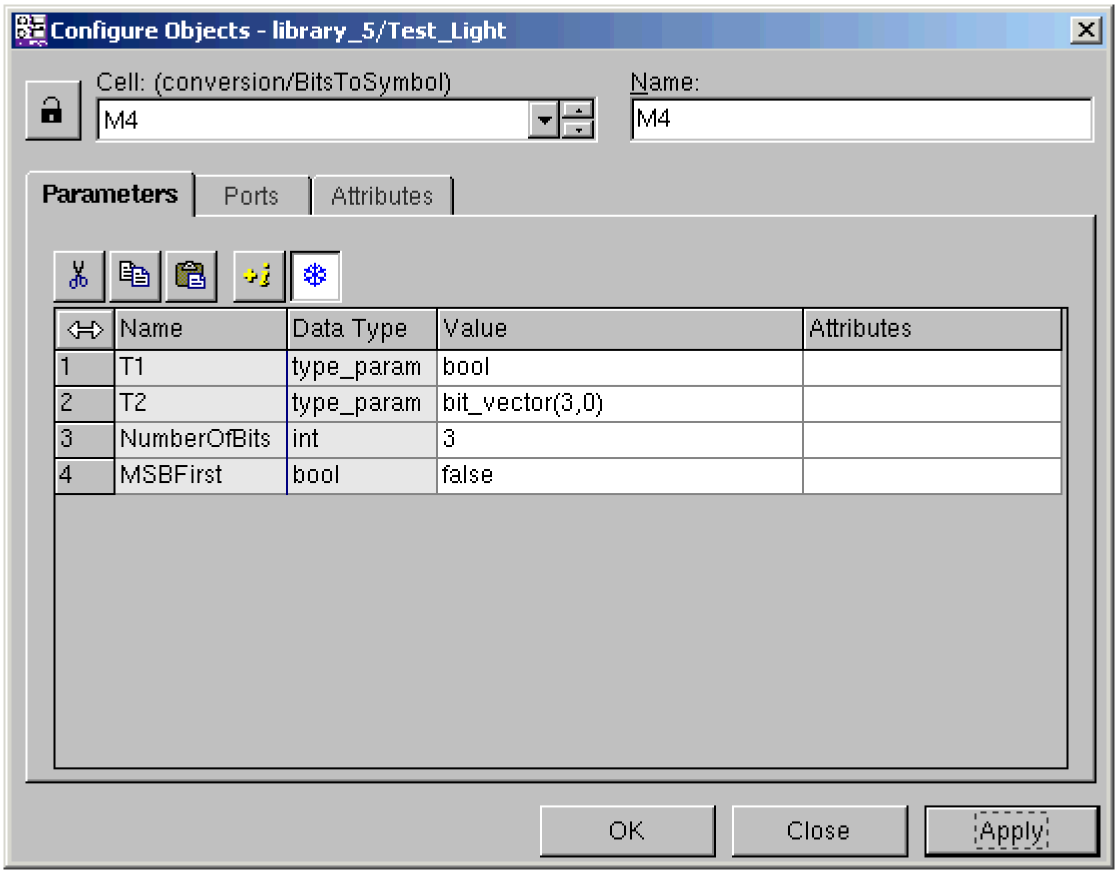

Configure Objects - library_5/Test_Light
Cell: (flowcontrol/Multiplex3)
Name:
Parameters
Ports
Attributes
OK
Close
Apply

Cell: M3
Name: M3

	Name	Data Type	Value	Attributes
1	T	type_param	bool	
2	ItemsFromInput1	int	1	
3	ItemsFromInput2	int	1	
4	ItemsFromInput3	int	1	

- Select the BitsToSymbol block (M4). Click the right mouse button and choose Configure Object from the context-sensitive menu. In the Configure Objects dialog box, set the value of parameter T1 to bool, parameter T2 to bit_vector(3,0) and change the value of NumberOfBits to 3, as shown in Figure 4-25. Click OK.

Figure 4-25   Defining the BitsToSymbol Model Parameters



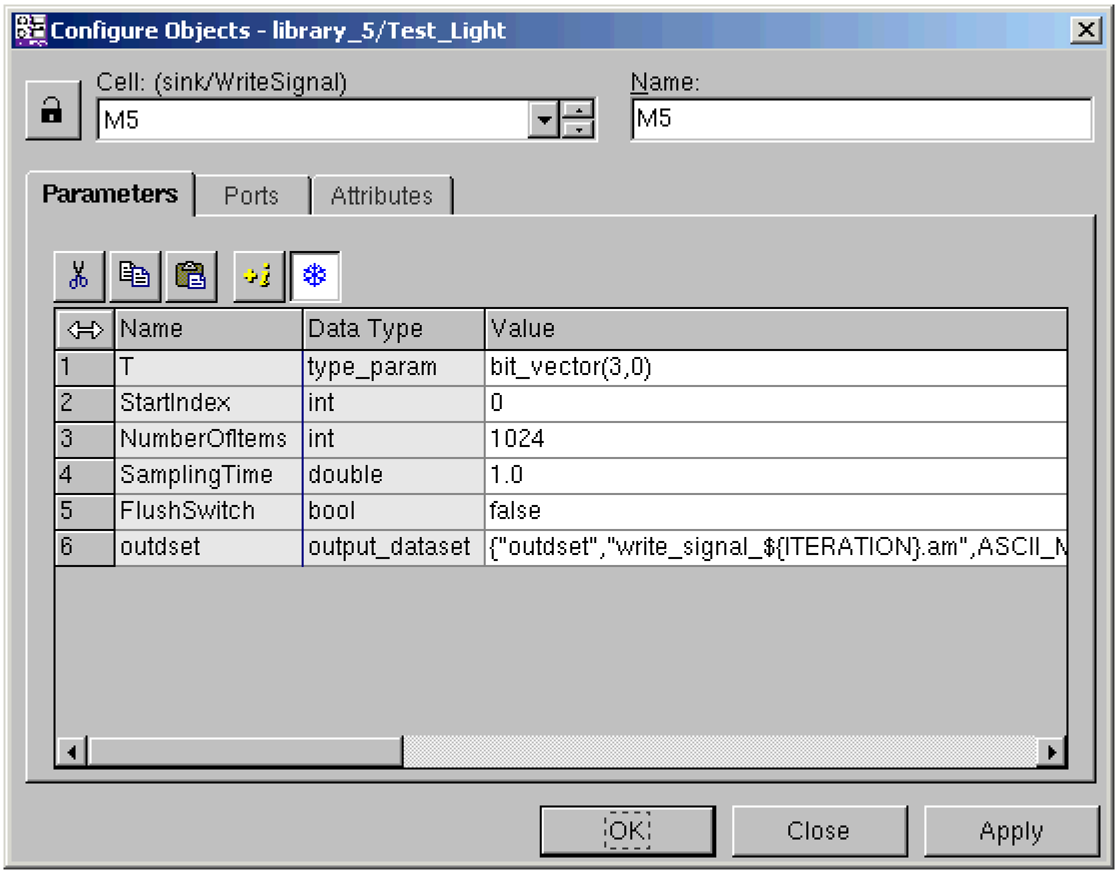
Cell: (conversion/BitsToSymbol) M4

Name: M4

	Name	Data Type	Value	Attributes
1	T1	type_param	bool	
2	T2	type_param	bit_vector(3,0)	
3	NumberOfBits	int	3	
4	MSBFirst	bool	false	

- Finally, select the WriteSignal block (M5). Click the right mouse button and choose Configure Object from the context-sensitive menu. In the Configure Objects dialog box, set the type of parameter T to bit_vector(3,0), as shown in Figure 4-26. Click OK.

Figure 4-26 Defining the WriteSignal Model Parameters



Cell: (sink/WriteSignal) M5

Name: M5

	Name	Data Type	Value
1	T	type_param	bit_vector(3,0)
2	StartIndex	int	0
3	NumberOfItems	int	1024
4	SamplingTime	double	1.0
5	FlushSwitch	bool	false
6	outdset	output_dataset	{"outdset","write_signal_${ITERATION}.am",ASCII_N

Checking the Model

Click the Check Model button in the System Studio Hierarchy Navigation toolbar to check the model for errors.

If you have made any errors in creating the model, an error message will be displayed in the Check Errors message window. Double-click on an error message to go to the likely source of the error. Once the model is free of errors, you are ready to create and run the simulation.

Building and Running the Model Simulation

You can create the model simulation either automatically or manually, this section will take you through both methods.

Automatically Building and Running the Simulation

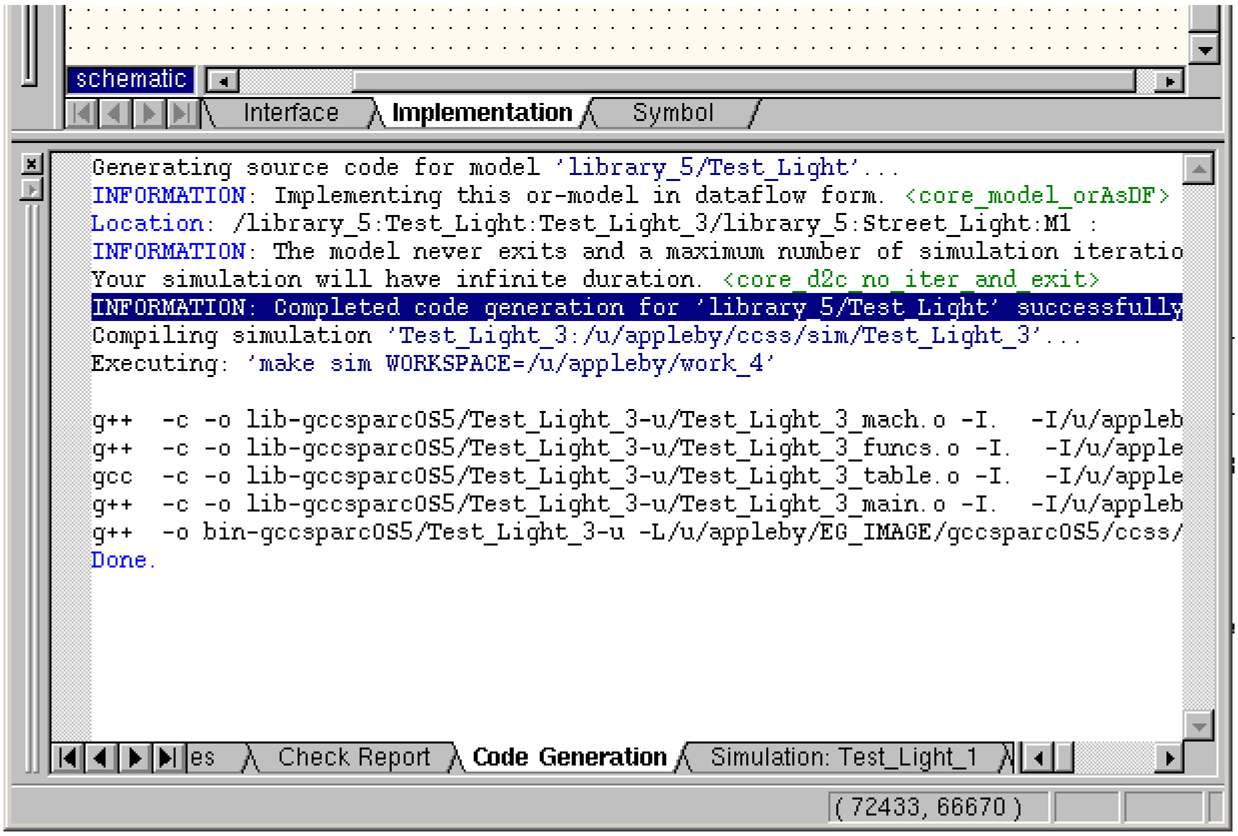
The steps to build and run the simulation include generating the source code, compiling and linking the model, and starting the simulation. Instead of entering all the steps manually, you can perform the steps required to build and start the simulation automatically with one button click.

- To simulate the model automatically, click the Simulate Model button, or choose Simulation › Simulate Model from the menu bar, and all the steps will be performed automatically. Do this now, and then click the Code generation tab to observe the messages in the Message window as the code is generated and the simulation started (see Figure 4-27).

  Note:

  You may notice that the OR model has no exit state and will run continually; however, the testbench simulation will automatically terminate after a predetermined number of cycles.

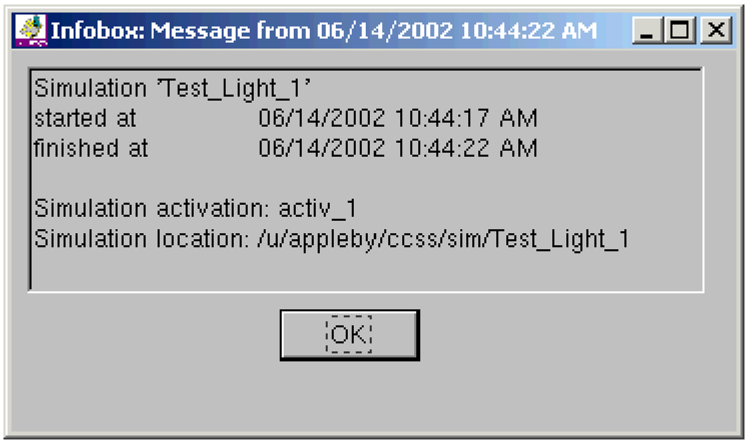
Figure 4-27   Message Window



```
Generating source code for model 'library_5/Test_Light'...
INFORMATION: Implementing this or-model in dataflow form. <core_model_orAsDF>
Location: /library_5:Test_Light:Test_Light_3/library_5:Street_Light:M1 :
INFORMATION: The model never exits and a maximum number of simulation iteratio
Your simulation will have infinite duration. <core_d2c_no_iter_and_exit>
INFORMATION: Completed code generation for 'library_5/Test_Light' successfully
Compiling simulation 'Test_Light_3:/u/appleby/ccss/sim/Test_Light_3'...
Executing: 'make sim WORKSPACE=/u/appleby/work_4'

g++  -c -o lib-gccsparcOS5/Test_Light_3-u/Test_Light_3_mach.o -I.  -I/u/appleb
g++  -c -o lib-gccsparcOS5/Test_Light_3-u/Test_Light_3_funcs.o -I.  -I/u/apple
gcc  -c -o lib-gccsparcOS5/Test_Light_3-u/Test_Light_3_table.o -I.  -I/u/apple
g++  -c -o lib-gccsparcOS5/Test_Light_3-u/Test_Light_3_main.o -I.  -I/u/appleb
g++  -o bin-gccsparcOS5/Test_Light_3-u -L/u/appleby/EG_IMAGE/gccsparcOS5/ccss/
Done.
```

- When the simulation is finished, you will see the message shown in Figure 4-28; click OK.

Figure 4-28 Simulation Message



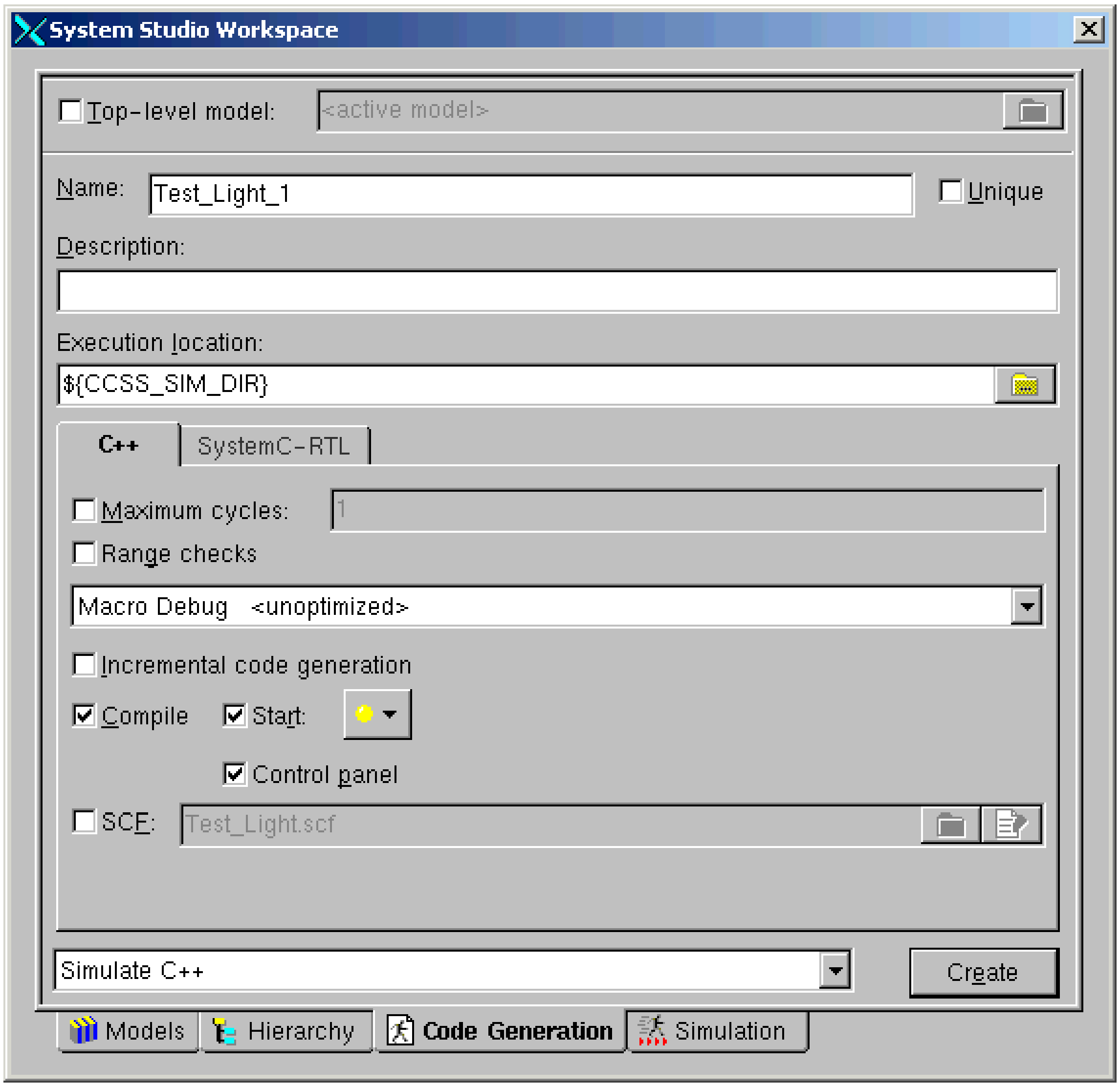
```
Simulation 'Test_Light_1'
started at          06/14/2002 10:44:17 AM
finished at         06/14/2002 10:44:22 AM

Simulation activation: activ_1
Simulation location: /u/appleby/ccss/sim/Test_Light_1
```

Manually Building the Simulation

As an alternative to creating the simulation automatically, you can start code generation and control the execution of the simulation manually, using various simulation options.

1. Click the Code Generation tab and make sure that the Compile, Start, and Control panel check boxes are checked as shown in Figure 4-29.
2. Click the Start button and select Paused so that when the simulation is started, it will be in the paused mode and under the control of the simulation control panel. Notice that the light on the Start button is now at yellow to indicate the paused mode. For more information about the code generation options, refer to the *CoCentric System Studio User Guide*.

Figure 4-29   Controlling the Code Generation


System Studio Workspace

☐ Top-level model: <active model>

Name: Test_Light_1 ☐ Unique

Description:

Execution location:
${CCSS_SIM_DIR}

C++ | SystemC-RTL

☐ Maximum cycles: 1

☐ Range checks

Macro Debug <unoptimized>

☐ Incremental code generation

☑ Compile ☑ Start:

☑ Control panel

☐ SCF: Test_Light.scf

Simulate C++ | Create

Models | Hierarchy | Code Generation | Simulation

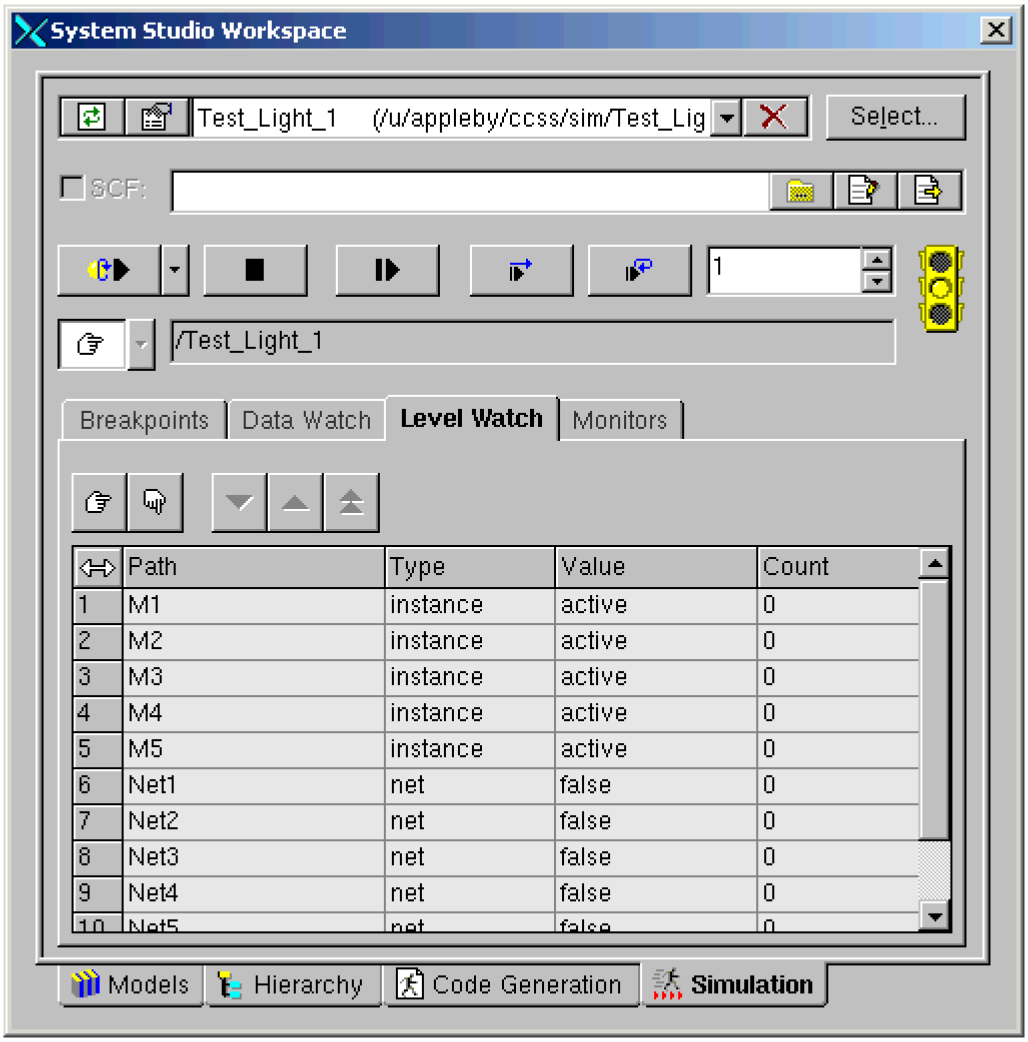
3. Click Create in the code generation window to generate the code and to start the simulation.

4. A warning dialog box may appear to warn you that the simulation directory already exists. Click OK to overwrite the existing simulation data.

The code will be generated again and the simulation will start, but this time it will not complete. After a short while, the simulation control panel will be opened as shown in Figure 4-30.

Notice that the traffic light symbol in the simulation control panel is at yellow to indicate a paused simulation. The traffic lights indicate green when the simulation is running, yellow when the simulation is paused, and red when the simulation is interrupted. The traffic lights show no color when the simulation is finished.

Figure 4-30 Simulation Started in Paused Mode



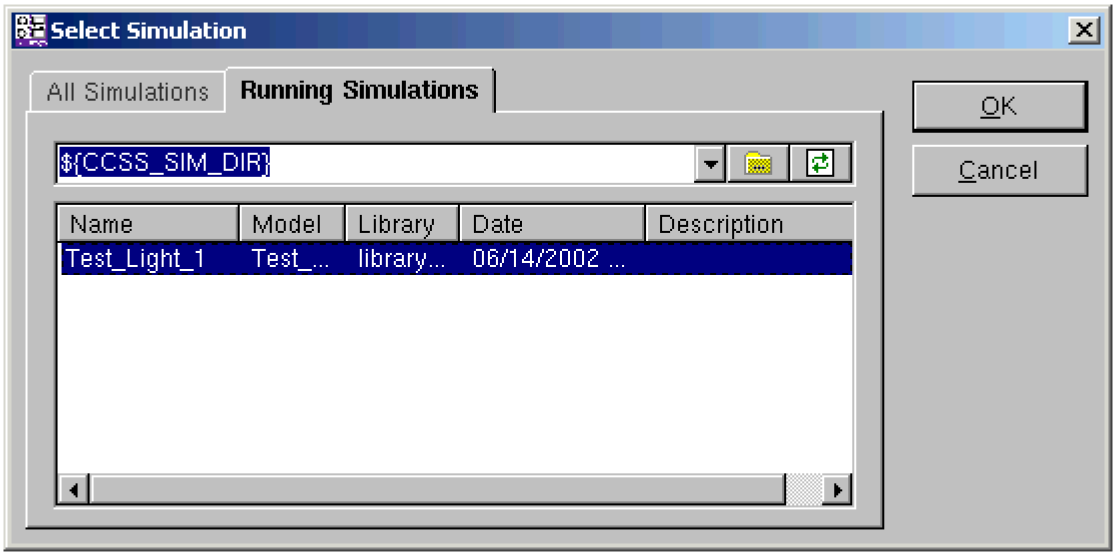
	Path	Type	Value	Count
1	M1	instance	active	0
2	M2	instance	active	0
3	M3	instance	active	0
4	M4	instance	active	0
5	M5	instance	active	0
6	Net1	net	false	0
7	Net2	net	false	0
8	Net3	net	false	0
9	Net4	net	false	0
10	Net5	net	false	0

Controlling the Simulation

This section will take you through the basics of using the simulation control panel. First, it may be interesting to see how many different simulations are running. To do this:

- Click Select in the simulation control panel to display the Select Simulation dialog box as shown in Figure 4-31.

Figure 4-31 Selecting the Simulation



${CCSS_SIM_DIR}

Name	Model	Library	Date	Description
Test_Light_1	Test_...	library...	06/14/2002 ...	

- Click the Running Simulations tab if it is not already highlighted.
- In this case, there is only one simulation running and it is highlighted, so click OK.

The Simulation control panel not only gives you full control over the simulation, it allows you to set breakpoints and watch points in the code execution, just as you would expect from any debugging tool.

Through a command-line interface you can also enter text commands to directly control the simulation without using any part of the System Studio graphical user interface.

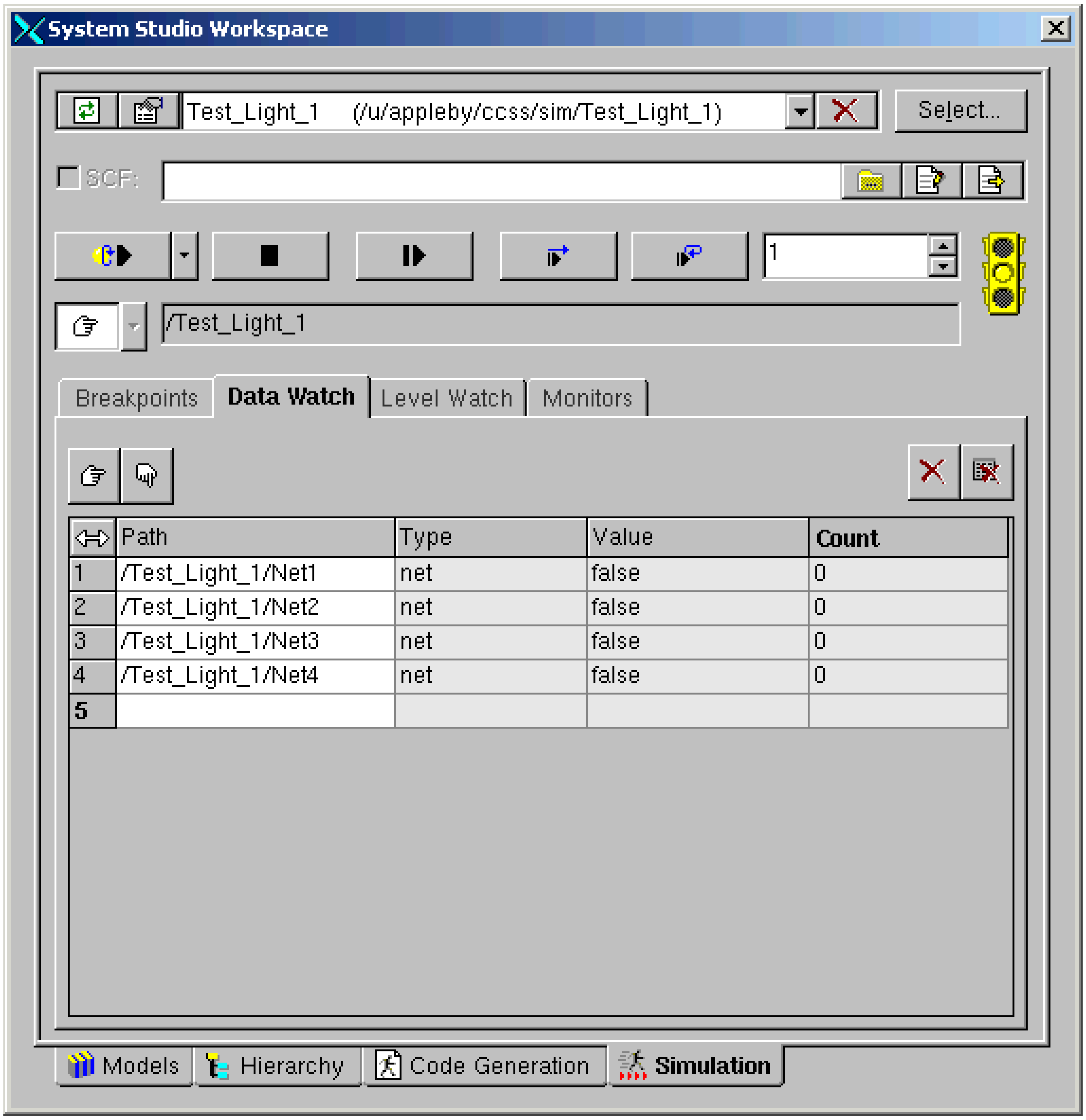
The text commands can also be read in from an external simulation control file (.scf). The System Studio simulation control language includes flow control constructions such as loops (for, while, break, continue) and conditional expressions (if, switch, else), which give you almost complete batch control over the simulation. The System Studio simulation control language consists of the Tcl programming language with the addition of a few specific System Studio commands.

You can display and modify the values of most objects, such as nets, at any level of the model. Values that you cannot modify are shown dimmed in the list. The fields that you can modify have a white background.

You can examine nets, ports, parameters, and variables anywhere in the hierarchy of the active instance. For example, you can monitor the status of the street light model by setting data watch points on the nets leading into and out of it:

- First, click the Level Watch tab.
- Right click on the following nets and select “Add to Data Watch” from the pop-up convenience menu:
  - /Test_Light_1/Net1 (the input signal to the OR model)
  - /Test_Light_1/Net2 (one of the OR model output signals)
  - /Test_Light_1/Net3 (another of the OR model output signals)
  - /Test_Light_1/Net4 (the third OR model output signal)
- Click the Data Watch tab to see the selected data watch points as shown in Figure 4-32.

Figure 4-32   Setting Data Watch Points



Test_Light_1   (/u/appleby/ccss/sim/Test_Light_1)

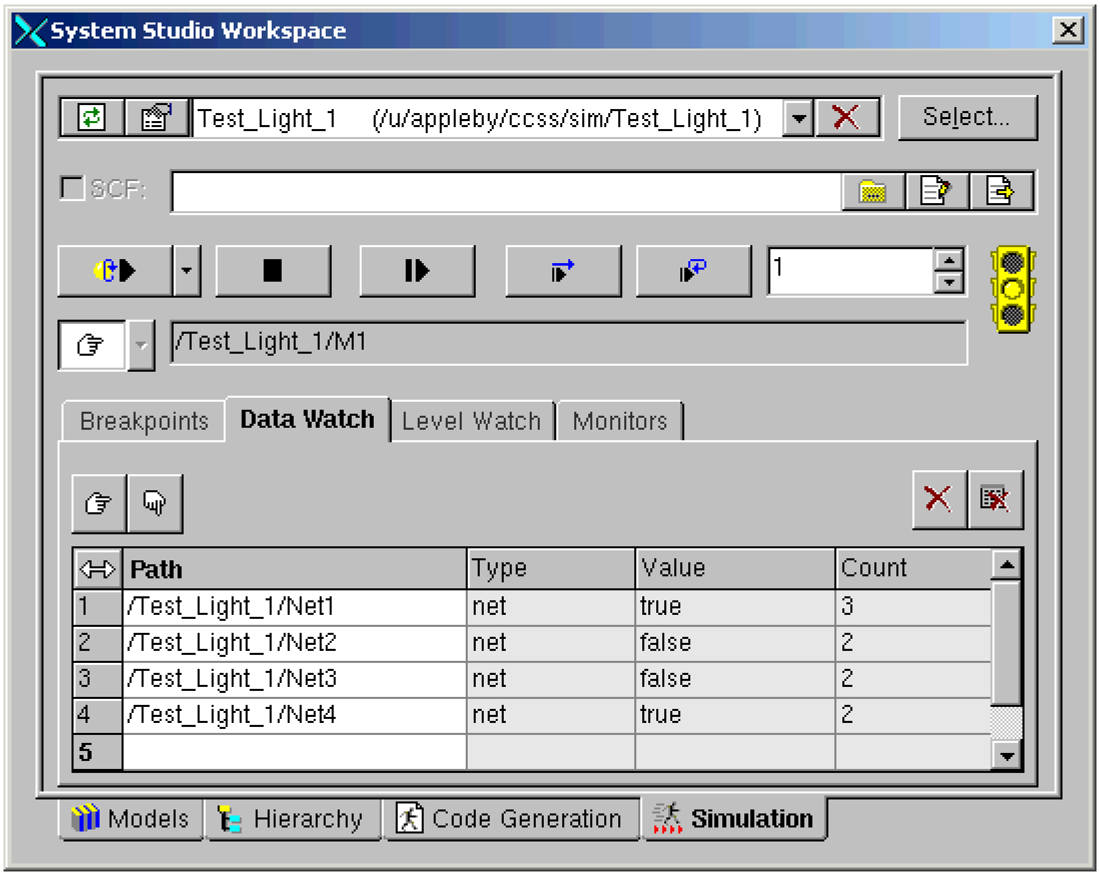
1

/Test_Light_1

	Path	Type	Value	Count
1	/Test_Light_1/Net1	net	false	0
2	/Test_Light_1/Net2	net	false	0
3	/Test_Light_1/Net3	net	false	0
4	/Test_Light_1/Net4	net	false	0
5				

If you now repeatedly click the Instance Entry button in the simulation control panel, you will be able to observe the values of the RedState, YellowState, and GreenState signal outputs cycle through their values, as shown in Figure 4-33.

Figure 4-33 Watching Data Points



	Path	Type	Value	Count
1	/Test_Light_1/Net1	net	true	3
2	/Test_Light_1/Net2	net	false	2
3	/Test_Light_1/Net3	net	false	2
4	/Test_Light_1/Net4	net	true	2
5				

Notice that each time you step through the simulation, the traffic light symbol changes from yellow to green and then back to yellow again, to indicate that the simulation runs and then pauses.

Checking the Output

While you were configuring the WriteSignal instance, you may have noticed that the output is written to a data-set file. In addition to using DAVIS to visualize the output from the model, the contents of a data-set file can be a very useful tool in checking and debugging a model.

Go to the directory ${CCSS_SIM_DIR}/TestLight_1/activ_1/iter_1. In this directory you will find a file called outdset.am; this is the output data-set file written by the WriteSignal instance for iteration 1 of simulation activation 1 (for later activations and iterations, this number will increase accordingly). A short extract of this file is shown in Example 4-1.

Example 4-1   Extract from the Output Data-Set File

```
100
100
100
100
100
001
001
001
001
001
010
100
100
100
100
100
001
001
001
001
001
010
100
100
100
100
100
001
```

The contents of the output data-set confirm that the model is behaving as you intended: five red states (100) followed by five green states (001), followed by one yellow state (010), and then the cycle starts to repeat again with another five red states and so on until the maximum number of values (1024) is reached.

If you want to check the details of the model you have created, you can now create and view the documentation for this model. System Studio will generate a documentation page from the contents of the model, just as it does for the models contained in the model libraries. For details, see “Creating and Viewing Model Documentation” on page 3-24.

This completes this tutorial. You can now save your workspace and exit System Studio at this point if you want.

5

GATED Model Tutorial

This tutorial describes the steps required to create the design for a GATED model. It contains the following sections:

The Gated Model

A GATED model is a hierarchical model that consists of

- One or two implementations
- A gating condition

A GATED model is particularly suitable for creating models in which dynamic switching is needed (such as from acquisition to tracking), or whenever a data flow must be switched, but the internal state (such as a loop filter) must be maintained. (In an OR model, the state is lost when it is reset.)

The following characteristics describe a GATED model:

- If there is only one implementation, the implementation is active only when the gating condition is true.
- When there are two implementations, only one implementation is active at a time, depending on the gating condition. The other implementation is suspended. (The internal state of the suspended implementation is preserved.)
- Output ports that are not driven by the active implementation are latched to their last value.
- Input ports that are not read by the active implementation read and discard samples.

Creating the Model

Use the following procedure to create the model.

1. Start System Studio with the following command:

```
% ccss &
```

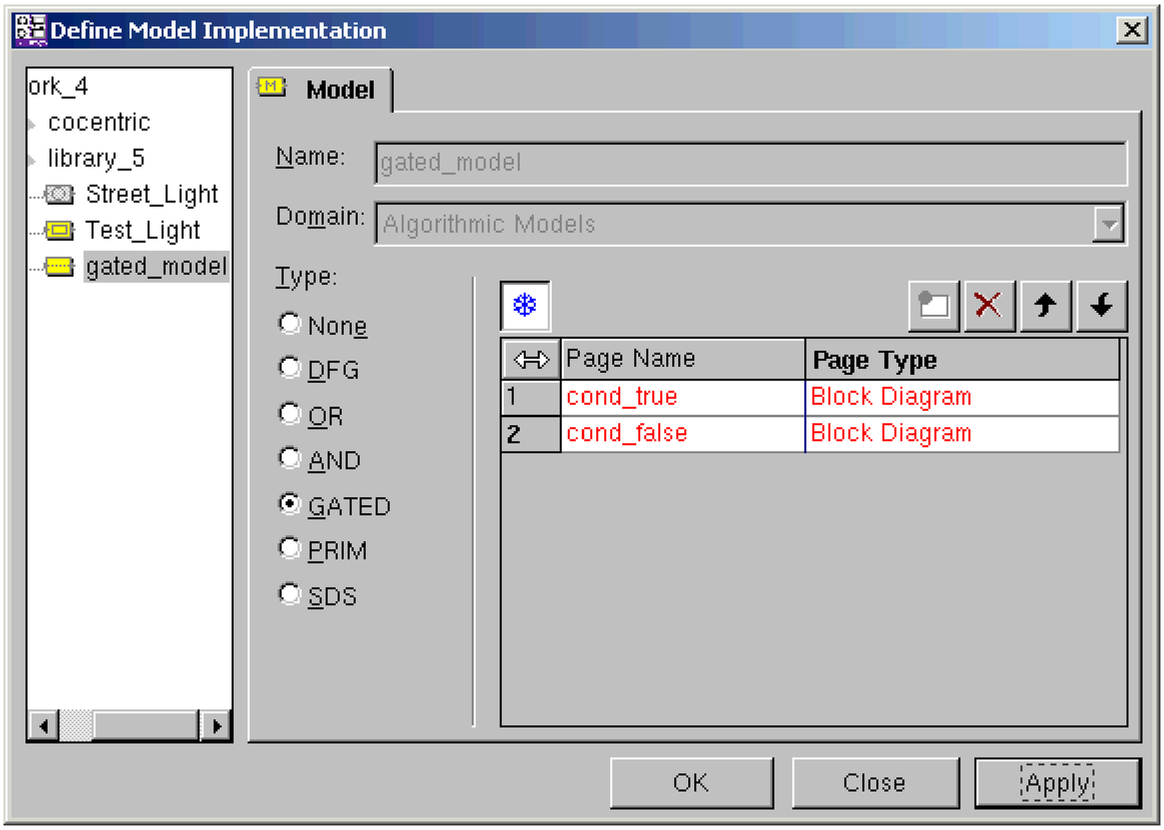
2. Open a workspace and select the library in which you want to create the new model.
3. Choose File > New from the Design Center menu bar. The Create Model dialog box is displayed.
4. Click the Model tab and enter a new name for your model, or use the default name. Click OK.

Defining the Model Behavior

Use the following procedure to define the behavior of the model.

1. Choose Model › Redefine Implementation from the main menu.
2. In the Define Model Implementation dialog box, select GATED.
3. The model will be created as a multipage design, so click the Create New Page button to add a second design page to this model design.
4. Set the Page Type of both pages to Block Diagram (indicating a data flow page), as shown in Figure 5-1, then click OK.

Figure 5-1 Defining the Design Page Types



Name: gated_model

Domain: Algorithmic Models

Type: None [✓ GATED] DFG OR AND PRIM SDS

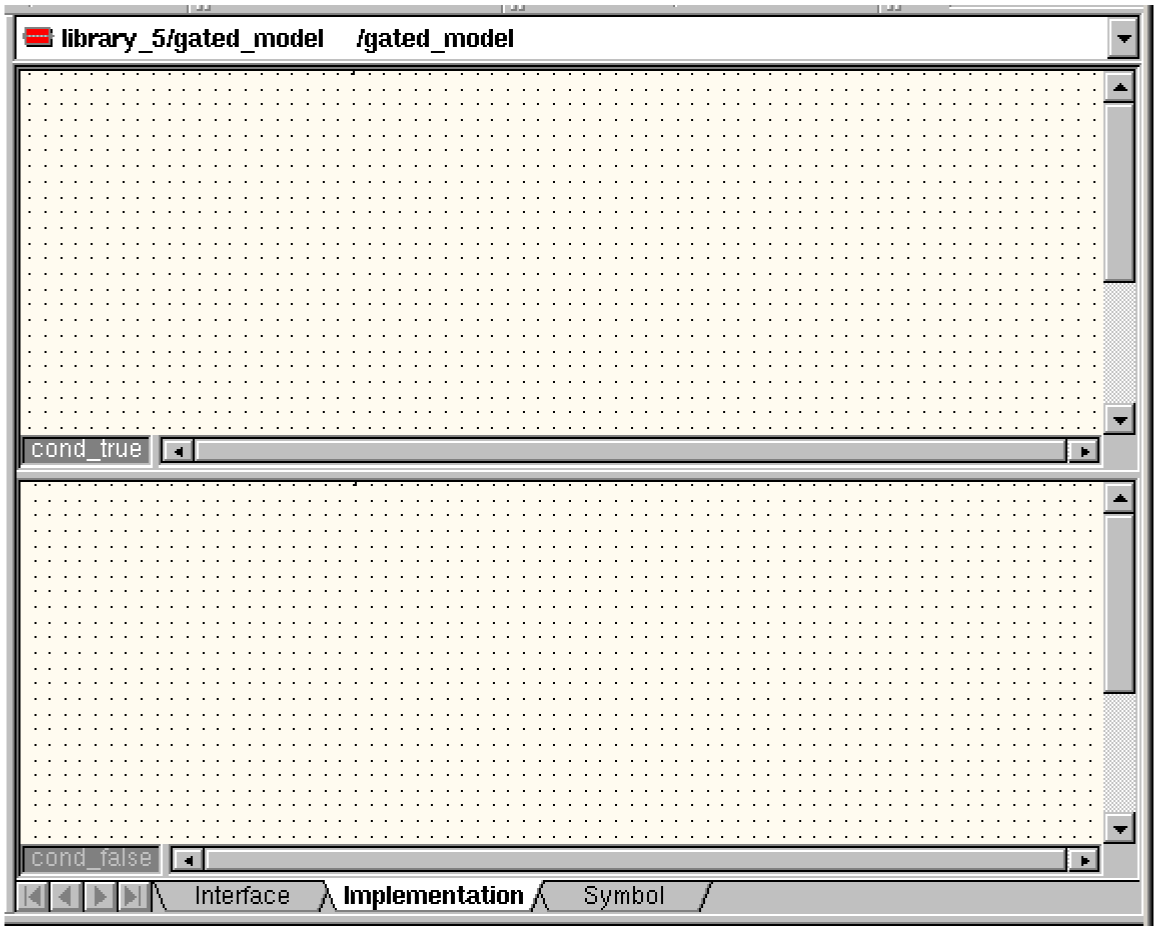
	Page Name	Page Type
1	cond_true	Block Diagram
2	cond_false	Block Diagram

The schematic editor appears in the design window, with one page for the gating_condition_true state, and one for the gating_condition_false state, as shown in Figure 5-2.

The model in a page whose gating condition becomes false is suspended, and will resume when the gating condition becomes true again.

The Dataflow Objects toolbar is now active.

Figure 5-2 Schematic Editor



Defining the Interface

Now you need to define the ports. This gated model will have four ports.

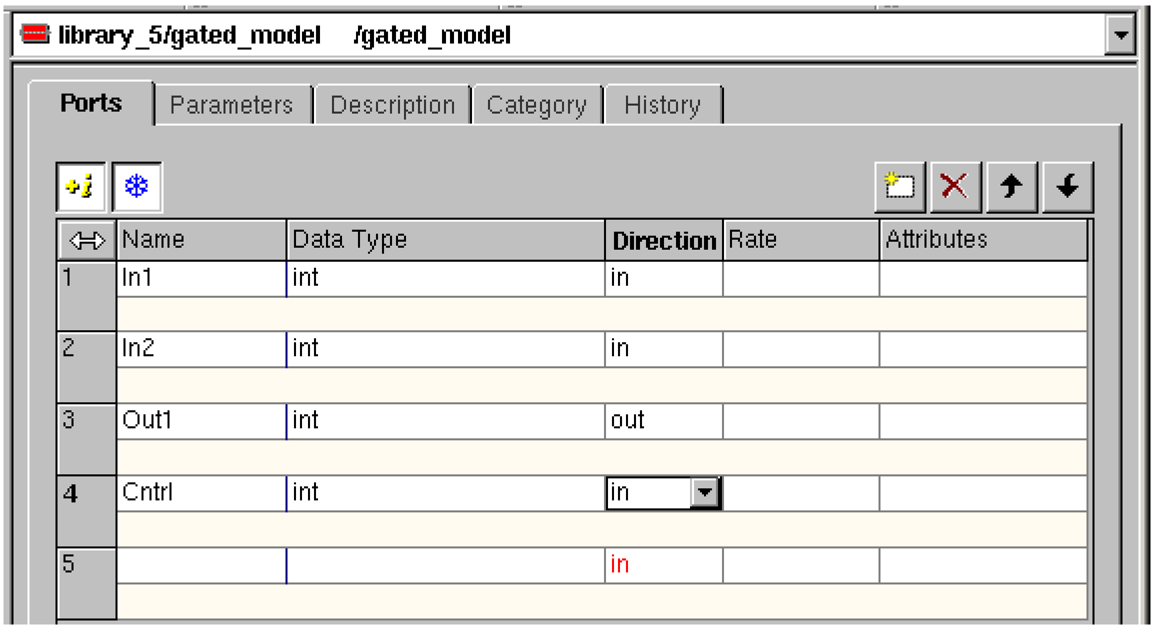
- Click the Interface tab.
- Click the Ports tab.

On the Ports page, the data type and direction fields display a list of valid options for the field. Use the keyboard Tab key to move between fields.

- Create four ports with the following characteristics:
  - Input port 1
  - Name: In1
  - Data Type: int
  - Direction: in
  - Input port 2
  - Name: In2
  - Data Type: int
  - Direction: in
  - Output port 1
  - Name: Out1
  - Data Type: int
  - Direction: out
  - Control port
  - Name: Cntrl
  - Data Type: int
  - Direction: in

  The end result should look like Figure 5-3.

Figure 5-3 Gated Model Interface Definition

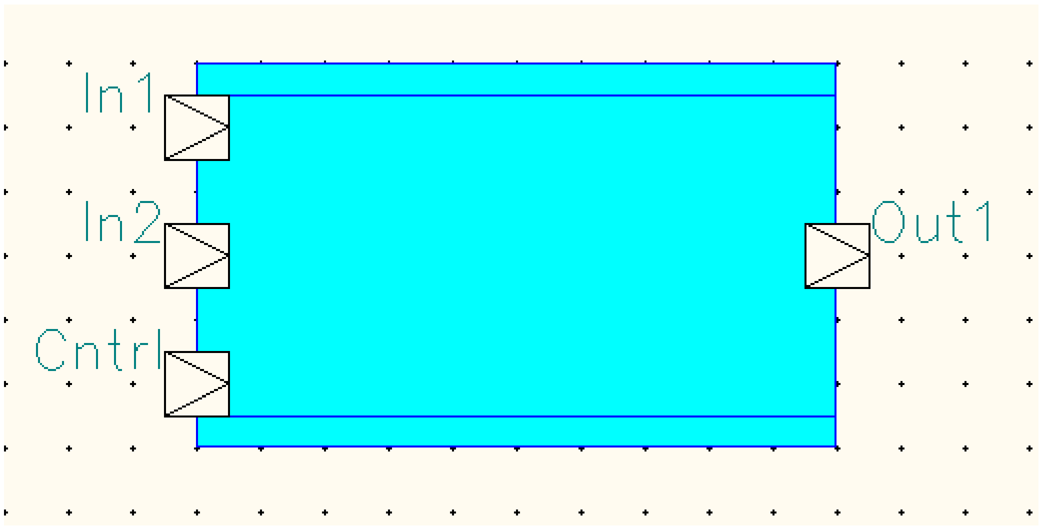

library_5/gated_model /gated_model

	Name	Data Type	Direction	Rate	Attributes
1	In1	int	in		
2	In2	int	in		
3	Out1	int	out		
4	Cntrl	int	in		
5			in		

After you have defined the ports, look at the symbol that was generated for the model by clicking the Symbol tab in the design area.

You will see a symbol with three input ports and one output port, as shown in Figure 5-4.

Figure 5-4   The Model Symbol



The model interface and symbol are now complete.

Creating the Model Schematic

The behavior of this GATED model is as follows:

- If the gating condition is true, the two input ports will be added and the sum placed on the output port.
- If the gating condition is false, the two input ports will be multiplied and the product put on the output port.

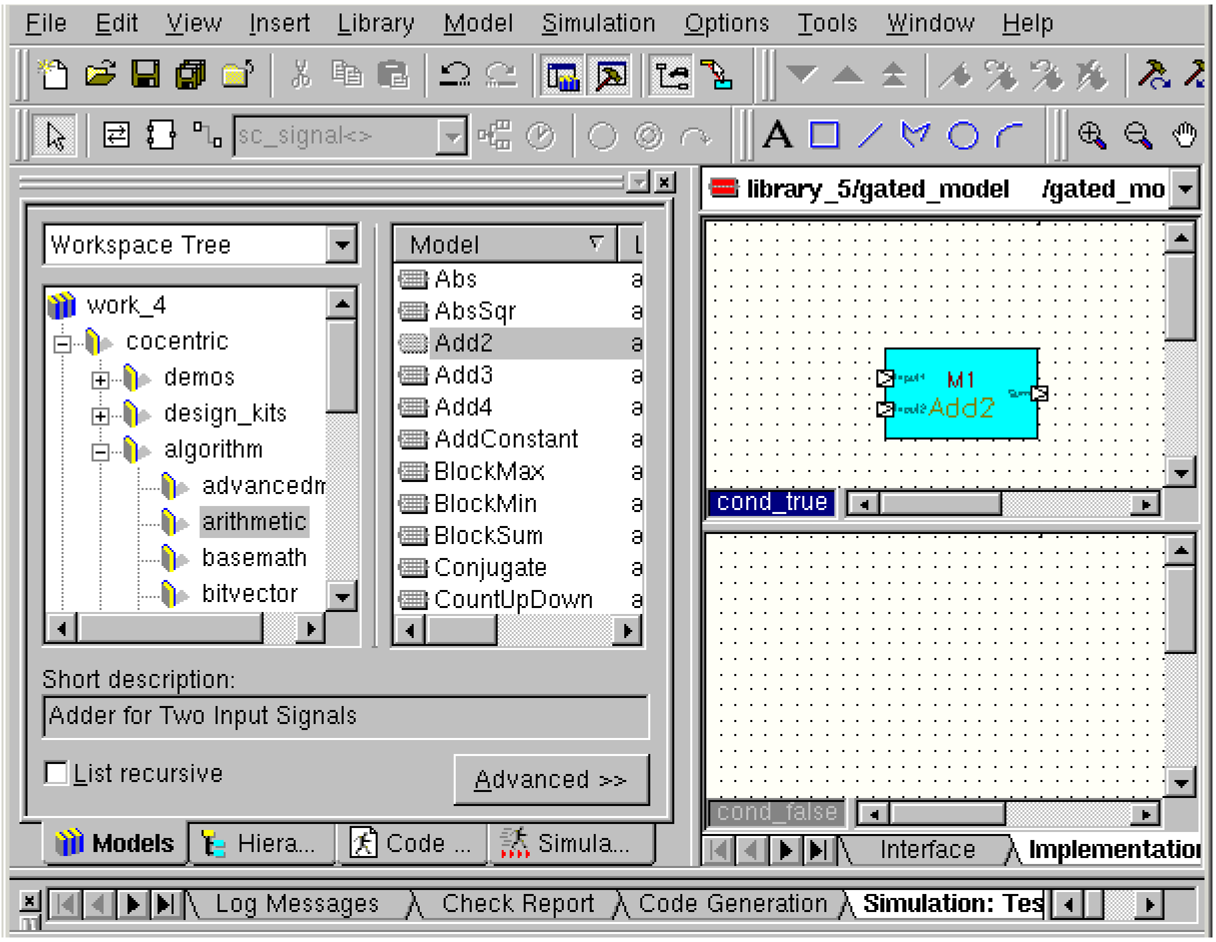
Each condition is represented by a design page that is displayed in a separate part of the design area.

Creating the First Model Page

Select the Implementation tab. The first model page is the GATING_CONDITION_TRUE (cond_true) page. In the window marked cond_true, enter the model schematic as follows:

1. Click the Instance button and add the model Add2 from the System Studio model library “arithmetic,” as shown in Figure 5-5.

Figure 5-5 Inserting a Block

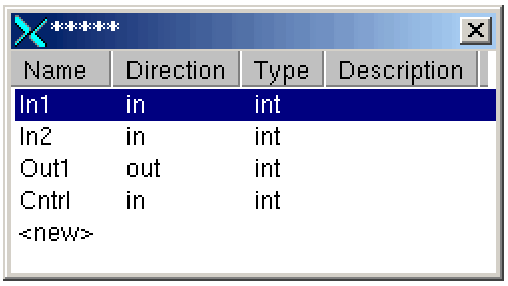


Short description:
Adder for Two Input Signals

2. Select the Add2 block, click the right mouse button and choose Configure Object. Change the value of type parameter T to int. Click OK.

3. Click the insert Port button and select port In1 from the pop-up selection window, as shown in Figure 5-6.

Figure 5-6 Selecting a Port to Insert



Name	Direction	Type	Description
In1	in	int	
In2	in	int	
Out1	out	int	
Cntrl	in	int	
<new>			

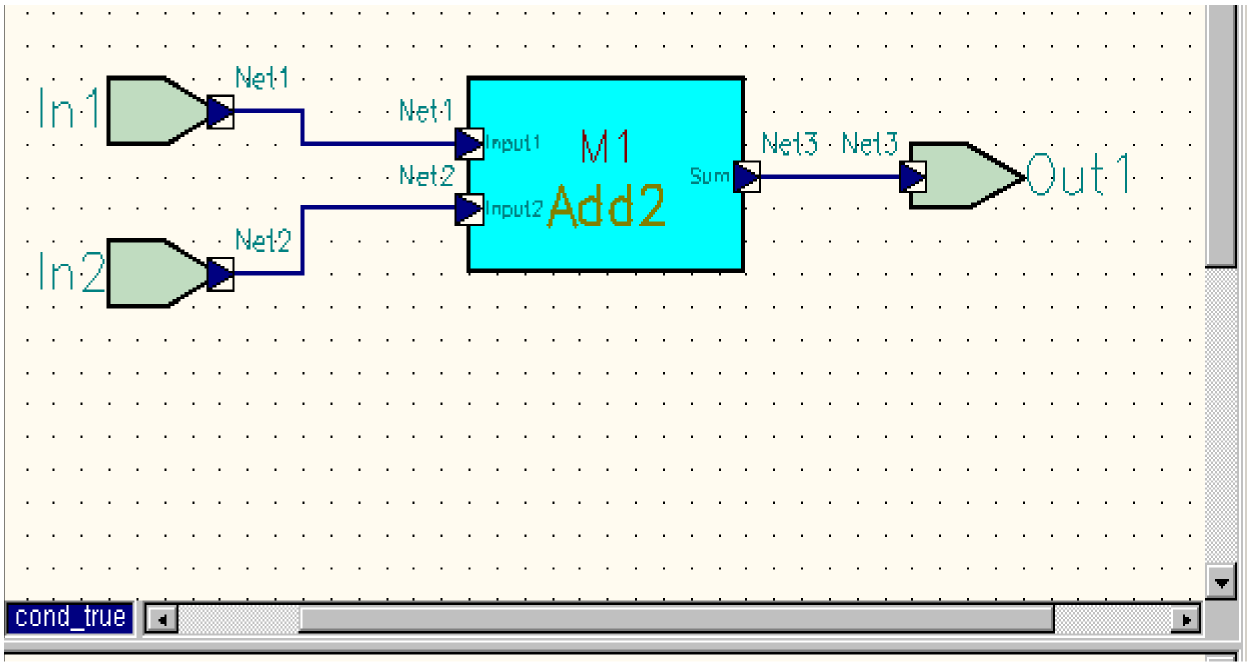
Move the mouse pointer to the design area and position the selected input port in the design, as shown in Figure 5-7.

Repeat these steps for the second input port.

4. Click the insert Port button and position the output port in the design.
5. Make sure that autorouting is enabled.
6. Click the insert Net button and connect the input ports to the input ports of the add model. Then connect the output port of the add model to the output port.

The completed design should look like Figure 5-7.

Figure 5-7 The First Page Completed



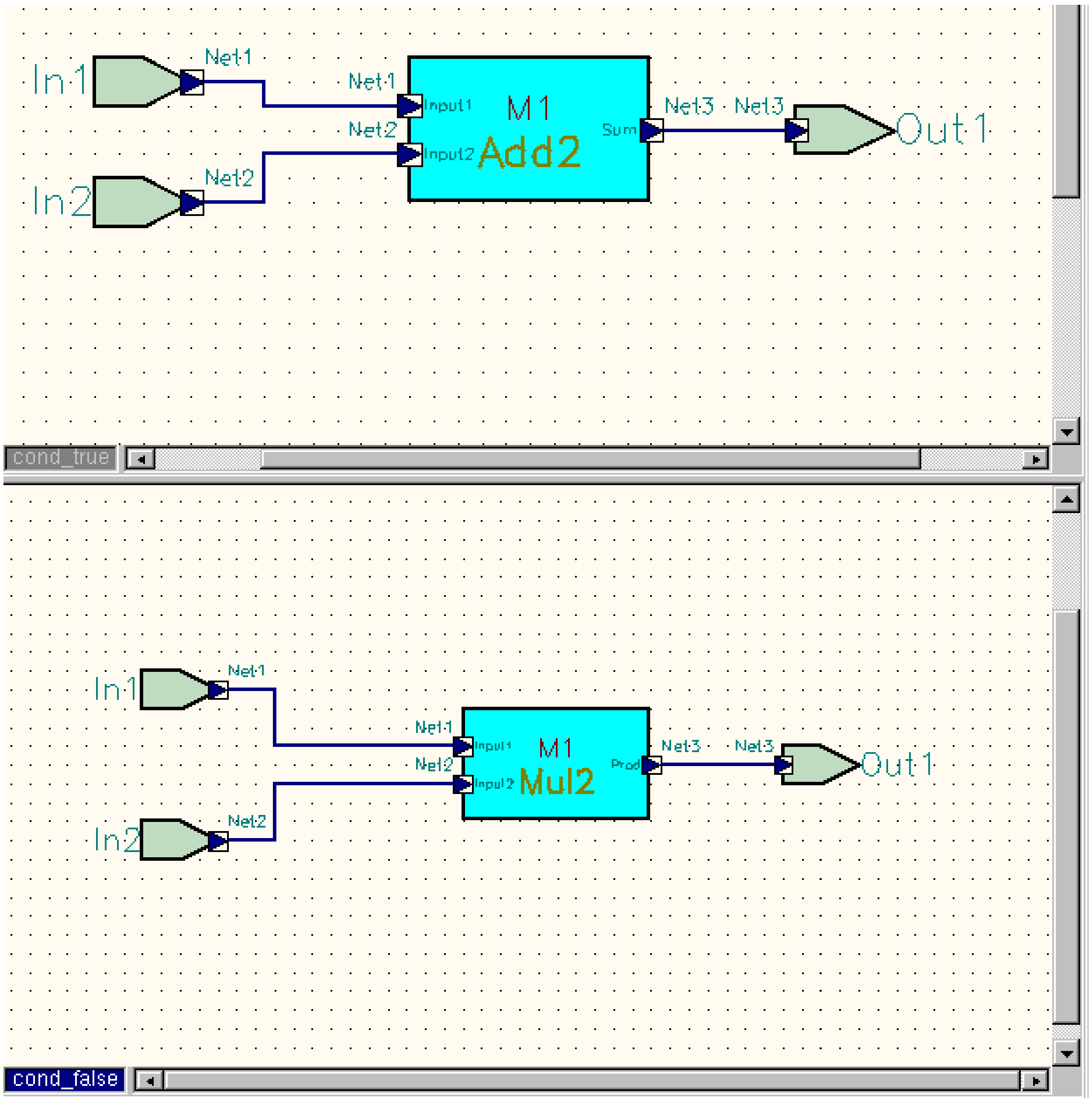
Creating the Second Model Page

The second design page is the GATING_CONDITION_FALSE (cond_false) page. This design contains a Mul2 model from the cocentric/arithmetic library, two input ports, and one output port.

- First click in the cond_false section of design area, then select and position the components in the design, as you did for the first page.
- Change the value of the type parameter T for the Mul2 instance from float to int.
- Connect the input ports to the input ports of the Mul2 model and output of the Mul2 model to the output port.

The completed design should look like Figure 5-8.

Figure 5-8 The Second Page Completed



The model design is now complete.

Defining the Gating Condition

The next step is to define the gating condition for the model. You need to enter only the `gating_condition_true` expression. System Studio will determine the `gating_condition_false` condition itself.

- Choose Model › Configure Implementation from the Design Center main menu. The Configure Implementation dialog box is displayed
- Click the Gating Condition tab. In the input area, enter the following expression:

  ```
  Cntrl == 1
  ```

- Click OK.

Finishing Up

✓ Click the Check Model button to check the model for errors.

To simulate the model, you need to connect the model's input ports to a testbench. For example, create a new model and instantiate this model in that new model. You could then choose a simple 1.0 pulse generator for the control input, set some constants for the values of a and b, and then monitor the output with DAVIS.

This completes this tutorial. You can now save your workspace and exit System Studio if you want.

6

AND Model Tutorial

This tutorial describes the steps to follow to create a simple AND model. It contains the following sections:

The AND Model

An AND model is a hierarchical model used for modeling concurrency.

AND models define multiple implementations (each implementation is called a page). The implementations can be FSM (Finite State Machine) models, DFG (data flow graph) models, or a mixture of both. An AND model contains at least 2 pages, but may have up to 16 pages.

The pages (implementations) of an AND model execute concurrently. The pages communicate internally by using signals. For example, one FSM page can use a signal to control the state transitions of another page. Each FSM page can have local variables.

Note:

Each FSM must exit before the AND model itself exits.

In this tutorial you will create a simple two-page AND model to modulate the output of a sine wave generator. This example demonstrates the features of an AND model that uses a global signal to transfer information between two pages.

The AND model’s pages are an OR page and a DFG page:

- The OR page of the AND model consists of a two-state FSM. On entry, the model increments the value of a local variable value by 0.5 until a maximum value (20.0) is reached. The OR model then switches to the second state and decrements the local variable value by 0.5. When the value of the local variable is zero, the FSM switches back to the first state. This state switching is repeated endlessly.

- The value of the local variable is copied to the signal sig1, which is accessible in the DFG page.
- The DFG page contains a SineGenerator block, Mul2, and a WriteSignal block. The output of the SineGenerator is multiplied by the value of signal sig1.

Creating the AND Model

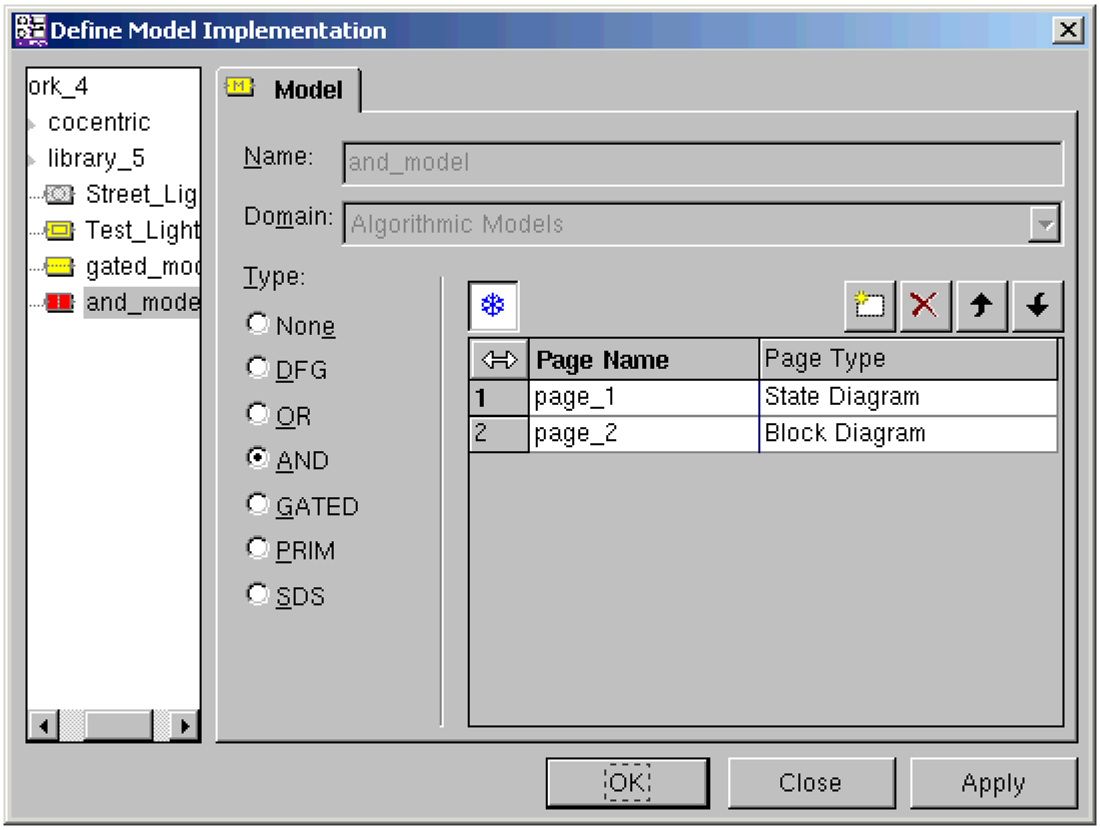
Use the following procedure to create the AND model.

1. Start System Studio by entering the following command:

   ```
   % ccss &
   ```

2. If you have not already created a library, follow the instructions in “Creating a Library” on page 3-5.
3. Open the workspace and select the library in which you want to create the new model.
4. Choose File › New from the System Studio main menu. The Create Model dialog box opens.
5. Enter a new name for your model or use the default name, and select AND as the model type. For basic information on creating a model see “Creating the Model Structure” on page 3-3.
6. The model to be created has two pages. By default the page type of both pages is set to State Diagram (indicating a control flow page). Click Model › Redefine Model Implementation and set the Page Type of page_2 to Block Diagram (indicating a data flow page), as shown in Figure 6-1.

Figure 6-1   Creating the AND Model



Define Model Implementation

ork_4
cocentric
library_5
Street_Lig
Test_Light
gated_moc
and_mode

Model

Name: and_model

Domain: Algorithmic Models

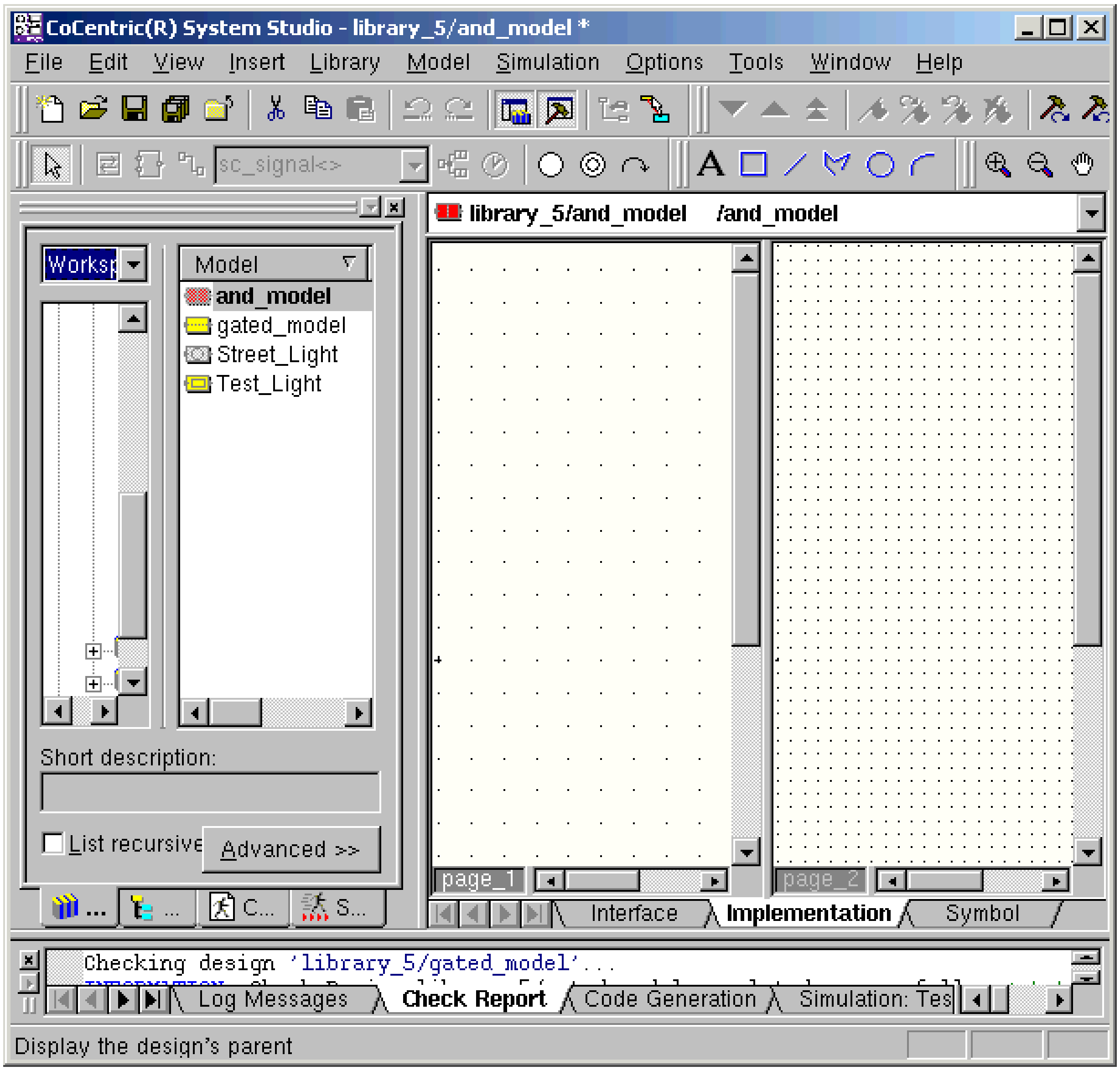
Type:
None
DFG
OR
[✓ AND]
GATED
PRIM
SDS

	Page Name	Page Type
1	page_1	State Diagram
2	page_2	Block Diagram

7. Click OK.

   The model is then opened with a window displayed for each of the two pages of the model, as shown in Figure 6-2.

Figure 6-2 The Empty AND Model



```
Checking design 'library_5/gated_model'...
```

Display the design's parent

Note that the FSM objects toolbar is now active.

Defining the Model Parameters

The next step is to define the parameters of the model.

- In the design area, click the Interface tab and then click the Parameter tab.
- To add a parameter, click the “Create new parameter” button, or enter the name of the new parameter in the next empty parameter name field. You can then complete the fields for the new parameter.
- If you wish, you can add a description of the parameter in the comments field; this description will help to document your design and will appear in the model’s documentation (choose Model › Show Documentation).

  You can show or hide the descriptions by clicking the “show descriptions ...” button on or off.

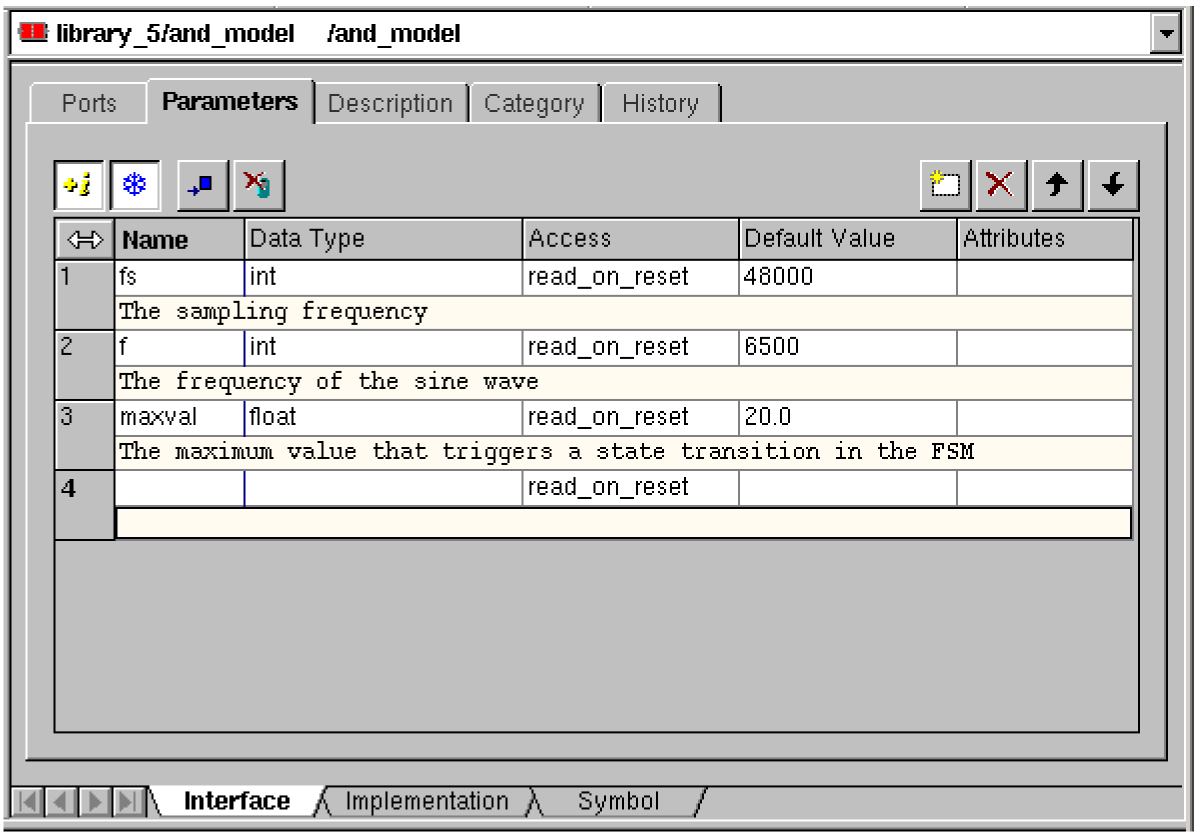
- Create three parameters with the following characteristics (see Figure 6-3):

  - Name: fs
    Data Type: int
    Default Value: 48000
    Comments (optional): The sampling frequency.

  - Name: f
    Data Type: int
    Default Type: 6500
    Comments (optional): The frequency of the sine wave.

- Name: maxval
  Data Type: float
  Default Type: 20.0
  Comments (optional): The maximum value, which triggers a state transition in the FSM.

Figure 6-3 Defining the Model Parameters



	Name	Data Type	Access	Default Value	Attributes
1	fs	int	read_on_reset	48000	
	The sampling frequency				
2	f	int	read_on_reset	6500	
	The frequency of the sine wave				
3	maxval	float	read_on_reset	20.0	
	The maximum value that triggers a state transition in the FSM				
4			read_on_reset		

Declaring the Local Variables

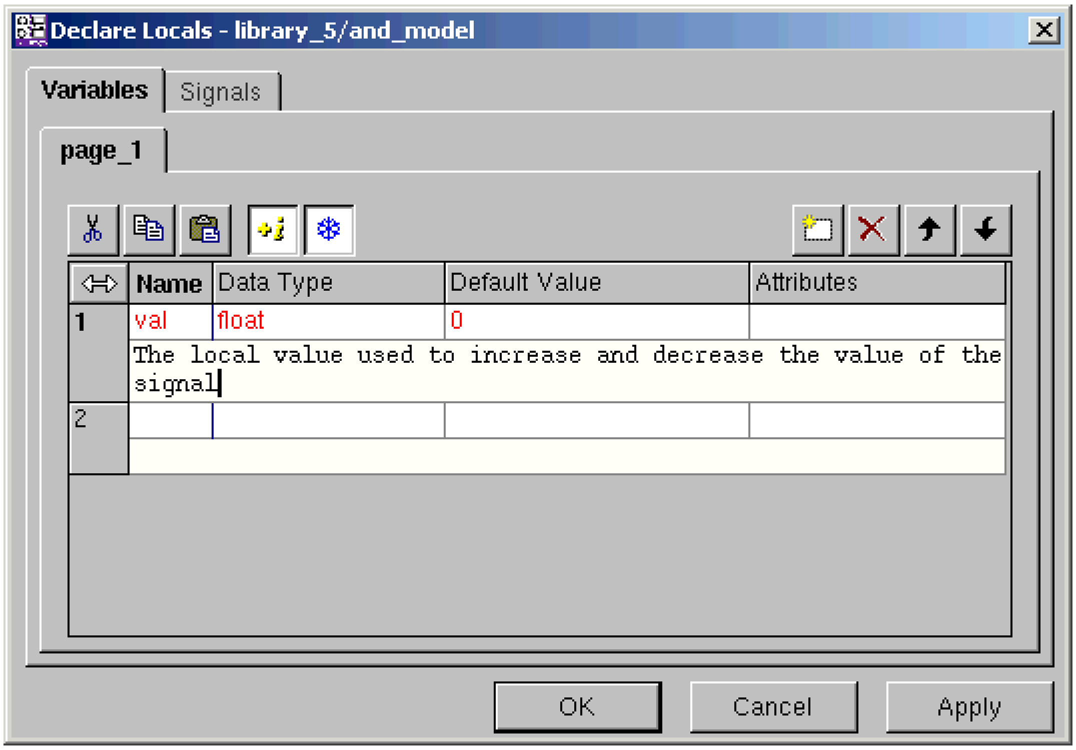
The next step is to define the local variables for the FSM page of the model.

Note:

The scope of a local variable is limited to the current page. If there was more than one FSM page, you would have to declare the variables separately for each FSM page.

- In the Implementation view, select the first page (page_1) of the model by clicking anywhere in the page.
- Choose Model › Declare Locals from the System Studio menu bar. The Declare Locals dialog box opens.
- In the Declare Locals dialog box, click the Variables tab and define a variable with the following characteristics (see Figure 6-4):
  - Name: val
  - Data Type: float
  - Default Value: 0
  - Description (optional): The local value used to increase and decrease the value of the signal.
- Click Apply.

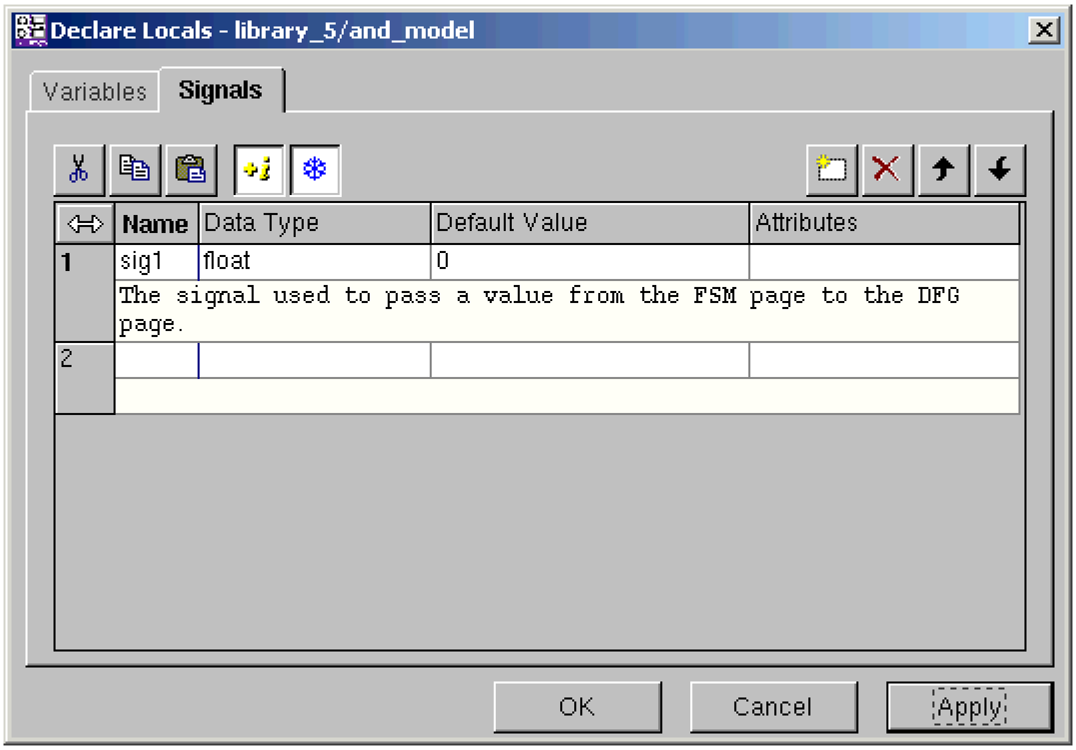
Figure 6-4 Defining the Variables



	Name	Data Type	Default Value	Attributes
1	val	float	0	
	The local value used to increase and decrease the value of the signal			
2				

- Click the Signals tab and define a signal with the following characteristics (see Figure 6-5):
  - Name: sig1
  - Data Type: float
  - Default Value: 0
  - Description (optional): The signal used to pass a value from the FSM page to the DFG page.
- Click OK.

Figure 6-5 Defining the Signals



	Name	Data Type	Default Value	Attributes
1	sig1	float	0	
	The signal used to pass a value from the FSM page to the DFG page.			
2				

Unlike local variables, whose scope is limited to the current page, the scope of a signal is the whole of the current model. Therefore, you can use signals to communicate between the pages of a model.

After the signal has been declared, you will notice that it is visible in the Signals page of the Declare Locals dialog box for both pages of the model.

The model definition phase is now complete.

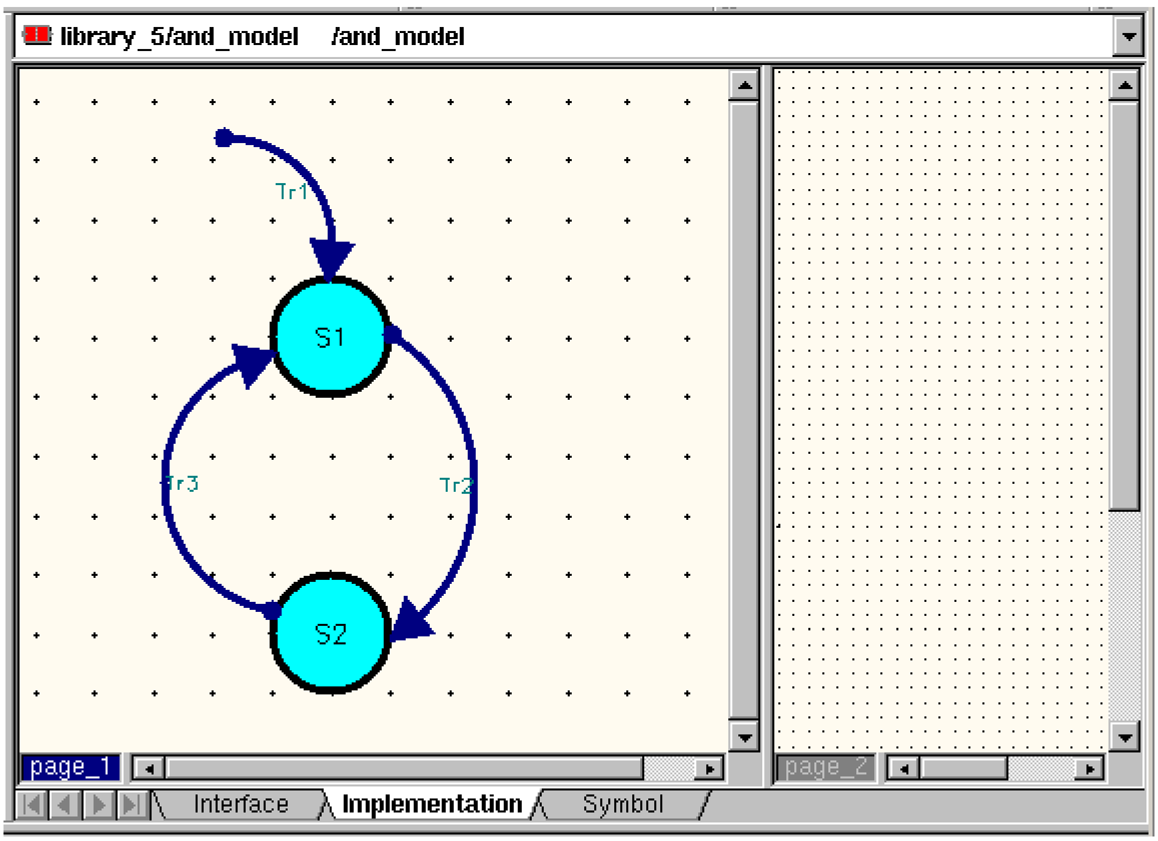
Creating the FSM Model

The FSM in page_1 of the AND model consists of two atomic states (S1 and S2) and three transitions (Tr1, Tr2 and Tr3). You will now add these to page_1 of the AND model.

- Select the required object from the FSM objects toolbar, move the cursor to the location in the design where you want to position it, and then click the left mouse button.
- To enter a transition, click the Transition button in the FSM objects toolbar. Draw the transition from one state to another. For more detailed description of these steps, see "Creating the Model Schematic" on page 4-7.

At this point, the FSM should look similar to the display shown in Figure 6-6.

Figure 6-6 The FSM Page



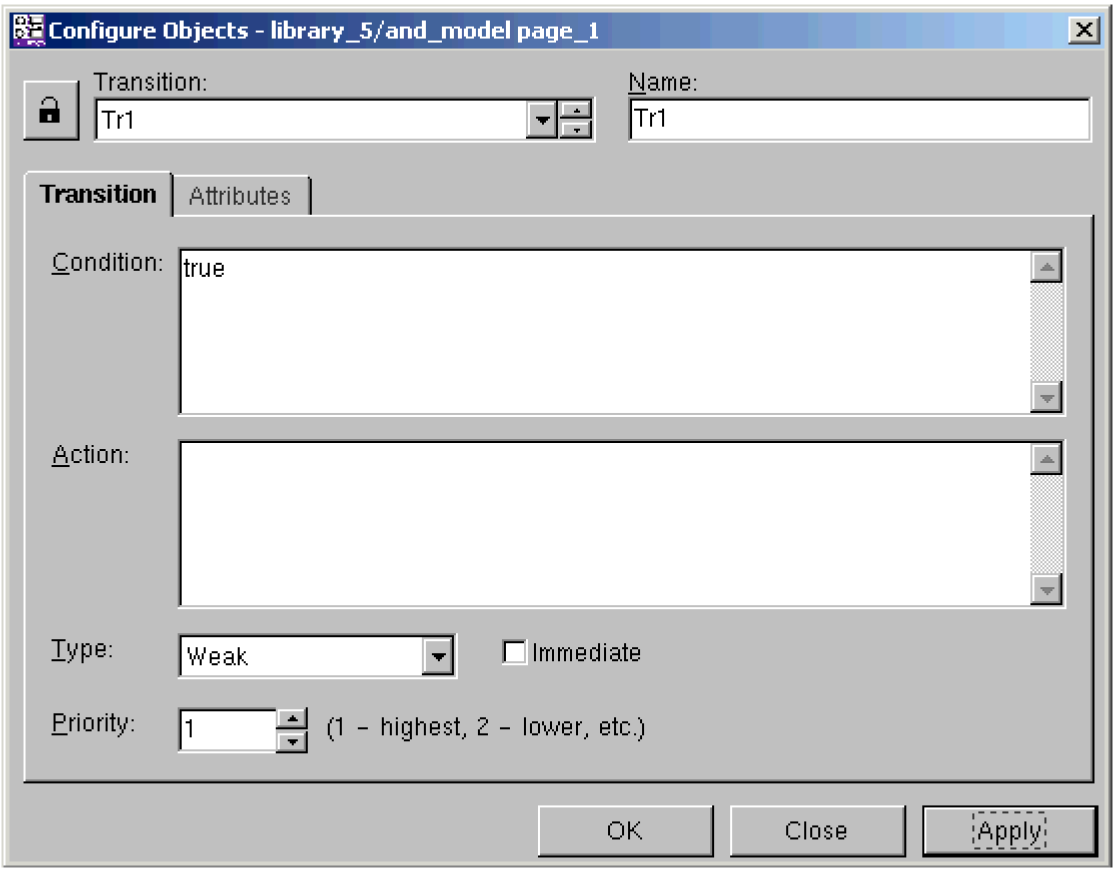
Defining the Transitions

- Select the transition that you want to define.
- Open the Configure Object dialog box by using one of the following two methods:

- Choose Model › Configure Object from the System Studio main menu.
- Click the right mouse button while the cursor is in the design and choose Configure Object from the pop-up menu.

The Configure Objects dialog box opens, as shown in Figure 6-7.

Figure 6-7 Defining a Transition



Configure Objects - library_5/and_model page_1

Transition: Tr1

Name: Tr1

Condition: true

Action:

Type: Weak

☐ Immediate

Priority: 1 (1 – highest, 2 – lower, etc.)

- Define the following transitions and set the following conditions and actions:

  Transition Tr1: start to S1

  - Condition: true (defaults to true)

- Action:
- Type: weak
- Priority: 1

Transition Tr2: S1 to S2

- Condition: val > maxval
- Action:
- Type: weak
- Priority: 1

Transition Tr3: S2 to S1

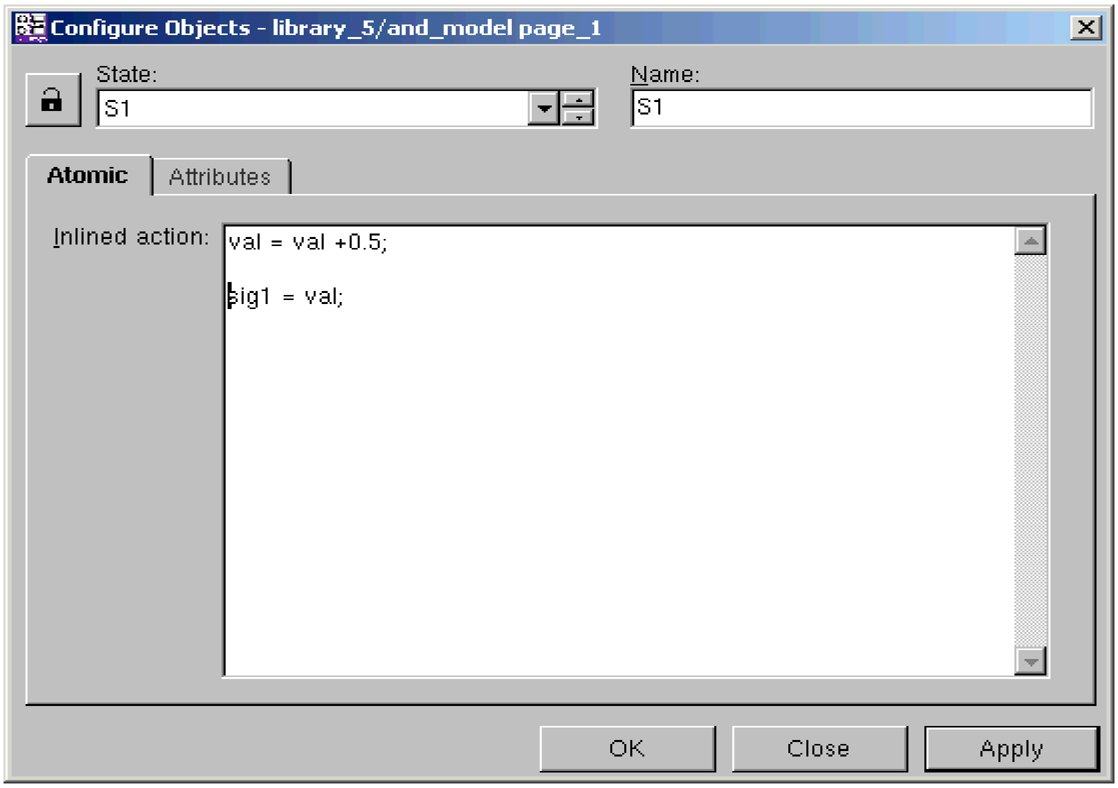
- Condition: val <=0
- Action:
- Type: weak
- Priority: 1

Defining the Inline Actions

- To define the inline actions for an atomic state, select the state in the design and check that the state is highlighted.
- Then open the Configure Object dialog box by either:
  - Choosing Model › Configure Object from the System Studio main menu.
  - Clicking the right mouse button while the cursor is in the design and choosing Configure Object from the pop-up menu.

The Configure Objects dialog box opens, as shown in Figure 6-8.

Figure 6-8 Defining an Inline Action


Configure Objects - library_5/and_model page_1

State: S1

Name: S1

Inlined action:

```
val = val +0.5;

sig1 = val;
```

- Enter the definitions for the atomic state in the Inlined Action field and then click OK.

  Note: It is through the definition of the inlined actions that the value of the signal, sig1, is bound to the value of the local variable, val.

- Define the following atomic state inline actions:

  For atomic state S1, define the inline action as follows:

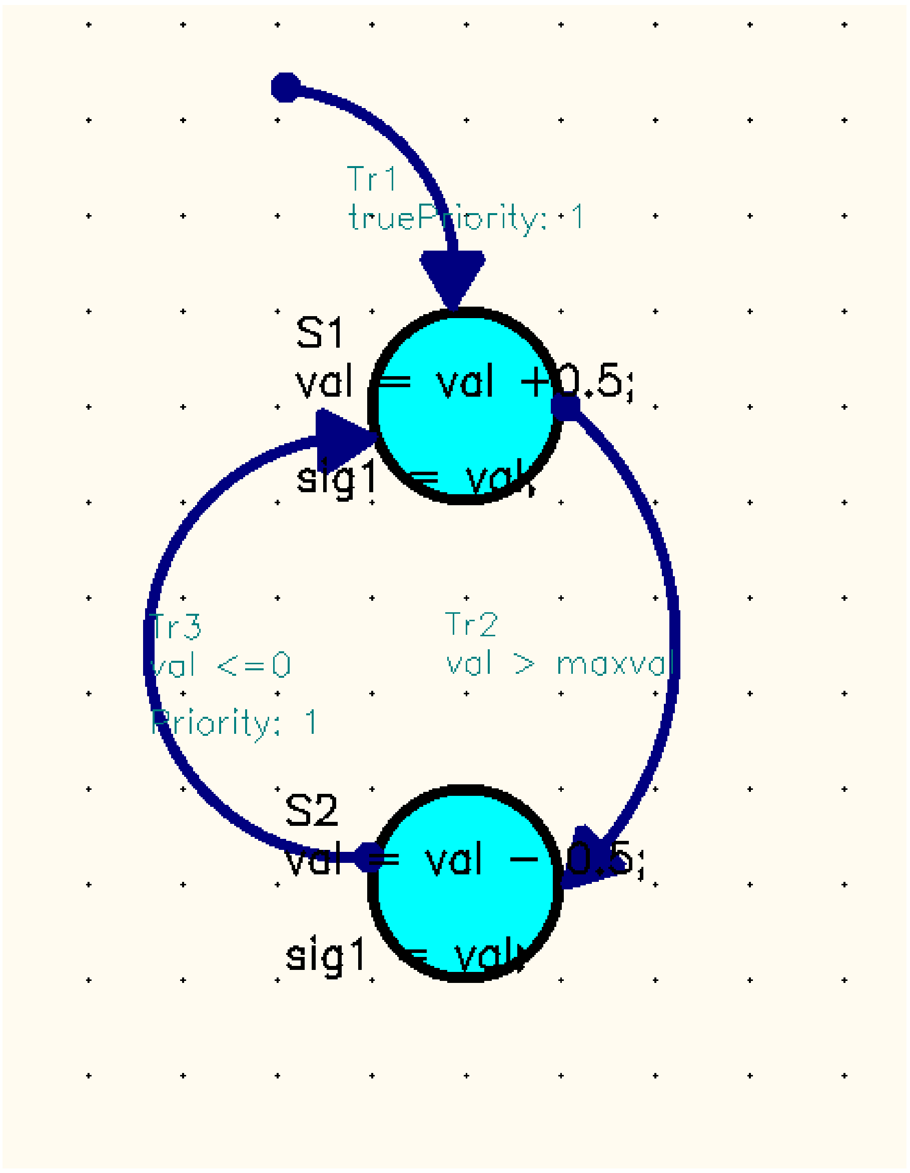
  - val = val +0.5;
  - sig1 = val;

  For the atomic state S2, define the inline action as follows:

- val = val - 0.5;
- sig1 = val;

The FSM model is now complete. It should now look something like the model shown in Figure 6-9.

Figure 6-9 The Completed FSM Model in Page 1



Creating The DFG Model

The DFG model consists of three blocks: a sine wave generator, a two-input multiplier, and a sink. All three blocks are existing models taken from the System Studio libraries.

Make page_2 the active page by clicking the cursor in page_2. You are now ready to create the DFG model.

Inserting the Blocks

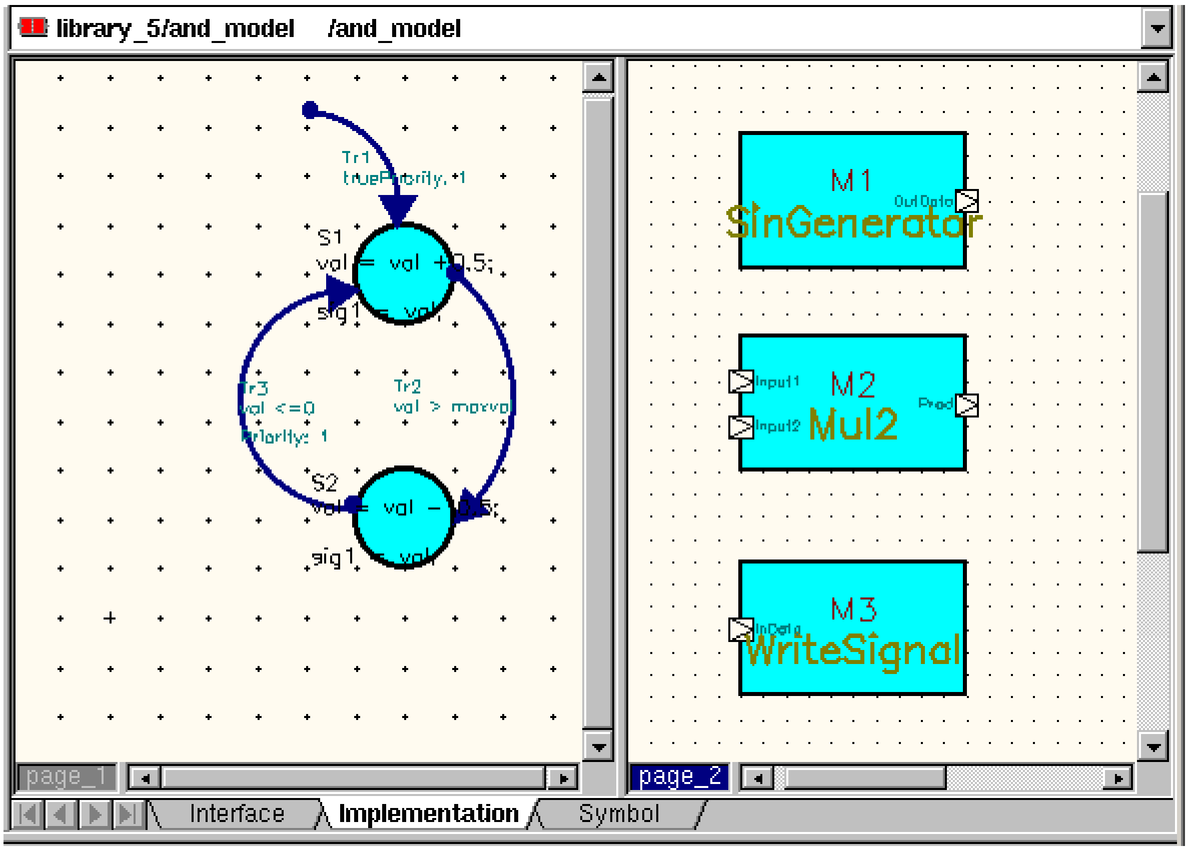
If the cocentric/algorithm library is collapsed, expand the library by clicking the plus sign (+) next to it in the Workspace window; then select the “source” library. The models in this library are shown in the Model area.

- Click the Instance button in the dataflow objects toolbar.
- Select the SinGenerator model from the source library.
- Move the cursor into the design area. You will see the outline of the block displayed as you move the cursor around.
- Click the left mouse button when the outline of the block is positioned where you want to insert the block.
- Now move the cursor back into the Workspace window and select the Mul2 block from the algorithm/arithmetic library.
- Position the Mul2 block below the SinGenerator block, as shown in Figure 6-10, and click the left mouse button.
- Now select the WriteSignal block from the algorithm/sink library.
- Position the WriteSignal block below the Mul2 block, as shown in Figure 6-10, and click the left mouse button.

Note:

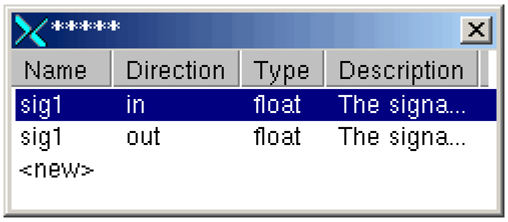
As an alternative, you can simply “drag and drop” the block you want by selecting it in the model list, dragging it into the design, and dropping it where you want it to be located.

Figure 6-10 Creating the DFG



- Click the insert Port button. A dialog box showing a list of available ports, is displayed, as shown in Figure 6-11.

Figure 6-11 Selecting the Port sig1



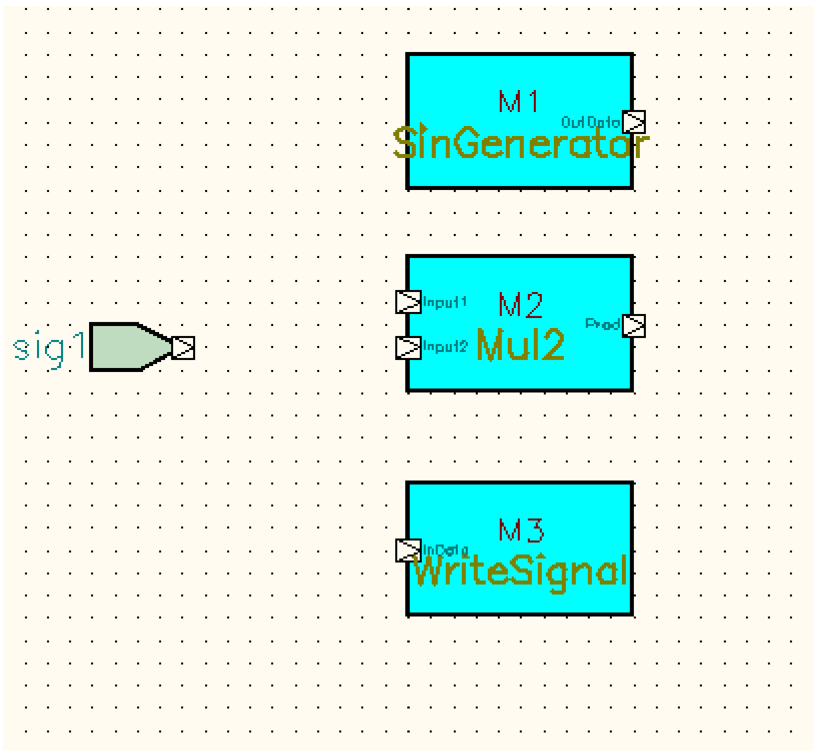
Name	Direction	Type	Description
sig1	in	float	The signa...
sig1	out	float	The signa...
<new>			

- Select port “sig1 in” and position it in the design as shown in Figure 6-12.
- Click the middle mouse button to cancel the insert Port mode.

Note:

In algorithmic models, signals are treated as ports. No ports have been defined for this model, but sig1 has been declared as a local variable and is therefore available under this menu option. The signal can be instantiated as either and input or an output. In this case it is an input.

Figure 6-12 Inserting the Port sig1



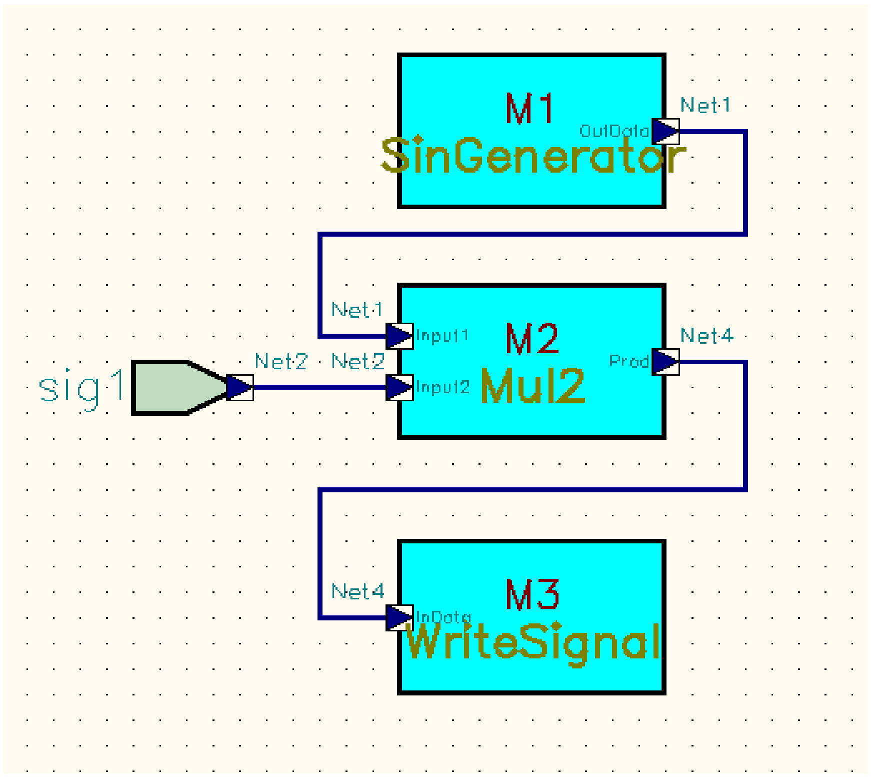
Inserting the Nets

The next step is to connect the blocks of the DFG together with nets:

- Click the insert Net button in the dataflow objects toolbar.
- Position the cursor over the output port of the SinGenerator block.
- Press and hold the left mouse button, and create a net by dragging a line from the SinGenerator output port to the first input port of the Mul2 block.
- Release the mouse button; the two blocks are now connected.
- Now position the cursor over port sig1.

- Press and hold the left mouse button, and create a net by dragging a line from port sig1 to the second input port of the Mul2 block.
- Release the mouse button; the two blocks are now connected.
- Now position the cursor over the output port of the Mul2 block.
- Press and hold the left mouse button, and create a net by dragging a line from the Mul2 output port to the input port of the WriteSignal block.
- Release the mouse button; the two blocks are now connected.
- Click the middle mouse button to cancel the Insert Net mode. The end result should look similar to the example shown in Figure 6-13.

Figure 6-13 The Completed DFG

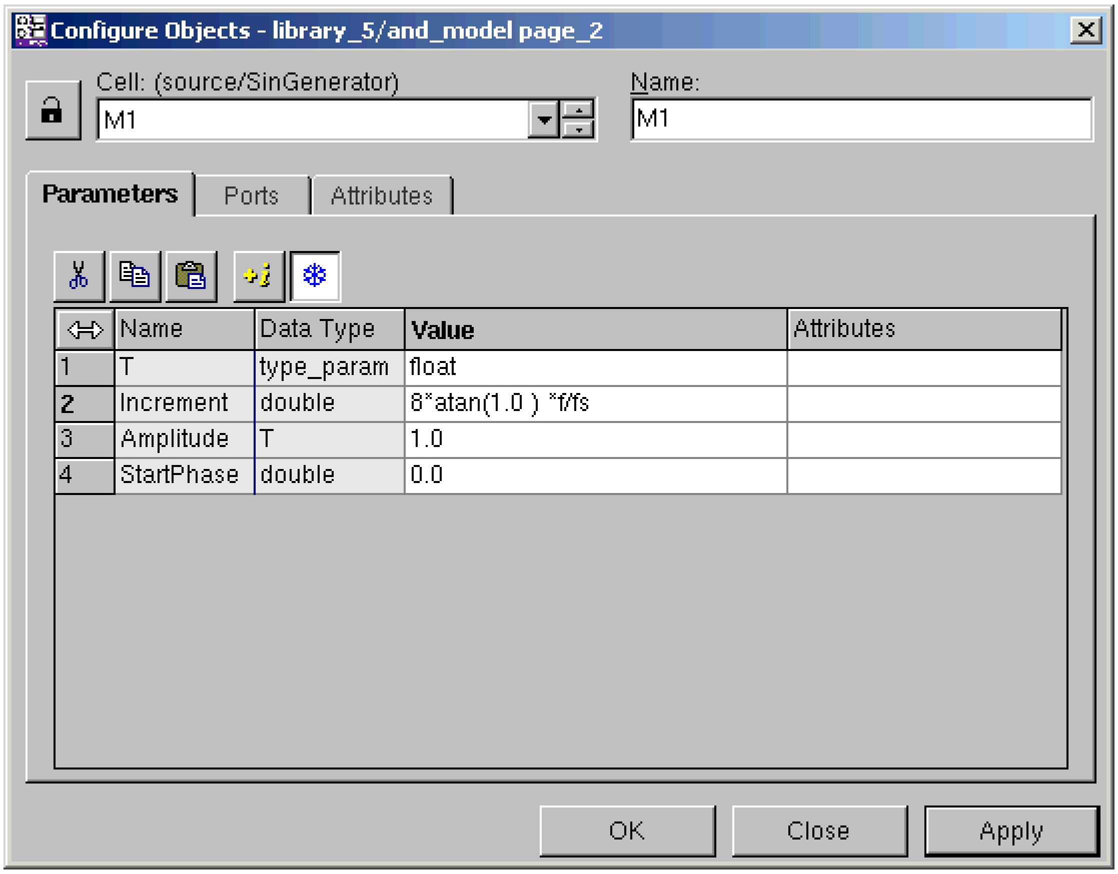


Setting the Parameter Values

To finish the design, the blocks of the DFG have to be parameterized.

- Position the mouse over the instance of the SinGenerator and click the right mouse button.
- Choose Configure Object from the context-sensitive menu. The Configure Objects dialog box opens as shown in Figure 6-14.

Figure 6-14 Configuring a DFG Block



Cell: (source/SinGenerator) M1

Name: M1

⟺	Name	Data Type	Value	Attributes
1	T	type_param	float	
2	Increment	double	8*atan(1.0) *f/fs	
3	Amplitude	T	1.0	
4	StartPhase	double	0.0	

For your convenience, some of the parameters have default values. You can ignore these values.

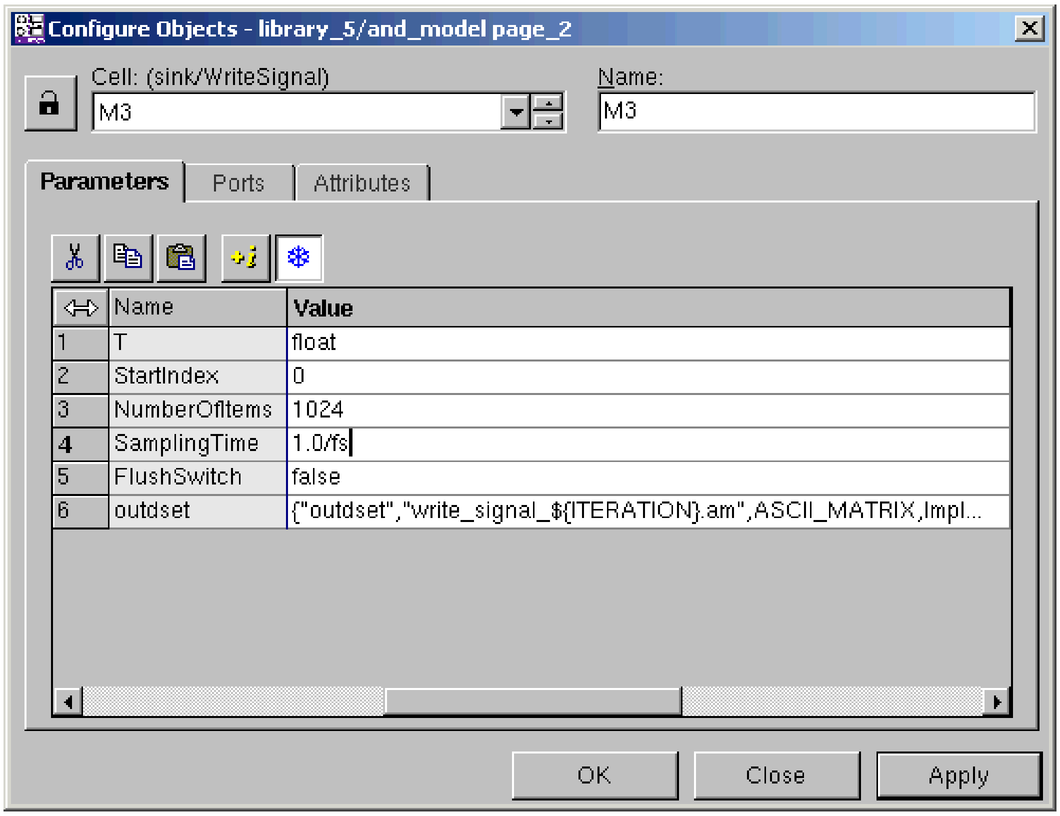
- Assign the following value to the parameter Increment of the SinGenerator block (as shown in Figure 6-14):

  Block: SinGenerator

  - Parameter: Increment
  - Value: 8*atan(1.0)*f/fs
- Click OK.
- Repeat these steps for the WriteSignal block to assign the following value to the parameter SamplingTime (as shown in Figure 6-15):

  Block: WriteSignal

Figure 6-15 Configuring the WriteSignal Block



Cell: (sink/WriteSignal) M3

Name: M3

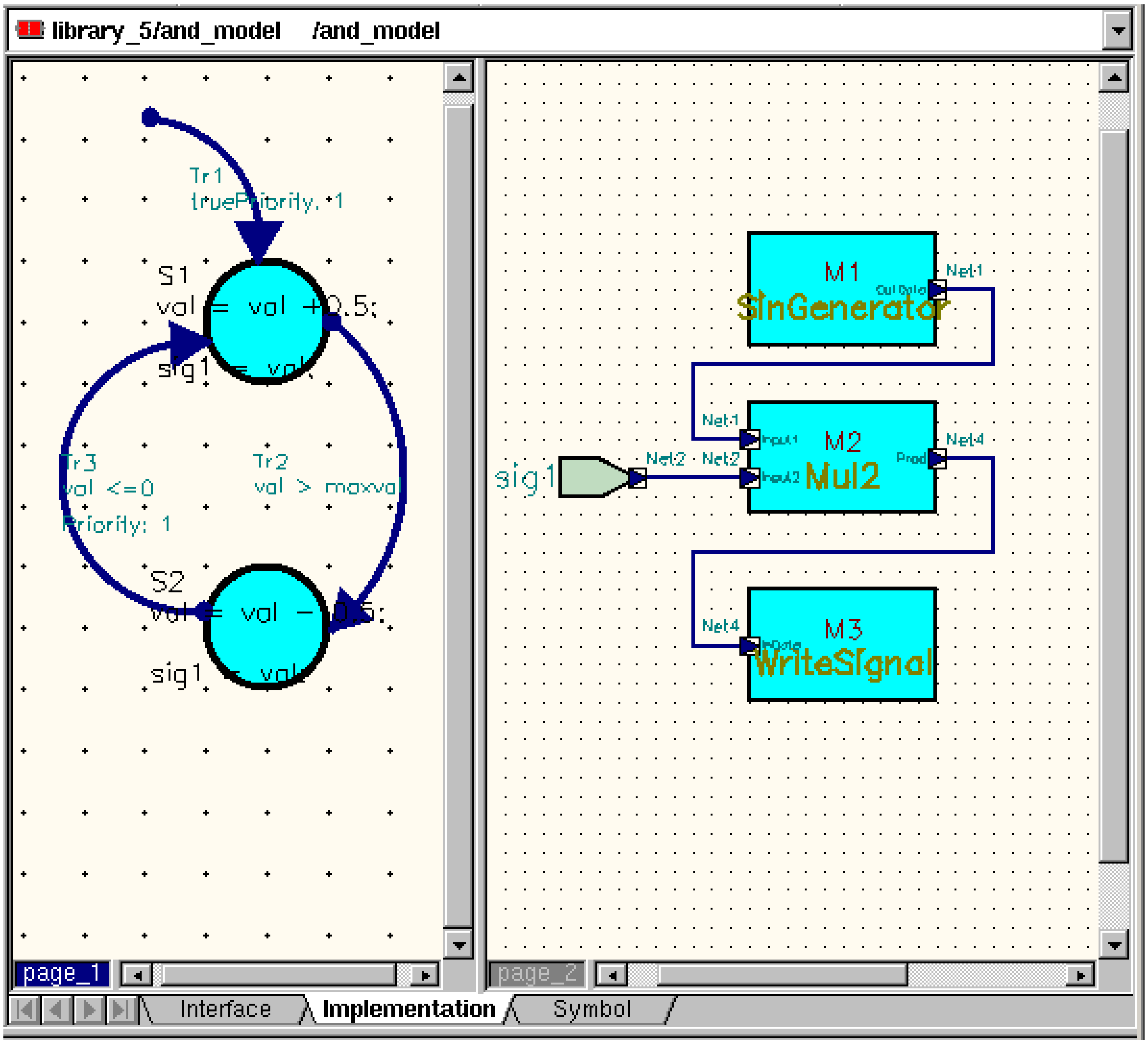
	Name	Value
1	T	float
2	StartIndex	0
3	NumberOfItems	1024
4	SamplingTime	1.0/fs
5	FlushSwitch	false
6	outdset	{"outdset","write_signal_${ITERATION}.am",ASCII_MATRIX,Impl...

- Parameter: SamplingTime
- Value: 1.0/fs

- Click OK.

The AND model is now complete; it should look like the design shown in Figure 6-16.

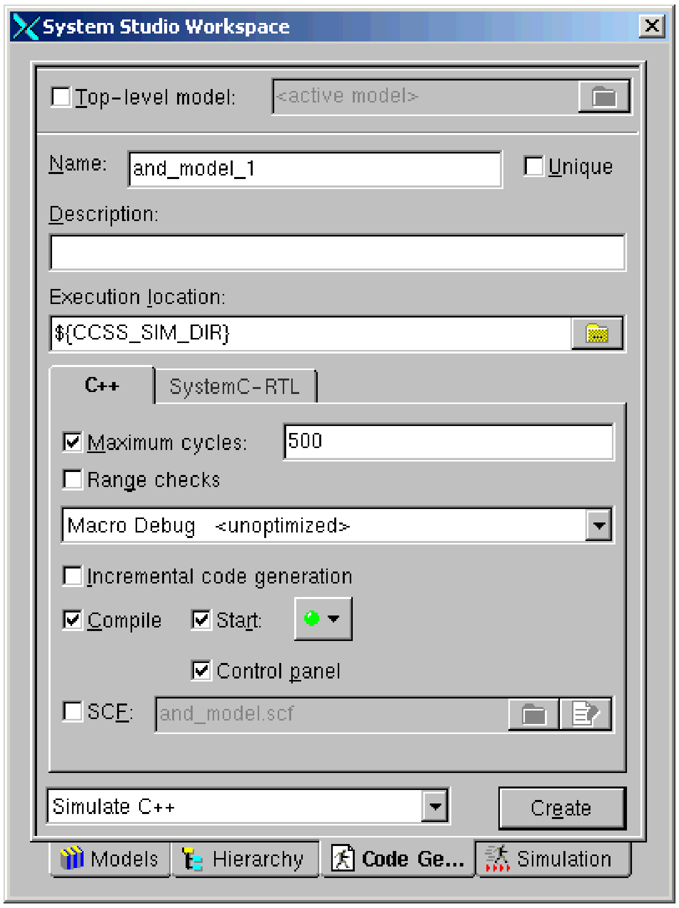
Figure 6-16 The Completed AND Model



Simulating the Design

- Check that the design has no errors by clicking the Check Design button in the System Studio toolbar and observing any messages that appear in the Check Report message window.
- Click the Code Generation tab in the Workspace window.
- Check the Max Cycles checkbox and enter 500 in the field next to it.
- Make sure that the Compile and Start check boxes are selected, as shown in Figure 6-17.

Figure 6-17   Preparing for Code Generation


System Studio Workspace

☐ Top-level model: <active model>

Name: and_model_1 ☐ Unique

Description:

Execution location: ${CCSS_SIM_DIR}

C++ | SystemC-RTL

☑ Maximum cycles: 500

☐ Range checks

Macro Debug <unoptimized>

☐ Incremental code generation

☑ Compile ☑ Start:

☑ Control panel

☐ SCF: and_model.scf

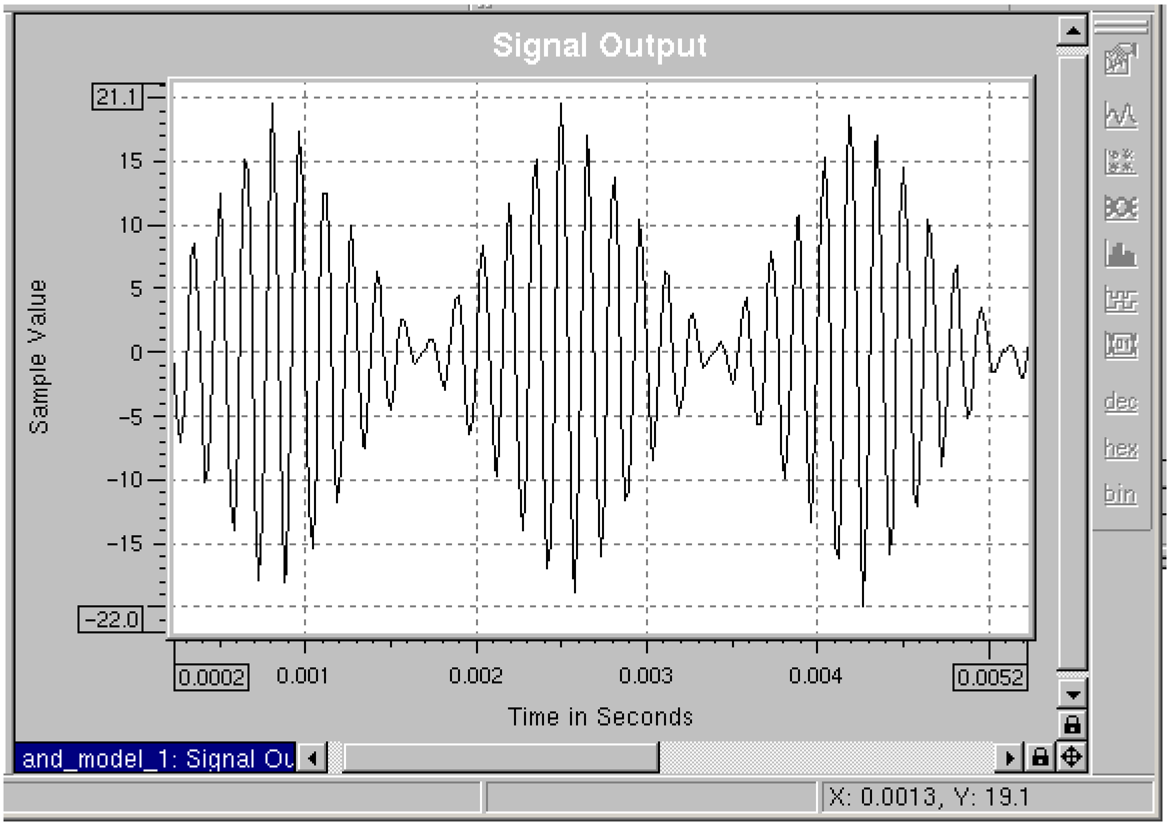
Simulate C++ | Create

Models | Hierarchy | Code Ge... | Simulation

- Click the Create button. The model source code will be generated and compiled, and the simulation will be executed and run to completion. The code generation messages will be displayed in the message window, followed by the simulation messages. An Infobox message will appear when the simulation completes successfully.

After the simulation execution has completed, you can open the System Studio simulation and check the output data file using DAVIS. The output waveform of the modulated sine wave should look like the waveform shown in Figure 6-18. For information about using DAVIS, refer to the CoCentric DAVIS User Guide.

Figure 6-18 The Output Waveform



7

Architectural Modeling

Architectural (SystemC) models capture the architecture of a system at various levels of granularity and abstraction. These models allow you to describe the overall platform architecture in terms of its buses, memories, processors, and ASIC content, as well as making it possible for you to describe the internal architecture of the individual components.

This chapter takes you step-by-step through the stages of creating a very simple design that consists of two SystemC primitive models and a hierarchical model. The information in this chapter is presented in the following sections:

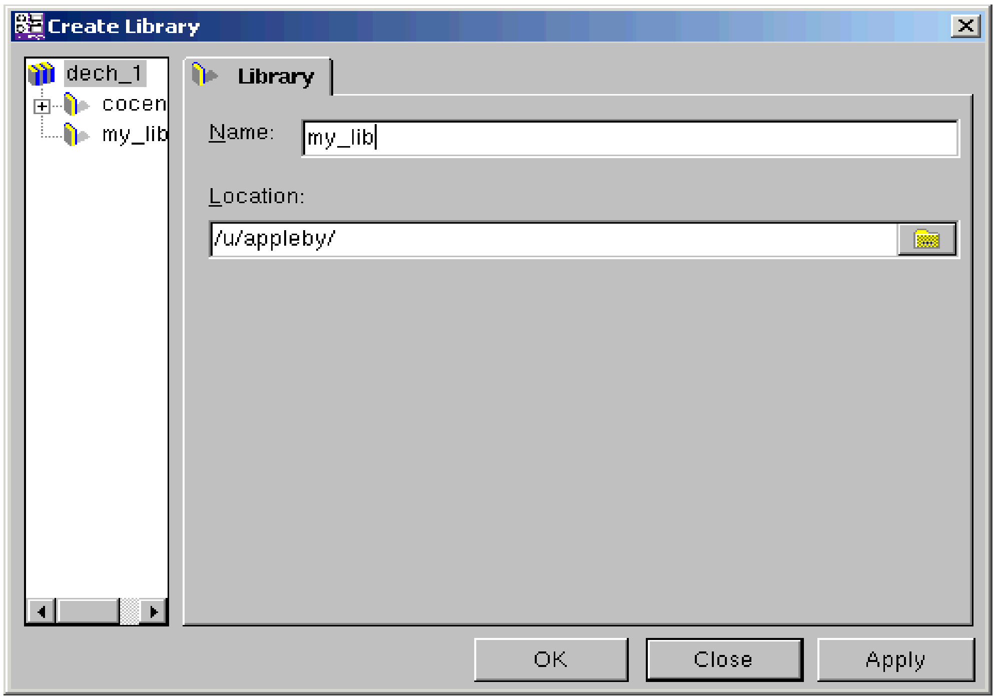
- Port Cloning and MultiPort Adapters
- Using CoCentric System Studio VirSim
- Simple Bus Example

Creating an Architectural Primitive Model

This example illustrates the essentials of creating a simple counter model.

1. The first step is to create the library that will contain the model (you can, of course, just choose an existing library, in which case you can skip this step). Choose File New from the menu and in the Create Library dialog box, as shown in Figure 7-1, enter the name of the library.

Figure 7-1   Creating the Library



Name: my_lib

Location: /u/appleby/

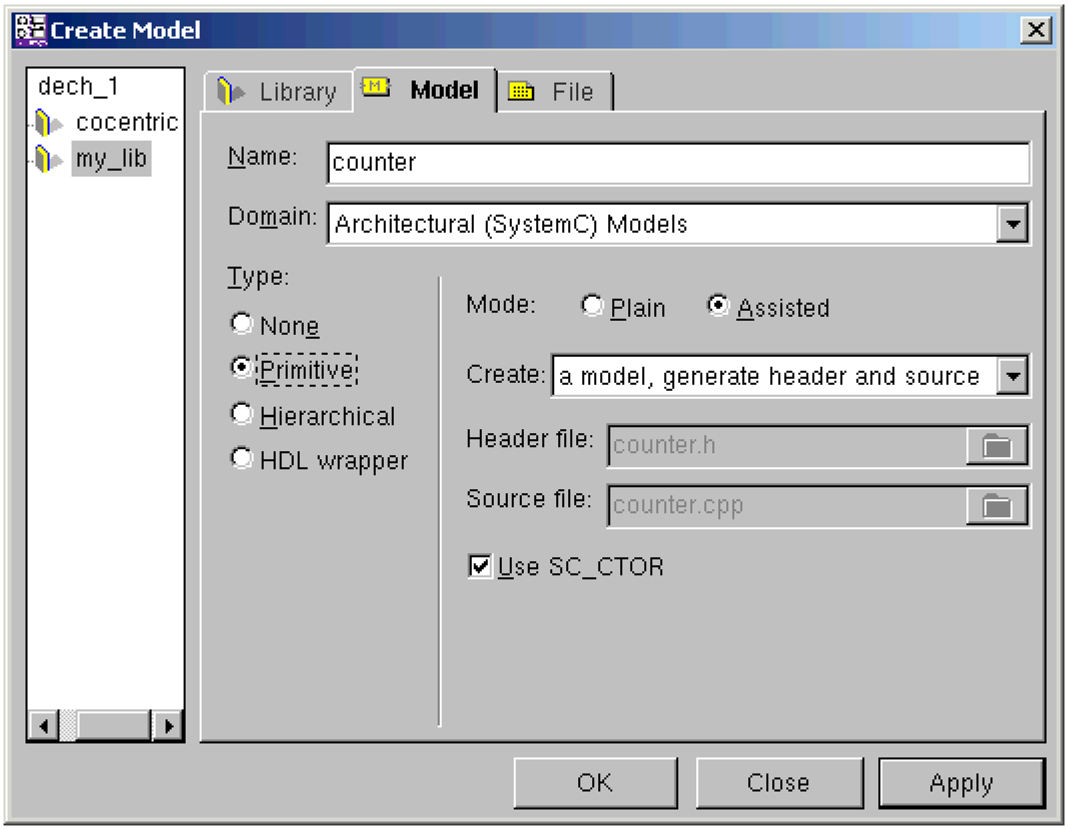
2. Click Apply.

3. Click the new library in the workspace window to highlight it and then choose File › New.

4. With the Create Library dialog box open, click the Model tab to open the Create Model page as shown in Figure 7-2.

5. Enter the name of the model (for example, counter), choose the Architectural (SystemC) Models domain, and choose Primitive as the model type. Leave the other option boxes at their default settings.

Figure 7-2 Creating the Model



Create Model

dech_1
cocentric
my_lib

Library | **Model** | File

Name: counter

Domain: Architectural (SystemC) Models

Type: ( ) None [✓ Primitive] ( ) Hierarchical ( ) HDL wrapper

Mode: ( ) Plain [✓ Assisted]

Create: a model, generate header and source

Header file: counter.h

Source file: counter.cpp

[✓] Use SC_CTOR

OK | Close | Apply

6. Click OK. The dialog box closes and the model Source View opens.

7. Click the Interface tab to open the Interface View. The port page is displayed.

8. Create two ports:

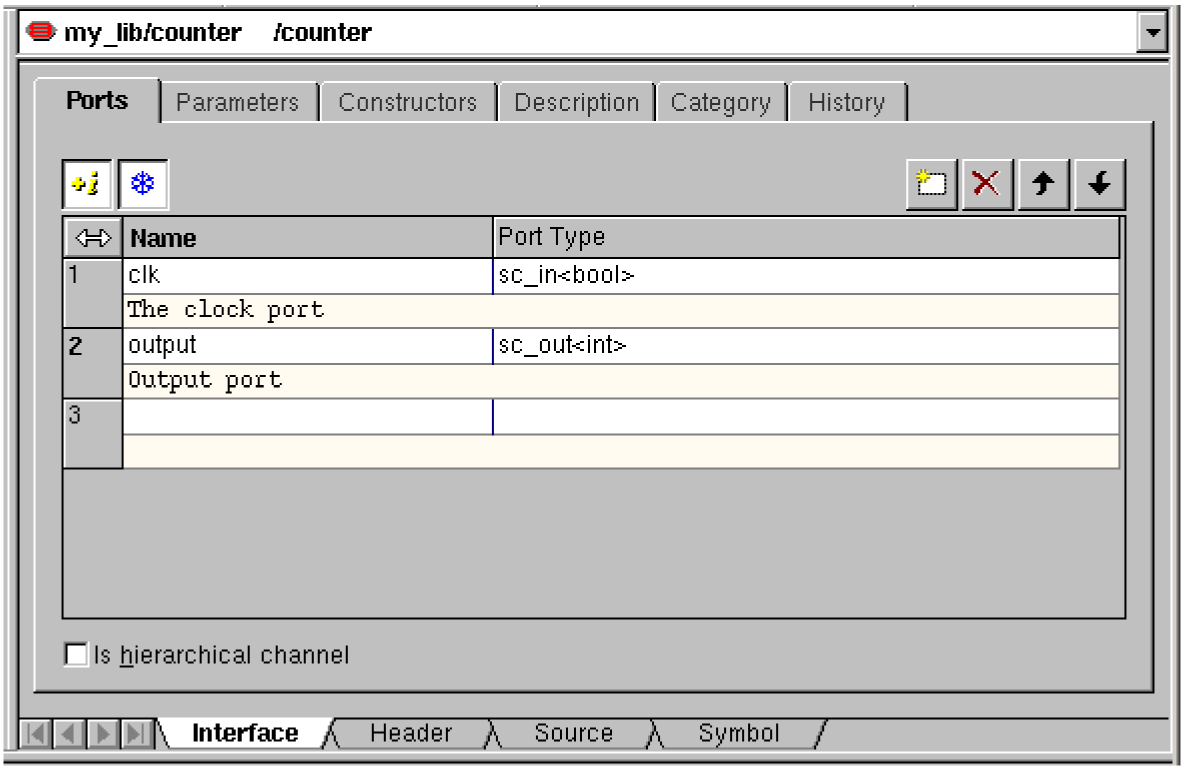
   - Name: clk - Port Type: sc_in<bool>

     This is the clock signal. As an alternative, you could select sc_in_clk as the Port Type for the clock signal.

   - Name: output - Port Type: sc_out<int>

     You can include your own comments for model documentation; see Figure 7-3.

Figure 7-3 Creating the Ports



	Name	Port Type
1	clk	sc_in<bool>
	The clock port	
2	output	sc_out<int>
	Output port	
3		

9. Now click the Parameter tab to open the Parameter page.

10. Create one parameter:

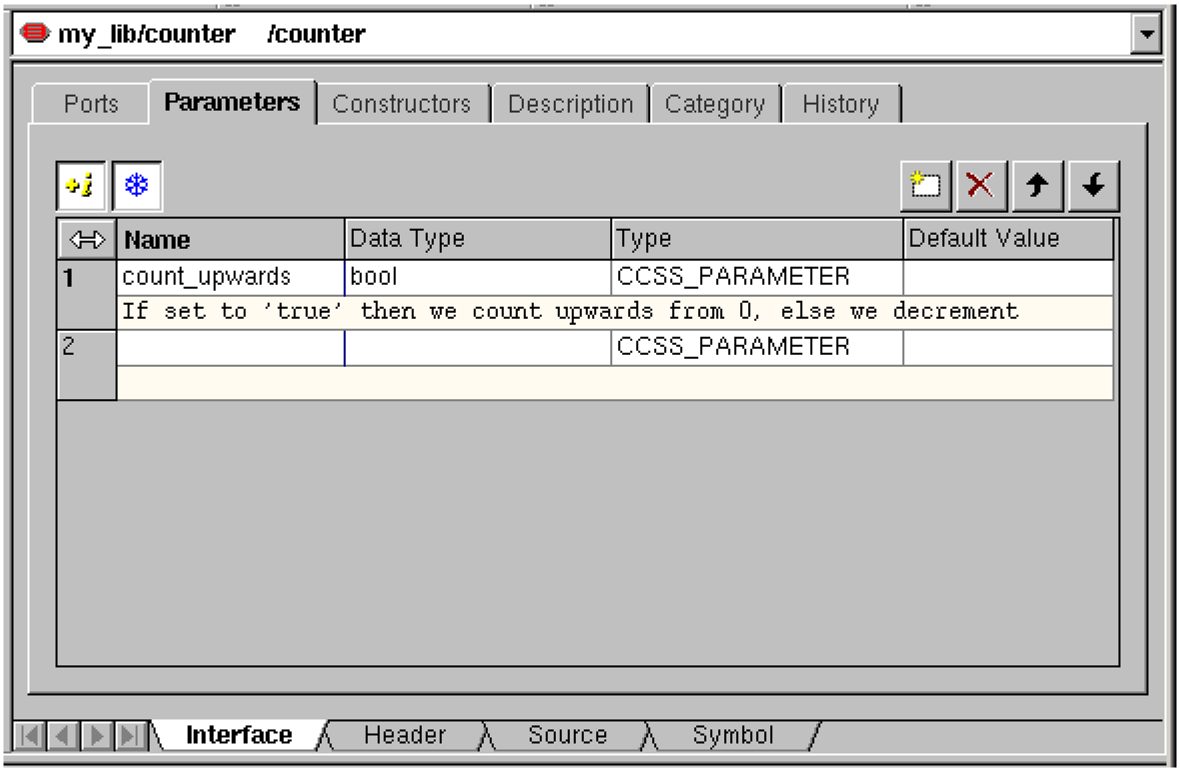
    - Name: count_upwards

      Data Type: bool

      Type: CCSS_PARAMETER.

      Add your own comments; see Figure 7-4.

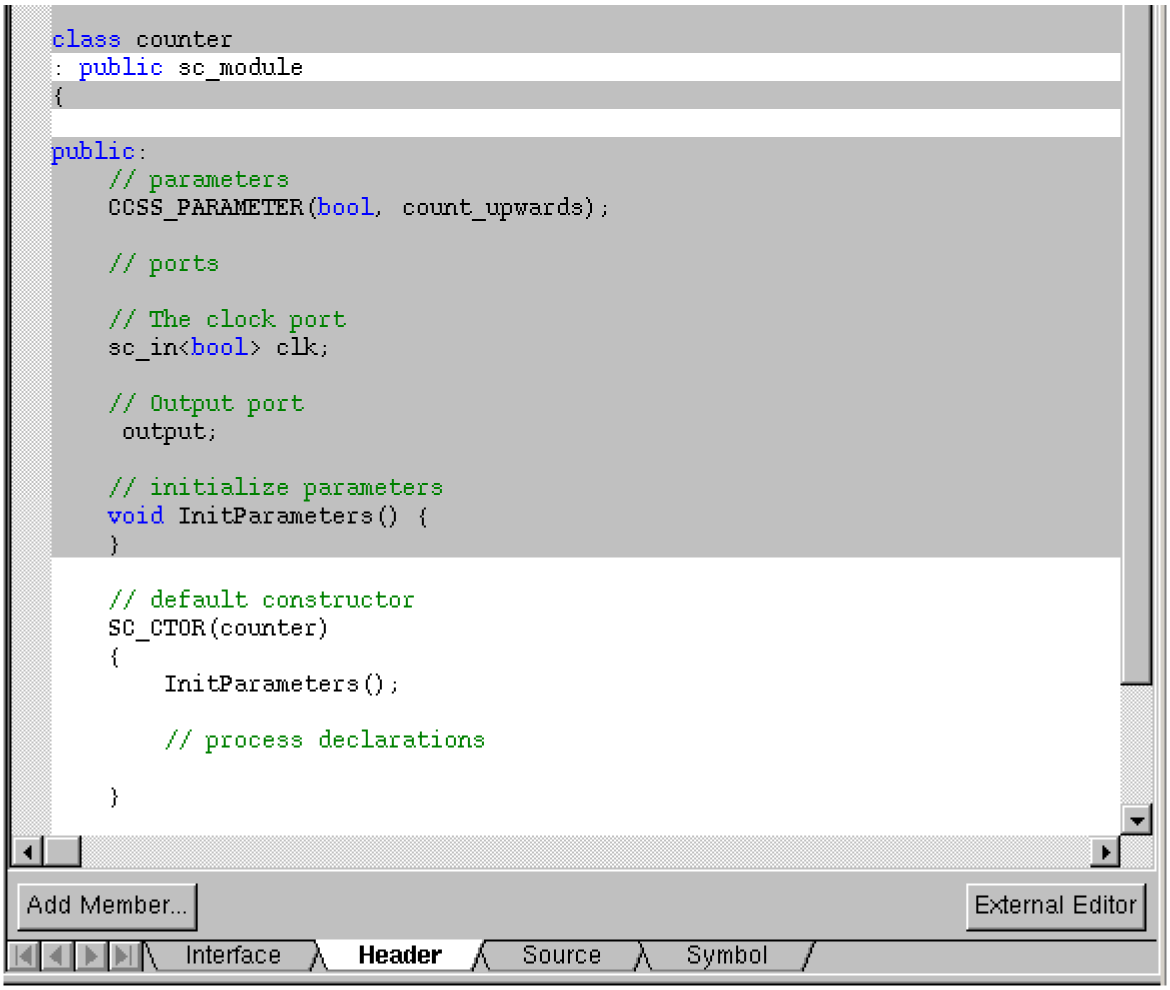
Figure 7-4 Creating the Parameter



	Name	Data Type	Type	Default Value
1	count_upwards	bool	CCSS_PARAMETER	
	If set to 'true' then we count upwards from 0, else we decrement			
2			CCSS_PARAMETER	

Click the Header tab to see the model code taking shape, as shown in Figure 7-5.

Figure 7-5 The Header View



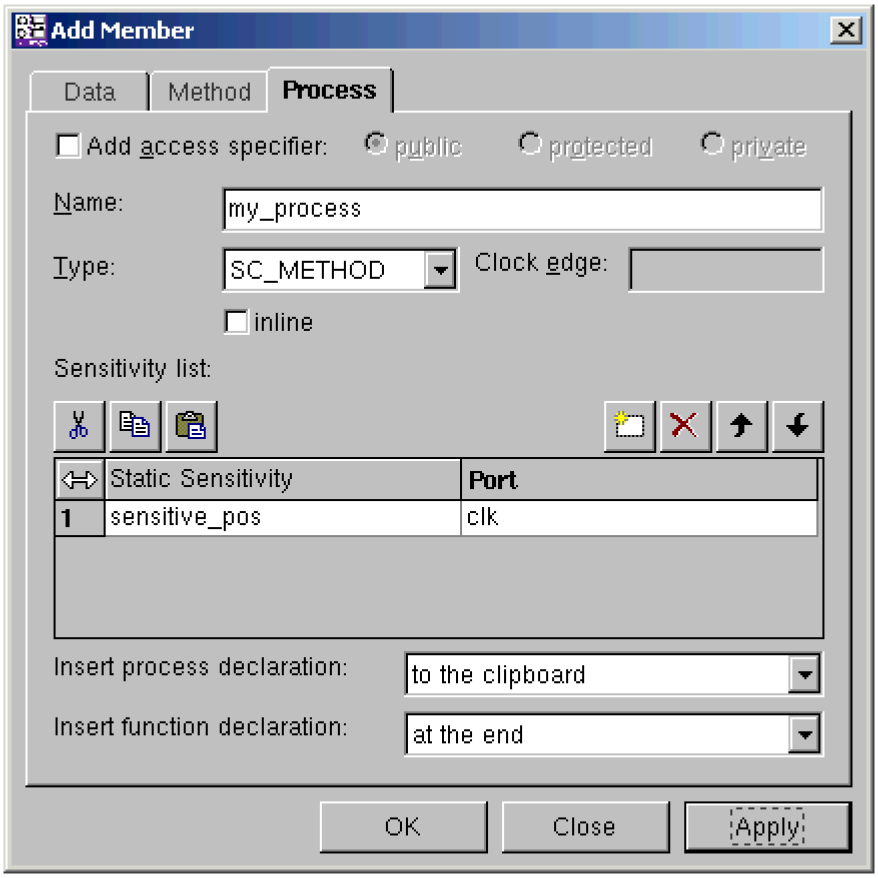
```
class counter
: public sc_module
{

public:
    // parameters
    CCSS_PARAMETER(bool, count_upwards);

    // ports

    // The clock port
    sc_in<bool> clk;

    // Output port
     output;

    // initialize parameters
    void InitParameters() {
    }

    // default constructor
    SC_CTOR(counter)
    {
        InitParameters();

        // process declarations

    }
```

11. Now you need to add a member process. In the Header View, click the Add Member button. The Add Member dialog box opens.

12. Click the Process tab to open the Add Member Process page.

13. Create a member process my_process and define it as being sensitive to the positive edge of the clk port (see Figure 7-6) and click OK.

Figure 7-6 Adding a Member Process


Add Member

Data | Method | **Process**

☐ Add access specifier: ● public ○ protected ○ private

Name: my_process

Type: SC_METHOD

Clock edge:

☐ inline

Sensitivity list:

	Static Sensitivity	Port
1	sensitive_pos	clk

Insert process declaration: to the clipboard

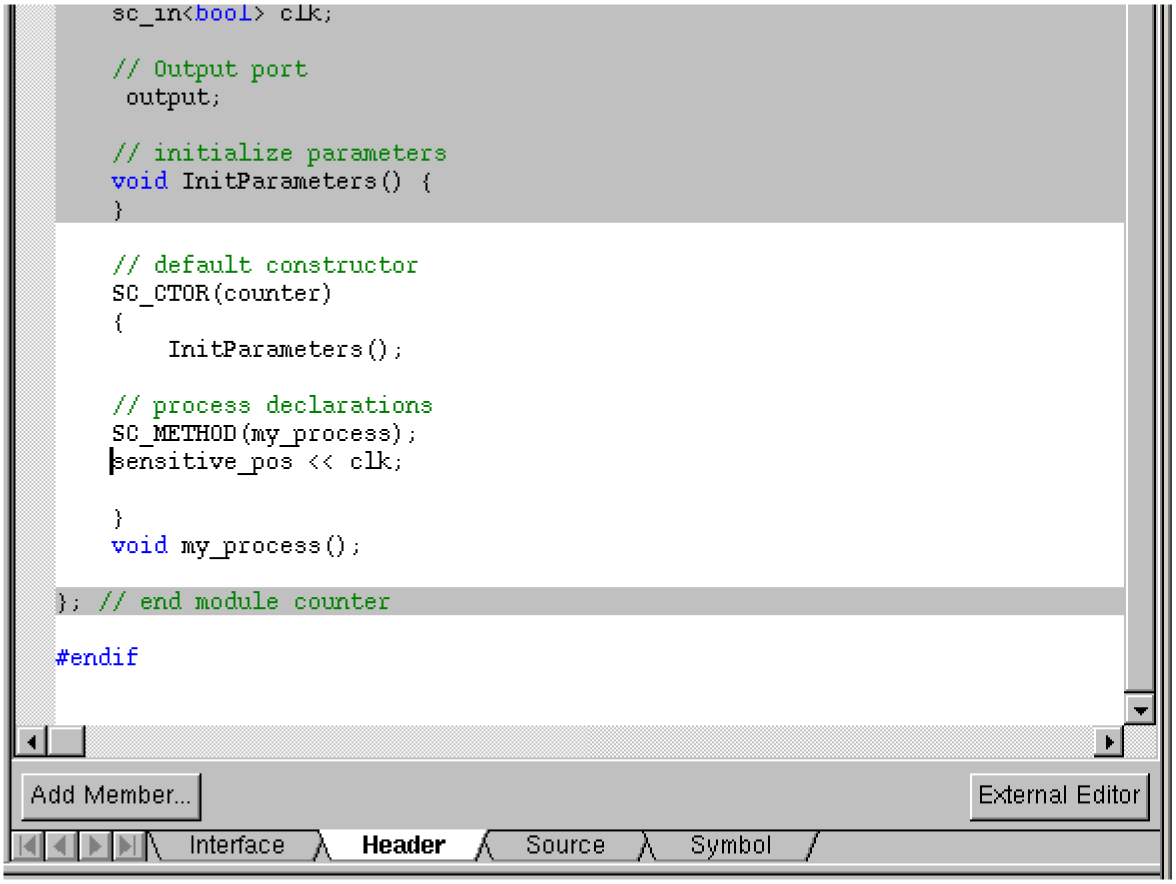
Insert function declaration: at the end

Note:

Using the “Insert Process” options, you can specify whether the initialization and declaration should be pasted into the code (and, if so, where), or copied to clipboard. If you copy to the clipboard, you can paste into the code by placing the cursor where you want to make the insertion, and pressing Control-v.

14. Open the Header View. Position the cursor in the line after “// process declarations” and paste the process declaration into the file (either use the Paste button or press Control-v). You will see that the model code now contains the process declaration as shown in Figure 7-7.

Figure 7-7 The Header View



```
    sc_in<bool> clk;

    // Output port
     output;

    // initialize parameters
    void InitParameters() {
    }

    // default constructor
    SC_CTOR(counter)
    {
        InitParameters();

    // process declarations
    SC_METHOD(my_process);
    sensitive_pos << clk;

    }
    void my_process();

}; // end module counter

#endif
```

15. From the Header View, click the Add Member button once again to open the Add Member dialog box. This time you will add member data.

16. Click the Data tab to open the Add Member Data page as shown in Figure 7-8.

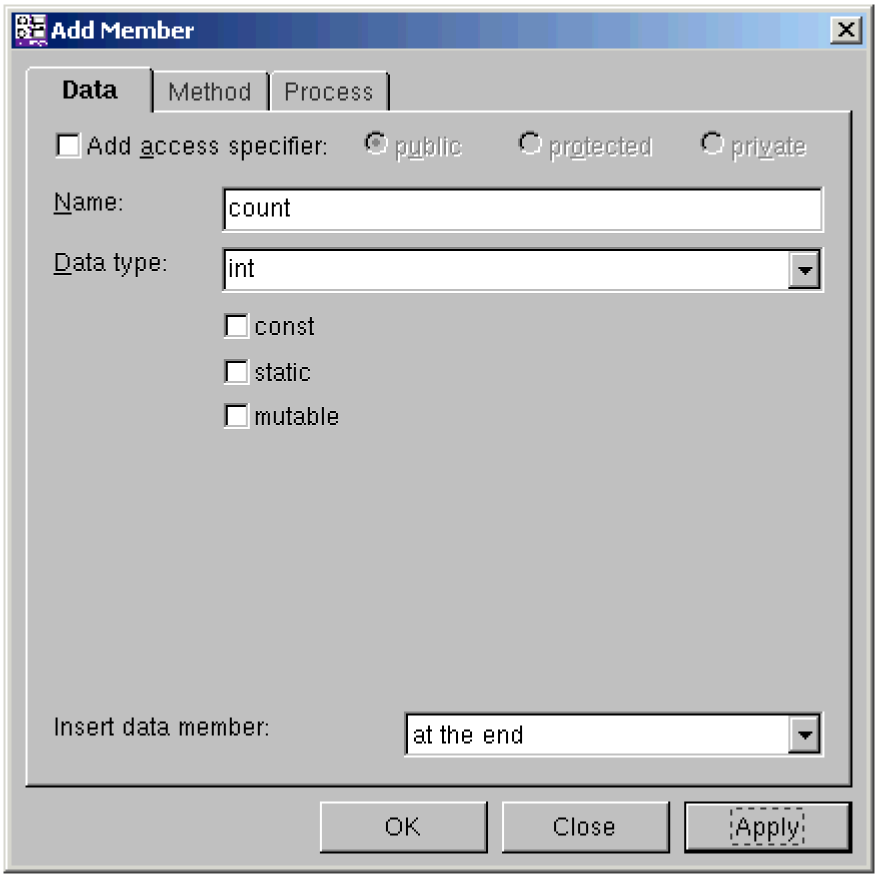
17. Create a member data

    - Name: count

      Data type: int

18. Click OK.

Figure 7-8 Creating the Member Data


Add Member

Data | Method | Process

☐ Add access specifier: ● public ○ protected ○ private

Name: count

Data type: int

☐ const

☐ static

☐ mutable

Insert data member: at the end

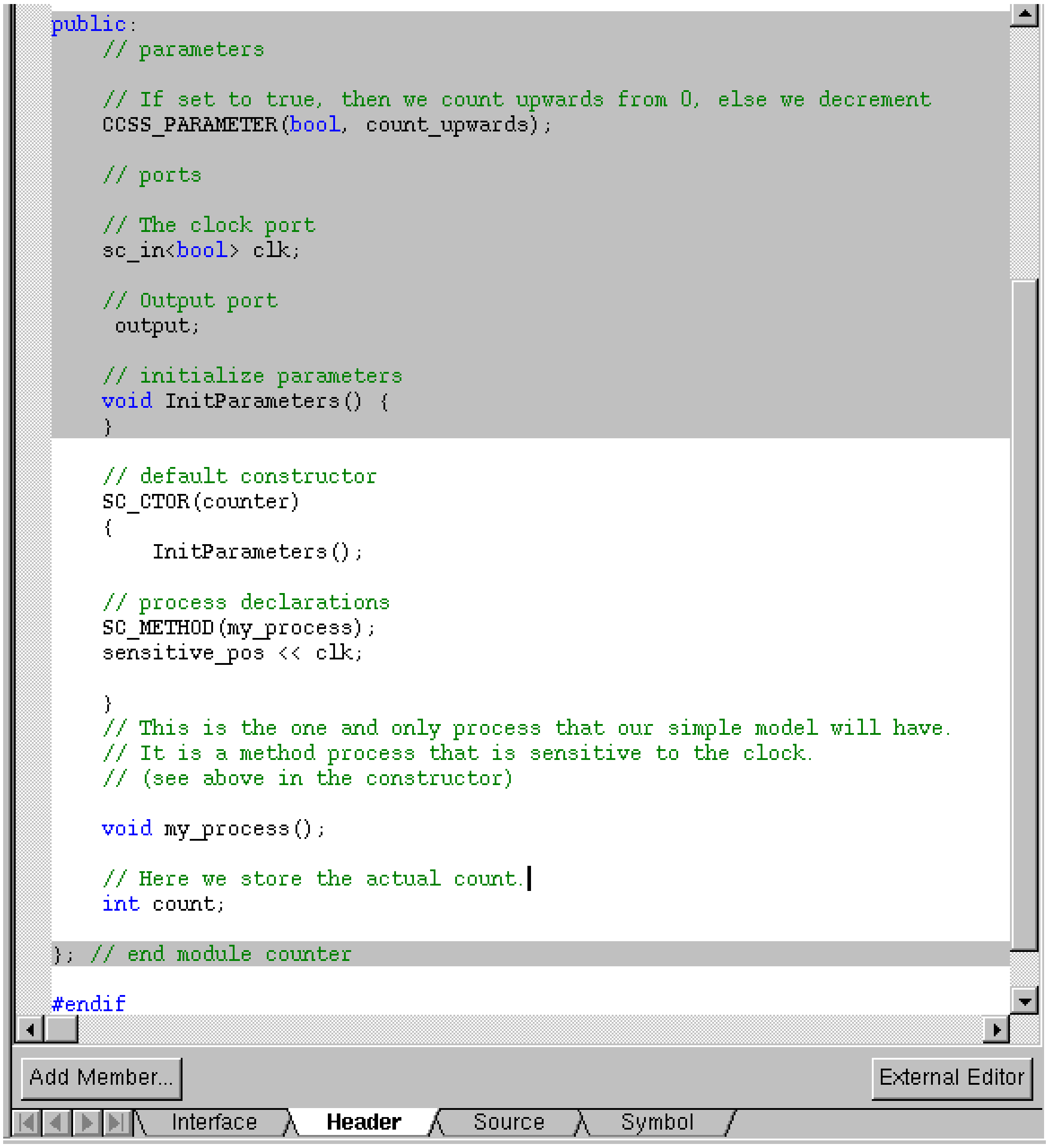
OK | Close | Apply

Note:

Leave the “Insert Data Member” option set to “at the end”; any other selection may result in unexpected behavior.

The declaration is now added to the header and is visible in the Header View. You will see that some explanatory comments have been added to the header in the version shown in Figure 7-9.

Figure 7-9   The Header View



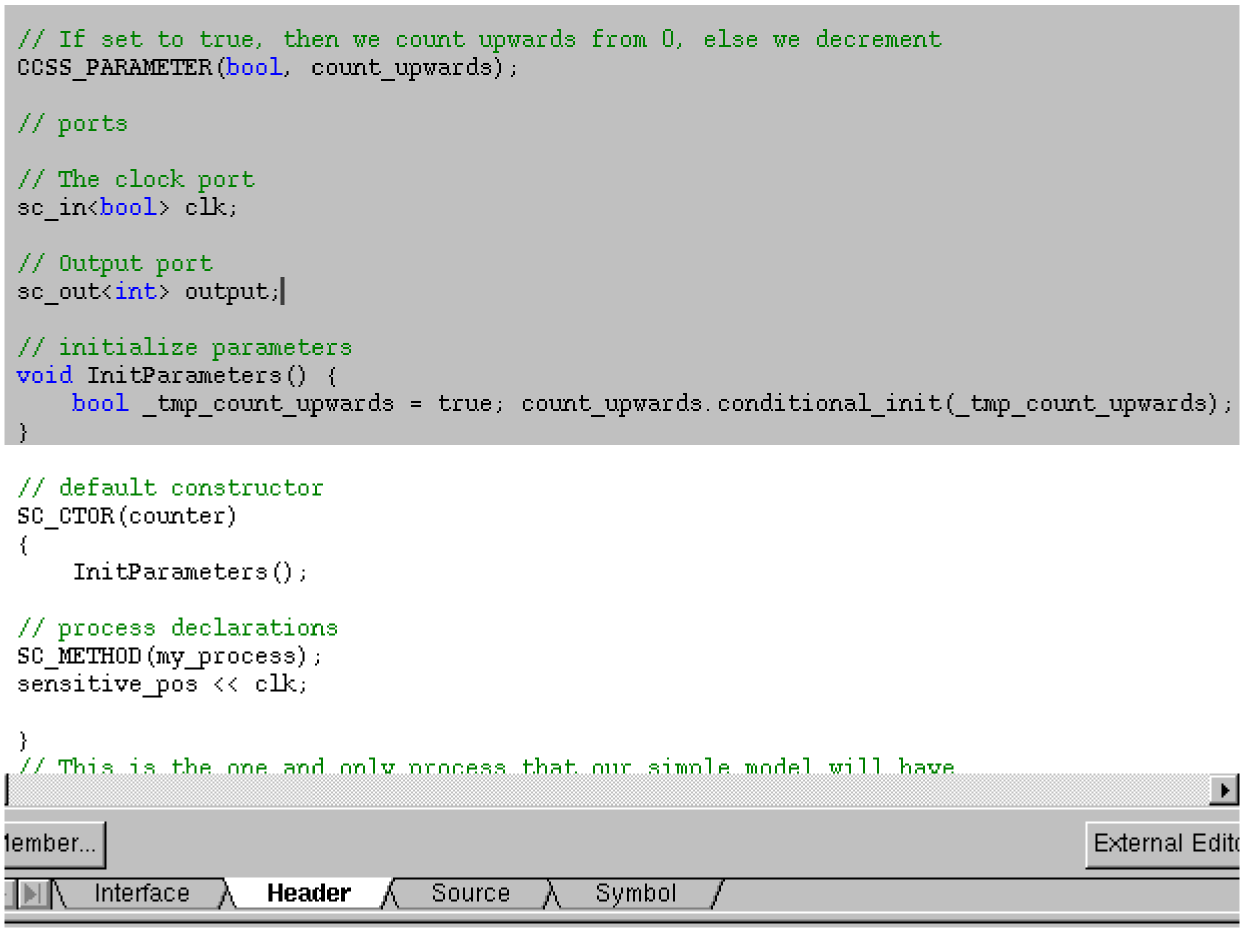
```
public:
    // parameters

    // If set to true, then we count upwards from 0, else we decrement
    CCSS_PARAMETER(bool, count_upwards);

    // ports

    // The clock port
    sc_in<bool> clk;

    // Output port
     output;

    // initialize parameters
    void InitParameters() {
    }

    // default constructor
    SC_CTOR(counter)
    {
        InitParameters();

    // process declarations
    SC_METHOD(my_process);
    sensitive_pos << clk;

    }
    // This is the one and only process that our simple model will have.
    // It is a method process that is sensitive to the clock.
    // (see above in the constructor)

    void my_process();

    // Here we store the actual count.
    int count;

}; // end module counter

#endif
```

You can now add the parameter initialization to the source code.

19. To add the parameter initialization, click Parameter in the interface view and set the Default value to true.

If you now look at the header view, you will see that the parameter count_upwards has been initialized, see Figure 7-10.

Figure 7-10 The Header View (Parameter Initialization)

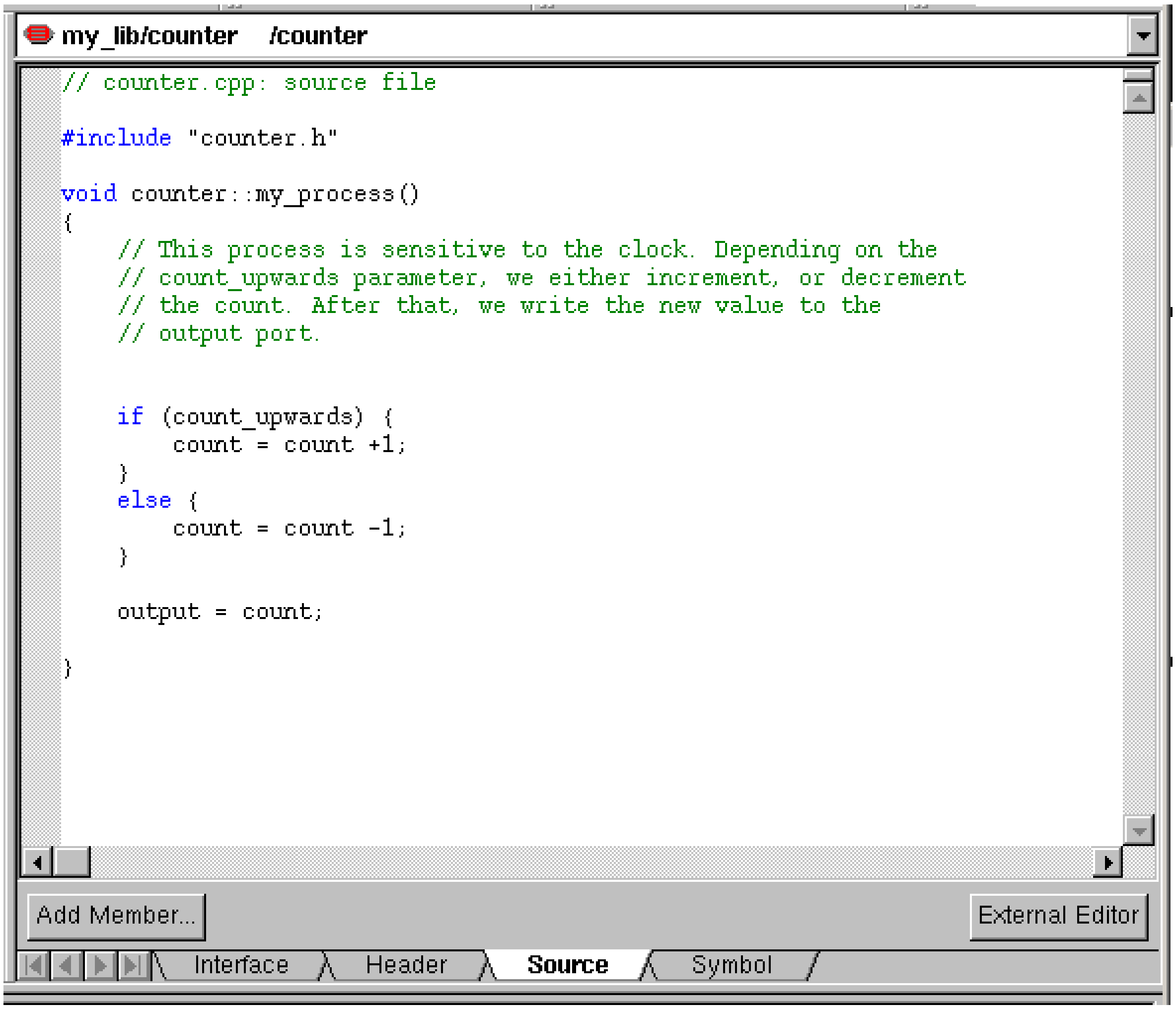


```
// If set to true, then we count upwards from 0, else we decrement
CCSS_PARAMETER(bool, count_upwards);

// ports

// The clock port
sc_in<bool> clk;

// Output port
sc_out<int> output;

// initialize parameters
void InitParameters() {
    bool _tmp_count_upwards = true; count_upwards.conditional_init(_tmp_count_upwards);
}

// default constructor
SC_CTOR(counter)
{
    InitParameters();

// process declarations
SC_METHOD(my_process);
sensitive_pos << clk;

}
// This is the one and only process that our simple model will have
```

20. You now need to add the process implementation code. Click the Source tab.

21. In the source view, type the code to implement the process as shown in Figure 7-11 and Example 7-1.

Figure 7-11 Implementing the Process



```
// counter.cpp: source file

#include "counter.h"

void counter::my_process()
{
    // This process is sensitive to the clock. Depending on the
    // count_upwards parameter, we either increment, or decrement
    // the count. After that, we write the new value to the
    // output port.


    if (count_upwards) {
        count = count +1;
    }
    else {
        count = count -1;
    }

    output = count;

}
```

22. Do not forget to remove the "#error" preprocessor directive used to remind you to implement any processes you declared earlier.

23. At this point in the design you can check your code. Click the Check Design button and look at the message in the Check Report view.

Note:

> If you make a mistake, such as a syntax error or a typing error when entering the code, it will be discovered when the design is checked and will be reported in the Check Report message window.

If you double-click the error message, the Source View opens with the cursor located as close to the source of the error as possible so that you can quickly correct the error and then recheck the design.

This completes the first model instance. Save the model. The completed source code for this model is shown in Example 7-1, and the header file is shown in Example 7-2.

Example 7-1 The Model Source Code

```
// counter.cpp: source file

#include "counter.h"

void counter::my_process()
{
     // This process is sensitive to the clock. Depending on the
     // count_upwards parameter we either increment or decrement
     // the count. After that, we write the new value to the
     // output port.

             if (count_upwards) {
                  count = count + 1;
        }
             else {
                       count = count - 1;
             }

             output = count;

}
```

Example 7-2 The Model Header File

```
// counter.h: header file

#ifndef __counter_h
#define __counter_h

#include <systemc.h>

#ifndef SYNTHESIS
#include <ccss_systemc.h>
#endif

class counter
: public sc_module
{

public:
    // parameters

    // If set to true, then we count upwards from 0, else we decrement
    CCSS_PARAMETER(bool, count_upwards);

    // ports

    // The clock port
    sc_in<bool> clk;

    // Output port
    sc_out<int> output;

    // initialize parameters
       void InitParameters() {
       bool _tmp_count_upwards = true;
       count_upwards.conditional_init(_tmp_count_upwards);
    }
       // default constructor
        SC_CTOR(counter)
        {
                InitParameters();

       // process declarations
          SC_METHOD(my_process);
          sensitive_pos << clk;

        }
       // This is the one and only process that our simple model will have
       // It is a method process that is sensitive to the clock.
       // (see above in the constructor)

          void my_process();
       // Here we store the actual count.
          int count;

};     // end module counter

#endif
```

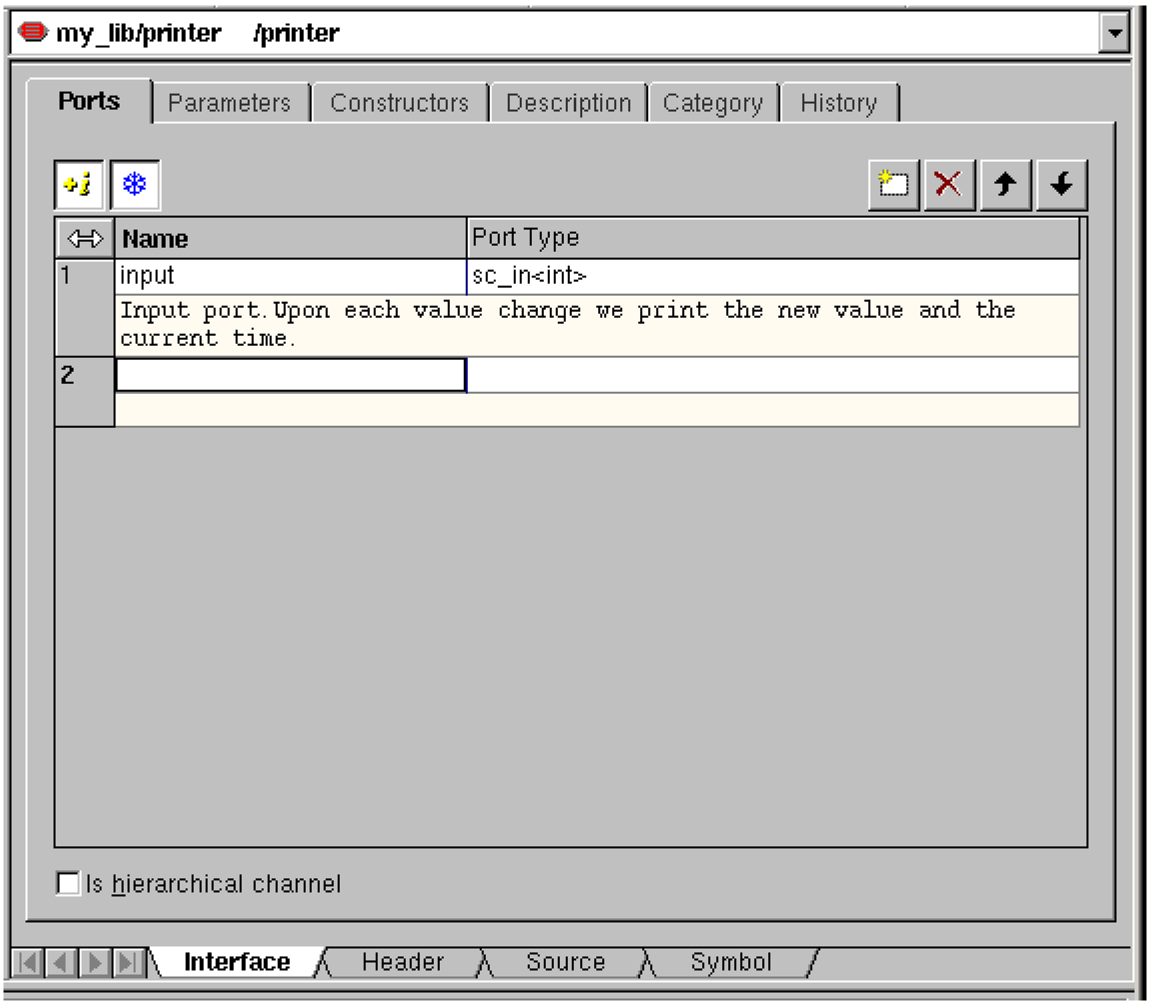
Creating the Printer Model

The next step is to create a second architectural (SystemC) primitive model in the same library as the first model. This is a simple printer that takes an integer value as its input. The steps to create this model are the same as for the first model, so some of the explicit detail will be left out.

- Create this model in the same way as the first model and in the same library. If necessary, refer to "Creating an Architectural Primitive Model" on page 7-2.
- For this second model, create one port:
  - Name: input

    Port Type: sc_in<int> (see Figure 7-12)

Figure 7-12   Creating the Port

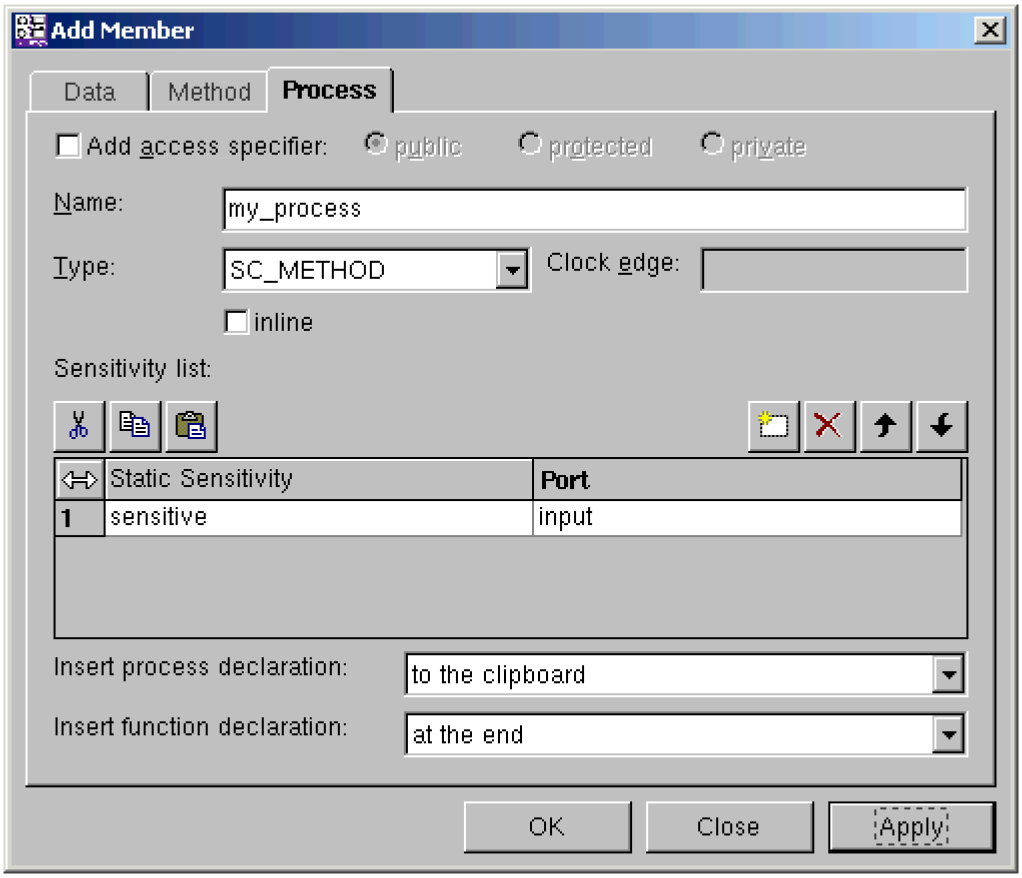


	Name	Port Type
1	input	sc_in<int>
	Input port. Upon each value change we print the new value and the current time.	
2		

☐ Is hierarchical channel

- Now create a member process for this second model and make it sensitive to the input signal (see Figure 7-13).

Figure 7-13 Creating the Member Process



Add Member

☐ Add access specifier: ● public ○ protected ○ private

Name: my_process

Type: SC_METHOD

Clock edge:

☐ inline

Sensitivity list:

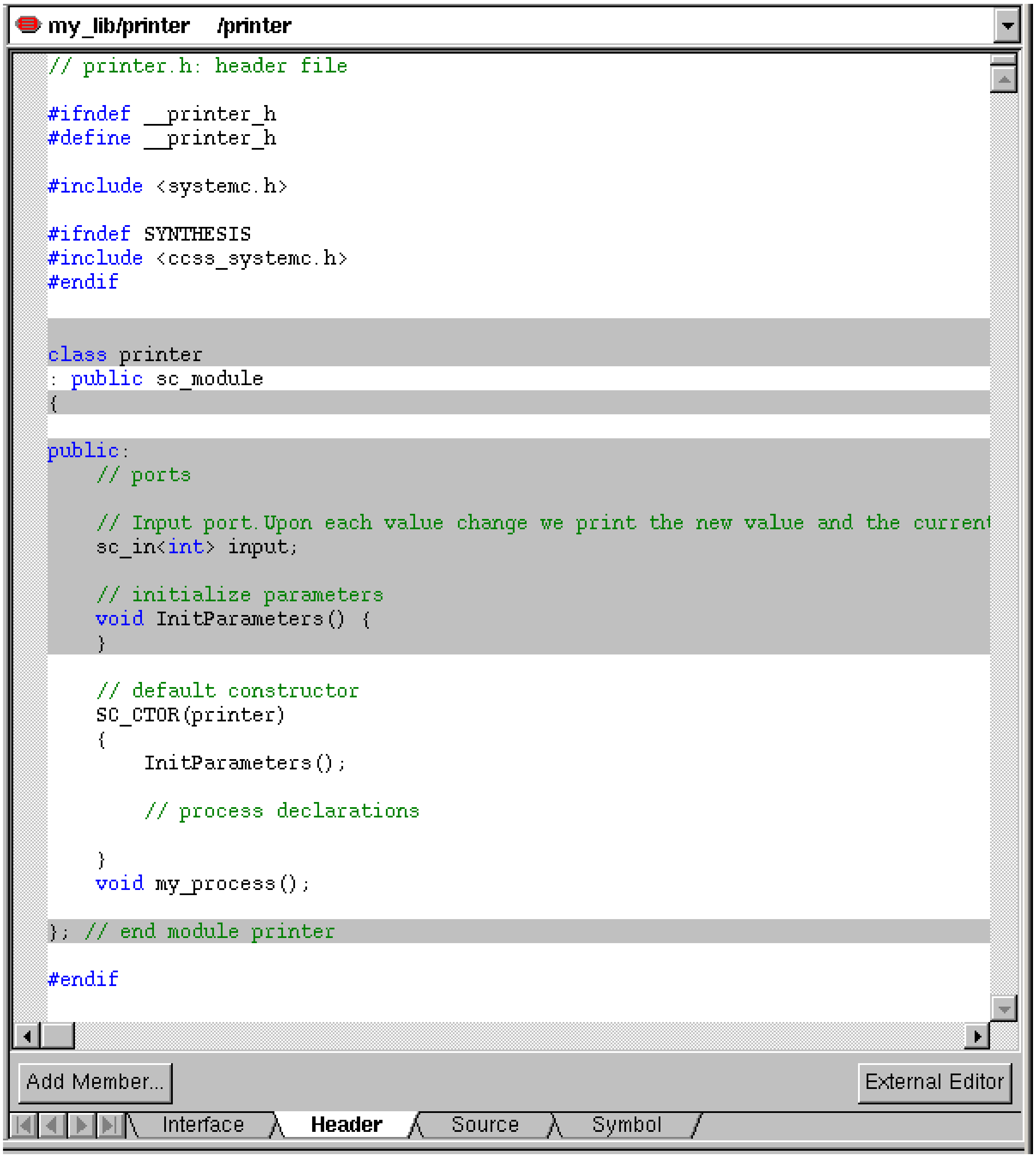
	Static Sensitivity	Port
1	sensitive	input

Insert process declaration: to the clipboard

Insert function declaration: at the end

In the Header View, you will now see the code for this second model taking shape (see Figure 7-14).

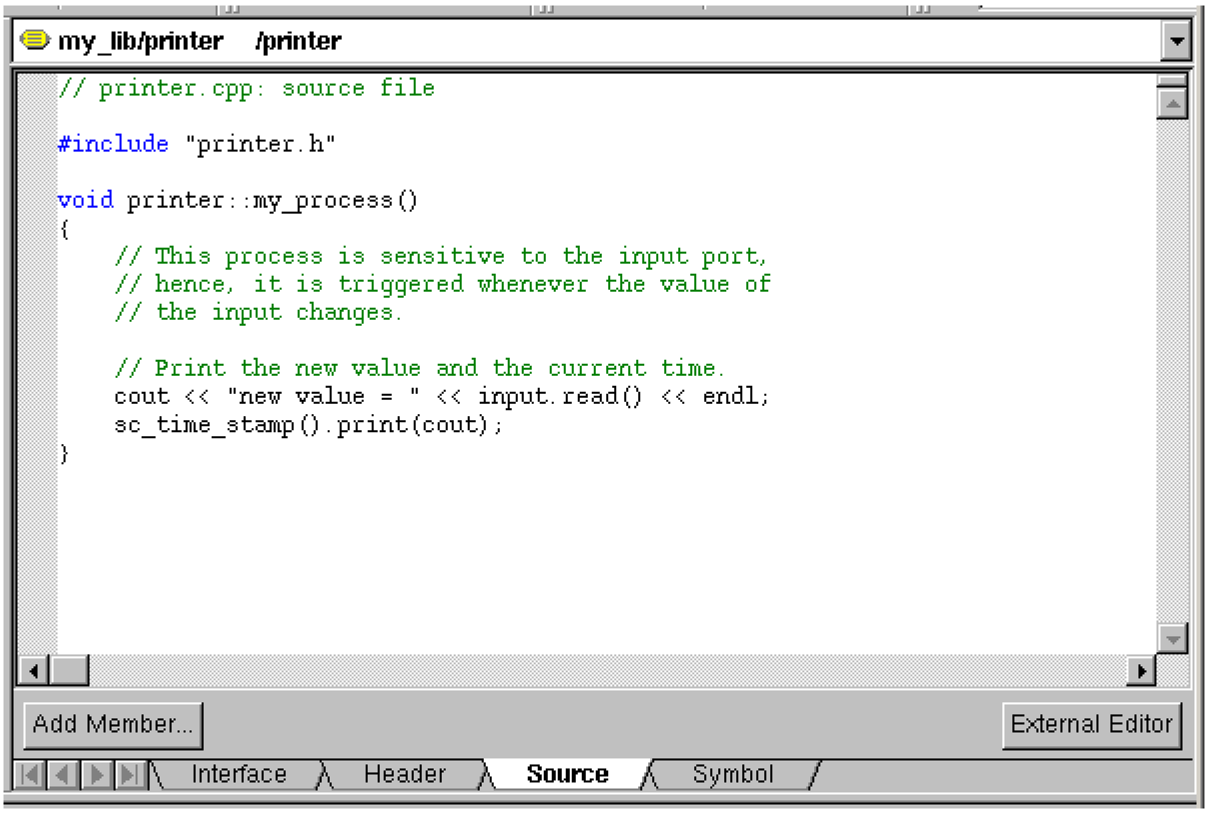
Figure 7-14   The Header File of the Second Instance



```
// printer.h: header file

#ifndef __printer_h
#define __printer_h

#include <systemc.h>

#ifndef SYNTHESIS
#include <ccss_systemc.h>
#endif


class printer
: public sc_module
{

public:
    // ports

    // Input port.Upon each value change we print the new value and the current
    sc_in<int> input;

    // initialize parameters
    void InitParameters() {
    }

    // default constructor
    SC_CTOR(printer)
    {
        InitParameters();

        // process declarations

    }
    void my_process();

}; // end module printer

#endif
```

- As with the first model, the next step is to add the process implementation, see Figure 7-15 and Example 7-3.

Figure 7-15   Creating the Process Implementation



```
// printer.cpp: source file

#include "printer.h"

void printer::my_process()
{
    // This process is sensitive to the input port,
    // hence, it is triggered whenever the value of
    // the input changes.

    // Print the new value and the current time.
    cout << "new value = " << input.read() << endl;
    sc_time_stamp().print(cout);
}
```

You have now completed the creation of the second primitive architectural (SystemC) model. The source code for the printer model is shown in Example 7-3, and the header file is shown in Example 7-4.

- Check the design to make sure that there are no errors and save the model.

Example 7-3   The Printer Model Source Code

```
// printer.cpp: source file

# include "printer.h"

void printer::my_process()
{
     // This process is sensitive to the input port.
     // Hence it is triggered whenever the value of the
     // input signal changes.

     // Print the new value and the current time.
     cout << "new value = " << input.read() << endl;
     sc_time_stamp().print(cout);
}
```

Example 7-4   The Printer Model Header File

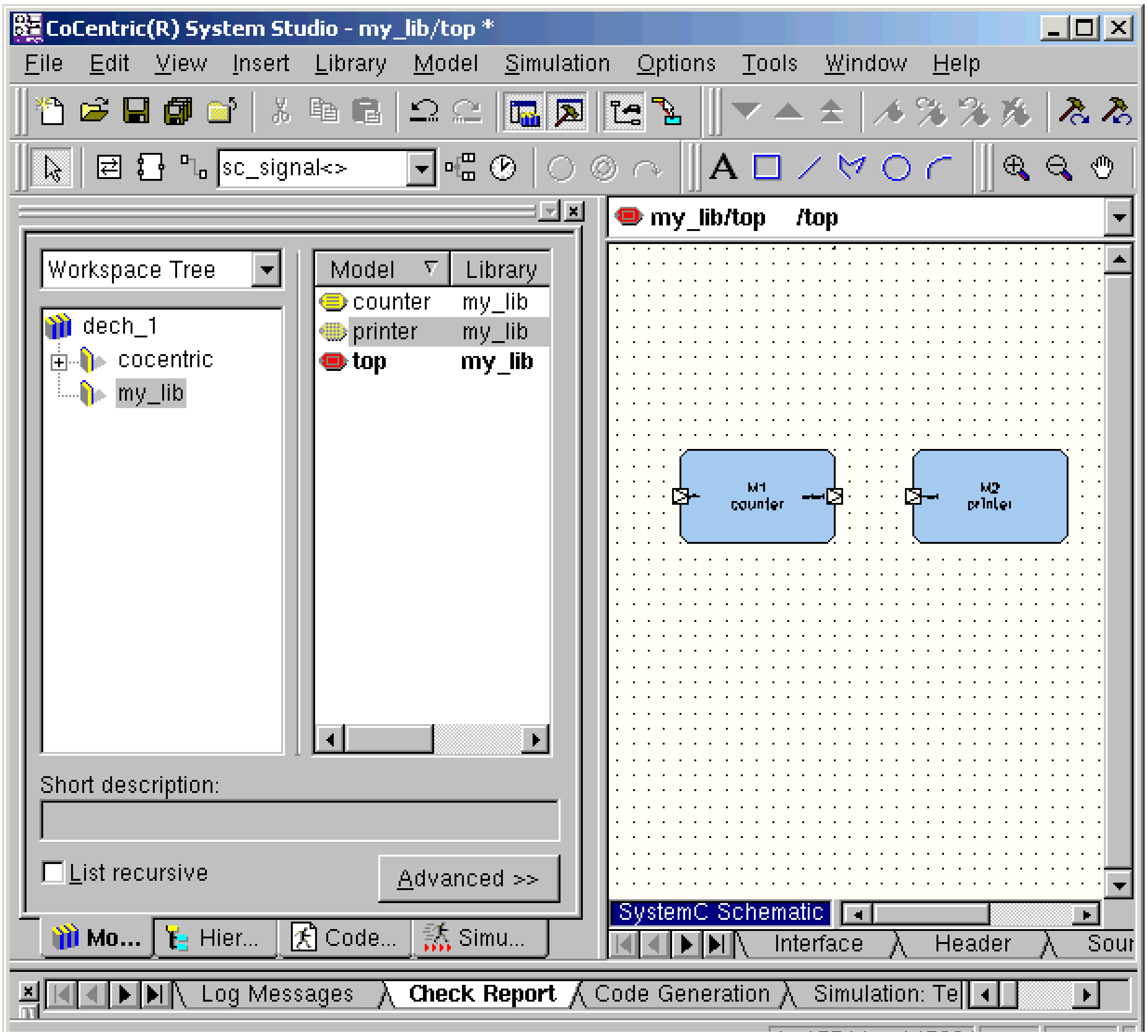
```
// printer.h: header file

#ifndef __printer_h
#define __printer_h

#include <systemc.h>

#ifndef SYNTHESIS
#include <ccss_systemc.h>
#endif


class printer
: public sc_module
{

public:
    // ports

    // Input port.Upon each value change we print the new value
    //and the current time.
    sc_in<int> input;

    // initialize parameters
    void InitParameters() {
    }

        // default constructor
        SC_CTOR(printer)
        {
                InitParameters();
                // process declarations
                SC_METHOD(my_process);
                sensitive << input;

        }
        void my_process();

}; // end module printer

#endif
```

Creating a Hierarchical Model

Now you will create a third architectural model. In this case, the model is a hierarchical model. You will then instantiate the other two models in this model.

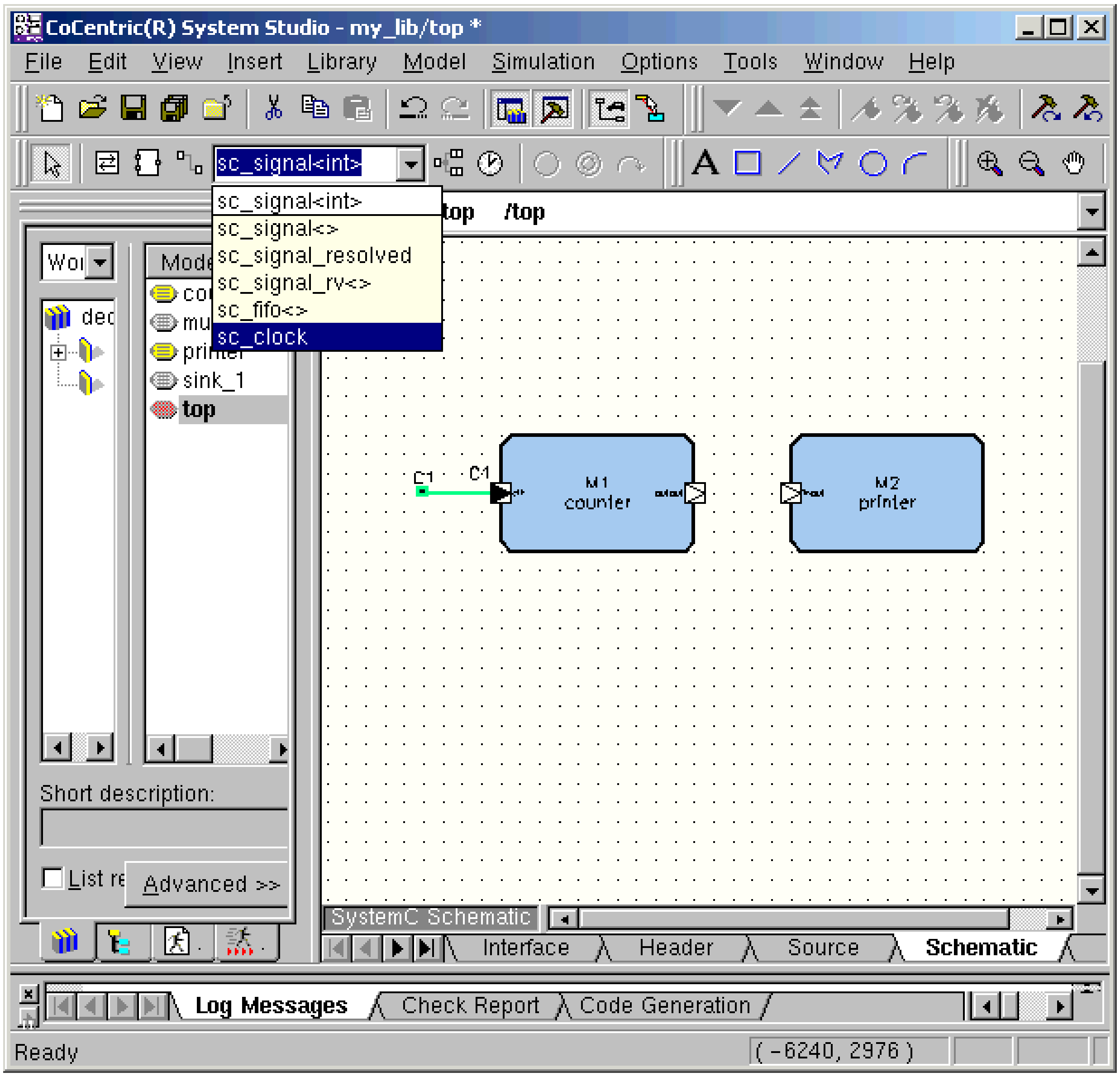
- Create a new hierarchical architectural model in the same library as your primitive models.
- Instantiate your two new primitive models in the new schematic, as shown in Figure 7-16. You can do this by simply dragging and dropping the model instances you want from the workspace window into the design area.

Figure 7-16 Instantiating the Models



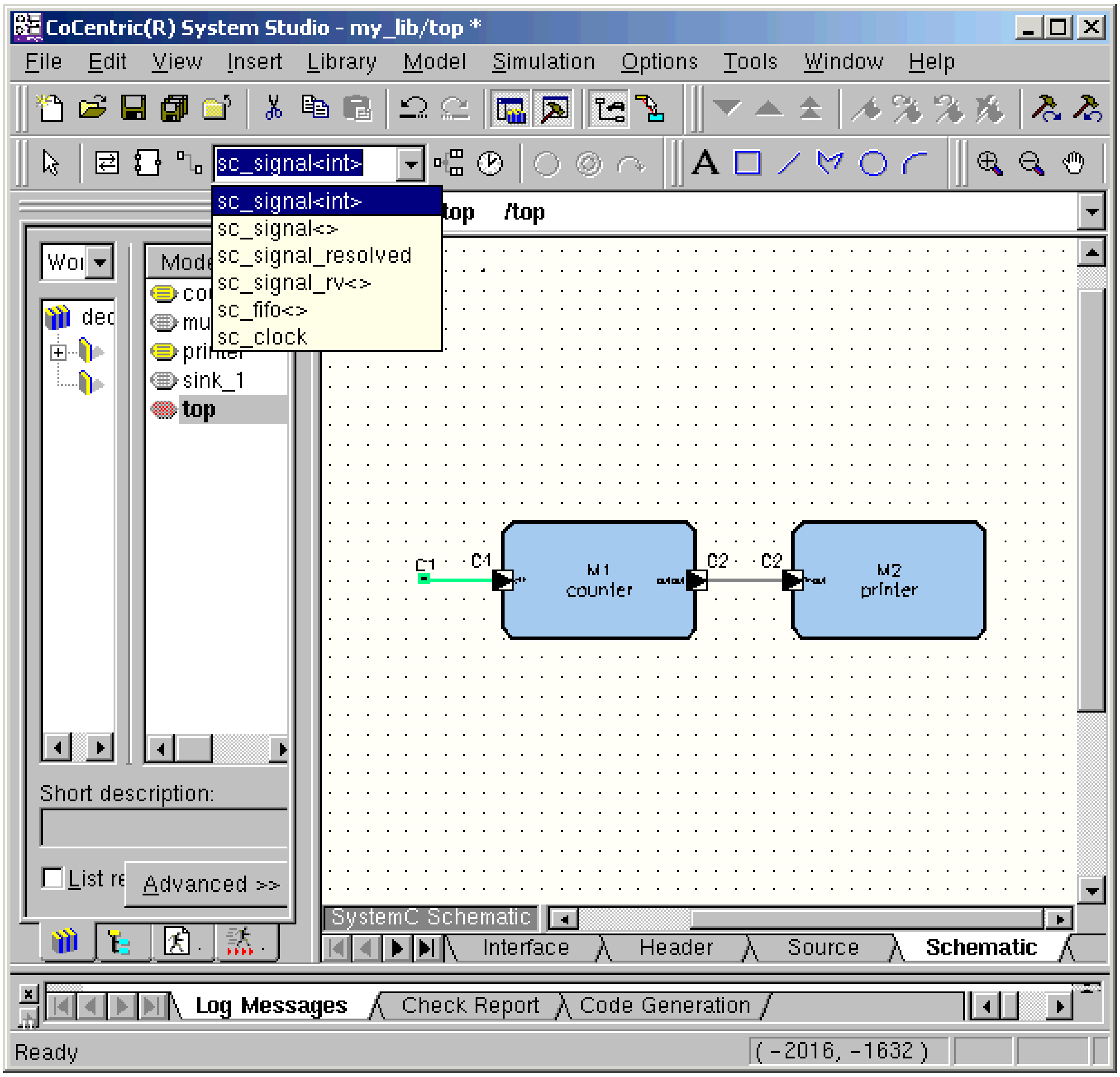
- Click the Channel button on the schematic toolbar. On the drop-down menu, choose sc_clock as the channel type.
- Draw a clock channel (sc_clock) from some point in the design area to the input (clk) of the counter block as shown in Figure 7-17.

Figure 7-17 Connecting a Clock Channel



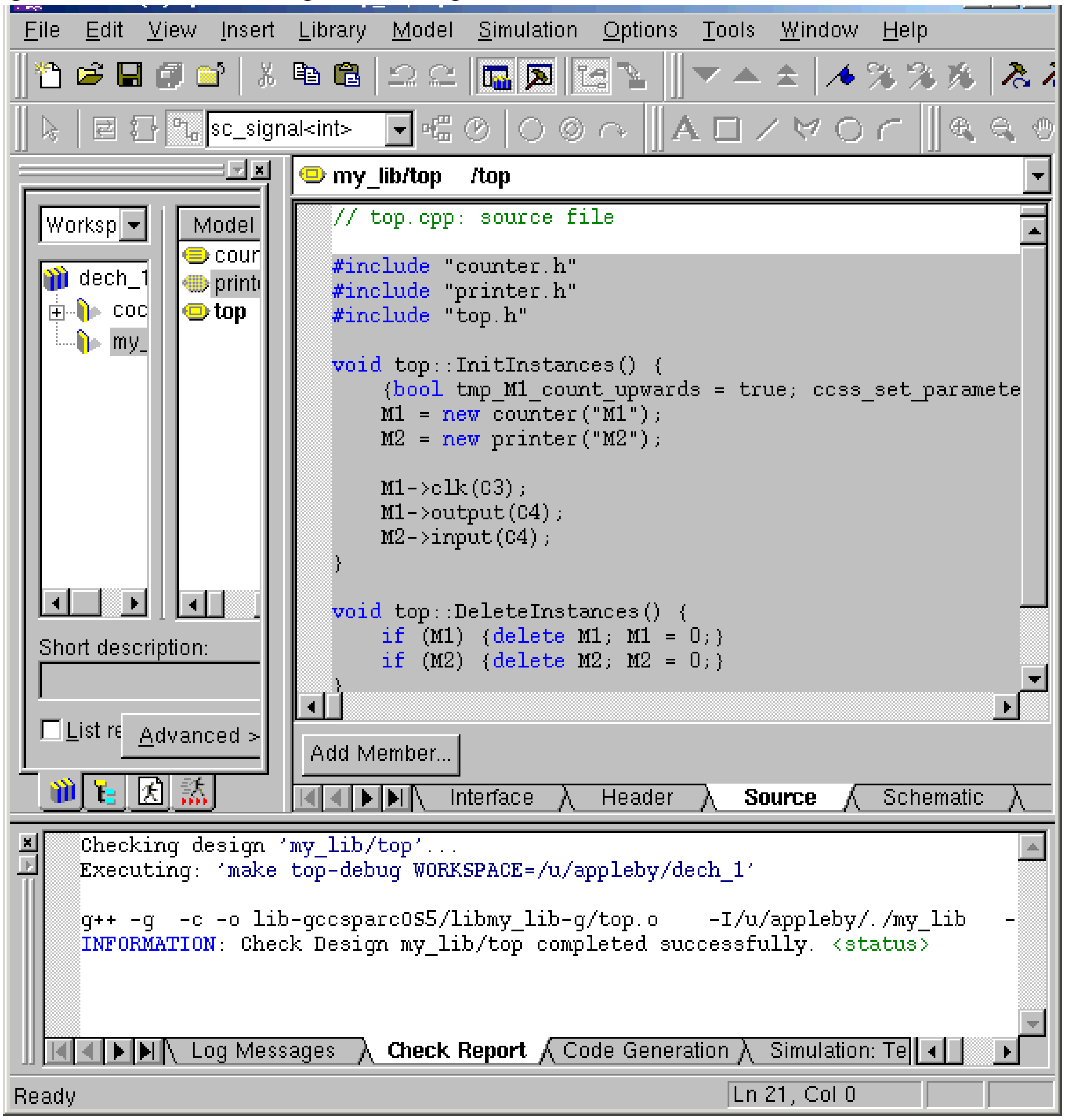
- Change the channel type to sc_signal<int>, and then draw a channel between the output of the counter block and the input of the printer block as shown in Figure 7-18.

Figure 7-18   Connecting the Instances



- To check your design, click the Check Design button in the toolbar.
- In the source view, click the Check Report tab. If there are no errors, as shown in Figure 7-19, you can save the hierarchical model.

Figure 7-19   Checking the Design



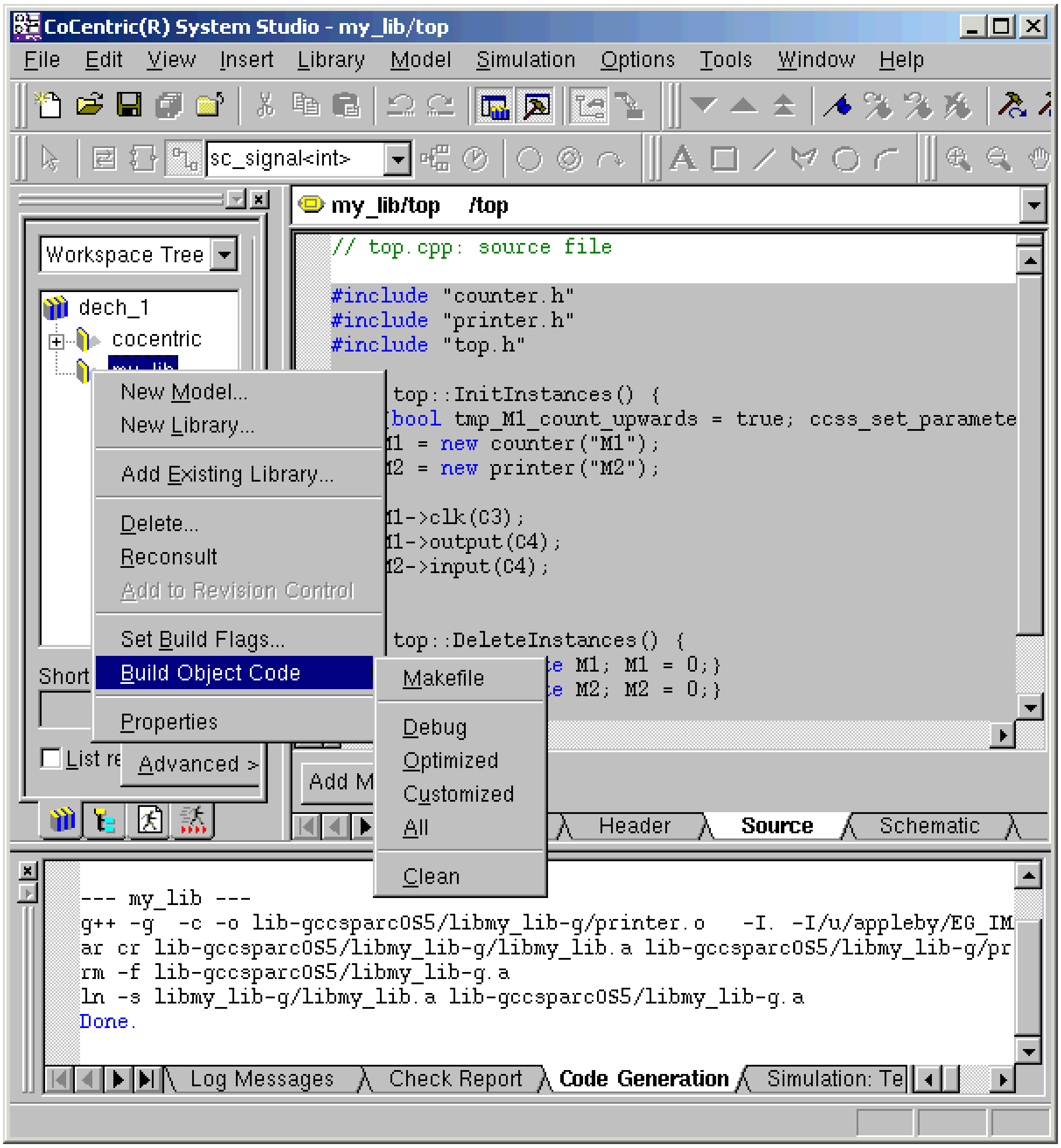
```
// top.cpp: source file

#include "counter.h"
#include "printer.h"
#include "top.h"

void top::InitInstances() {
    {bool tmp_M1_count_upwards = true; ccss_set_paramete
    M1 = new counter("M1");
    M2 = new printer("M2");

    M1->clk(C3);
    M1->output(C4);
    M2->input(C4);
}

void top::DeleteInstances() {
    if (M1) {delete M1; M1 = 0;}
    if (M2) {delete M2; M2 = 0;}
}
```

```
Checking design 'my_lib/top'...
Executing: 'make top-debug WORKSPACE=/u/appleby/dech_1'

g++ -g  -c -o lib-gccsparcOS5/libmy_lib-g/top.o    -I/u/appleby/./my_lib   -
INFORMATION: Check Design my_lib/top completed successfully. <status>
```

The design is now complete and ready for simulation.

- Select the library containing all three instances. Click the right mouse button, choose Build Object Code, and then choose Debug, as shown in Figure 7-20.

Figure 7-20 Creating the Library Object Code



```
// top.cpp: source file

#include "counter.h"
#include "printer.h"
#include "top.h"

top::InitInstances() {
bool tmp_M1_count_upwards = true; ccss_set_paramete
M1 = new counter("M1");
M2 = new printer("M2");

M1->clk(C3);
M1->output(C4);
M2->input(C4);

top::DeleteInstances() {
te M1; M1 = 0;}
te M2; M2 = 0;}
```

```
--- my_lib ---
g++ -g  -c -o lib-gccsparcOS5/libmy_lib-g/printer.o   -I. -I/u/appleby/EG_IM
ar cr lib-gccsparcOS5/libmy_lib-g/libmy_lib.a lib-gccsparcOS5/libmy_lib-g/pr
rm -f lib-gccsparcOS5/libmy_lib-g.a
ln -s libmy_lib-g/libmy_lib.a lib-gccsparcOS5/libmy_lib-g.a
Done.
```

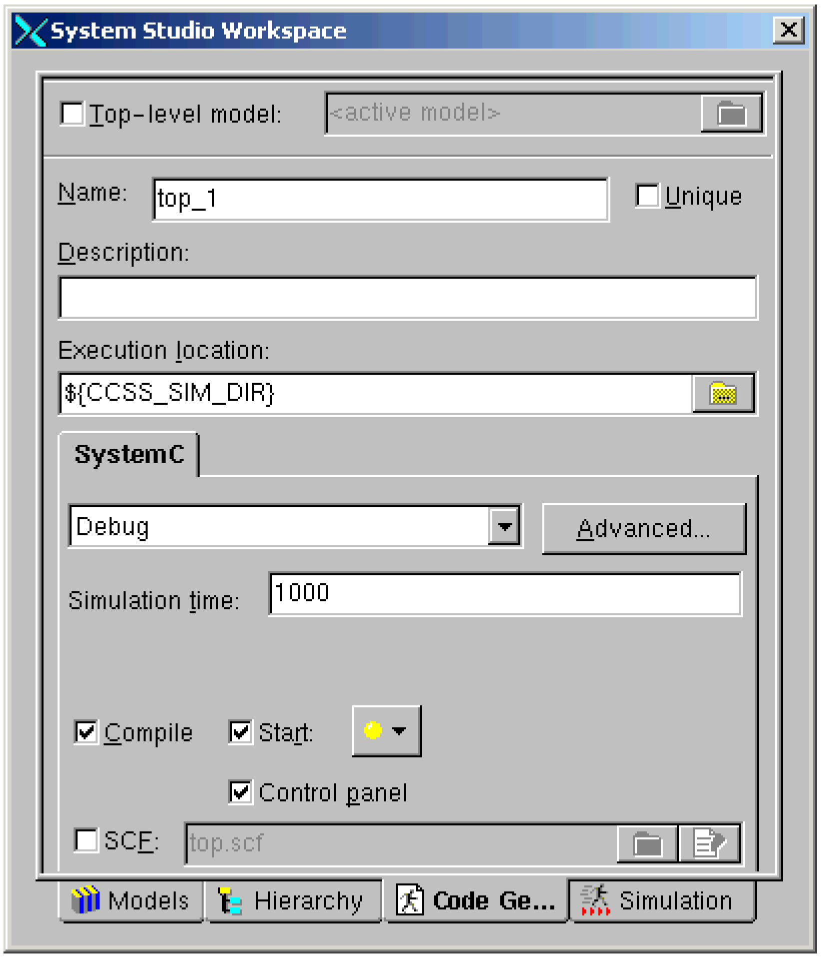
Simulating the Model

If there are no error messages displayed in the message window, the object code generation has succeeded. You can now generate the code and run the simulation.

- Click the Code Generation tab.
- In the code generation window (see Figure 7-21), select the Compile, Start, and Control panel check boxes.
- Click the Start button and select Paused to start the simulation in paused mode.
- Enter a simulation time, for example 1000, and then click Create.

In the messages window you should see a log window containing the code generation progress.

Figure 7-21 Creating the Simulation


System Studio Workspace

☐ Top-level model: <active model>

Name: top_1 ☐ Unique

Description:

Execution location: ${CCSS_SIM_DIR}

SystemC

Debug

Advanced...

Simulation time: 1000

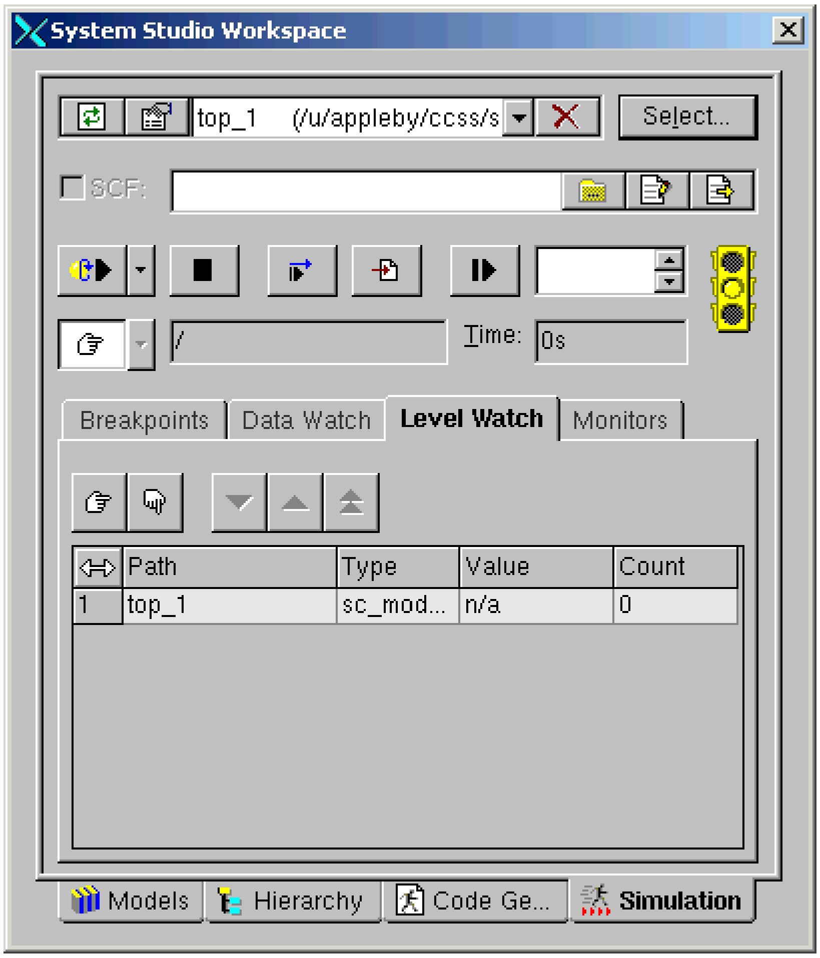
☑ Compile ☑ Start:

☑ Control panel

☐ SCF: top.scf

The simulation will now be compiled and started in paused mode. After a short time, the simulation control panel will open, as shown in Figure 7-22, allowing you to interact with the simulation.

Figure 7-22   Controlling the Simulation



	Path	Type	Value	Count
1	top_1	sc_mod...	n/a	0

Visualizing the Simulation With DAVIS

With the simulation running under the control of the simulation control panel, you can start DAVIS, connect to the simulation, and look at the output of the counter M1.

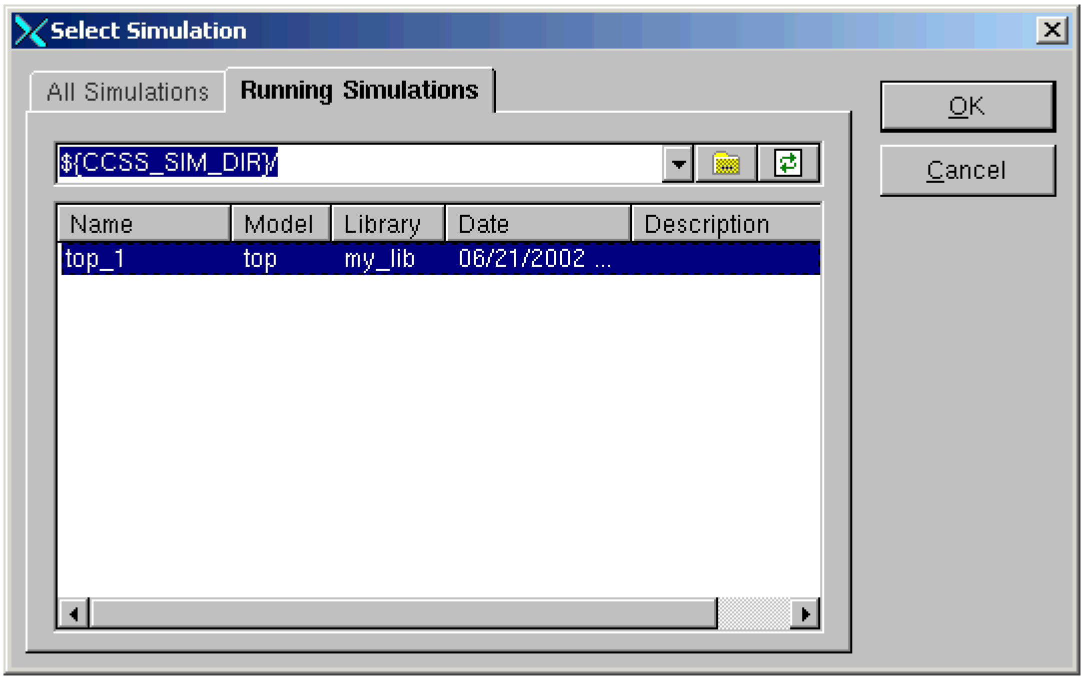
- First open a DAVIS tool and then from the DAVIS drop-down menu, choose File › Connect to Simulation.

Note:

For details of using DAVIS, see the *CoCentric DAVIS User Guide*.

- Connect to your running simulation, as shown in Figure 7-23, and click OK.

Figure 7-23 The Select Simulation Dialog Box

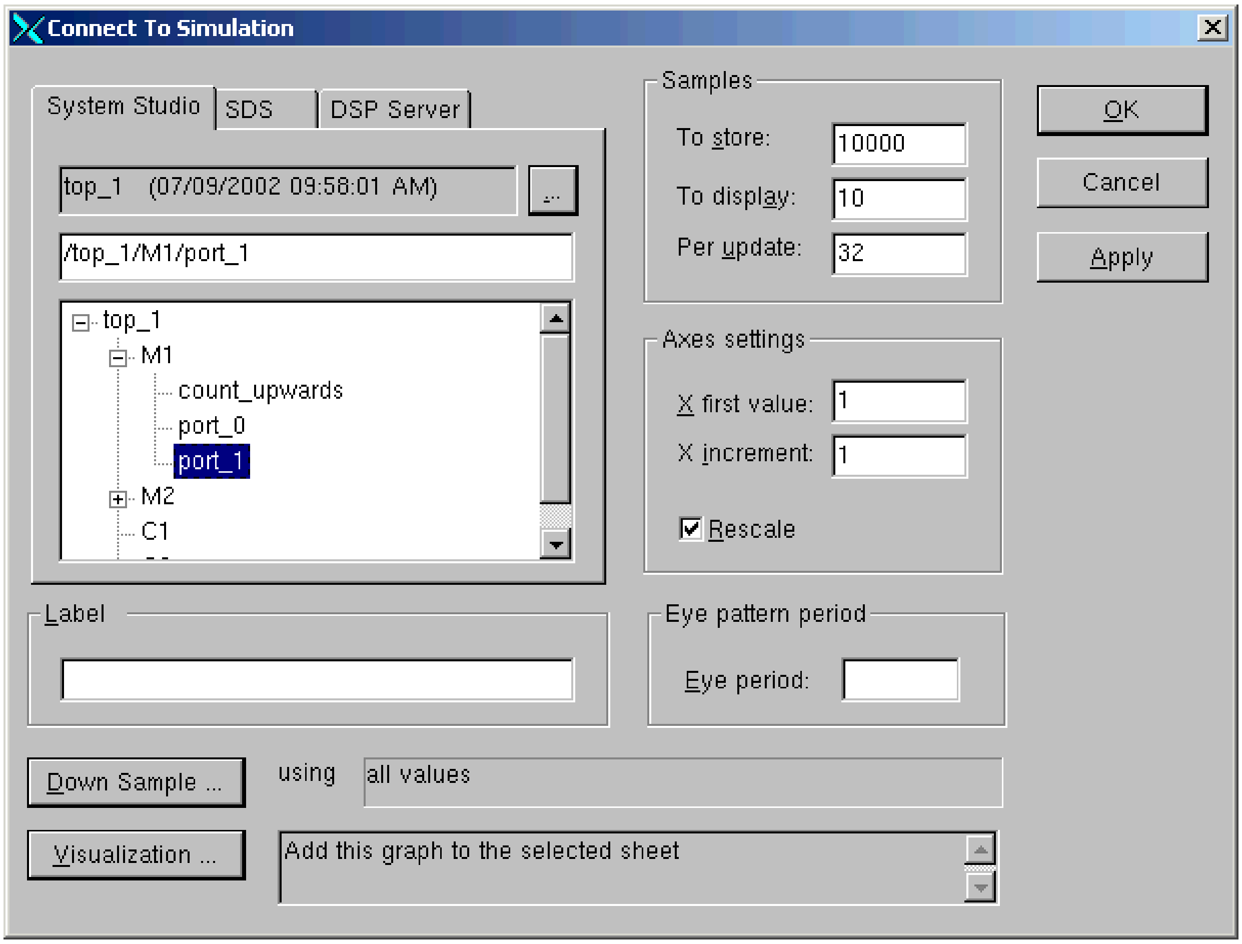


${CCSS_SIM_DIR}/

Name	Model	Library	Date	Description
top_1	top	my_lib	06/21/2002 ...	

- To connect to the output of the counter (port_1), navigate down through the model hierarchy and select M1/port_1.
- To make this simulation easier to view, type 10 in the Samples to display box, as shown in Figure 7-24, and click OK.

Figure 7-24   The Connect To Simulation Dialog Box


Connect To Simulation

System Studio | SDS | DSP Server

top_1   (07/09/2002 09:58:01 AM)

/top_1/M1/port_1

```
⊟ top_1
   ⊟ M1
      count_upwards
      port_0
      port_1
   ⊞ M2
     C1
```

Samples
- To store: 10000
- To display: 10
- Per update: 32

Axes settings
- X first value: 1
- X increment: 1
- ☑ Rescale

Label

Eye pattern period
- Eye period:

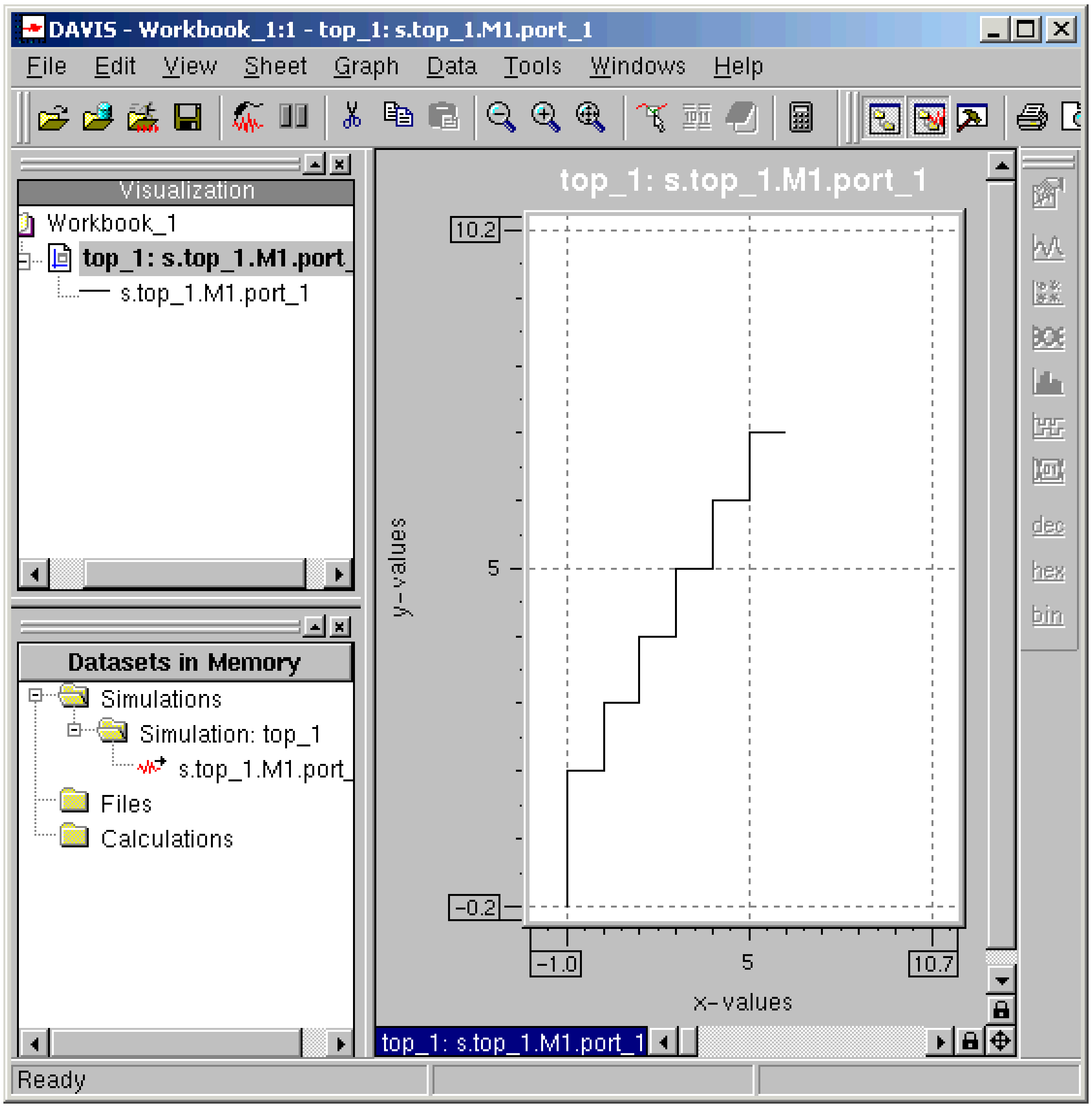
Down Sample ... using all values

Visualization ... Add this graph to the selected sheet

A DAVIS window appears showing the port_1 output data, although at this point, no data has yet been generated.

- Using the Step button on the simulation control panel, step through the simulation a few times and observe the DAVIS display.
- If necessary, use the zoom controls and the scroll bars to resize the DAVIS display to view a waveform similar to the one shown in Figure 7-25.

Figure 7-25 DAVIS Display Counter Output



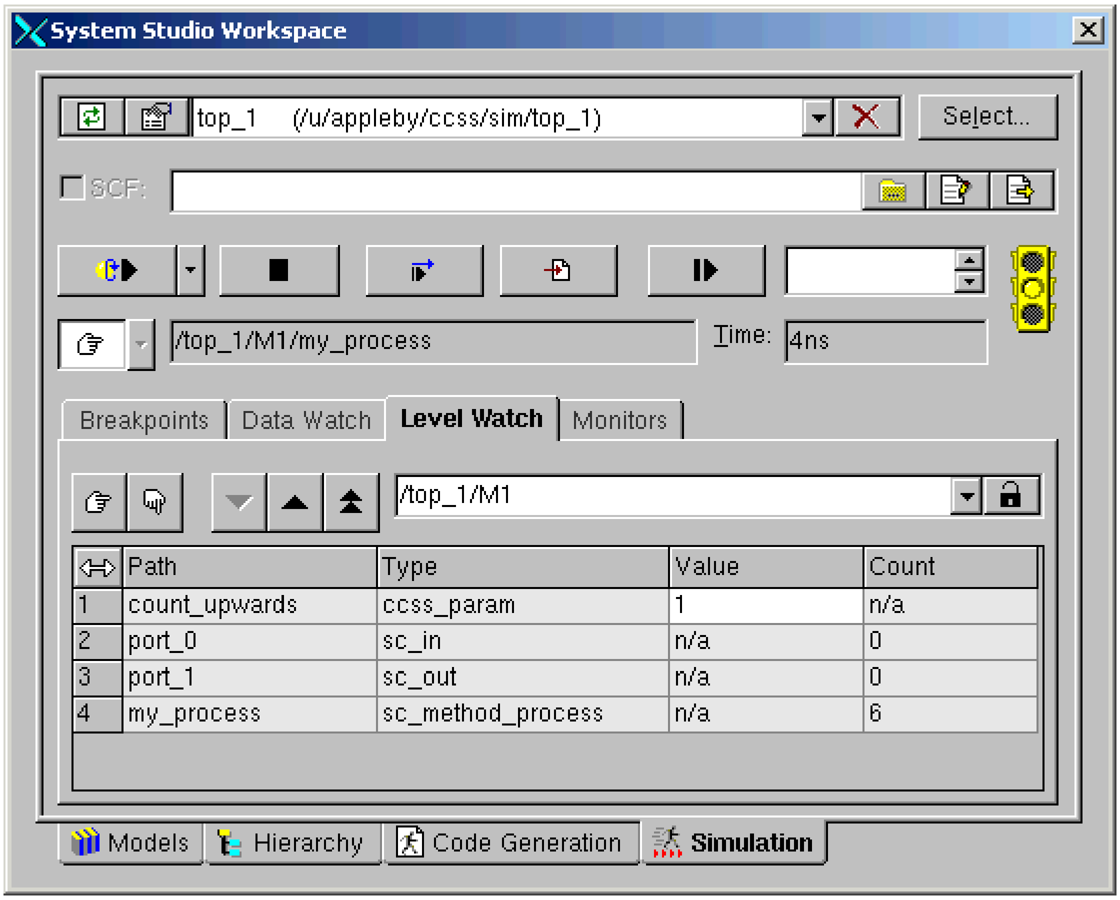
Changing the Simulation Data

In addition to using the simulation control panel to start, stop, or step through the simulation execution, you can also use it to change simulation data.

In this example, you can use the simulation control panel to change the value of the count_upwards parameter from 1 to 0 so that the counter starts to count down instead of up.

- In the simulation control panel, click the Level Watch tab and then step through the simulation until the Level Watch details for the M1 counter are available. You will see a display similar to that shown in Figure 7-26.

Figure 7-26 Changing Simulation Parameters

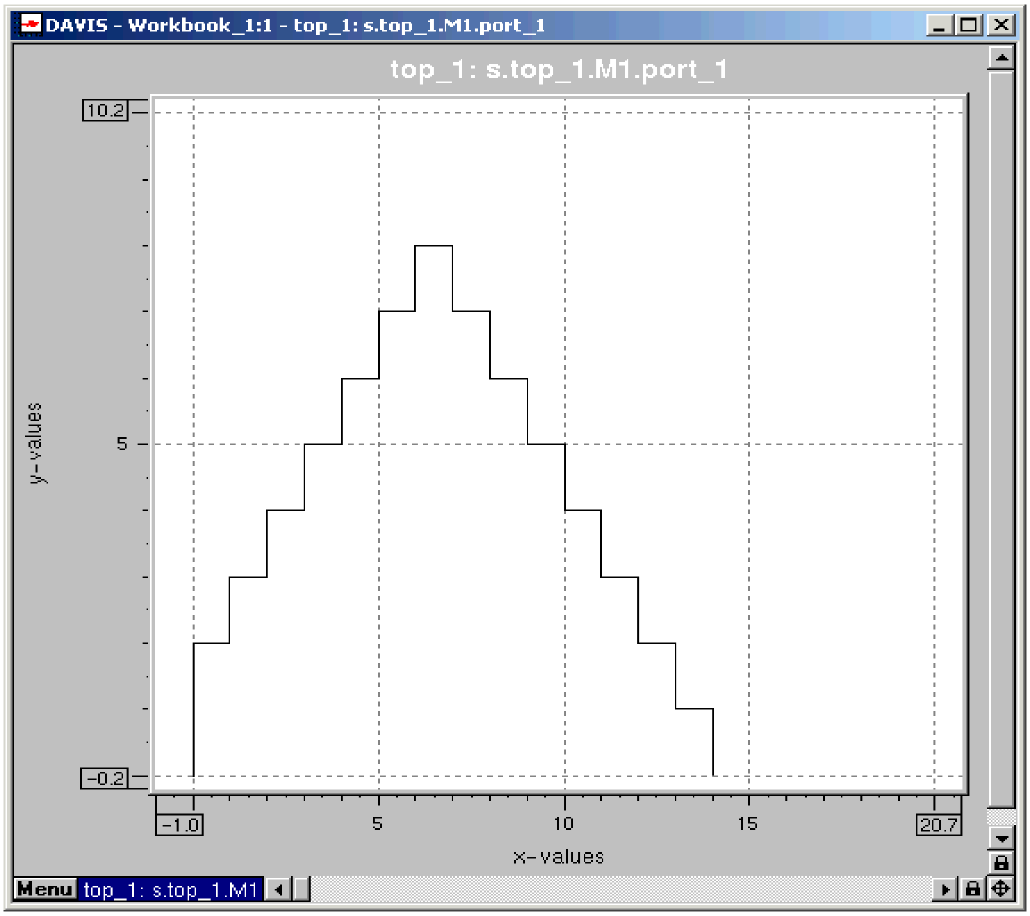


	Path	Type	Value	Count
1	count_upwards	ccss_param	1	n/a
2	port_0	sc_in	n/a	0
3	port_1	sc_out	n/a	0
4	my_process	sc_method_process	n/a	6

- To change the value of the count_upwards parameter from true to false, type 0 in the Value field.

- Step through a few more cycles using the simulation control panel and observe the DAVIS display as the counter decrements the count instead of incrementing it, as shown in Figure 7-27.

Figure 7-27 Results of Changing the Parameter.



This concludes this part of the tutorial on using the simulation control panel. You can save and exit the tutorial, or you can experiment with the other features of the simulation control panel, such as setting breakpoints, and using data watch to monitor the execution of the simulation.

Port Cloning and MultiPort Adapters

In SystemC, a port can address multiple channels. In System Studio, there are two ways to implement multiple port-to-channel connections. The preferred way is to use a system of copying and pasting ports known as “cloning.” The second way is to use a multiport adapter. Both methods are described in this section.

In general, a port in SystemC is derived from the port base class sc_port<IF,N>, where IF is an interface, and N is the maximum number of interfaces that can be connected to the port. The port classes you have encountered so far in this tutorial (sc_in<type> and sc_out<type>) are a shortcut for the fuller and more accurate definitions of sc_port<sc_port_in_if<type>,1> and sc_port<sc_port_out_if<type>,1> respectively.

Based on the knowledge you have gained in the first part of this chapter, you are now familiar with the basics of creating a SystemC model. Therefore, the next part of this tutorial deals only with the unique aspects of creating multiple port-to-channel connections. The example will, as usual, use a very simple design to clarify the process.

Port Cloning

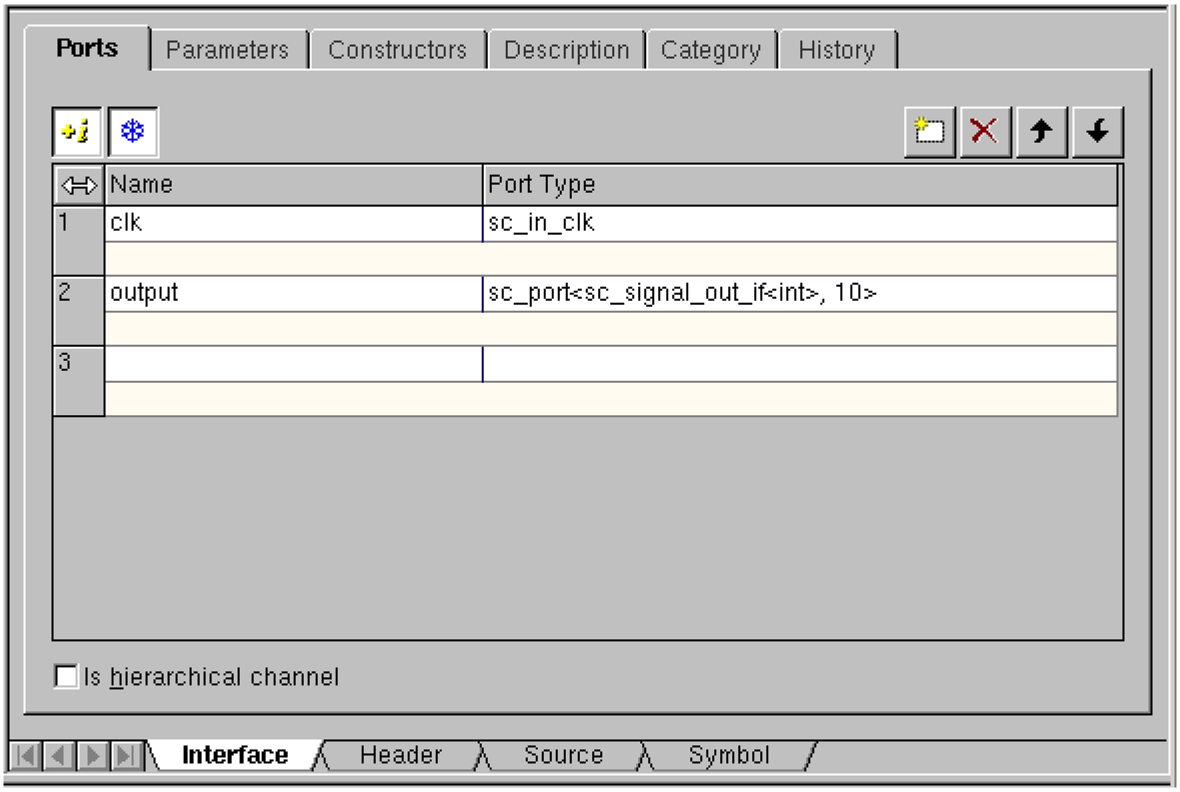
The first step in this design will be to create a source model. This will be a very simple, primitive, clock-driven source; the most important aspect of this source model is that, in the port declaration, you will define the number of channels that it can output. The output port is declared as having type sc_port<sc_signal_out_if<int>, 10>, thus supporting up to 10 outputs.

Note:

A special case would be to declare N=0 (sc_port<IF, 0>). In this case you can connect an arbitrary number of interfaces to the port.

- Create an architectural primitive model as your multiport source, with the port declarations shown in Figure 7-28.

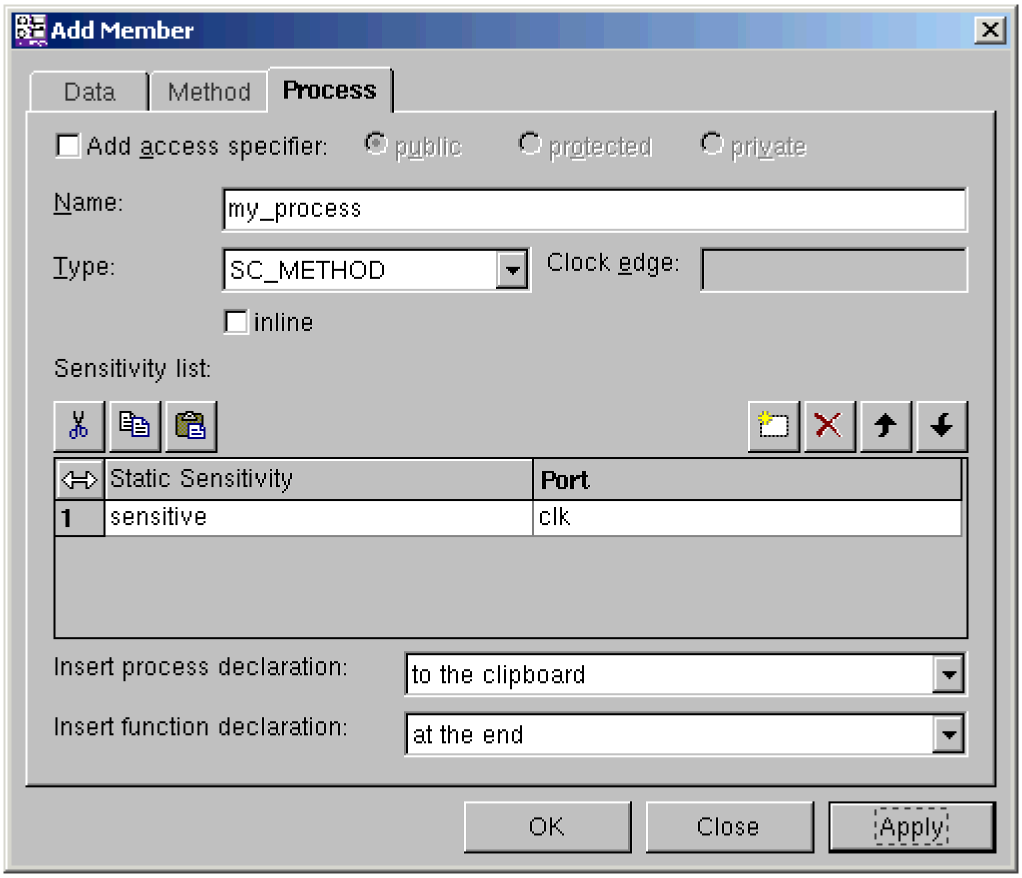
Figure 7-28 Creating a Multiport Source Model



	Name	Port Type
1	clk	sc_in_clk
2	output	sc_port<sc_signal_out_if<int>, 10>
3		

- Next, create a member process sensitive to the clock as shown in Figure 7-29.

Figure 7-29 The Member Process



[ ] Add access specifier: ● public ○ protected ○ private

Name: my_process

Type: SC_METHOD

Clock edge:

[ ] inline

Sensitivity list:

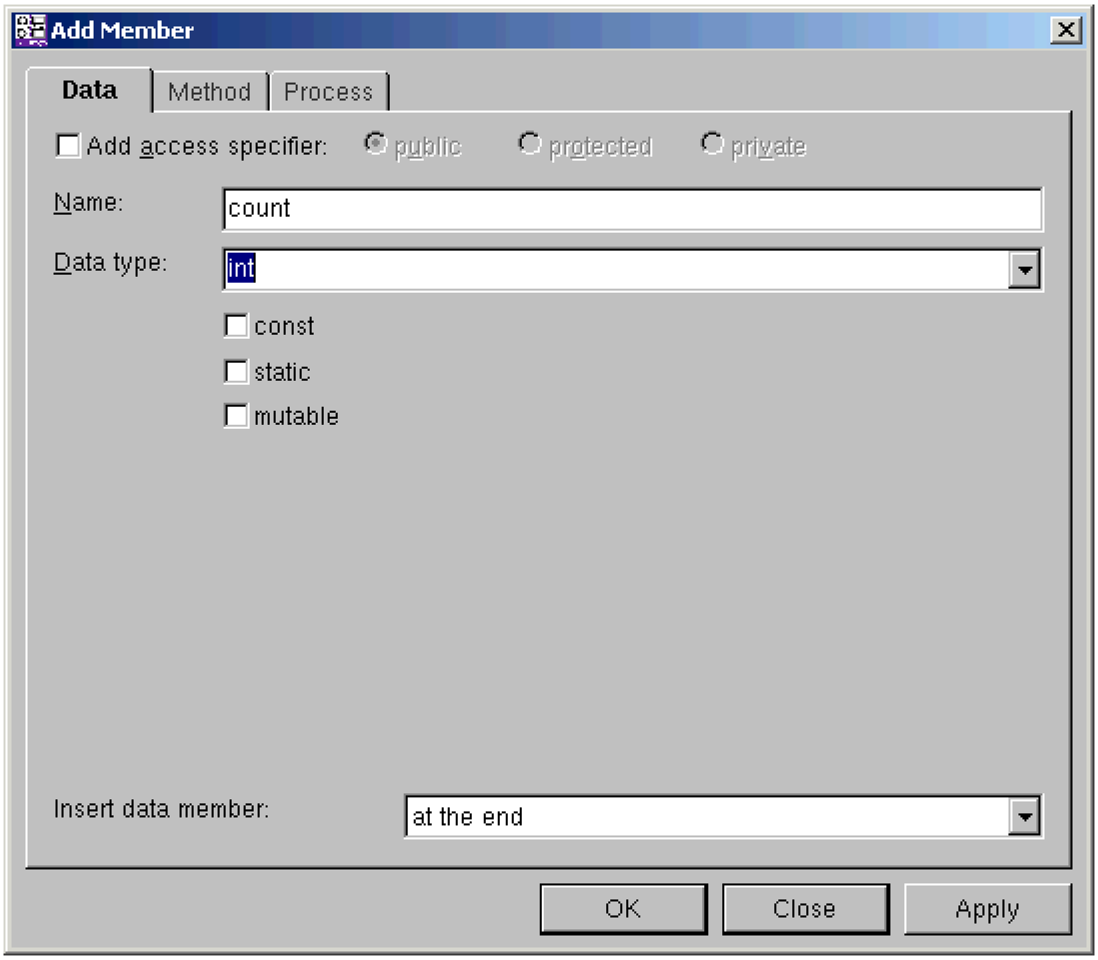
	Static Sensitivity	Port
1	sensitive	clk

Insert process declaration: to the clipboard

Insert function declaration: at the end

- Create a data member named count of type int, as you did for the counter model in the previous section (see Figure 7-30).

Figure 7-30 The Data Member



☐ Add access specifier: ● public ○ protected ○ private

Name: count

Data type: int

☐ const

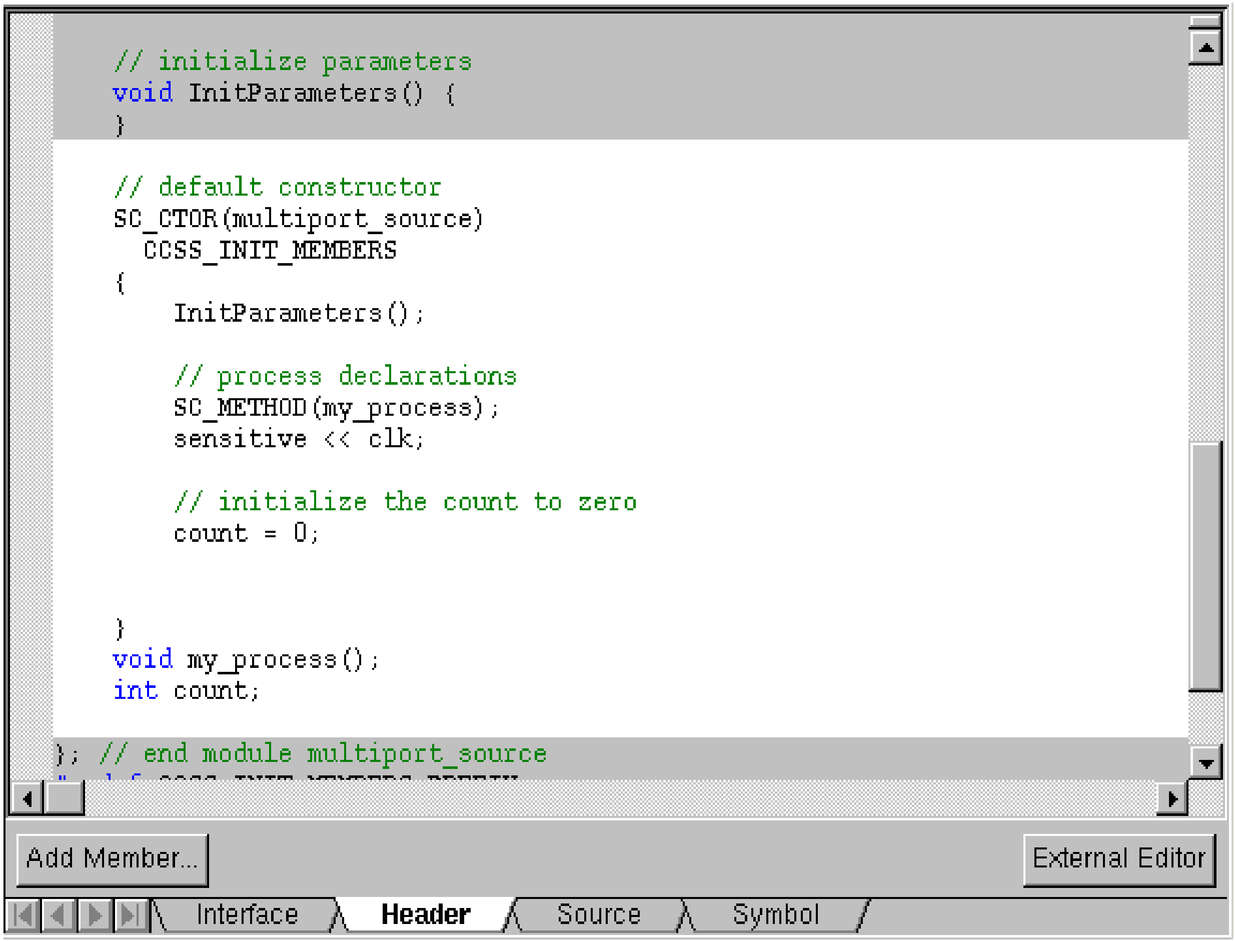
☐ static

☐ mutable

Insert data member: at the end

- In the Header View, initialize count to 0 in the model constructor, as shown in Figure 7-31.

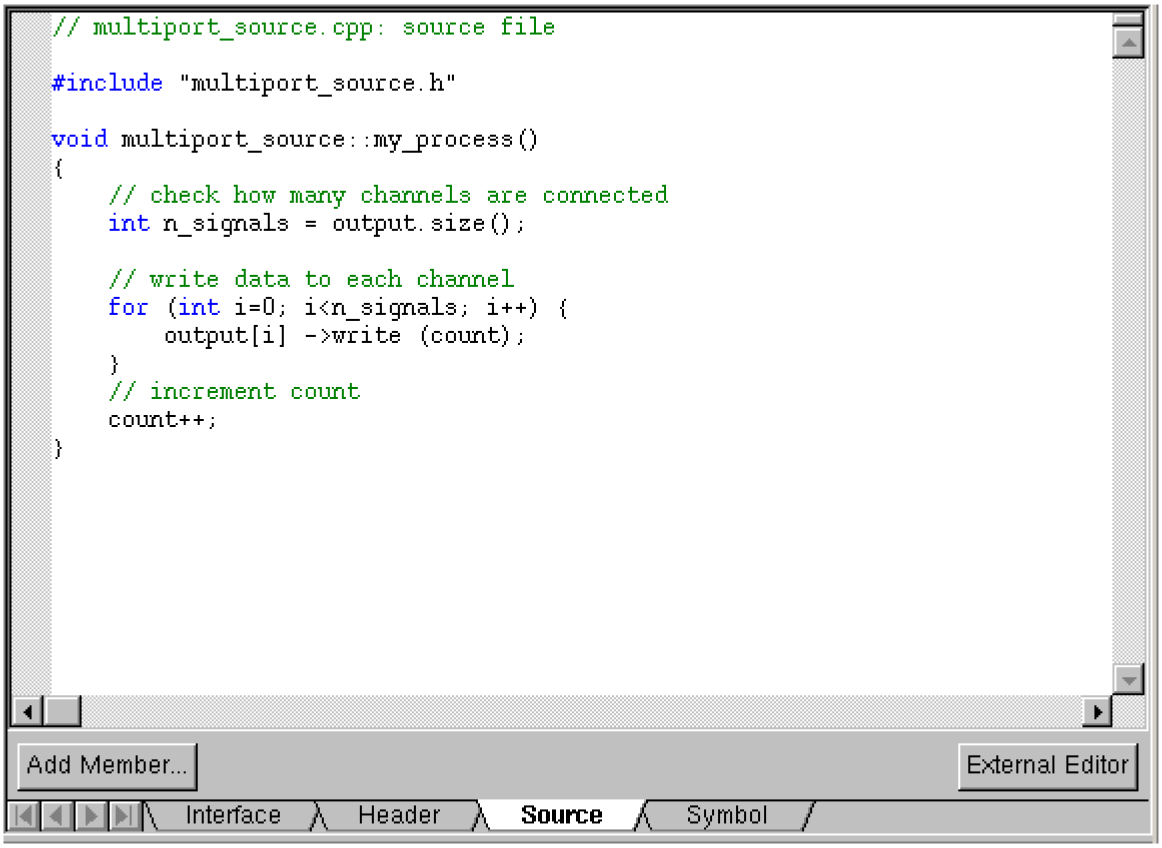
Figure 7-31 Initializing the Count Parameter



```
    // initialize parameters
    void InitParameters() {
    }

    // default constructor
    SC_CTOR(multiport_source)
      CCSS_INIT_MEMBERS
    {
        InitParameters();

        // process declarations
        SC_METHOD(my_process);
        sensitive << clk;

        // initialize the count to zero
        count = 0;


    }
    void my_process();
    int count;

}; // end module multiport_source
```

- In the source view, implement the code as shown in Figure 7-32.

Figure 7-32   The Model Source View

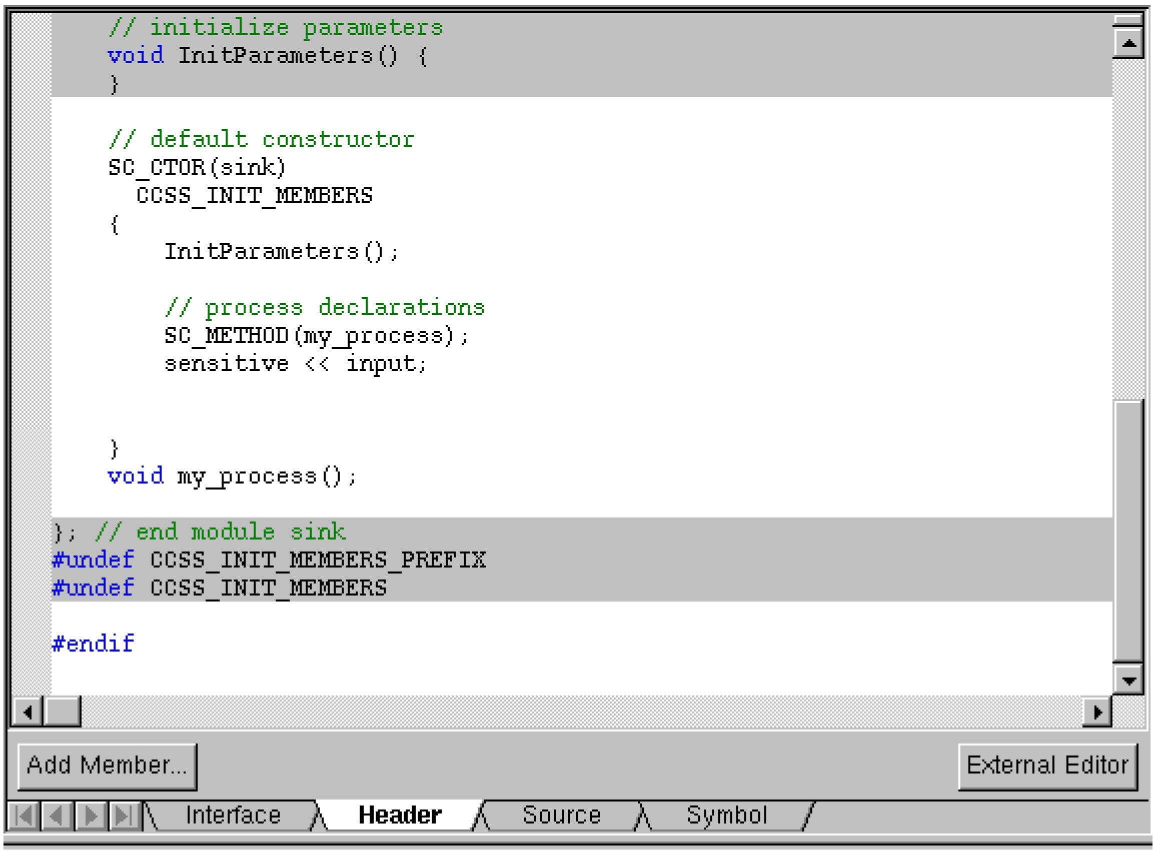


```
// multiport_source.cpp: source file

#include "multiport_source.h"

void multiport_source::my_process()
{
    // check how many channels are connected
    int n_signals = output.size();

    // write data to each channel
    for (int i=0; i<n_signals; i++) {
        output[i] ->write (count);
    }
    // increment count
    count++;
}
```

Next, create the sink model that will be connected to the source model.

- Create a model with one input port of type sc_<int>, with a process sensitive to the input. This model uses a simple sink that writes the value that it receives. The header view is shown in Figure 7-33.

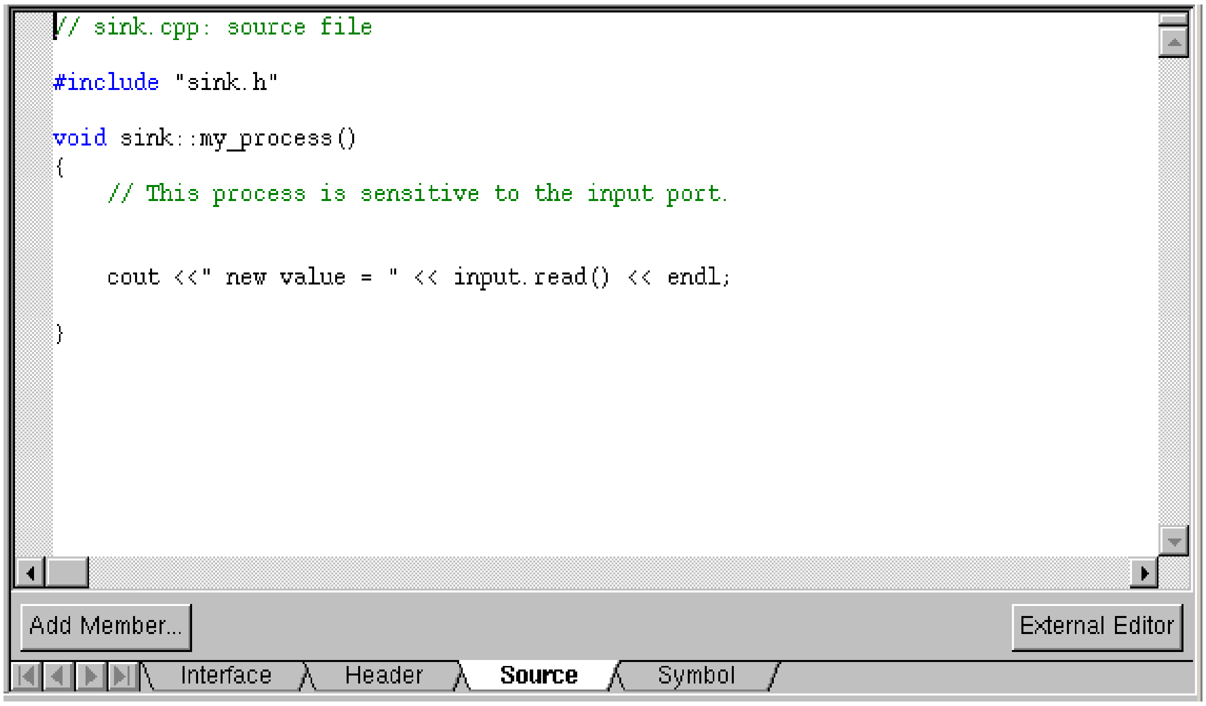
Figure 7-33 The Sink Model Header View



```
    // initialize parameters
    void InitParameters() {
    }

    // default constructor
    SC_CTOR(sink)
      CCSS_INIT_MEMBERS
    {
        InitParameters();

        // process declarations
        SC_METHOD(my_process);
        sensitive << input;


    }
    void my_process();

}; // end module sink
#undef CCSS_INIT_MEMBERS_PREFIX
#undef CCSS_INIT_MEMBERS

#endif
```

- Implement the code shown in the source view in Figure 7-34.

Figure 7-34   The Sink Model Source View



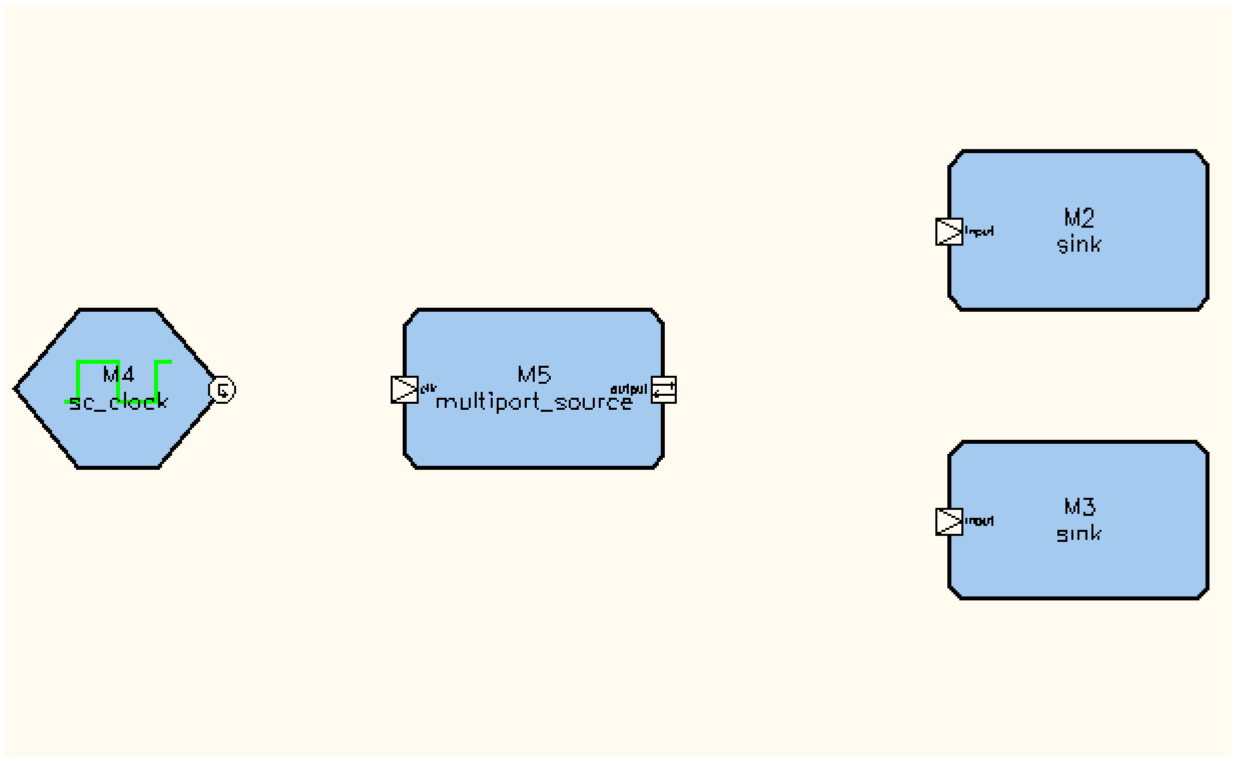
```
// sink.cpp: source file

#include "sink.h"

void sink::my_process()
{
    // This process is sensitive to the input port.


    cout <<" new value = " << input.read() << endl;

}
```

Now that the models have been created, the next step is to create a test bench model to hold them.

- Create an architectural hierarchical model named multiport_clone.
- Open multiport_clone and instantiate one instance of the source model and two instances of the sink model into the test bench design.

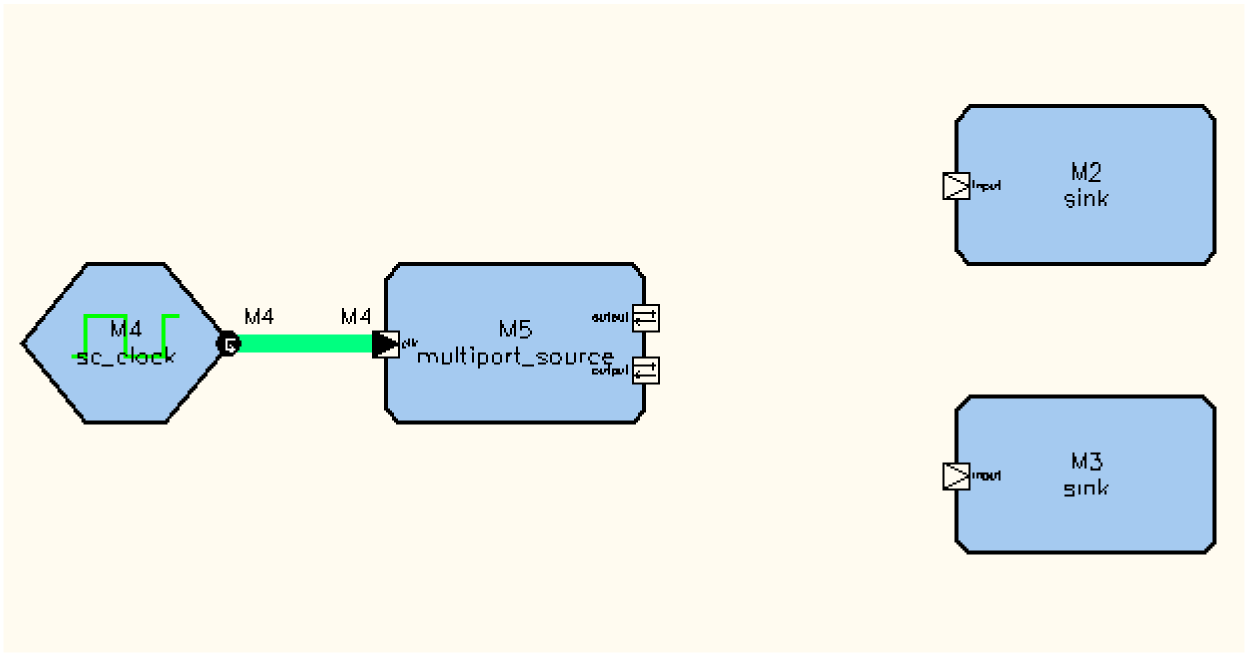
- Navigate to the library architectural/systemc, and drag-and-drop an instance of the model sc_clock into your test bench. The model should now look like Figure 7-35.

Figure 7-35 Instantiate the Models



- Connect a channel between the sc_clock and the input of the source model.
- Select the output port of your source model to highlight it.
- From the main menu choose Edit › Copy › Edit › Paste and then move the mouse pointer over the position on the source module where you require the cloned port. When you have the correct position, left-click to instantiate the cloned port. Your source model should look like the model in Figure 7-36.

Figure 7-36   Multiport Clone Test Bench



- Connect channels from the two output ports to the inputs of the two sink models. Use type sc_signal<int> for these channels.
- Finally, check the design and, if there are no errors, simulate the design using the simulation control panel.

By single-stepping through the simulation, you will see that each instance of the sink model is activated in turn.

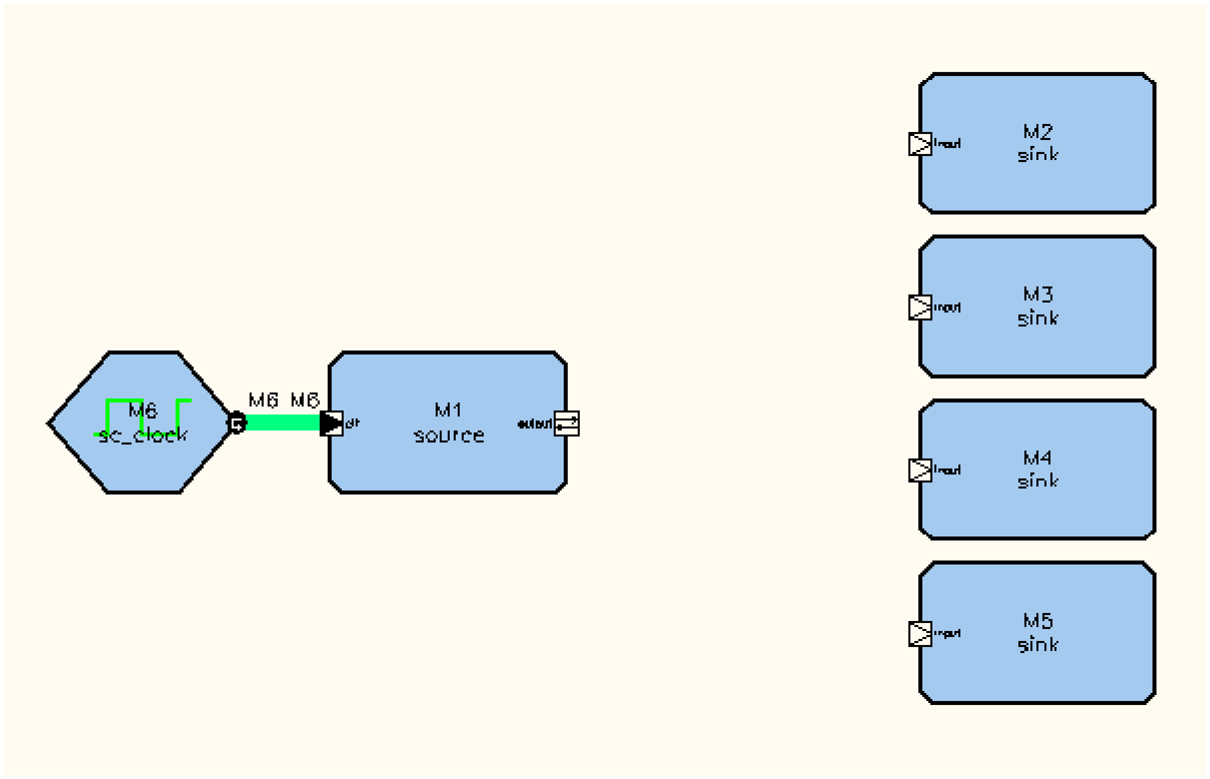
This method of cloning ports or interfaces works very well when there is sufficient space on the module to copy and paste the required number of ports or interfaces. However, in designs where space is limited, you can use a multiport adapter to fanout the ports.

Using Multiport Adapters

The next design uses the same source and sink models as in the last design, and the output is the same. You will create a new test bench for this design.

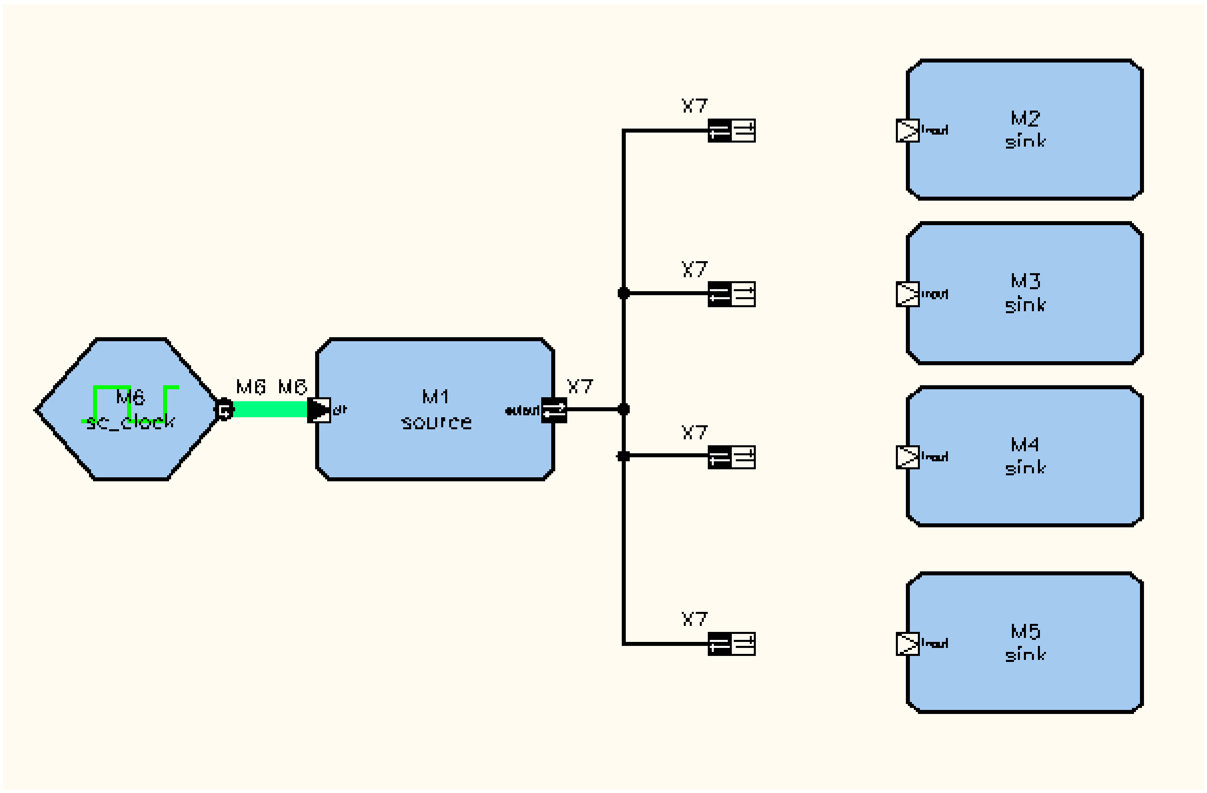
- Create an architectural hierarchical model named multiport_adapter.
- Instantiate the clock, the multiport_source module, and the sink modules as you did in the last design, but this time you can add tree or four sink models if you want.
- Lay out your test bench as shown in Figure 7-37.

Figure 7-37 Multiport Adapter Test Bench



- From the main menu, choose Insert › Adapter.
- Move the mouse pointer to the source output port, and draw a channel from the output port to a position in line with the input port of one of the sink modules.
- Repeat the previous step for the other sink modules until your design looks like Figure 7-38.

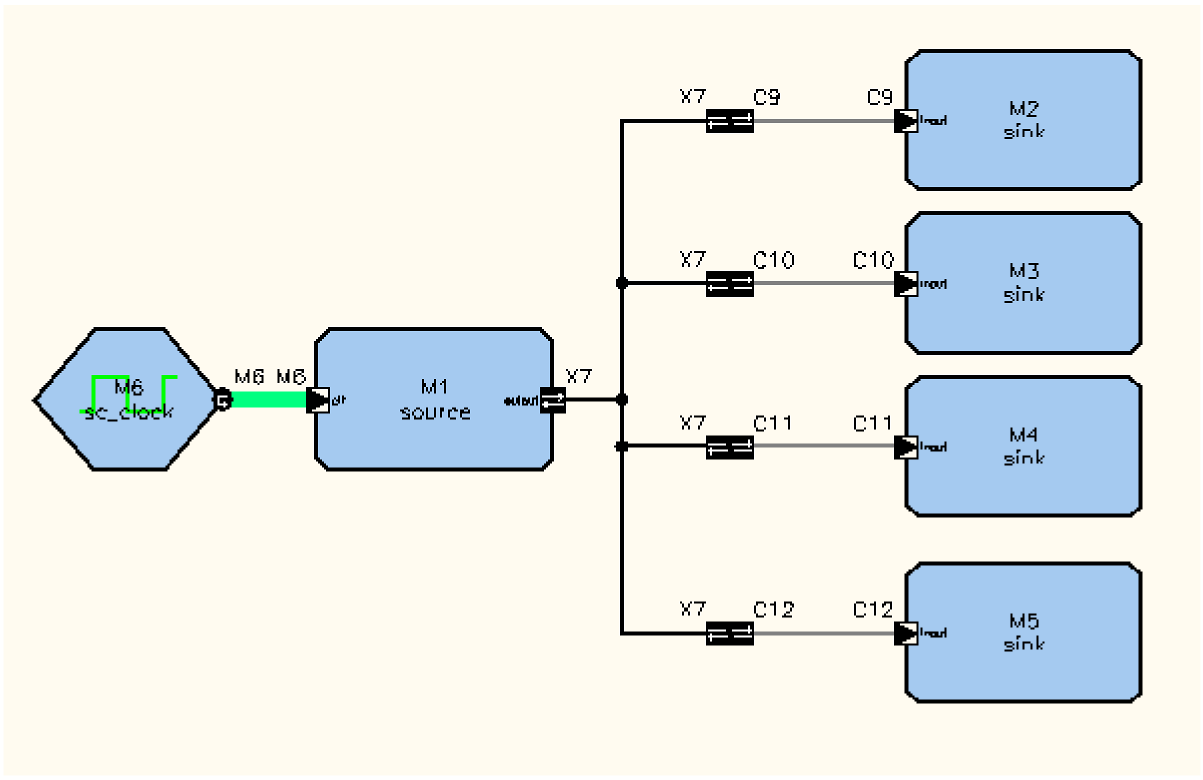
Figure 7-38 Creating the Multiport Channels - Step1



- From the toolbar, select the channel button and draw a channel from one of the adapter symbols to the corresponding sink input port.

- Repeat the previous step for the other channels until your test bench looks like Figure 7-39.

Figure 7-39 Creating the Multiport Channels - Step 2



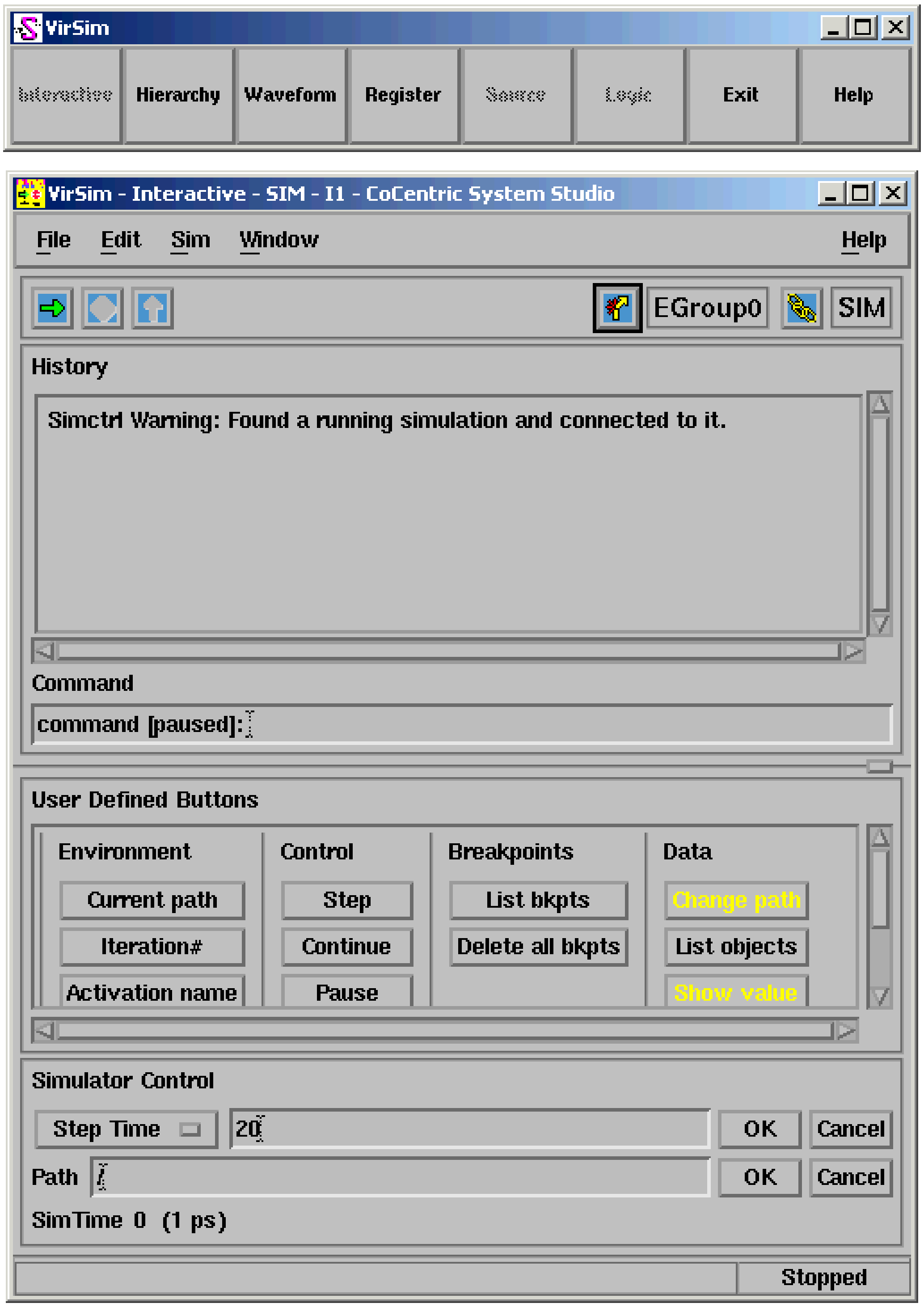
You can now check and run the model. The results should be the same as for the previous model.

Using CoCentric System Studio VirSim

In addition to using DAVIS to visualize the simulation output, you can use CoCentric System Studio VirSim to control, monitor, and debug an architectural simulation. This section gives you a simple introduction to the use of VirSim, and shows you how to connect to a running simulation. For more in-depth information on the extensive features of VirSim, see the *CoCentric System Studio VirSim Guide*, and the VirSim online Help.

- First, if you terminated your simulation at the end of the last section, you must regenerate the code and start the simulation again in paused mode.
- From the System Studio main menu, choose Simulation › Open VirSim. The Select Simulation dialog box appears.
- Select the running simulation and click OK. VirSim automatically connects to the running simulation. The VirSim Interactive window and the VirSim main menu bar appear, as shown in Figure 7-40.

Figure 7-40 VirSim Main Menu and Interactive Window



Simctrl Warning: Found a running simulation and connected to it.

command [paused]:

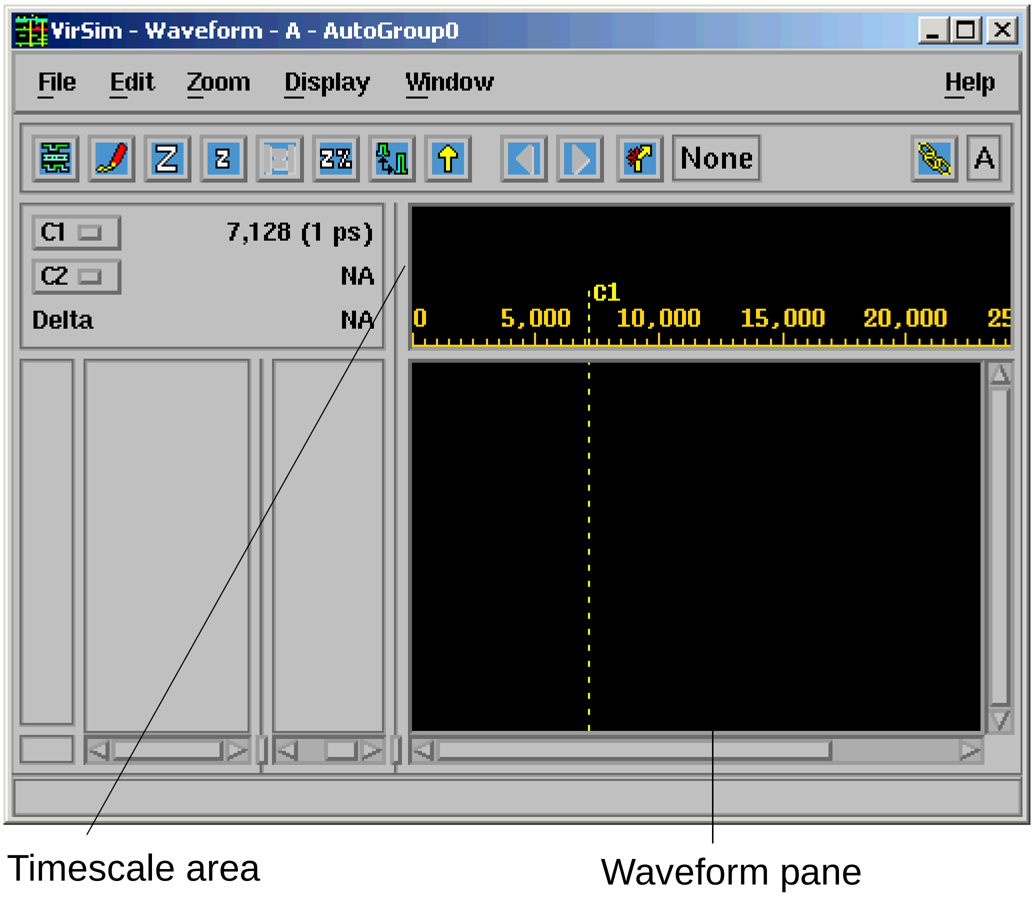
Step Time: 20

Path: /

SimTime 0 (1 ps)

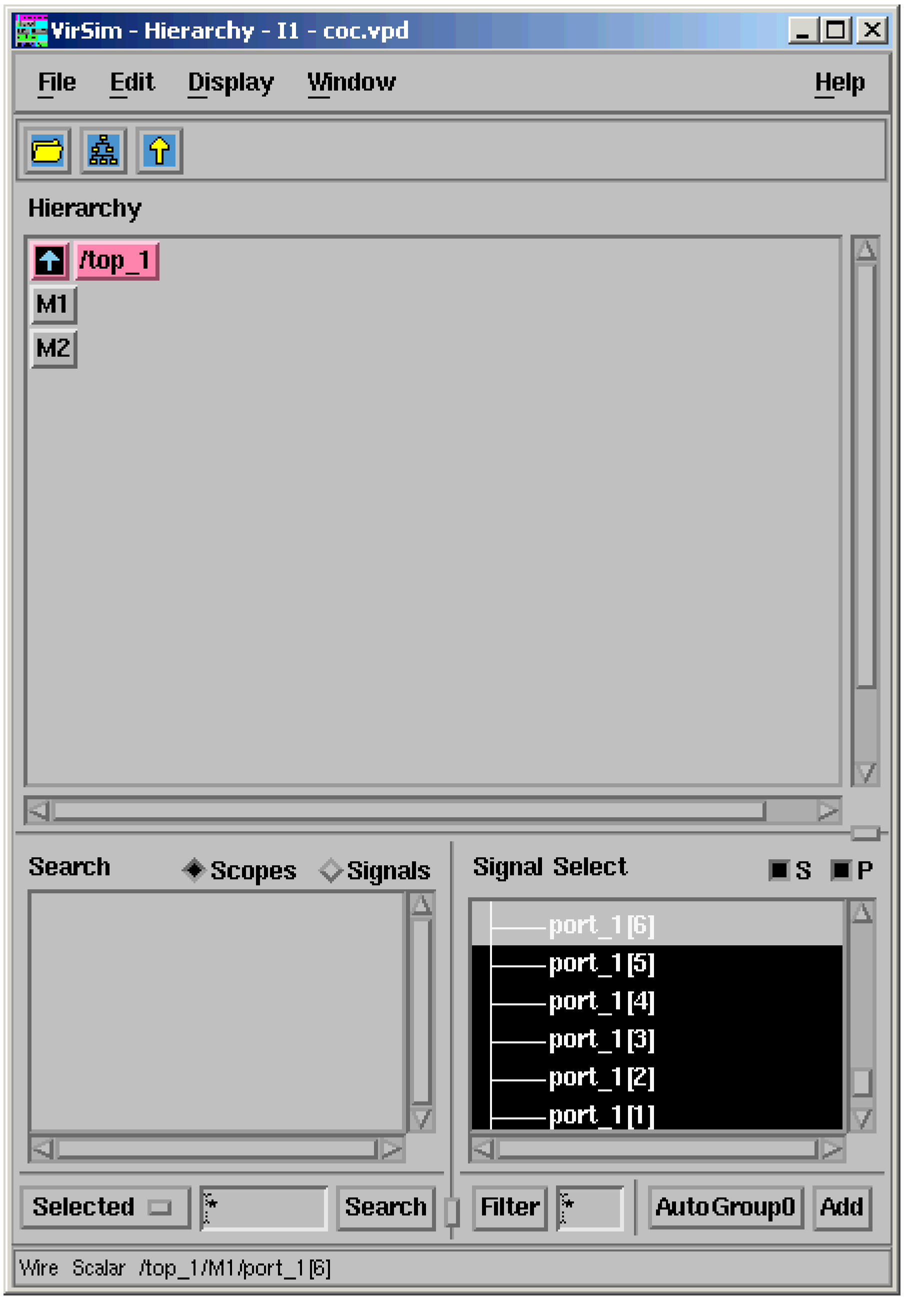
- From the VirSim main menu bar, choose Hierarchy. The Hierarchy window appears.
- From the VirSim main menu bar, choose Waveform, The Waveform window appears; see Figure 7-41.

Figure 7-41 Waveform Window



- In the Hierarchy window, select the down arrow next to the model name, then select the M1 instance.
- In the Hierarchy window Signal Select pane, click the + symbol next to the port_1[31:0] signal to expand the signal (see Figure 7-42).

Figure 7-42 VirSim Hierarchy Window



Wire Scalar /top_1/M1/port_1[6]

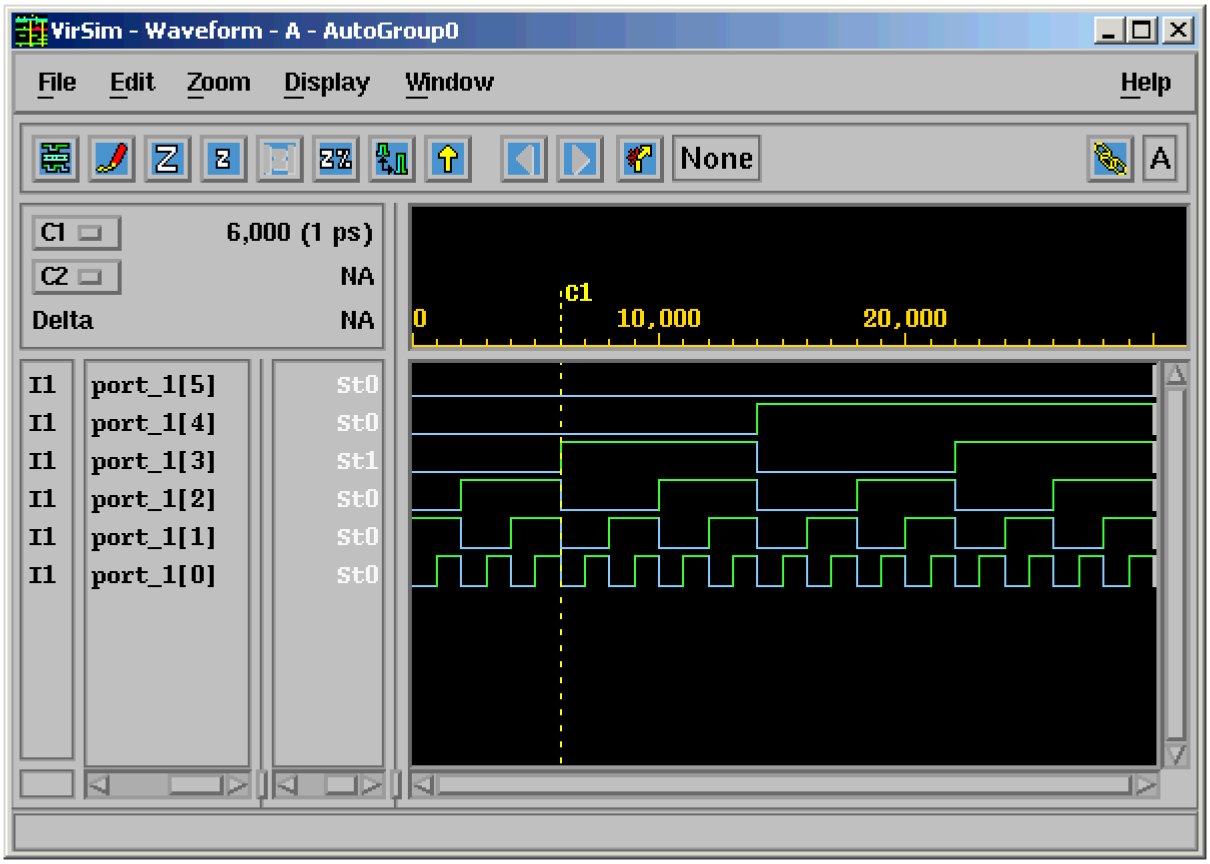
- Select the bits that you are interested in (use Shift + the left mouse button to highlight multiple bits).

- In the Hierarchy window, click the Add button to view the selected bits in the Waveform window. Add

  Alternatively, use the middle mouse button to drag and drop the highlighted bits into the Waveform pane of the Waveform window below the Timescale area.

- In the Waveform window, click the Zoom Percent button and select 100%.
- In the VirSim Interactive window, use the Step button to step through the simulation a few times.
- The resulting display will look similar to Figure 7-43. If necessary, resize the Waveform window and use the scroll bar to view the desired signals.

Figure 7-43 VirSim Waveform Window



You can continue to step through the simulation using the simple preconfigured commands provided in the Virsim Interactive window, or you can enter Tcl commands at the command line.

You can use VirSim concurrently with DAVIS to visualize the signal in different ways.

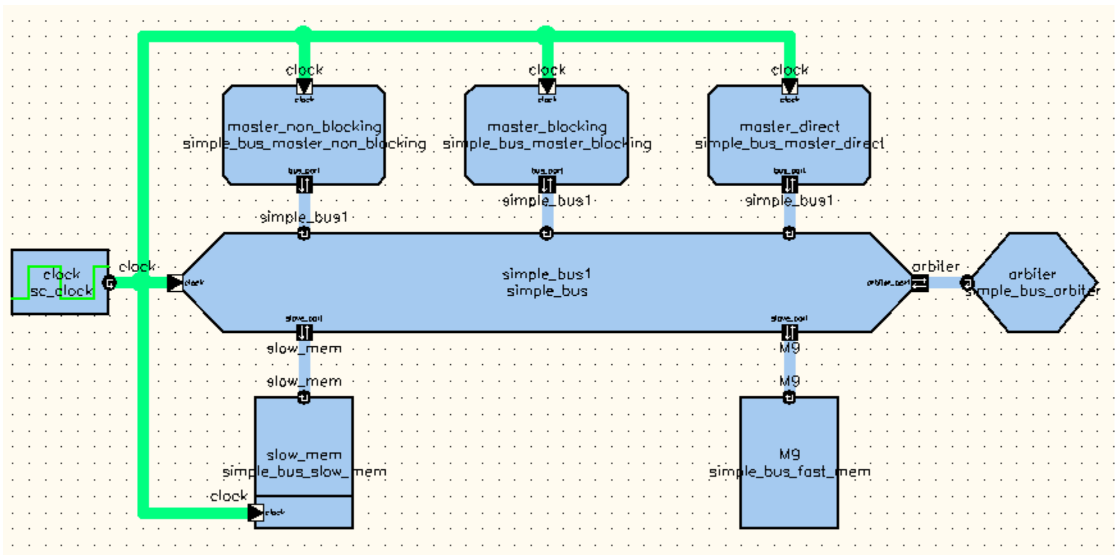
You can use either the VirSim Interactive window or the System Studio simulation control panel to control the simulation.

Simple Bus Example

You have now covered the basic steps in creating a simple SystemC architectural model. You will find more detailed information in the System Studio User Guide however, at this stage, it will be useful to examine the source code and header files of a more complex architectural model.

Figure 7-44 shows the schematic of a simple bus test model. You can find this example in the cocentric/architectural/simple_bus/ simple_bus_examples library. The building-block modules for this example can be found at the next level up in the simple_bus library.

Figure 7-44 Simple Bus Test Example



Open this test example and examine the information in the Interface, Source, and Header views.

8

Exploring the Float-to-Fixed Capabilities

This information in this chapter is presented in the following sections:

One of the first steps in moving from a software algorithm to a hardware implementation is to convert the design from a floating-point representation to a fixed-point representation. This will inevitably introduce quantization and other errors.

CoCentric System Studio’s floating-point-to-fixed-point exploration capabilities allow you to create a properties database for a design in which you can save the configuration parameter values that you consider are key factors in the floating-point-to-fixed-point conversion. By specifying values for the parameters (such as the word length for ports) or enabling the statistics collection function for a given symbol, you can run a simulation, merge the results of that simulation into the current model, and then compare the results with those of other simulations. This approach allows you to experiment with the values of the parameters until you find the best fit. At this point you can then “refine” the design to create a new design that incorporates the design improvements that you have introduced.

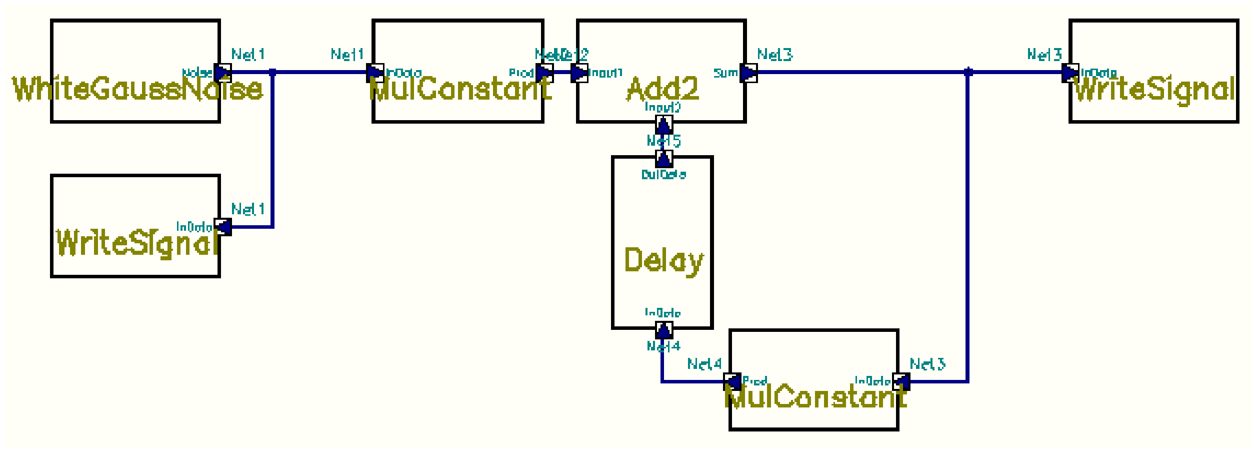
It is always possible in System Studio to use the available type mechanisms to change a model from a floating point to a fixed point representation. However, there are important advantages to be gained from using properties instead of type parameters. For example, in the simple example demonstrated in this chapter, the example model (add2) only has one type parameter. To create a usable add2 model three type parameters would be required, one for each port. If you were to consider a more complicated model, such as a filter/Biquad, you would need a type parameter for every port, every variable, and every temporary variable in the model. Working with a realistic model would require you to set a bewildering number of parameters during the exploration, which would be totally unacceptable. In addition, these models are too complicated for work without fixed-point types.

About the Demo

This chapter steps through a simple demonstration that illustrates System Studio's floating-point-to-fixed-point capabilities.

The design consists of a first order IIR filter. You simulate a double precision version of an IIR filter design using a testbench that consists of a White Gaussian Noise source block and a WriteSignal sink block as shown in Figure 8-1.

Figure 8-1 The IIR Filter Design



The simulation provides the reference output data as well as some useful statistical information such as the mean, variance, maximum value, minimum value, proposed fixed type, and so on at the different locations of the design.

Preparing the Demo

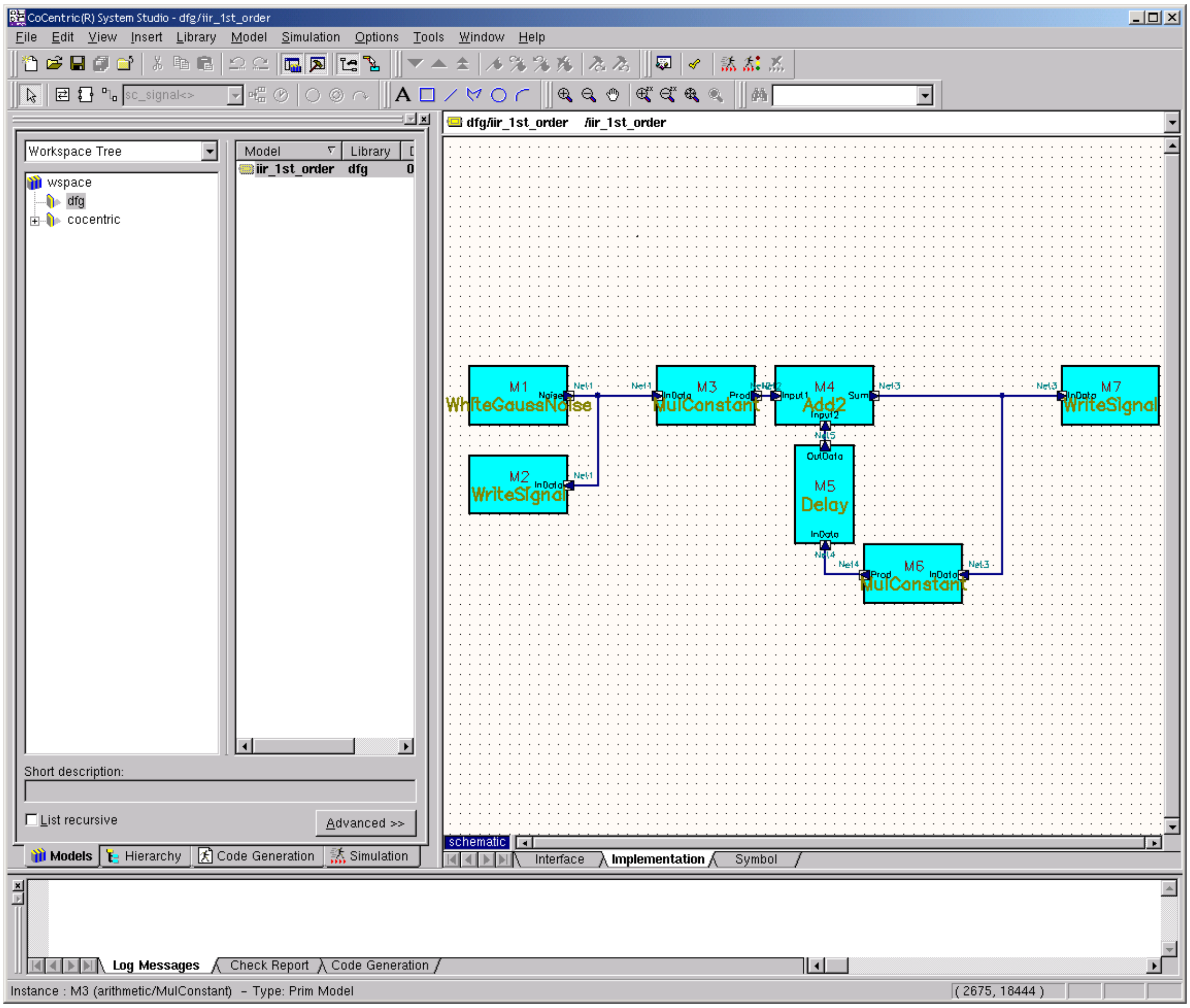
To set up and load the demo, you need to run a script to start System Studio and make the reference design available.

- Move to the directory from which you want to run the demo. In this chapter it is the directory referred to as $DEMODIR.
- Make sure that the `SYNOPSYS_CCSS` environment variable points to your System Studio installation.
- Execute the following command to start System Studio and open the reference workspace.

  ```
  $SYNOPSYS_CCSS/../../ccss/models/demos/properties/run_me.sh
  ```

- From System Studio, open the double-type reference design `dfg/iir_1st_order` by double-clicking it in the workspace view as shown in Figure 8-2.

Figure 8-2 The Loaded Design


CoCentric(R) System Studio - dfg/iir_1st_order
File Edit View Insert Library Model Simulation Options Tools Window Help
Workspace Tree
wspace
dfg
cocentric
Model Library
iir_1st_order dfg
Short description:
List recursive
Advanced >>
Models Hierarchy Code Generation Simulation
dfg/iir_1st_order /iir_1st_order
M1 WhiteGaussNoise
M2 WriteSignal
M3 MulConstant
M4 Add2
M5 Delay
M6 MulConstant
M7 WriteSignal
schematic
Interface Implementation Symbol
Log Messages Check Report Code Generation
Instance : M3 (arithmetic/MulConstant) – Type: Prim Model
(2675, 18444)

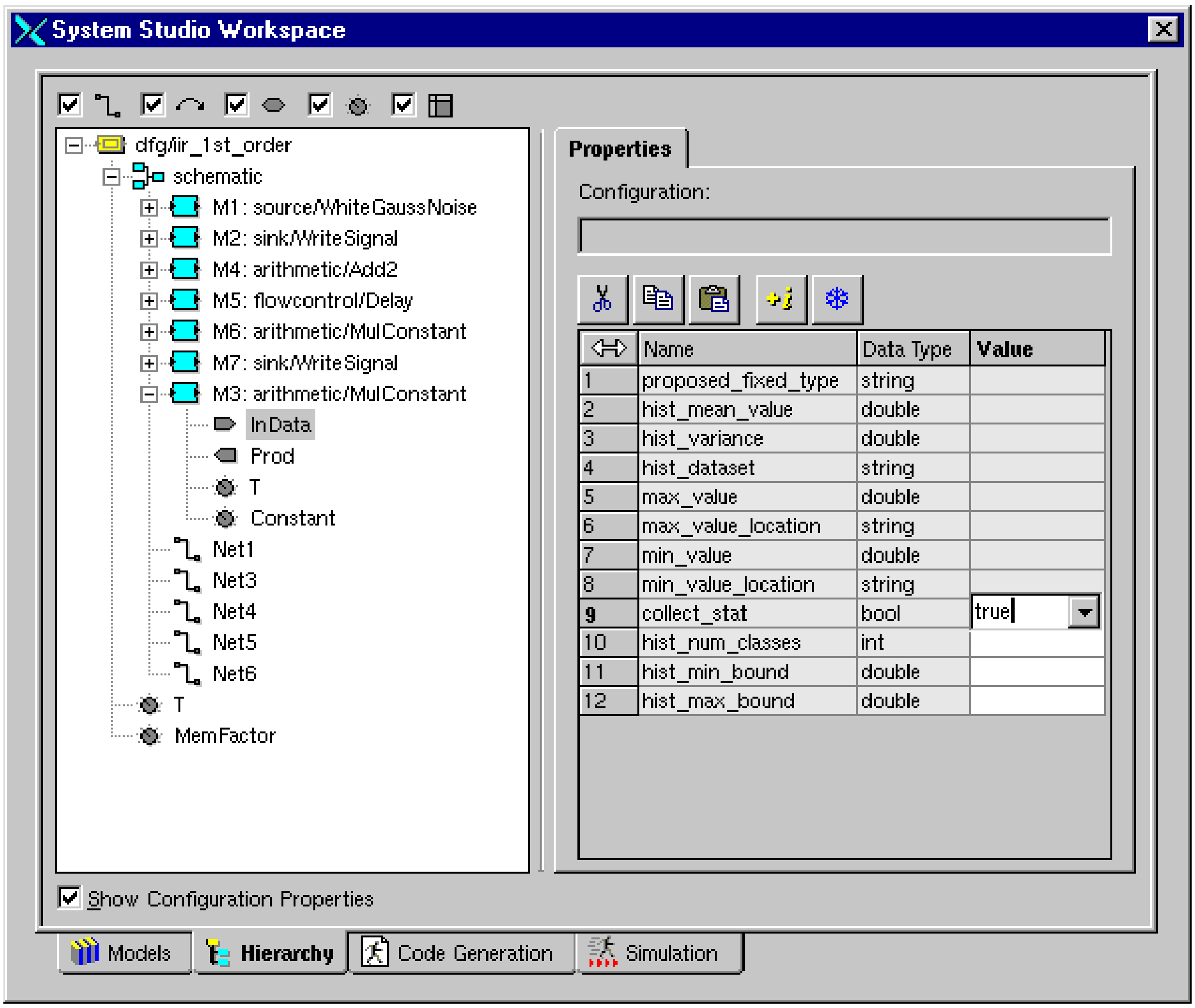
Statistic Functions

Open the hierarchy browser by choosing View › Workspace Tab › Hierarchy from the System Studio main menu bar, or by clicking the Hierarchy tab of the Workspace window.

To activate the statistic collection function, you need to select the relevant ports of an instance in the model, and set the value of the `collect_stat` property to true.

1. In the hierarchy browser, select the Show configuration Properties check box if it is not already selected. The Properties Configuration pane is displayed next to the hierarchy browser.
2. In the hierarchy browser, select the input port (`InData`) of instance M3 (`dfg/iir_1st_order/schematic/M3:arithmetic/MulConstant/InData`)
3. In the Properties Configuration pane, set the value of the `collect_stat` property to true, and then press Enter (see Figure 8-3).

Figure 8-3 Enabling Statistics Collection



System Studio Workspace

```
dfg/iir_1st_order
  schematic
    M1: source/WhiteGaussNoise
    M2: sink/WriteSignal
    M4: arithmetic/Add2
    M5: flowcontrol/Delay
    M6: arithmetic/MulConstant
    M7: sink/WriteSignal
    M3: arithmetic/MulConstant
      InData
      Prod
      T
      Constant
    Net1
    Net3
    Net4
    Net5
    Net6
  T
  MemFactor
```

Properties

Configuration:

	Name	Data Type	Value
1	proposed_fixed_type	string	
2	hist_mean_value	double	
3	hist_variance	double	
4	hist_dataset	string	
5	max_value	double	
6	max_value_location	string	
7	min_value	double	
8	min_value_location	string	
9	collect_stat	bool	true
10	hist_num_classes	int	
11	hist_min_bound	double	
12	hist_max_bound	double	

☑ Show Configuration Properties

4. Set the property value to true for the two input ports (`Input1` and `Input2`), and the output port (`Sum`) of instance M4. Do not forget to press Enter after each change.

5. You must now save the property database to a file:

   From the System Studio main menu, choose Model › Configuration Properties › Save As, to open the Save Configuration Properties dialog box.

6. In the File Name field of the Save Configuration Properties dialog box, enter `result0.ocd` as the name of the object configuration database file, and click save.

Note:

Appropriate values can also be set for the `hist_num_classes`, `hist_min_bound` and `hist_max_bound` properties if desired; otherwise a default number of classes (64) will be used and the minimum and maximum bounds will be computed using the data collected.

Activating statistic collection for a symbol provides the following information after simulation:

- The proposed fixed type representation for the symbol (`proposed_fixed_type`).
- The mean of the values collected at the symbol (`hist_mean_value`).
- The variance of the values collected at the symbol (`hist_variance`).
- The name of the data set containing the histogram classes (`hist_dataset`).
- The highest value collected at the symbol (`max_value`) and its location (`max_value_location`).
- The lowest value collected at the symbol (`min_value`) and its location (`min_value_location`).

Simulating and Back-Annotating the Design

On the System Studio main menu, click the Pop Top button to return to the top level view of the model, and then simulate the design by choosing Simulation › Simulate Model from the main menu.

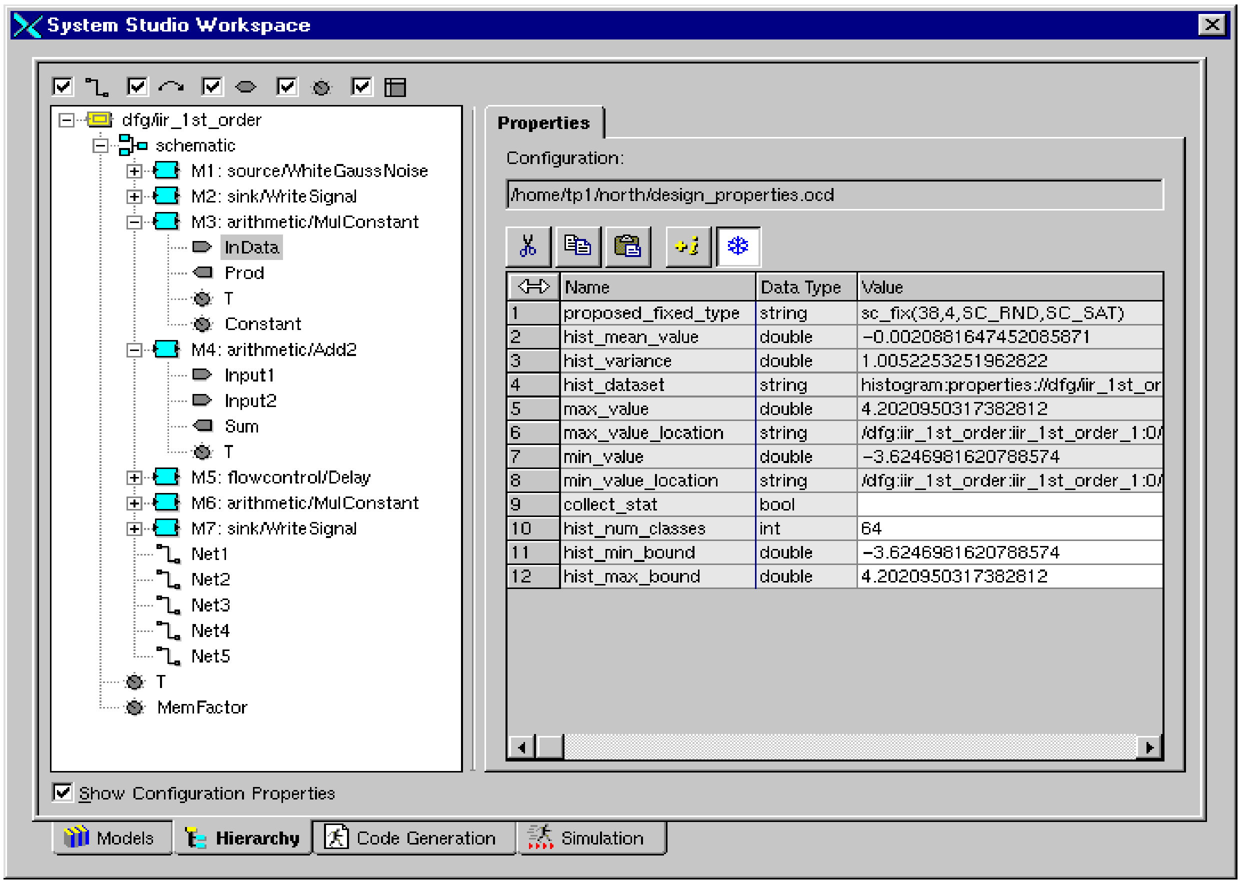
Because this is the first simulation of the design, the simulation name should be `iir_1st_order_`1, which will be referred to in this chapter as `$`*`SIMNAME`*`_1`. The simulation is only activated once, and only one iteration is executed, so the statistics information generated is stored in the file $DEMODIR/$SIMNAME_1/activ_1/iter_1/design_properties.ocd.

When the simulation execution has completed, you need to back-annotate the simulation statistics information into the Properties Configuration dialog box by merging the simulation results into the current properties database.

1. From the System Studio main menu, choose Model › Configuration Properties › Merge, to open the Merge Configuration Properties dialog box.
2. In the Merge Configuration Properties dialog box, use the file browser to select the file $DEMODIR/$SIMNAME_1/activ_1/iter_1/design_properties.ocd, which is the statistic information file generated by the simulation. Click Open.
3. Choose View › Workspace › Hierarchy to open the hierarchy browser.
4. From the hierarchy browser, select the input port (`InData`) for instance M3 (`dfg/iir_1st_order/schematic/M3:arithmetic/MulConstant/InData`).

   Note that the properties now have values as shown in Figure 8-4.

Figure 8-4 The Configuration Properties after Merging


System Studio Workspace

```
dfg/iir_1st_order
  schematic
    M1: source/WhiteGaussNoise
    M2: sink/WriteSignal
    M3: arithmetic/MulConstant
      InData
      Prod
      T
      Constant
    M4: arithmetic/Add2
      Input1
      Input2
      Sum
      T
    M5: flowcontrol/Delay
    M6: arithmetic/MulConstant
    M7: sink/WriteSignal
    Net1
    Net2
    Net3
    Net4
    Net5
  T
  MemFactor
```

Properties

Configuration:

/home/tp1/north/design_properties.ocd

	Name	Data Type	Value
1	proposed_fixed_type	string	sc_fix(38,4,SC_RND,SC_SAT)
2	hist_mean_value	double	−0.0020881647452085871
3	hist_variance	double	1.0052253251962822
4	hist_dataset	string	histogram:properties://dfg/iir_1st_or
5	max_value	double	4.2020950317382812
6	max_value_location	string	/dfg:iir_1st_order:iir_1st_order_1:0/
7	min_value	double	−3.624698162078857 4
8	min_value_location	string	/dfg:iir_1st_order:iir_1st_order_1:0/
9	collect_stat	bool	
10	hist_num_classes	int	64
11	hist_min_bound	double	−3.6246981620788574
12	hist_max_bound	double	4.2020950317382812

☑ Show Configuration Properties

Take a look at the values of some useful properties such as:

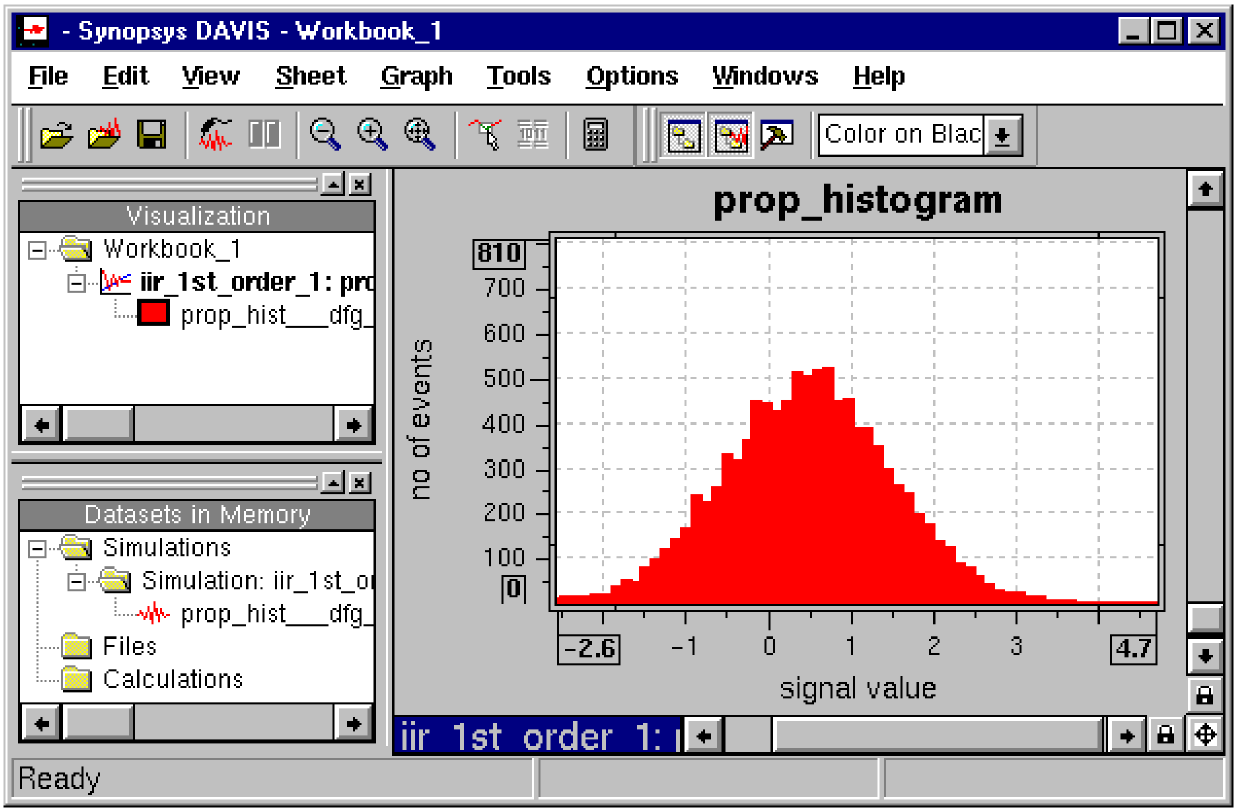
- `hist_dataset`, which shows the histogram data set (`histogram:properties://dfg/iir_1st_order/schematic/M3//arithmetic/MulConstant/*/InData//`)
- `proposed_fixed_type`, which is the proposed fixed type representation to be used for the input port

Using DAVIS to Visualize the Output

You can use DAVIS to visualize the simulation data. For example, to display the histogram data set, do the following:

- From the System Studio main menu, choose Tools › Davis to open the CoCentric DAVIS main window.
- From the DAVIS main window, choose File › Open System Studio Simulation to open the Open Simulation dialog box.
- From the Open Simulation dialog box, use the hierarchy browser to select “$SIMNAME_1/activ_1/iter_1/histogram:properties://dfg/iir_1st_order/schematic/M3//arithmetic/MulConstant/*/InData//”.
- Click OK to display the plot. If you zoom the display and, for example, select the horizontal range from -2.6 to 4.7, you will see a histogram similar to the example shown in Figure 8-5.

Figure 8-5 The Histogram Dataset Displayed in DAVIS

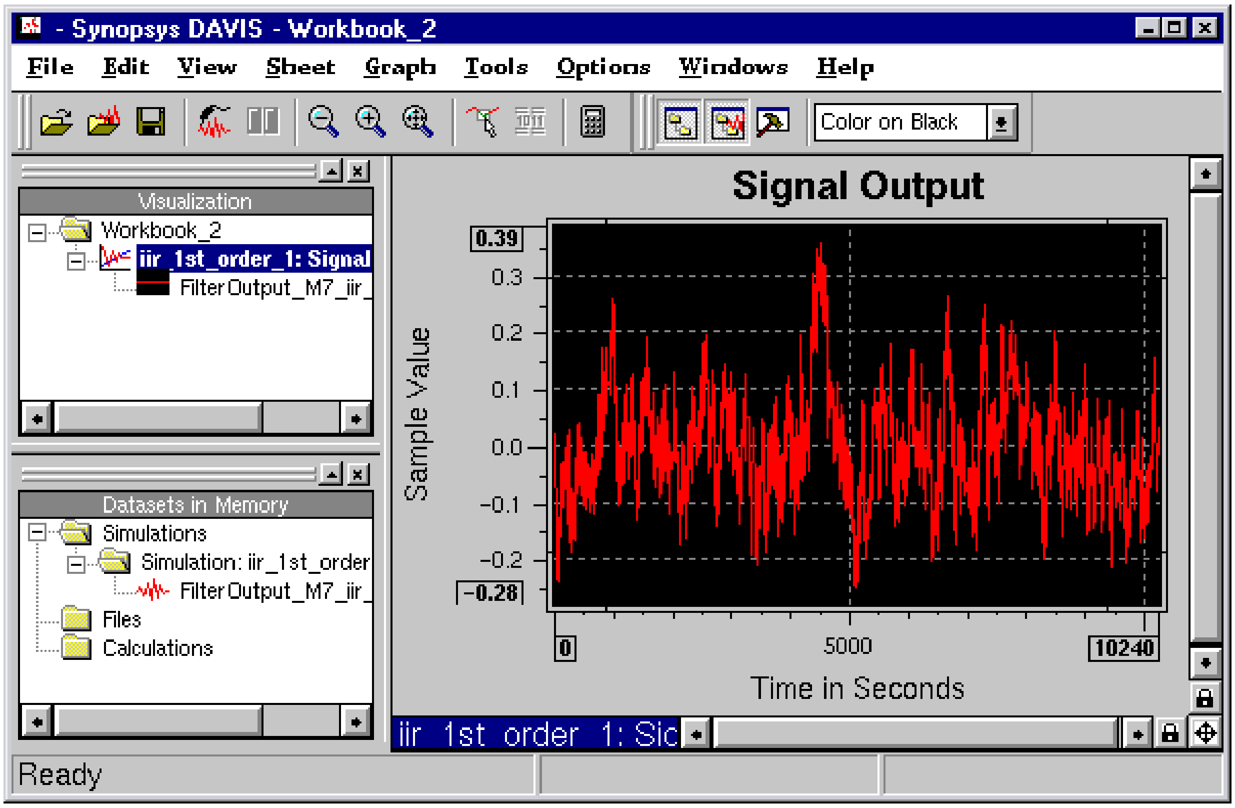


Use DAVIS to visualize the filter result by performing the following steps:

- From the main window of the first DAVIS workbook, choose File › New Workbook to open a second workbook.
- From the main window of the second DAVIS workbook, open the Open Simulation dialog box and select $SIMNAME_1/activ_1/iter_1/outdset:WriteSignal:M7.

  Click OK to see a display similar to Figure 8-6.

Figure 8-6 The Filter Output Displayed in DAVIS



9

Importing From COSSAP

This chapter describes how to import COSSAP models into CoCentric System Studio. It contains the following sections:

You can import various COSSAP files, as well as the COSSAP models themselves, into CoCentric System Studio. You can import the following models:

- A COSSAP primitive model into a System Studio SDS model
- A COSSAP hierarchical model or a pure COSSAP schematic into an System Studio dataflow (DFG) model

- A COSSAP assignment file into a System Studio simulation control file (.scf)
- A COSSAP generic C (.gc/.gcc) file into a System Studio SDS model

Setting Up Your Environment

Before you can import COSSAP files into System Studio, you must ensure that the following two COSSAP variables have been set before starting System Studio:

- `COSSAP_PROJECT`, which specifies the name of your current COSSAP project
- `COSSAP_DIR`, which specifies your COSSAP installation directory

Importing COSSAP Models and Schematics

To import COSSAP models and schematics into System Studio:

1. Choose File > Open to open a workspace that will contain the converted design.
2. Choose File > Import to open the Import dialog box, as shown in Figure 9-1.
3. In the Import dialog box, select the symbol file corresponding to the COSSAP model (hierarchical or primitive model), or select the schematic file corresponding to the COSSAP model.

To select the file to be imported, proceed as follows:

1. In the Source directory tree, select the directory containing the symbol or schematic file.
2. In the File List, select “COSSAP Files” to display all the symbols or schematics contained in the selected directory.
3. Select the symbol or schematic file from the files displayed in the File Type list (center part of the import dialog box).
4. In the Destination Library field, select the library in which you want the converted top-level model to be located.

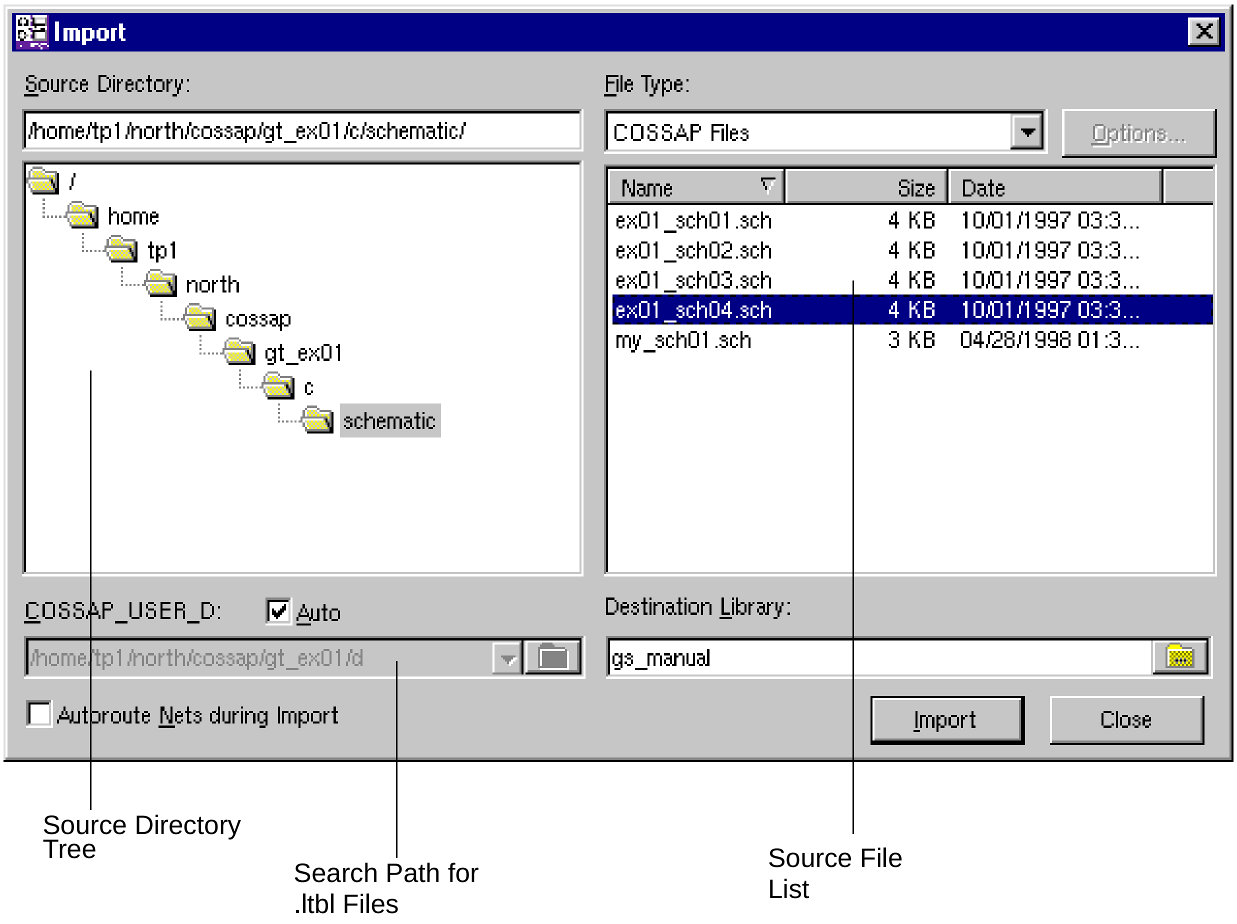
   If you do not want to use the default COSSAP development directory location, you can change the location by editing the entry in the COSSAP_USER_D field, shown below the Source directory tree.

5. To autoroute the nets while importing the COSSAP schematic, check the selection box.

   Due to differences between native COSSAP designs and System Studio designs, it may not be possible to automatically route the nets in the COSSAP design during import. If problems occur, deselect autorouting and try importing the design again. Note, however, that after you have imported such a design it may still be difficult to route the nets manually.

6. Click the Import button to import the model.

Figure 9-1 Importing a COSSAP File



Import

Source Directory: /home/tp1/north/cossap/gt_ex01/c/schematic/

File Type: COSSAP Files

Name	Size	Date
ex01_sch01.sch	4 KB	10/01/1997 03:3...
ex01_sch02.sch	4 KB	10/01/1997 03:3...
ex01_sch03.sch	4 KB	10/01/1997 03:3...
ex01_sch04.sch	4 KB	10/01/1997 03:3...
my_sch01.sch	3 KB	04/28/1998 01:3...

COSSAP_USER_D: [✓ Auto] /home/tp1/north/cossap/gt_ex01/d

Destination Library: gs_manual

Autoroute Nets during Import

Note:

When you import a COSSAP model or schematic, a new top-level node is always created. However, lower-level models are not overwritten if they have already been imported. To overwrite a specific model, you must import it as a top-level node.

Importing a COSSAP Assignment File

To import a COSSAP assignment file:

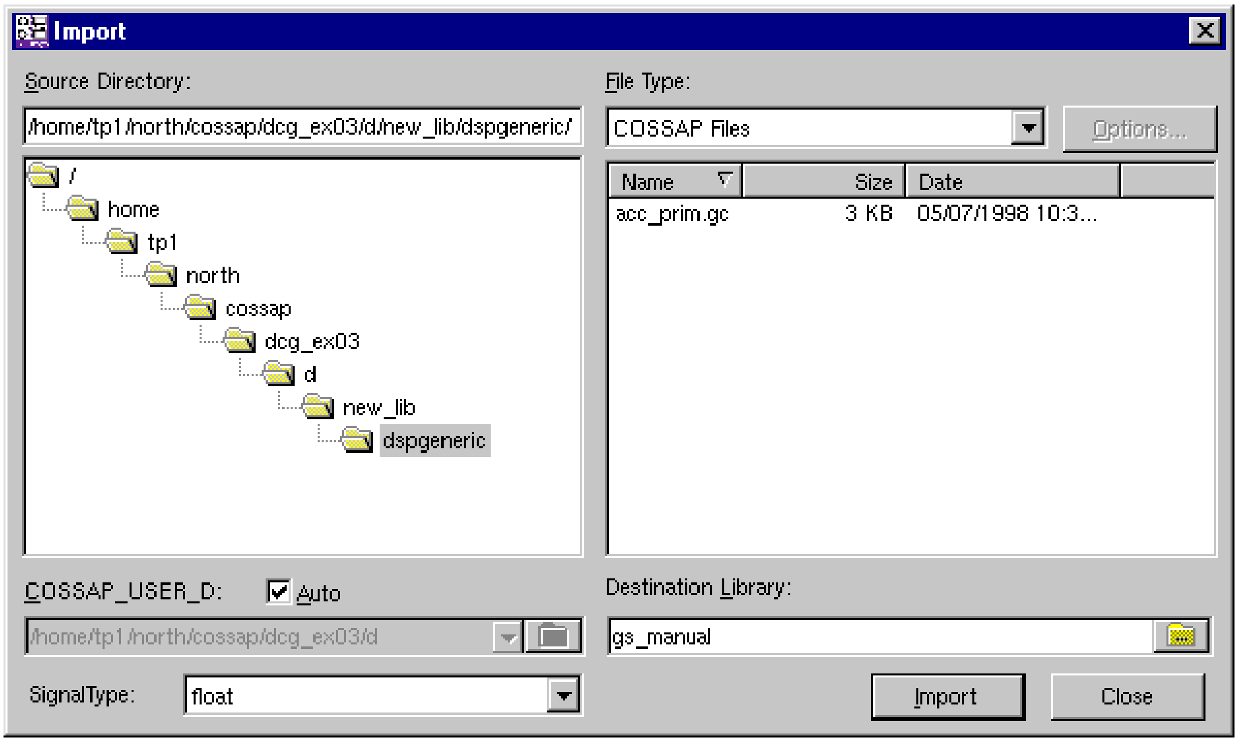
1. Choose File › Open to open a workspace that will contain the converted design.
2. Select File › Import to open the Import dialog box.
3. In the Source directory tree (left side of the Import dialog box), select the directory containing the assignment file.
4. In the File Type part (center of the Import dialog box), select “COSSAP Files” to display all the assignment files contained in the selected directory.
5. Select the assignment file from the files displayed in the File Type list.
6. In the Destination Library field, select the library in which to write the System Studio simulation control file.
7. Click the Import button.

Importing a COSSAP Generic C (.gc) File

To import a COSSAP generic C (.gc) file:

1. Choose File › Open to open a workspace that will contain the converted design.
2. Choose File › Import to open the Import dialog box as shown in Figure 9-2).

Figure 9-2 Importing a COSSAP GC File



Source Directory: /home/tp1/north/cossap/dcg_ex03/d/new_lib/dspgeneric/

```
/
  home
    tp1
      north
        cossap
          dcg_ex03
            d
              new_lib
                dspgeneric
```

File Type: COSSAP Files

Name	Size	Date
acc_prim.gc	3 KB	05/07/1998 10:3...

COSSAP_USER_D: [✓ Auto] /home/tp1/north/cossap/dcg_ex03/d

Destination Library: gs_manual

SignalType: float

3. In the Source directory tree (left side of the Import dialog box), select the directory containing the generic C file.
4. In the File Type list (center of the import dialog box), select “COSSAP Files” to display all the COSSAP files contained in the selected directory.
5. Select the generic C file from the files displayed in the File Type list.
6. Select the Signal Type according to the contents of the GC code: select “float” for GC code that contains `float` signal types, select “int” for GC code that contains `long` signal types.
7. In the Destination Library field, select the library in which you want the generated System Studio model to be located.
8. Click the Import button.

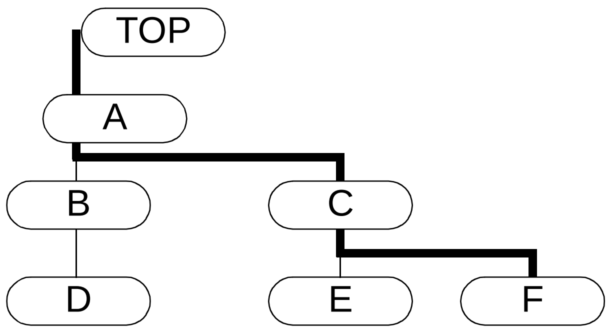
Glossary

ancestor

An ancestor is an instance from which the parent instance of the current instance is descended.

Instance A is an ancestor of another instance F when the path from the top level of the design down to F goes through A, as shown in Figure G-1. If this relationship is satisfied, F is a descendant of A.

Figure G-1 Ancestors and Descendants



AND model

An AND model is a hierarchical model in which all the instances execute in parallel lockstep with synchronous-reactive communication between them.

atomic state

An atomic state is the most primitive component of a control model. It can be empty, as a state of an OR model is, or it can have an action associated with it that is executed on each execution step that the state is active.

const

A const is a static constant parameter. These parameters are not visible from the outside and must have a specified value. They are equivalent to `const` variables in C++.

In control models, local signals and ports can be used in parameter binding expressions.

In dataflow models, nets and ports cannot be used in parameter binding expressions because this makes the parameter a “pseudo port” and makes scheduling very difficult.

You do not have to specify the class of a parameter. The real class of a parameter that is left unspecified is inferred by analyzing the expression bound to it. Optionally you can force a parameter to be either structural, read_on_reset, dynamic or a const. If a parameter class is explicitly specified, it will be taken as a constraint and a check will be made during design elaboration to ensure that the constraint is satisfied. For example, if a parameter is specified as structural and if the expression bound to the parameter does not resolve to a constant, it will be flagged as an error. On the other hand, it is legal to bind a constant expression to a read_on_reset or dynamic parameter. The code generator will optimize this to a structural case. All parameters can be given a default value.

control model

Control model is a term used to describe all hierarchical models other than dataflow graphs.

DDK or DSP Developer Kit

A DDK is a mechanism permitting the execution and debugging of assembly programs for DSP processors or cores in the System Studio environment. System Studio sees the DDK as a primitive dataflow block.

DFG model or dataflow model

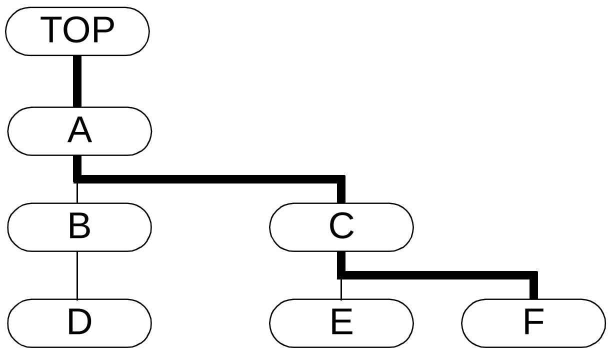
A dataflow graph (DFG) model or dataflow model is a hierarchical model in which the constituent instances communicate by means of FIFO queues of data travelling on nets.

descendant

A descendant is an instance that is directly descended from another instance.

An instance F is a descendant of another instance A if and only if the path from the top level of the design down to F goes through A, as shown in Figure G-2. If this relationship is satisfied, A is an ancestor of F.

Figure G-2 Descendants



dynamic data flow or DDF

Dynamic data flow is the most general form of data flow, in which the flow rates into or out of constituent elements is not predictable. A runtime simulation kernel is needed to implement it.

dynamic parameter
A dynamic parameter is a parameter whose value is refreshed (read from memory) every time it is used and hence it is possible to poke its value from the surrounding environment.

If a dynamic parameter is used in an expression that is bound to another parameter that is not marked dynamic, it will behave like a read_on_reset parameter and the values will be sampled only when the corresponding model is reset.

exiting; to exit
A model exits when it voluntarily ends its own execution.

exit condition
An exit condition is a condition in a model that, when true, causes the model to exit.

exit-handling transition
In an OR state, an exit-handling transition specifies what state to change to when an instance exits. Like other transitions, there may be a condition and an action, though calling an exit-handling transition immediate is meaningless.

hierarchical model
A hierarchical model is a model that contains instances of other models.

GATED model
A GATED model is a hierarchical model with one or two instances and a gating condition. When the gating condition is true, the first instance executes and the second is suspended. When the gating condition is false, the second instance (if it exists) is executed and the first is suspended.

immediate, immediate transition
If a transition is marked as immediate, it is enabled and can occur at the point its source state is entered, as well as at subsequent times. If a state has no outgoing immediate transitions, it is always the

active state for at least one execution step; however, multiple immediate transitions can be taken in no time (thus cycles of immediate transitions are forbidden).

instance

An instance is an instantiation of a model: a particular occurrence of a model at some position in a design. The distinction between a model and an instance is essentially the same as the distinction between a class and an object of that class.

Mealy machine

In hardware design, a Mealy machine is a finite state machine in which the outputs depend on both the inputs and the states. An OR model in which the instances are atomic states and the actions on transitions only set the outputs is a Mealy machine.

model

A model is a design in the System Studio environment. There may be many instances of a given model.

Moore machine

In hardware design, a Moore machine is a finite state machine in which the outputs depend only on the states. An OR model in which the instances have inline actions and there are no actions on transitions is a Moore machine.

net

In a dataflow graph model, a net is an interconnection between the ports of instances. A net must be connected to one output port and may be connected to any number of input ports.

OR model

An OR model is a control-oriented hierarchical model; it specifies a state transition diagram in which its contained instances play the role of states. Only one instance of the model is active at a time (except for boundary cases where one model is exiting as another one is starting).

parameter
A symbolic name associated with a model that may be bound to an expression composed of symbols defined in the parent model.

Parameters enable the creation of generic models (or templates) and the values of the parameters can be specified when the model is instantiated (configured). An System Studio parameter can belong to one of five classes: structural, read_on_reset, dynamic, const, or “unspecified” (this is the default).

parent
A parent is an instance from which the current instance is immediately descended. A model or instance A is the parent of an instance B if and only if B is directly contained in A.

port
A port is a connection between a model and an external signal or stream. In addition to a data type, ports have a mode which is one of (input, output, inout). A dataflow model cannot have inout ports.

All models (except perhaps at the top level) have one or more ports.

PRIM model
A PRIM (or primitive) model is also called a primitive dataflow model. This model type steps through a fixed circular sequence of states, reading inputs, computing, updating internal state and producing outputs. Intended for use in dataflow graph models, it can also be used in other hierarchical models if it is uniform-rate.

pure signal
A pure signal is a local signal or port of Boolean type. Unlike signals of other types, which must be assigned exactly once at any time step, pure signals may have multiple sources (for example, can be driven from both sides of an AND model at the same time; the values are combined with an OR operation). Also, pure signals do not latch; if a value is not set, its value is false.

read_on_reset parameter

A read_on_reset parameter is a parameter whose value is refreshed (read from memory) every time the corresponding model is reset. This type of parameter can be bound to expressions involving signals at the control - dataflow boundary enabling a control model to control the parameters of a dataflow model.

shared instance

A shared instance is an instance that is treated as if it belongs to more than one hierarchical model. The true parent of a shared instance is always an OR model.

SDS, SDS model

An SDS model is a primitive dataflow model that runs under the COSSAP Stream Driven Simulator (a dynamic dataflow simulation kernel). System Studio can produce simulations that combine COSSAP Stream Driven Simulator models with “native” System Studio models.

signal

A signal is a local storage element that can be shared among parallel branches and that can be assigned only one value at a given execution step.

Signals are the communication medium between concurrent objects in the control flow domain. Signals can be thought of as <event, value> tuples. The event1 is the result of an object emitting a value that is consistent with the data type of the signal. A pure (or Boolean) signal is either present or absent and has no value associated with it. Signals are broadcast instantaneously and pure signals have wired-or semantics when more than one object is writing to them.

For non-Boolean signals, System Studio does not have an associated Boolean signal that indicates whether that signal was emitted in a tick. You will have to explicitly use a Boolean signal if you want to test for the presence of a non-Boolean signal.

state
In the context of OR models, member instances are often referred to as states because the model is essentially a state transition diagram.

strong termination
When a model is subject to strong termination, it performs no action in the execution step in which it is terminated (this can be compared to a UNIX `kill -9` command). It is a logical contradiction for a signal produced by a model to cause a strong termination (the effect cancels the cause), and this is treated as an error.

structural parameter
A structural parameter is a parameter that resolves to a constant value at the end of design elaboration; that is, one that is bound to expressions involving constants or other structural parameters.

suspension, to suspend
When an instance is suspended, its execution is frozen; it holds its internal state until the suspension ends. This resembles gating the clock of a piece of hardware or suspending a UNIX process by pressing Control-z.

synchronous data flow
Synchronous data flow is a form of data flow in which the data flow rate of the constituent elements (the number of values read by input ports and written by output ports) is fixed and known by the tool (it may depend on parameters).

synchronous-reactive
Synchronous-reactive is a model of computation in which entities compute their reaction to their inputs instantaneously; there is broadcast communication and a global notion of simultaneity. Instantaneous cyclic communication is permitted as long as there is a unique fixed point.

termination; to terminate

Termination is the ending of a model’s execution by its environment (an ancestor). A model cannot prevent this. Whenever a model is terminated, all of its descendants are terminated as well.

transition

A transition is a path between two instances of an OR state, between the start symbol and an instance, or between an instance and an exit state. It may be annotated with a condition, an action, a priority, and/or an “immediate” attribute. Transitions come in three flavors, indicating strong termination, weak termination, and exit-handling.

uniform rate

A model or instance is said to be uniform rate if on each execution step, it reads one value from each input and writes one value to each output, when used inside a dataflow graph model. All control models are uniform rate. Dataflow models may or may not be uniform rate, and the answer may depend on parameter values.

update instance

If you change the interface of a model (for example, if you delete part of it) that is instantiated in a hierarchical model, the model in which that model is instantiated will be updated only after you invoke “Update instance.”

variable

A variable is a local storage element that has sequential assignment semantics, like a variable in a conventional programming language. Variables cannot be shared by AND models and are local to OR models or dataflow models.

VSI or VHDL/Verilog Simulation Interface

VSI is an interface to a HDL simulator that looks like a primitive dataflow block to System Studio and permits a fairly arbitrary VHDL or Verilog code to be cosimulated with a system-level simulation.

Index