Agilent Ptolemy Simulation

December 2003
Notice

The information contained in this document is subject to change without notice.

Agilent Technologies makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Agilent Technologies shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Warranty

A copy of the specific warranty terms that apply to this software product is available upon request from your Agilent Technologies representative.

Restricted Rights Legend

Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 for DoD agencies, and subparagraphs (c) (1) and (c) (2) of the Commercial Computer Software Restricted Rights clause at FAR 52.227-19 for other agencies.

Agilent Technologies
395 Page Mill Road
Palo Alto, CA 94304 U.S.A.


Acknowledgments

Mentor Graphics is a trademark of Mentor Graphics Corporation in the U.S. and other countries.

Microsoft®, Windows®, MS Windows®, Windows NT®, and MS-DOS® are U.S. registered trademarks of Microsoft Corporation.

Pentium® is a U.S. registered trademark of Intel Corporation.

PostScript® and Acrobat® are trademarks of Adobe Systems Incorporated.

UNIX® is a registered trademark of the Open Group.

Java™ is a U.S. trademark of Sun Microsystems, Inc.
Portions of the documentation

Copyright © 1990-1996 The Regents of the University of California. All rights reserved.

In no event shall the University of California be liable to any party for direct, indirect, special, incidental, or consequential damages arising out of the use of this software and its documentation, even if the University of California has been advised of the possibility of such damage. The University of California specifically disclaims any warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The software provided hereunder is on an “as is” basis and the University of California has no obligation to provide maintenance, support, updates, enhancements, or modifications.
Contents

1 Overview of Agilent Ptolemy
   Agilent Ptolemy and UC Berkeley Ptolemy ............................................................... 1-2
   Timed Synchronous Dataflow Simulator .............................................................. 1-2
   Terminology .............................................................................................................. 1-3

2 Integrator Example
   Copying and Opening Example Project .............................................................. 2-1
      Copy the Project ................................................................................................. 2-1
      Open the Project ............................................................................................... 2-2
   Selecting and Placing Components .................................................................... 2-4
      Add a Source ...................................................................................................... 2-4
      Add an Output Display ...................................................................................... 2-5
      Modify Component Parameters ........................................................................ 2-5
      Connect Components with Wires ...................................................................... 2-6
      Add a Controller ............................................................................................... 2-6
   Starting Simulation ................................................................................................... 2-8
      Simulate and Display Data Directly .................................................................... 2-8
      Simulate and Save Data ..................................................................................... 2-9

3 Data, Controllers, Sinks, and Components
   Representation of Data Types .................................................................................. 3-1
      Single Versus Multiple Arrowheads .................................................................... 3-2
   Automatic or Manual Data Type Conversion ............................................................. 3-3
      What Happens During Conversion? ................................................................... 3-4
   Controllers ................................................................................................................ 3-6
      DF Controller ...................................................................................................... 3-6
      OutputOption Controller ..................................................................................... 3-15
      WTB Controller ................................................................................................... 3-16
   Sources and Sinks Control the Simulation ............................................................... 3-17
   Agilent Ptolemy Components ................................................................................... 3-20

4 Understanding Parameters
   Value Types .............................................................................................................. 4-1
   Parameter Editing ..................................................................................................... 4-7
   Parameter Expressions ............................................................................................ 4-9
   Complex-Valued Parameters .................................................................................... 4-9
   Parameters for Fixed-Point Components .................................................................. 4-10
   String Parameters ..................................................................................................... 4-16
   Filename Parameters ................................................................................................ 4-16
   Array Parameters ...................................................................................................... 4-17
   Reading Array Parameter Values From Files ............................................................ 4-18
5 Using Data Types

Representation of Data Types ................................................................. 5-1
  Stem Thickness ....................................................................................... 5-2
  Single and Multiple Arrowheads ............................................................. 5-3

Data Types Defined .................................................................................. 5-3
  Numeric Scalar Data ............................................................................... 5-4
  Numeric Matrix Data ............................................................................... 5-4
  Timed Data .............................................................................................. 5-5

Conversion of Data Types ....................................................................... 5-6
  What Happens During Conversion? ........................................................... 5-6
  Numeric Scalar and Matrix Conversions ................................................ 5-6
  Timed Data Conversions ......................................................................... 5-7
  Rules and Exceptions ............................................................................. 5-9
  Automatic or Manual Data Type Conversion ......................................... 5-10

6 Understanding File Formats

  Real Array Data ....................................................................................... 6-1
  Complex Array Data ................................................................................ 6-1
  String Array Data .................................................................................... 6-1
  Real Matrix Data ..................................................................................... 6-2
  Fixed-Point Matrix Data ......................................................................... 6-2
  Integer Matrix Data ................................................................................ 6-2
  Complex Matrix Data .............................................................................. 6-3

  SPW (.ascsig and .sig) File Formats ......................................................... 6-3
  Time-Domain Waveform Data (.tim) File, MDIF ASCII Format ............... 6-5
  Time-Domain Waveform Data (.bintim) File in Binary Format ............... 6-7
  Agilent Standard Data Format (.dat) Files .............................................. 6-7

7 Performing Parameter Sweeps

  Introduction ............................................................................................ 7-1
  Sweeping Parameters Using the ParamSweep Component ........................ 7-2
    Procedure ............................................................................................. 7-3
  Sweeping Parameters Using the VAR Component ..................................... 7-5
    Procedure ............................................................................................. 7-5
  Using a Sweep Plan ................................................................................ 7-6
    Procedure ............................................................................................. 7-6
  Sweeping Various Parameter Types ......................................................... 7-10
    Sweeping a Complex Waveform Component ....................................... 7-10
    Performing Multidimensional Sweeps ................................................. 7-13

8 Using Nominal Optimization

  Optimizing Various Parameter Types ..................................................... 8-2
9 Theory of Operation
Synchronous Dataflow (SDF) ................................................................. 9-2
Basic Dataflow Terminology .............................................................. 9-3
Balancing Production and Consumption of Tokens ......................... 9-3
How Schedulers Work in Agilent Ptolemy ................................. 9-5
Iterations in SDF ................................................................. 9-6
Inconsistency ............................................................................. 9-7
Deadlocks .................................................................................. 9-7
Deadlock Resolution .................................................................. 9-8
Timed Synchronous Dataflow (TSDF) .................................................. 9-9
Time Step Resolution ................................................................ 9-10
Carrier Frequency Resolution ..................................................... 9-11
Input/Output Resistance ............................................................. 9-11
Multithreaded Synchronous Dataflow .................................................. 9-13
Parallelism with Clustering ......................................................... 9-13
Memory Overhead ....................................................................... 9-14
Sample Results ........................................................................ 9-15
Reentrancy ............................................................................... 9-16
Thread Safe Programming ............................................................. 9-17
References ............................................................................. 9-18

10 Introduction to MATLAB Cosimulation
Setting Up MATLAB ................................................................. 10-1
Simulating with MATLAB (Script-Interpreting) .................... 10-3
Writing Functions for the MATLAB Models (Script-Interpreting) 10-4
Simulating with MATLAB (Library-Importing) ......................... 10-5
Examples ............................................................................... 10-6

11 Cosimulation with Analog/RF Systems
Setting Up the Analog/RF Circuit Schematic ................................. 11-3
Setting Up the Signal Processing Schematic .................................. 11-4
Circuit Simulation Controllers ..................................................... 11-5
Numeric-to-Timed Converters ......................................................... 11-5
Enabling Automatic Verification Modeling (Fast Cosimulation) .................................................. 11-5
Clustering of Circuit Subnetworks .............................................. 11-8
Connected Circuit Subnetworks ................................................. 11-9
Connected Resistors .................................................................. 11-9
Feedback Loops ....................................................................................................... 11-10
Named Connections and Measurements in Circuit Designs .................................. 11-11
Circuit Envelope Specific Rules .............................................................................. 11-11
Transient Simulation Specific Rules ....................................................................... 11-12
Nested Simulation Approach .................................................................................. 11-12
  Signal Processing Model of the Circuit Network ............................................... 11-12
  Circuit Model of the Signal Processing Network ................................................. 11-13
Interface Issues ..................................................................................................... 11-13
  Time Step ........................................................................................................... 11-13
  Delays in Feedback Loops .............................................................................. 11-15
  Time Converters ............................................................................................... 11-15
  Carrier Frequency ............................................................................................ 11-15
  EnvOutSelector and EnvOutShort Components .............................................. 11-16
Preparing Wireless Test Bench Designs for RFDE ............................................... 11-17
  Creating a Wireless Test Bench Design ............................................................ 11-18
  Exporting the WTB Design to RFDE ................................................................. 11-19
  WTB Design User Interface Attributes .............................................................. 11-20
  Verifying a WTB Design in ADS ...................................................................... 11-20
  Creating a Results Display for WTB Designs ................................................... 11-21
  Circuit Envelope Parameters ............................................................................ 11-22
  Wireless Test Bench Design Examples ............................................................. 11-23
Troubleshooting Common Problems .................................................................... 11-24
Cosimulation Example .......................................................................................... 11-25
  Copying and Opening the Project .................................................................... 11-25
  Rectifier Schematic ......................................................................................... 11-26

12 Using Interactive Controls and Displays
  List of Interactive Control and Display Components ......................................... 12-3
  TkSlider and TkPlot Components Example ...................................................... 12-4
    Placing Multiple TkSlider Components ......................................................... 12-7
  TkText and TkShowValues ............................................................................... 12-8
  TkXYPlot Component ....................................................................................... 12-9
  TkBarGraph ....................................................................................................... 12-11
  LMS Adaptive Filter Components .................................................................... 12-12
  TkButtons .......................................................................................................... 12-14
  TkBreakPt .......................................................................................................... 12-14
  TkMeter ............................................................................................................. 12-15
  TkShowBooleans .............................................................................................. 12-15
  Baseband Equivalent Channel ......................................................................... 12-16
  Invoke Tcl Script ............................................................................................. 12-17
  TkEye, TkConstellation, TkHistogram, TkIqRms, and TkPower .......................... 12-17

Index
Chapter 1: Overview of Agilent Ptolemy

Part of Advanced Design System, Agilent Ptolemy software gives you the simulation tools you need to evaluate and design modern communication systems products. Today's designs call for implementing DSP algorithms in an increasing number of portions in the total communications system path, from baseband processing, to adaptive equalizers and phase-locked loops in the RF chain. Agilent Ptolemy software gives you the most complete tool set available to do your job—including cosimulating with the Advanced Design System's RF and analog simulators from the same schematic.

Agilent Ptolemy Major Benefits

The Agilent Ptolemy simulator allows you to:

- Find the best design topology using state-of-the-art technology with over 500 behavioral DSP and communication systems models
- Cosimulate with RF and analog simulators
- Integrate your intellectual property from previous designs
- Reduce the time-to-market for your products

Agilent Ptolemy Major Features

- Timed Synchronous Dataflow simulation
- Cosimulation capability with RF and analog simulators
- Easy-to-use interface for adding and sharing your own custom models
- Interface to test instruments
- Data Display with post-processing capability
- Integration with ADS DSP Synthesis
Agilent Ptolemy and UC Berkeley Ptolemy

The Ptolemy signal processing simulator has its roots at the University of California at Berkeley. UC Berkeley Ptolemy is a third-generation software environment that began in January of 1990. It is an outgrowth of two previous generations of design environments, Blosim and Gabriel, that were aimed at digital signal processing. Both environments use dataflow semantics with block-diagram syntax for the description of algorithms.

Agilent EEsof has built on the UC Berkeley Ptolemy code in developing Agilent Ptolemy software. We have added a large number of behavioral, time-domain models, which are critical to communication systems designers. These include antenna and propagation models. For DSP designers, we have upgraded fixed point analysis to be scalable up to 256 bits. Agilent Ptolemy software runs under Advanced Design System's intuitive user interface, which includes post-processing capability, cosimulation with analog/RF simulators, links to test instruments, online help, and a host of other features.

In Ptolemy, different specialized design environments are called domains. Agilent Ptolemy has modified the proven Synchronous Dataflow domain to include timed components. This domain is called the Timed Synchronous Dataflow domain.

Timed Synchronous Dataflow Simulator

The Timed Synchronous Dataflow domain captures years of Agilent EEsof expertise in system-level analog/RF simulation, while adding the benefits of dataflow technology. This domain enables fast RF simulation, integration with signal processing simulation, and cosimulation with Agilent EEsof circuit simulators. For more information on the Timed Synchronous Dataflow simulator and the Synchronous Dataflow domain, refer to Chapter 9, Theory of Operation.
Terminology

Throughout most of the Agilent Ptolemy documentation, we use the Advanced Design System terminology, which is standard EDA terminology. However, UC Berkeley Ptolemy has its own terminology and for users familiar with this terminology, or those who are writing their own models, the following table compares the terms. The UC Berkeley Ptolemy terminology is used only in Chapter 9, Theory of Operation and in the chapters on building signal processing models found in the User-Defined Models documentation.

Table 1-1. Terminology Comparison

<table>
<thead>
<tr>
<th>Agilent Ptolemy</th>
<th>UC Berkeley Ptolemy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>Star</td>
</tr>
<tr>
<td>Network (or circuit)</td>
<td>Galaxy</td>
</tr>
<tr>
<td>Top-level System</td>
<td>Universe</td>
</tr>
<tr>
<td>Controller</td>
<td>Target</td>
</tr>
<tr>
<td>Wire</td>
<td>Arc</td>
</tr>
<tr>
<td>Data (or signals)</td>
<td>Particles (or tokens)</td>
</tr>
</tbody>
</table>

$HPEESOF_DIR

In UNIX installations, the environment variable specifying the directory in which the Advanced Design System software is installed. In Windows installations, the syntax, when needed, is %HPEESOF_DIR%.

actor

An atomic (indivisible) function in a dataflow model of computation. An actor is called a component in Agilent Ptolemy and a star in UCB Ptolemy.

arc

A wire that connects the output of one star or component with the input of another.

base class

A C++ object used to define common interfaces and common code for a set of derived classes. An object may be a base class and a derived class simultaneously.
Overview of Agilent Ptolemy

**behavioral modeling**

System modeling consisting of functional specification plus modeling of the timing of an implementation (compare to functional modeling).

**Block**

The base class defined in the kernel for stars, galaxies, universes, and targets.

**block**

A star or a galaxy.

**compile-time scheduling**

A scheduling policy in which the order of block execution is pre-computed when the execution is started. The execution of the blocks thus involves only sequencing through this pre-computed order one or more times (compare to run-time scheduling).

**derived class**

A C++ object derived from some base class. It inherits all of the members and methods of the base class.

**dataflow**

A model of computation in which actors process streams of tokens. Each actor has one or more firing rules. Actors that are enabled by a firing rule may fire in any order.

**domain**

A specific implementation of a computation model.

**Domain**

The base class in the Agilent Ptolemy kernel from which all domains are derived.

**drag**

The action of holding a mouse button while moving the mouse.

**FFT**

The Fast Fourier Transform (FFT) is an efficient way to implement the discrete Fourier transform in digital hardware.

**firing**

A unit invocation of an actor in a dataflow model of computation.
firing rule
A rule that specifies how many tokens are required on each input of a dataflow actor for that actor to be enabled for firing.

fork star
A star that reads one input particle and replicates it on any number of outputs.

functional modeling
System modeling that specifies input/output behavior without specifying timing (compare to behavioral modeling).

galaxy
A block that contains a network of other blocks.

Gantt chart
A graphical display of a parallel schedule of tasks. In Agilent Ptolemy, the tasks are the firings of stars and galaxies.

homogeneous synchronous dataflow
A particular case of the synchronous dataflow model of computation, where actors produce and consume exactly one token on each input and output.

hpeesoftlang
(1) A schema language used to define stars in Agilent Ptolemy. (2) The program that translates stars written in the hpeesoftlang language to C++. In UCB Ptolemy, the equivalent language is called ptlang.

iteration
A set of executions of blocks that constitutes one pass through the pre-computed order of a compile-time schedule.

kernel
The set of classes defined in the Agilent Ptolemy kernel.

layer
In the Schematic, a color with a given precedence. Colors with higher precedence will obscure colors with lower precedence.

member
A C++ object that forms a portion of another object.
Overview of Agilent Ptolemy

**method**
A function defined to be part of an object in C++.

**model of computation**
A set of semantic rules defining the behavior of a network of blocks.

**net**
A graphical connection between ports in the schematic.

**object**
A data type in C++ consisting of members and methods. These members and methods may be private, protected, or public. If they are private, they can only be accessed by methods defined in the object. If they are protected, they can also be accessed by methods in derived classes. If they are public, they can be accessed by any C++ code.

**palette**
A schematic area that contains a library of block icons.

**parameter**
The initial value of a state.

**particle**
Data (for example, a floating-point value) communicated between blocks.

**port**
A star or galaxy input or output.

**PortHole**
The base class in the Agilent Ptolemy kernel for all ports.

**Ptolemy**
A design environment that supports simultaneous mixtures of different computation models. Ptolemy, named after the second-century Greek astronomer, mathematician, and geographer, was developed at the University of California at Berkeley.

**real time**
The actual time (compare to simulated time).
**RTL**

Register-transfer level description of digital systems. This kind of description is used by DSP Synthesis.

**run-time scheduling**

A scheduling policy in which the order of block execution is determined “on-the-fly,” as they are executed (compare to compile-time scheduling).

**Scheduler**

An object associated with a domain that determines the order of block execution within the domain. Domains may have multiple schedulers.

**schematic**

A block diagram.

**SDF**

A simulation domain using the synchronous dataflow model of computation.

**simulated time**

In a simulation domain, the real number representing time in the simulated system (compare to real time).

**simulation**

The execution of a system specification (an Agilent Ptolemy block diagram) from within the Agilent Ptolemy process (that is, execution without generating code and spawning a new process to execute that code).

**simulation domain**

A domain that supports simulation, but not code generation.

**star**

A component in Agilent Ptolemy. An atomic (indivisible) unit of computation in an Agilent Ptolemy application. Every Agilent Ptolemy simulation ultimately consists of executing the methods of the stars used to define the simulation.

**Star**

The base class in the Agilent Ptolemy kernel for all stars.
Overview of Agilent Ptolemy

state
A member of a block that stores data values from one invocation of the block to the next.

State
The base class in the Agilent Ptolemy kernel for all states.

stop time
Within a timed domain, the time at which a simulation halts.

symbol
A graphical object that represents a single block.

synchronous dataflow
A dataflow model of computation where the firing rules are particularly simple. Every input of every actor requires a fixed, pre-specified number of tokens for the actor to fire. Moreover, when the actor fires, a fixed, pre-specified number of tokens is produced on each output. This model of computation is particularly well-suited to compile-time scheduling.

target
An object that manages the execution of a simulation or code generation process. In Agilent Ptolemy this is called a controller. For example, in code generation, the target would be responsible for compiling the generated code and spawning the process to execute that code.

Target
The base class in the kernel for all targets.

Tcl
Tool command language—a textual, interpreted language developed by John Ousterhout at UC Berkeley. Tcl is embedded in Agilent Ptolemy.

timestamp
A real number associated with a particle in timed domains that indicates the point in simulated time at which the particle is valid.

timed domain
A domain that models the evolution of a system in time.
Tk
A Windows and X-Windows toolkit for Tcl. The interactive sliders, buttons, and plotting capabilities of Agilent Ptolemy are implemented in Tcl/Tk.

token
A unit of data in a dataflow model of computation. Tokens are implemented as particles in Agilent Ptolemy.

universe
An entire Agilent Ptolemy design.

VHDL
The VHSIC hardware description language, a standardized language for specifying hardware designs at multiple levels of abstraction.

wormhole
A star in a particular domain that internally contains a galaxy in another domain.
Overview of Agilent Ptolemy
Chapter 2: Integrator Example

This chapter is designed for the new user. If you know how to use Advanced Design System for Analog/RF Systems design, you can read quickly through this chapter noting the differences to Agilent Ptolemy, such as the use of sinks, Data Flow controller, and Interactive Controls and Displays components.

To learn how to use Agilent Ptolemy, let's load a simple integrator example. We will add a source, an output display item, and a controller, then simulate and view the results.

Copying and Opening Example Project

First, we will copy an example project.

Copy the Project

1. From the Main window, choose File > Copy Project. A dialog box appears.

   ![Copy Project Dialog](image)

   **Note** On UNIX platforms, you do not generally have permission to work in the Advanced Design System Example directories. You must copy the example project to a directory for which you have write permission. On PC platforms, while you can work in the Example directories if you want, it's better to copy the examples to another directory.
Integrator Example

2. In the From Project field, click the Examples Directory button, and then the Browse button. The File Browse dialog box appears with the Example directories listed.

3. Scroll to the bottom of the Directories list and select the /Tutorial directory.

4. Select integrator_prj from the list of files in the Files field.

5. In the To Project field, click the Startup Directory or Working Directory button (depending on where you want to copy the project to) or choose the Browse button to select another directory.

6. Choose Copy Project Hierarchy. This ensures that all the appropriate directories and files will be copied.

7. Click OK to copy the project and close the dialog box.

Open the Project

1. From the Main window, choose File > Open Project. When the Open Project dialog box appears, select <the directory you copied the example to> in the Directories field, then double-click integrator_prj in the Files field. The project will appear in the File Browser field of the Main window.

2. Under the integrator_prj project directory, click the Networks subdirectory to open the various schematics within this project. These will all have the extension .dsn.

3. Double-click integrator1.dsn. In the Design Information field at the right, one item appears.

4. Double-click integrator1 (Schematic) to open the design. The schematic appears:
Figure 2-1. Integrator Design
Selecting and Placing Components

We will add a sine wave source, an output display item, and a controller to the integrator schematic. There are two ways to choose library components:

- **From a Palette List**  The far left side of the window contains a palette list and icons for each item in the palette. You choose items by first selecting a palette and then clicking on the icons in the palette.

- **From a Library**  You can also select components by choosing Insert > Component > Component Library. A dialog box appears that displays components in each component library. You select items from the list.

Add a Source

1. We will use the Palette List method first (left side of the window). Since the component we want is in the default library, called Common Components, simply click the SinGen icon (near bottom of list). Crosshairs and a ghost image of the component appear as you move the pointer over the design window.

2. Move the crosshairs to the upper left part of the schematic (to the left of the Fork2 component), then click once. A schematic representation of the source component is placed in the design window. Beneath the schematic is block representing the component's name and editable parameters. We will accept the default values.

3. If you are finished placing components, click the **End Command** arrow on the toolbar, or press **Escape**. The crosshairs disappear. If you are continuing to place components, as in this example, you can skip this step.

---

**Note**  If you continue to click without deselecting, you will place a new component with each click.
Add an Output Display

We will continue by adding the output display item. But this time we will use the Library method of selecting components.

1. Choose Insert > Component > Component Library. A dialog box appears that displays components in each component library. From the Libraries list box, select Interactive Controls and Displays (resize the dialog box to show long names).

2. From the right side, select TkPlot. Crosshairs and a ghost image of the component appear as you move the pointer over the design window. (Another TkPlot item is already in the schematic to display the input signal.)

3. Move the crosshairs to the upper right part of the schematic (to the right of the Fork2 component), then click once. A schematic representation of the TkPlot display component is placed in the design window. Beneath the schematic is an options block representing the component’s name and editable parameters.

4. As explained before, click the End Command arrow on the toolbar, or press Escape. The crosshairs disappear.

5. Close the Component Library dialog box by choosing OK.

Modify Component Parameters

We will modify two parameters for this item. There are several ways to edit parameters, including:

- Double-click the top of the component.
- Choose Edit > Component > Edit Component Parameters.
- Click the Edit Component Parameters button on the toolbar.
- Type the parameter value directly on the schematic page. The text changes color. Then edit the value and press Return at the right of the new value. Pressing Return also takes you through subsequent parameters.

We will use the dialog box method.

1. Double-click the TkPlot item. A dialog box appears.

2. Select the xRange parameter (left side). On the right side, backspace over the 100 and type 400.
3. Select the yRange parameter (left side). On the right side, backspace over the -1.5 1.5 and type 0 32.
4. Similarly, select the Persistence parameter and change the value from 100 to 300.
5. Type Output in the Label field (top of list) so we can keep track of the input and output plots.
6. Choose OK.

Connect Components with Wires

1. Choose Insert > Wire or click the Insert Wire button on the toolbar (bottom row). Connect a wire from the port on the SineGen source to the input port on the Fork2 component. When a port is successfully connected, its color changes from red to blue.
2. Connect a wire from the top port of the Fork2 component to the port on the TkPlot display component.

**Note**  Wires must connect ports in pairs, and you must place at least two components before you can add a wire. You cannot add a wire to a component port first, and then add a second component to that wire.

Add a Controller

Controllers are used to specify the type of simulator you want to use and simulation parameters.

1. From the Palette List under Common Components, select the Data Flow Controller icon (right, near top). Crosshairs and a ghost image of the component appear as you move the pointer over the design window.
2. Move the crosshairs to the lower left part of the schematic, then click once.
3. A schematic representation of the controller component is placed in the design window. Controllers are not connected or wired to other components. We will accept the default values.
4. Click the deselect arrow, or press Escape. The crosshairs disappear.
There are several types of controllers, the one we have chosen is called the Data Flow controller, which is used to run mixed numeric and timed signal processing simulations.

At this point, your example should look similar to Figure 2-2.

Figure 2-2. Integrator Design After Source, Display, and Controller Items Added.
If you have had difficulty building the design, you can select the completed schematic from your directory you copied the example project to earlier. Select integrator1_complete.dsn.
Starting Simulation

Now that we have a completed schematic, we're ready to start a simulation. The Advanced Design System provides flexibility in this task. In our example, we have placed an interactive display item called TkPlot. This item quickly displays the results of your simulation. Later we will substitute a “Sink” item in the schematic that will save the simulation results to a file. We will then use the Data Display to review our results.

Simulate and Display Data Directly

1. Choose Simulate > Simulate. The simulation begins. A status window appears, which gives you information on your simulation or reports errors.

2. Two TkPlot windows appear showing you an animated display of both the sine wave input and the simulation results of the output, as shown below.

![Output](image)

Figure 2-3. Integrator Output Simulation Results

In this simple example, the sine wave source has been changed to a cosine wave (offset by 90 degrees) by the integrator.

The simulation must be stopped manually. Choose Quit from the small dialog box labeled Agilent Ptolemy when you are done reviewing the animated plots.
Simulate and Save Data

Now we will use the alternate approach where we substitute a “Sink” component in the schematic and save the data.

1. Click the output TkPlot item in your schematic to select it.
2. Press the Delete key or choose the Delete (trash can) icon from the toolbar.
4. Place the NumericSink where the TkPlot item was originally.
5. Double-click the NumericSink to edit the sink’s parameters.
6. Accept the Start default of DefaultNumericStart.
7. Select Stop and change the value to 200. Here we show that a sink can override the Data Flow controller’s Stop value. Typically, a sink can be left at its default and you can control simulation from the Data Flow controller.
8. Choose OK.
9. Choose Simulate > Simulation Setup. The Simulation Setup dialog box appears. This step is used when you want to explicitly name a dataset.
10. In the Dataset field, type `myresults`. This becomes the filename of your simulation results. Leave the other defaults as they are.

11. Choose the `Simulate` button (at bottom). The simulation begins. A status window appears, which gives you information on your simulation or reports errors.

This time, your data is saved to disk where it can be used to display results in a variety of formats, or be used in post-processing procedures. In addition, the input TkPlot displays an animated plot for the input. Click `Quit` in the Agilent Ptolemy controller to dismiss this display.


2. From the drop-down list next to Plot Types, select `myresults`. This list is called the Default Dataset list.

3. Click `Rectangular Plot` in the Plot Types palette list, at the left of the window. A ghost rectangular frame appears.
4. Click once to place the frame in the Data Display window. A dialog box appears; follow the instructions in the next figure:

5. Choose OK. Your data is plotted in the Data Display window, as shown in the next figure.
Integrator Example

To resize a plot, use the various zoom buttons in the toolbar, or drag a corner outward. A large variety of graphing, annotation, and post-processing tasks can be done from this window. Giving the data a unique name allows it to be archived as a reference in a suite of simulations.

We have seen two methods for displaying data, both of which start with the placement of an output item in your schematic:

- TkPlot (one of several interactive display items), which does not store data to disk.
- The Data Display window, which takes stored data and displays it in a variety of formats.
Chapter 3: Data, Controllers, Sinks, and Components

Before continuing to use Agilent Ptolemy, let's look at some of the Agilent Ptolemy concepts you may have questions about and introduce the signal processing components that Agilent Ptolemy uses.

Representation of Data Types

Agilent Ptolemy schematics contain component stems with single and double arrowheads as well as stems of different colors and thicknesses. This differs from the Analog/RF Systems schematics in ADS. Table 3-1 describes the data types and their representation in the Advanced Design System. A note on terminology: For some applications, particularly those using timed components, data types can be thought of as signal types. Regardless of the terminology, packets of data are passed from one component to another.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Stem Color</th>
<th>Stem Thickness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalar Fixed Point</td>
<td>Magenta</td>
<td>Thin</td>
</tr>
<tr>
<td>Scalar Floating Point (Real)</td>
<td>Blue</td>
<td>Thin</td>
</tr>
<tr>
<td>Scalar Integer</td>
<td>Orange</td>
<td>Thin</td>
</tr>
<tr>
<td>Scalar Complex</td>
<td>Green</td>
<td>Thin</td>
</tr>
<tr>
<td>Scalar Timed</td>
<td>Black</td>
<td>Thin</td>
</tr>
<tr>
<td>Matrix Fixed Point</td>
<td>Magenta</td>
<td>Thick</td>
</tr>
<tr>
<td>Matrix Floating Point (Real)</td>
<td>Blue</td>
<td>Thick</td>
</tr>
<tr>
<td>Matrix Integer</td>
<td>Orange</td>
<td>Thick</td>
</tr>
<tr>
<td>Matrix Complex</td>
<td>Green</td>
<td>Thick</td>
</tr>
<tr>
<td>Any Type</td>
<td>Red</td>
<td>Thin</td>
</tr>
</tbody>
</table>
Data, Controllers, Sinks, and Components

**Single Versus Multiple Arrowheads**

*Figure 3-1* shows the thicker stem width associated with matrix data (top) compared to the thinner stem width associated with scalar data (bottom).

Components appear on the schematic with single or multiple arrowheads at inputs or outputs.

- Single arrowheads carry only one distinct signal.
- Multiple arrowheads carry more than one distinct signal.

For example, an adder component (Add) has multiple arrowheads on the input pin, but a single output arrowhead.

BusMerge items can be used to connect multiple signals to a component when the signal order must be specified. Similarly, BusSplit items can be used to split signals to multiple outputs.
Automatic or Manual Data Type Conversion

When you connect components of the same data type (color), the data is copied from one component to another. If you connect components represented by different data types, such as scalar complex to scalar floating-point (real), or scalar integer to matrix integer, you need to consider two things about conversion:

Should I place a conversion component in the schematic or let the software automatically do the conversion? What will happen to my data? These questions are answered next.

Although the software will automatically convert dissimilar data types, such as complex to floating-point (real), we recommend that you place an appropriate converter (from the Signal Converters library) in your schematic. This acts as a visual reminder that a conversion is taking place, and also helps you decode error messages that may arise. Automatic conversion means that an appropriate converter is “spliced in” behind the scenes and is not shown on the schematic.

Automatic conversion is allowed among scalar data types and among matrix data types, but not between scalar and matrix data types.

Timed Component Exceptions

For Timed pins, there are two cases when automatic splicing produces an error message:

1. When either a Float (real) to Timed, Fixed to Timed, Integer to Timed, or Complex to Timed converter is placed (or spliced) in the design and there is no time step defined (via sources or other timed converters) in the design. You need to define the time step at least once in your design.

2. When a Complex port is connected to a Timed port. Automatic conversion from Complex to Timed is not supported. You need to place a Complex to Timed converter between the ports and enter appropriate parameters.
Data, Controllers, Sinks, and Components

Conversion Between Scalar and Matrix Types

When a scalar pin is directly connected to a matrix pin (or vice versa), without a Pack or Unpack converter, an error message is generated.

In the Numeric Matrix Library, four converters are used to “pack” scalar data into matrix data, such as Pack_M and PackCx_M. Similarly, four converters “unpack” the data (back to scalar), such as UnPk_M and UnPkCx_M. There is no automatic conversion between scalar and matrix data (or vice versa). You must place the converters where needed in your design.

What Happens During Conversion?

Most conversions do what you expect. For example, when converting from lower precision to higher precision data types, such as integer to floating-point (real), no data is lost; only the format is changed.

When converting from higher precision to lower precision data types, such as floating-point (real) to integer, the outcome is governed by your computer’s math rounding rules, with the following exceptions:

Complex Data Conversions

- Complex to Float (Real) Agilent Ptolemy computes the magnitude and ignores the phase.
- Complex to Fixed After computing the floating-point (real) magnitude, Agilent Ptolemy converts the floating-point (real) to fixed.
- Complex to Integer After computing the floating-point (real) magnitude, Agilent Ptolemy converts the floating-point (real) to integer.
**Timed Data Conversions**

The Timed data type represents the time-domain signal in either carrier-modulated (complex) or real-baseband flavors. The Timed data class members are $I$, $Q$, $F_c$, time, plus an Agilent Ptolemy member called Flavor. Flavor specifies whether the Timed data type is in a carrier-modulated or real-baseband format. When the carrier frequency is not specified (undefined) for a Timed port, an error message is generated.

You can convert between Timed and non-Timed ports by placing one of the following converters and supplying the parameters as needed:

- Timed to Complex or Complex to Timed
- Timed to Float (Real) or Float (Real) to Timed
- Timed to Fixed or Fixed to Timed
- Timed to Integer or Integer to Timed

Time-data conversion depends on the flavor of the Timed data and the carrier frequency.

For more detailed information on conversion of data types, refer to “Conversion of Data Types” on page 5-6.
Controllers

Controllers are used to control simulation, are placed unconnected anywhere in the schematic, and are found in the Controllers library or palette. Use the Data Flow controller to control the flow of mixed numeric and timed signals for all digital signal processing simulations within Advanced Design System. Other controllers are used to set up parameter sweeps, optimization, or statistical design. To set or modify the parameters using a dialog box, double-click the component in the schematic, or choose Edit > Component > Edit Component Parameters.

DF Controller

The main Agilent Ptolemy controller is the Data Flow (DF) controller, and it is required for all simulations. Use the Data Flow controller to control the flow of mixed numeric and timed signals for all digital signal processing simulations within Advanced Design System. This controller, together with the source and sink components, provide you the flexibility to control the duration of simulation globally or locally.

Note  Multiple DF controllers on the schematic are not allowed. Versions of Agilent Ptolemy released before ADS 1.5 allowed multiple Data Flow controllers on the same schematic. Starting with ADS 1.5, this is no longer possible. Multiple controllers were used to simulate the same design with different Data Flow parameters, for example with a different value of DefaultNumericStart. You can achieve the same effect by using single-point sweeps on the parameter you are interested in varying.

The Data Flow controller dialog box shown below has six tabs: Controls, Options, Output, Resistors, Debug, and Display.
The following information will help you understand the controller’s use:
Controls Tab

Agilent Ptolemy sinks have Start and Stop parameters that control when to start and stop data collection. Sinks collect from Start to Stop, inclusively.

In numeric sinks, these numbers are unitless because they represent sample numbers. The first datum that the sink receives is #0, the second is #1, etc. For example, a numeric sink with Start=3 and Stop=4 will skip the first three pieces of data and collect the next two.

In timed sinks, Start and Stop have timed units because the data has a time base. The amount of data that the sink collects is a function of both the data time base and the sink’s Start and Stop parameters. For example, if Start=0 msec, Stop=1 msec, and the data has a time base of 2 usec, the sink will collect 501 pieces of data.

The Controls tab contains global parameters that are the default values for the sink’s start and stop parameters. Numeric sinks’ start and stop parameters are set to DefaultNumericStart and DefaultNumericStop. Timed sinks’ start and stop parameters are set to DefaultTimedStart and DefaultTimedStop. The default values for these Data Flow controller values are 0, 100, 0 usec, and 100 usec, respectively.

Sinks can control simulation locally with their own start and stop times, or they can use the appropriate Data Flow parameter to inherit control. By default, all sinks inherit start and stop times from the controller. You can inherit none, one, or both of the start and stop times.

Because these Data Flow parameters function as variables inside the simulation, they can be used inside expressions or overridden in a hierarchal fashion. For
example, you could set a numeric sink’s parameters to Start=DefaultNumericStart and Stop=DefaultNumericStop*2.

**Options Tab**

The options tab has the following parameters:

**DefaultSeed**  Enter an integer to seed the random number generator. The default is 1234567.

The DefaultSeed parameter is used by all random number generators in the simulator, except those components that use their own specific seed parameter. DefaultSeed initializes the random number generation. The same seed value produces the same “random” results, thereby giving you predictable simulation results.

To generate repeatable “random” output from simulation to simulation, use any positive seed value. If you want the output to be truly random, enter the seed value of 0.

**OutVar**  OutVar is a space-separated list of variable names defined using variables and equations (VAR) components. The values of these variables will be sent to the Data Display window. In the case of hierarchical designs, in order to send variables that are at a level other than the top-most level, use the complete path to the variables, which must be period ‘.’ delimited.
Example:

OutVar="freq1 freq2 X1.amplitude X2.X4.temp"

In this case, there are four variables to be sent to the Data Display: freq1, freq2, amplitude, and temp, each separated with a space. The variable amplitude is contained in subnetwork X1, while the variable temp is contained in subnetwork X4, which in turn is contained in subnetwork X2. These subnetworks are delimited with periods.

The global character * can also be used to specify a list of names that have the same initial part of the string. For example, to send variables freq1, freq2, freq3, and freq4 to the data display, specify OutVar="freq*". Similarly, to send all variables, specify OutVar="*". To send all variables defined in a subnetwork named X1, specify OutVar="X1.*". The global character can be used only at the end of a string.

**Note**
ADS puts a set of quotes around the OutVar variable. Do not type your own quotes or the double set will cause simulations to fail.

**SchedulerType**  Select a Scheduler Type from the drop-down list. The choices are:

- Cluster Loop Scheduler (default)—Optimized for multirate graphs with feedback cycles.
- Classical Scheduler—Better for uni-rate graphs with cycles.
- Acyclic Loop Scheduler—Better for multirate graphs with no cycles.
- Multithreaded Scheduler—Optimized for multiprocessor machines.

The Scheduler Type option enables you to run the simulation based on one of three options. No matter which scheduler is chosen, the simulation results will be the same. The difference is in the time and memory needed to set up and run the simulation. It's best to start with the default, and experiment with the others as needed. For more information on these schedulers, refer to "How Schedulers Work in Agilent Ptolemy" on page 9-5.

**DeadlockManager**  The Deadlock Manager enables you to manage deadlocks in designs. A deadlock in a design occurs when a feedback loop does not have a delay in its feedback path, or when a Delay item does not initialize the proper number of signal tokens. A static schedule (required for simulation) can only be derived in a design with no schedule deadlocks.
Select the type of deadlock management from the drop-down list:

• **Report deadlock (default)**—Indicates the design includes deadlocks.

• **Identify deadlocked loops**—Enables you to spot which loops are deadlocked. These loops can be highlighted on the schematic page by double-clicking on the error message or Status window.

• **Resolve deadlock by inserting tokens**—Adds delays to deadlock loops and allows the simulator to proceed.

**Output Tab**

The Output tab enables you to selectively save simulation data to a dataset. For details, refer to Selectively Saving and Controlling Simulation Data in ADS in Chapter 1 of the Circuit Simulation documentation.

Note that Node Voltages are supported only on A/RF controllers; therefore, you will not find this option available on the Data Flow Controller’s Output tab.
Resistors Tab

The Resistors tab controls global parameters related to resistor behavior. As in the Controls tab, these parameters act as variables inside the simulation. Overriding the resistor values in a hierarchical fashion can be especially useful. For example, a large design can have a subcircuit representing a component being tested. By setting the DefaultRTemp inside the Data Flow controller to -273.15, and placing a VAR block with a DefaultRTemp setting inside the subcircuit, you can easily add resistor noise to the subcircuit and nowhere else.

- **DefaultRIn** is the default input impedance of timed components. Its value is 50 ohms.
- **DefaultROut** is the default output impedance of timed components. Its value is also 50 ohms.
- **DefaultRLoad** is the default input impedance of timed sinks and the default impedance of solitary resistors (the R component). Its value is 1.0e18 ohms, representing an infinite load.
- **DefaultRTemp** is the default temperature of resistors. Its value is -273.15 Celsius (0 K), so by default there is no thermal noise.
Debug Tab

The Debug tab can be used to provide the ability to debug your design and its custom components. There are two fields in this tab:

**Schedule Log File** enables you to specify the file name for a log file. After a simulation is finished, the log file you specified will be generated under the /data directory of the project. It will log the firing schedule of the components in your design.

**Profile Times File** enables you to specify the file name for a file containing simulation information. After a simulation is finished, the file will also be located under the /data directory of the project. It provides run-time information on the components in your design during simulation. For example, information may include the number of times a component is fired or the average time.
Data, Controllers, Sinks, and Components

Display Tab

From the Display tab, you can choose which parameters to display on the schematic. By default, only the four start and stop parameters are displayed:

- DefaultNumericStart
- DefaultNumericStop
- DefaultTimedStart
- DefaultTimedStop

You can add or subtract parameters you want to be displayed on the schematic by scrolling through the list and selecting or unselecting parameters.
OutputOption Controller

Description: Output Option for Dataset Templates
Library: Controllers

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Default</th>
<th>Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>DatasetTemplate</td>
<td>Dataset Template name (repeatable)</td>
<td></td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>

Notes/Equations

1. The OutputOption controller is used to specify the data display template(s) for Wireless Test Bench (WTB) used in RFDE.

2. All listed data display template(s) will be automatically inserted into an Autoplot data display window in RFDE after simulation.

3. The string value of DatasetTemplate should be the name of only one data display template file and should not contain the data display template file extension (.ddt).

4. To list more than one template name, use the Add button on the component dialog box to add additional DatasetTemplate parameters, each of which should have a string value for only one template.

5. A blank space (“ ”) value for DatasetTemplate will be ignored.
WTB Controller

**Description:** Controller for Wireless Test Bench

**Library:** Controllers

**Parameters**

There are no parameters for this controller.

**Notes/Equations**

1. The WTB controller is used to verify the Wireless Test Bench (WTB) Design created in ADS, before exporting the design to RFDE as a WTB model. See “Preparing Wireless Test Bench Designs for RFDE” on page 11-17.

2. The WTB model created in ADS is a subcircuit that must contain output pins and input pins followed by EnvOutShort or EnvOutSelector. The WTB model subcircuit must also have a DF controller with the correct setup. The WTB model simulates a RFIC Design Under Test (DUT) and reads the data back from the DUT to do complex measurements.

3. The WTB subcircuit created in ADS should be verified before exporting the WTB design to RFDE. To verify the WTB subcircuit open a new DSP schematic window, and add a WTB controller along with the instance of the WTB model subcircuit, a DUT, and an Envelope controller. The WTB controller enables a cosimulation between the Envelope controller and the DF controller inside the WTB model subcircuit.

4. The Envelope controller must be added along with an instance of the DUT subcircuit created using Analog/RF components. The Envelope controller can be added to the WTB verification design (DSP schematic) by opening an Analog/RF schematic window, adding an Envelope controller in this window, then copy and paste the Envelope controller into the WTB verification design window.

5. The WTB simulation in ADS will ignore the OutputOption controller that has been placed in the WTB subcircuit (see “OutputOption Controller” on page 3-15). This means that the data display window with the templates defined in the OutputOption controller will not be opened automatically in ADS. These templates open only in RFDE.
Sources and Sinks Control the Simulation

Agilent Ptolemy simulation is controlled by the sources and sinks you place on your schematic. All sinks and many sources have a ControlSimulation parameter that is either YES or NO. Controlling sinks and sources keep the simulation running, and non-controlling sinks and sources do not. There must be at least one source or sink that is controlling the simulation.

Sinks

Sinks are components with no outputs. When a sink is controlling the simulation, it will keep the simulation running long enough to satisfy its start and stop times. (One or both of the start and stop times might be inherited by the Data Flow controller.) By default, a sink's ControlSimulation parameter is set to YES. When a sink is not controlling the simulation, it will start collecting data at Start, and then collect as much data as the simulation produces. Consider the following example:
As shown, both sinks will collect 101 data samples (0 to 100 inclusive). They are both controlling sinks so they will obey their start and stop times. Because of the DownSample, sink N2 will receive more data but it will not collect it.

Changing one of the sinks ControlSimulation parameters to NO will cause N2 to collect twice as much data as N1. If N1 is the controller, then it will collect 101 samples, and N2 will collect 202. If N2 is the controller, then it will collect 101 samples, and N2 will collect 50.

This example demonstrates a useful way to design a schematic with multiple sinks. Choose one sink to control the simulation, and set all other sinks’ ControlSimulation parameters to NO. In this manner, your sinks will collect appropriate amounts of data according to the multirate characteristics of your schematic.

Sources

Sources are components with no inputs. Sources that read from files, instruments, and data sets also have a ControlSimulation parameter. By default, its value is NO. When a source is controlling the simulation, it will keep the simulation running long enough to output all its data. Controlling sources can be used to create designs that process all the data in a file, as shown in the next figure.
In this example, the SDFRead component is controlling the simulation, and the TimedSink parameter is not in control. The TimedSink will collect all the data available in the file. This example demonstrates another useful way to design schematics: control the simulation with a source, and set all the sinks’ ControlSimulation parameters to NO.

In the example, if both components’ ControlSimulation parameters were flipped so that only the TimedSink was in control, then it would collect enough data to meet its Start and Stop parameters. If that were more data than was available in the file, then the SDFRead component would repeat its data or zero pad according to its Periodic parameter. If that were less data than was available in the file, then the SDFRead would not output the entire file.

It’s possible to set both components’ ControlSimulation parameters to YES. In that case, and if the file had more data than the TimedSink’s Start and Stop required, then the SDFRead would output the entire file, but the TimedSink would ignore any data received after its Stop.
Agilent Ptolemy Components

The component libraries available for use with signal processing designs using the Agilent Ptolemy simulator are shown below. Reference information for each component is available by choosing Help, either from the parameters dialog box for a specific component, or from the Help menu.

Get to know the available components by choosing Insert > Component > Component Library, resizing the dialog box so you can read the complete names, and browsing through the list. Table 3-2 summarizes the libraries and their contents.

<table>
<thead>
<tr>
<th>Library</th>
<th>Summary of Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antennas &amp; Propagation</td>
<td>Contains components dealing with the radio channel, including antennas and propagation models. The channel models provide built-in functionality based on various standards, e.g., GSM, TDMA, CDMA.</td>
</tr>
<tr>
<td>Circuit Cosimulation</td>
<td>Contains items used to set up cosimulation with analog/RF circuits.</td>
</tr>
<tr>
<td>Common Components</td>
<td>A factory list of the most commonly used components taken from the remainder of the libraries.</td>
</tr>
<tr>
<td>Controllers</td>
<td>Items that control simulation parameters.</td>
</tr>
<tr>
<td>HDL Blocks</td>
<td>Components needed for HDL cosimulation.</td>
</tr>
<tr>
<td>Instruments</td>
<td>Contains components used to link data to instruments, such as the Agilent 89400 Vector Signal Analyzer.</td>
</tr>
<tr>
<td>Interactive Controls and Displays</td>
<td>Components that control and interactively display real-time simulation results. Data is not saved.</td>
</tr>
<tr>
<td>Numeric Communications</td>
<td>Contains numeric communications functions, such as ADPCM coder, QAM encoder, Viterbi decoder, modulation, demodulation, scrambler, spreader.</td>
</tr>
<tr>
<td>Numeric Control</td>
<td>Items that manipulate the flow of data during simulation, such as commutators, multiplexers, demultiplexers, upsamplers, forks, etc.</td>
</tr>
<tr>
<td>Numeric Logic</td>
<td>Contains Boolean operators, such as and, or, equals, greater than, etc.</td>
</tr>
<tr>
<td>Numeric Math</td>
<td>Contains math functions, such as adders, multipliers, integrators, log, sine, cosine, etc.</td>
</tr>
<tr>
<td>Library</td>
<td>Summary of Contents</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Numeric Matrix</td>
<td>Contains components that receive and/or produce vector or matrix signals at their input and output, such as add and multiply. Also contains MATLAB components and components used for converting scalar to matrix.</td>
</tr>
<tr>
<td>Numeric Signal Processing</td>
<td>Contains basic discrete-time DSP functions, such as FIR filter, IIR filter and adaptive filter, DTFT, etc.</td>
</tr>
<tr>
<td>Numeric Sources</td>
<td>Contains sources (items having only output) that produce numeric signals. This includes sources that output scalar, matrix, and random signals.</td>
</tr>
<tr>
<td>Numeric Special Functions</td>
<td>Miscellaneous items. Typically nonlinear operations such as quantizing, limiting, or triggering on input signals.</td>
</tr>
<tr>
<td>Numeric Synthesizable DSP</td>
<td>Bit-accurate DSP models (adders, registers, etc.) with behavioral C++ simulation code and Verilog and VHDL synthesizable code.</td>
</tr>
<tr>
<td>Signal Converters</td>
<td>Converts signal (data) types, from one type to another, for example, CxToFloat (complex to floating-point (real)). Others include integer, fixed, or timed.</td>
</tr>
<tr>
<td>Sinks</td>
<td>Data collection items or data processed as measurements, such as numeric sink, BER sinks, or EVM sink.</td>
</tr>
<tr>
<td>Timed Data Processing</td>
<td>Contains data processing components that operate on time-domain baseband waveforms, e.g., multilevel symbol coders and converters, IQ data coders.</td>
</tr>
<tr>
<td>Timed Filters</td>
<td>Contains time-domain lowpass and bandpass analog filters for filtering baseband or RF signals.</td>
</tr>
<tr>
<td>Timed Linear</td>
<td>Contains various linear operations for time-domain analog baseband and RF signals, e.g., waveform delay, split, sum, sample, switch.</td>
</tr>
<tr>
<td>Timed Modem</td>
<td>Contains analog RF modulators, demodulators, and carrier recovery for AM, FM, PM, QAM, QPSK, GMSK, MSK, DQPSK, and Pi/4 DQPSK formats.</td>
</tr>
<tr>
<td>Timed Nonlinear</td>
<td>Contains various nonlinear time-domain operations for time-domain analog baseband and RF signals, e.g., nonlinear gain, RF mixers, RF multipliers, rectifiers, signal sampling, or phase detectors.</td>
</tr>
</tbody>
</table>
Data, Controllers, Sinks, and Components

Table 3-2. Agilent Ptolemy Component Libraries (continued)

<table>
<thead>
<tr>
<th>Library</th>
<th>Summary of Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timed RF Subsystems</td>
<td>Contains various RF subsystem components, such as RF combiner, RF modulator, or RF demodulator.</td>
</tr>
<tr>
<td>Timed Sources</td>
<td>Contains various time-domain signal generators for baseband and RF signals, e.g., AM, FM, PM, QAM, clock, sinusoid, pulsed, or video.</td>
</tr>
</tbody>
</table>

**Note**: If you have purchased and installed ADS Design Library products, such as the CDMA, GSM, W-CDMA, cdma2000, EDGE, DTV, 1xEV, WLAN, or W-CDMA3G design libraries, they will be displayed in the list, in alphabetical order.
Chapter 4: Understanding Parameters

Value Types

Agilent Ptolemy requires specific parameter value types (such as string, real array, or complex) for the component parameter values you enter in schematic designs. They are called value types.

Component parameter values may be entered several ways by:

- Editing the Component Parameters dialog box. Double-click the component symbol on the schematic, a dialog box appears. Parameter values may be selected from lists or entered. The dialog box lists the value type expected, such as real or integer.
- Editing values directly on the schematic. Click the parameter value and type.
- Editing default values in the Design Definition dialog box. Choose File > Design/Parameters > Parameters tab. A type of parameter value can be selected from the Value Type list, and a default value can be entered in the Default Value field.

This section describes these value types. Table 4-1 lists each value type and its use:

Table 4-1. Ptolemy Parameter Value Types

<table>
<thead>
<tr>
<th>Value Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| Real       | Editing in Component Parameter dialog box:
  A. Enter real number.
  B. Enter expression for a real value—Example: X*cos(Y), where X and Y are defined expressions.
  Parameter editing on schematic: Highlight parameter value on schematic and enter real value or expression. |
| Integer    | Editing in Component Parameter dialog box:
  A. Enter integer.
  B. Enter expression for an integer value—Example: X+Y, where X and Y are defined expressions.
  Parameter editing on schematic: Highlight parameter value on schematic and enter integer value or expression. |
**Understanding Parameters**

### Table 4-1. Ptolemy Parameter Value Types (continued)

<table>
<thead>
<tr>
<th>Value Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| Fixed Point | Parameter editing in Component dialog box:  
A. Enter real value, but the value used will be based on the precision used with this parameter.  
B. Enter expression for a real value—Example: X*cos(Y), where X and Y are defined expressions.  
Parameter editing on schematic: Highlight parameter value on schematic and enter real value or expression. |
| Complex | Editing in Component Parameter dialog box:  
A. Enter a complex number using the form Re + j * Im.  
B. Enter expression for a complex value—Example: cos(X)+j*sin(Y), where X and Y are defined expressions, j is the imaginary operator.  
Parameter editing on schematic: Highlight parameter value on schematic and enter complex value or expression. |
| String | Editing in Component Parameter dialog box:  
A. Enter string. Do not enclose this string with any double quote symbols. Note for embedded double quotes ("), use double double quotes (\").  
B. Enter value by reference—Example: @Y, where Y is a the name of a Variable or Expression for a string value.  
Parameter editing on schematic: Highlight parameter value on schematic and enter string value enclosed with double quote symbols. |
| Precision | Editing in Component Parameter dialog box:  
A. Enter string in the form X.Y or Y/W. Do not enclose this string with any double quote symbols. The form X.Y, such as 8.24, means that there are X bits (including sign bit) to the left of the decimal point, and Y bits to the right of the decimal point. The form Y/W, such as or 24/32, means that there are Y bits to the right of the decimal point and W bits total. Note that X+Y=W.  
B. Enter value by reference—Example: @Y, where Y is a the name of a Variable or Expression for a precision value.  
Parameter editing on schematic: Highlight parameter value on schematic and enter string value enclosed with double quote symbols. |
Table 4-1. Ptolemy Parameter Value Types (continued)

<table>
<thead>
<tr>
<th>Value Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| Filename   | Editing in Component Parameter dialog box:  
A. Enter string for the name of a file including the pathname and select an extension type. The filename may include environment variables such as ~/,$HOME,$HPEESOF_DIR, or others.  
B. Enter value by reference—Example: @Y, where Y is a the name of a Variable or Expression for a filename value.  
Parameter editing on schematic: Highlight parameter value on schematic and enter string value enclosed with double quote symbols. |
| Integer Array | Editing in Component Parameter dialog box:  
A. Enter integer values directly—Example: 1 -2 5 2 (spaces separate data).  
B. Enter values from a file—Example: <filename. If the filename has no path specified, the project data directory is used. The content of the file must be numbers separated by spaces or on a new line. For example:  
\[
\begin{align*}
1 &\quad -2 \\
5 &\quad 2 \\
1 &\quad -2 \\
5 &\quad 2
\end{align*}
\]  
are equivalent.  
C. Enter values directly in addition to file data—Example: 1 <file1 2. If file1 contains -2 5, then the array would be the same as in A.  
D. Enter value by reference—Example: @Y, where Y is a the name of a Variable or Expression for an integer array.  
Example: @\{1,-2,5,2\}.  
Parameter editing on schematic: Highlight parameter value on schematic, then:  
- enter array values enclosed with double quote symbols  
or  
- enter array values as shown in E without the @ and double quote symbols. |
### Table 4-1. Ptolemy Parameter Value Types (continued)

<table>
<thead>
<tr>
<th>Value Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| Fixed Point Array or Real Array | Editing in Component Parameter dialog box:  
A. Enter fixed-point values directly—Example: 1.2 -2.3 5.6 2.8 (spaces separate data).  
B. Enter values from a file—Example: `<filename`. If the filename has no path specified, the project data directory is used. The content of the file must be numbers separated by spaces or on a new line. For example:  
1.2 -2.3  
5.6 2.8  
and  
1.2  
-2.3  
5.6  
2.8  
are equivalent.  
C. Enter values directly in addition to file data—Example: 1.2 `<file1 2.8`. If file1 contains -2.3 5.6, then the array would be the same as in A.  
D. Enter value by reference—Example: `@Y`, where Y is a name of a Variable or Expression for a fixed-point or real array.  
E. Enter values separated by commas, and surrounded by curly braces. Example: `@{1.2,-2.3,5.6,2.8}`.  
Parameter editing on schematic: Highlight parameter value on schematic, then:  
- enter array values enclosed with double quote symbols  
or  
- enter array values as shown in E without the `@` and double quote symbols. |
Complex Array

Editing in Component Parameter dialog box:
A. Enter complex values directly as ordered pairs separated by a comma (optional spaces may follow the comma). Each ordered pair must be enclosed in parentheses, and separated from other ordered pairs by spaces. Example:
(1.2, 2.5) (-2.3, 1.3) (5.6, -1.4) (2.8, 3.4)
B. Enter values from a file—Example: <filename. If the filename has no path specified, the project data directory is used. The content of the file must be ordered pairs of numbers separated by a comma (optional spaces may follow the comma). Each ordered pair must be enclosed in parentheses, and separated from other ordered pairs by spaces or on a new line. For example:
(1.2, 2.5) (-2.3, 1.3) (5.6, -1.4)
(2.8, 3.4) and
(1.2, 2.5)
(-2.3, 1.3)
(5.6, -1.4)
(2.8, 3.4)
are equivalent.
C. Enter values directly in addition to file data—Example: (1.2, 2.5) <file1
(2.8, 3.4). If file1 contains (-2.3, 1.3) (5.6, -1.4), then the array would be the same as in A.
D. Enter value by reference—Example: @Y, where Y is a the name of a Variable or Expression for a complex array.
E. Enter values separated by commas, and surrounded by curly braces. Complex values must be entered using the format a+j*b. Example:
@{1.2+j*2.5, -2.3+j*1.3, 5.6-j*1.4, 2.8+j*3.4}

Parameter editing on schematic: Highlight parameter value on schematic, then:
- enter array values enclosed with double quote symbols
or
- enter array values as shown in E without the @ and double quote symbols.

<table>
<thead>
<tr>
<th>Value Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complex Array</td>
<td>Editing in Component Parameter dialog box: A. Enter complex values directly as ordered pairs separated by a comma (optional spaces may follow the comma). Each ordered pair must be enclosed in parentheses, and separated from other ordered pairs by spaces. Example: (1.2, 2.5) (-2.3, 1.3) (5.6, -1.4) (2.8, 3.4) B. Enter values from a file—Example: &lt;filename. If the filename has no path specified, the project data directory is used. The content of the file must be ordered pairs of numbers separated by a comma (optional spaces may follow the comma). Each ordered pair must be enclosed in parentheses, and separated from other ordered pairs by spaces or on a new line. For example: (1.2, 2.5) (-2.3, 1.3) (5.6, -1.4) (2.8, 3.4) and (1.2, 2.5) (-2.3, 1.3) (5.6, -1.4) (2.8, 3.4) are equivalent. C. Enter values directly in addition to file data—Example: (1.2, 2.5) &lt;file1 (2.8, 3.4). If file1 contains (-2.3, 1.3) (5.6, -1.4), then the array would be the same as in A. D. Enter value by reference—Example: @Y, where Y is a the name of a Variable or Expression for a complex array. E. Enter values separated by commas, and surrounded by curly braces. Complex values must be entered using the format a+j<em>b. Example: @{1.2+j</em>2.5, -2.3+j<em>1.3, 5.6-j</em>1.4, 2.8+j*3.4} Parameter editing on schematic: Highlight parameter value on schematic, then: - enter array values enclosed with double quote symbols or - enter array values as shown in E without the @ and double quote symbols.</td>
</tr>
</tbody>
</table>
Table 4-1. Ptolemy Parameter Value Types (continued)

<table>
<thead>
<tr>
<th>Value Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| String Array | Editing in Component Parameter dialog box:  
A. Enter string values directly—Example: "Button 1" "Button 2" "Button 3" (each string is enclosed with double quote marks, spaces separate each string).  
B. Enter values from a file—Example: <filename>. If the filename has no path specified, the project data directory is used. The content of the file must be text separated by spaces or on a new line. For example:  
"Button 1" "Button 2" "Button 3"  
and  
"Button 1"  
"Button 2"  
"Button 3"  
are equivalent.  
C. Enter values directly in addition to file data—Example: "Button 1" <file1 "Button 3". If file1 contains "Button 2," then the array would be the same as in A.  
D. Enter value by reference—Example: @Y, where Y is a the name of a Variable or Expression for a string array.  
Parameter editing on schematic: Highlight parameter value on schematic and enter array value enclosed with double quote symbols. |
| Enumerated Type (with form specific to the component) | Editing in Component Parameter dialog box:  
A. Select enumerated type from selection list specific to the component parameter. For example, Time Unit (milliseconds, etc.) is an Enumerated Type you choose from a list. The Data Flow Controller parameter Scheduler Type is also an Enumerated Type.  
B. Select the "Standard" enumerated type and enter an integer value in the entry field provided. The integer value is associated with an option in the selection list with the first selection list entry associated with the integer 0, the second entry with the integer 1, etc.  
C. Select the "Standard" enumerated type and enter the expression in the entry field provided for an integer value—Example: X+Y, where X and Y are defined expressions.  
Parameter editing on schematic: Highlight parameter value on schematic and enter enumerated value, or use the up or down arrow keys on the keyboard to scroll through the enumerated options available. |

All of the above parameter types, except the Enumerated type, are directly available to define parameters in a schematic design.

To define or see the list of value types for a schematic design, from a Signal Processing Schematic window, choose File > Design/Parameters > Parameter. The list of parameter types available for a schematic design can be seen by selecting the Value Type drop-down list.

4-6 Value Types
To use the Enumerated type for a schematic design, you must edit the design AEL file (located in the project networks directory) and implement the AEL code for the desired Enumerated type. Examples of AEL for Enumerated types can be observed in the file: $HPEESOF_DIR/hptolemy/ael/stars.ael.

Parameter Editing

When an instance of a component or design is placed on the schematic, its parameters can be viewed below its symbol.

The default is for parameters to be visible on the schematic. To enable parameter visibility on the schematic, check two areas. From the Schematic window, choose Options > Layers, a dialog box appears. In the Layers list (left), select Parameters. Make sure that the Visible box is checked (center). Next double-click any component in the schematic. This displays the component parameters dialog box. Make sure that the Display parameter on schematic box (lower-center) is checked.

To illustrate this procedure, we will place an instance of the FreqPhase component.

1. Choose Insert > Component > Component Library.
2. From the Library list, select Numeric Communications and then FreqPhase from the Components list on the right.
3. Next place this component on the schematic.

Observe that its parameters (visible below its symbol on the schematic) are SampleRate, PhaseJitterFrequencyHz, FrequencyOffsetHz, and PhaseJitterAmplitudeDeg, as shown in the figure below. This component is actually a schematic design that we will look at in more detail:
Understanding Parameters

Notice that each parameter has a numeric value. These values could just as easily be an expression. Note that the SampleRate parameter is given as an expression, \(2\pi\). The full features of Advanced Design System expressions are discussed in the Simulator Expressions, and Measurement Expressions documentation.

To observe the detail of the schematic design associated with the FreqPhase component, click the symbol to highlight it, then choose View > Push into Hierarchy or click the Push into Hierarchy button (down arrow icon) from the toolbar. When this is done for FreqPhase, the schematic shown below appears.

Notice that the various components in this schematic reference the top-level FreqPhase parameters by name. The top RampFloat component has its Step parameter equal to \(2\pi\times\text{FreqOffsetHz}/\text{SampleRate}\), where FreqOffsetHz and SampleRate are value passed in from the top level.

To observe the parameters defined (or to define additional parameters) for this schematic design, choose File > Design/ Parameters. The Design Definition dialog box appears. Select the Parameters tab and you will observe fields to enter parameter definition information. For more information on the Design Definition dialog box refer to Schematic Capture and Layout > Creating Hierarchical Designs.
Parameter Expressions

Parameter values can be arithmetic expressions. This is particularly useful for propagating values down from a top-level system parameter to component parameters down in the hierarchy. An example of a valid parameter expression is:

\[ x = \frac{\pi}{2 \times \text{order}} \]

where \( \text{order} \) is a parameter defined in the network or top-level system, and \( \pi \) is the built-in constant. The basic arithmetic operators are addition (+), subtraction (-), multiplication (*), division (/), and exponentiation (^). These operators work on integers and floating-point (real) numbers. Currently, all intermediate expressions are calculated in double-precision values and only the final value is converted to the type of the parameter being computed. Hence, it is necessary to be very careful when, for example, using floating-point (real) expressions, to compute an integer parameter. In an integer parameter specification, all intermediate expressions will be calculated with double-precision floating-point (real) values and the final value is cast to an integer value.

Complex-Valued Parameters

When defining complex values, the basic syntax is

\[ \text{real} + j \times \text{imag} \]

where \( \text{real} \) and \( \text{imag} \) evaluate to double-precision, floating-point (real) values, which may be numbers or expressions, and where \( j \) is the imaginary operator.

There are also other functions in Agilent Ptolemy that can be used with complex values. These include:

- An expression/function that returns a Cartesian form: \( \text{complex} (X, Y) \).
- An expression/function that converts a polar form to Cartesian form: \( \text{polar} (X, Y) \), where \( X \) is magnitude and \( Y \) is in degrees.
- An expression/function that converts a decibel form to a Cartesian form: \( \text{dbpolar} (X, Y) \), where \( X \) is in decibels and \( Y \) is in degrees.
Parameters for Fixed-Point Components

Many of the fixed-point components used in Agilent Ptolemy utilize one or more common parameters that identify the specific characteristics of the finite-precision, fixed-point value. These include characteristics specifying overflow, overflow reporting, quantization, finite precision bit format, and more. The following describes several properties in common use by these components.

1. Parameters specifying fixed-point value precision are typically labeled “Precision,” “InputPrecision,” “OutputPrecision,” or some other token containing “Precision.”

   Fixed-point parameter precision is defined by either of two types of syntax:

   Syntax 1
   As a string such as “3.2”, or more generally “m.n”, where m is the number of integer bits (to the left of the binary point) and n is the number of fractional bits (to the right of the binary point). Thus length is m + n.

   Syntax 2
   A string like “24/32” which means 24 fraction bits from a total word length of 32. This format, n/w, is often more convenient because the word length often remains constant while the number of fraction bits changes with the normalization being used.

   In both cases, the sign bit counts as one of the integer bits, so this number must be at least one. The maximum value of w (or x+y) is 256.

   Thus, for example, a fixed-point value of 0.8 may have a precision defined as 2/4. This means that a 4-bit word will be used with two fraction bits. Since the value “0.8” cannot be represented precisely in this precision, the actual value of the parameter will be rounded to “0.75.”

   When an input pin with an associated fixed-point signal class (scalar or matrix) receives another class of signal (scalar or matrix, respectively), the received signal is automatically converted to the fixed-point class. A pin specified for use with fixed-point scalar signals does not accept any matrix class signals, and vice versa. The automatic conversion from timed, complex or floating-point (real) signals to a fixed-point signal uses a default bit width of 32 bits with the minimum number of integer bits needed to represent the value. For example, the automatic conversion of the floating-point (real) value of 1.0 creates a fixed-point value with precision of 2.30, and a value of 0.5 would create one of
precision of 1.31. For details on data/signal conversion rules, refer to "Conversion of Data Types" on page 5-6.

2. The parameter used to specify the arithmetic form of a fixed-point value is labeled ArithType. This parameter is an enumerated type with two options: TWOS_COMPLEMENT and UN_SIGNED. The fixed-point components in the Numeric Synthesizable DSP library may use either arithmetic form. However, fixed-point components outside the Numeric Synthesizable DSP library may only use the TWOS_COMPLEMENT form, which is the default.

3. The parameter used to specify the quantization property of a fixed-point value is labeled RoundFix. This parameter is an enumerated type with two options: ROUND and TRUNCATE. The quantization property is used to convert a floating-point (real) value to its fixed-point value. The ROUND quantization property causes this float-to-fixed transformation to occur such that the nearest fixed-point value to the floating-point (real) value is used. For example, consider the floating-point (real) value 0.1. It is not possible to represent this number exactly as a two's complement fixed-point value. Remember that a fractional decimal number is represented in its fixed-point form by composing it of the summation of fractional powers of two \(2^{-(N)}\). 0.1 is represented as 0.0001100110011... with an infinite number of fractional binary terms. If the precision is 2.8 and the quantization is ROUND, then this above fixed-point value is rounded up to the nearest fractional power of \(2^{-(8)}\) which is 0.00011010. If the precision remains at 2.8 and the quantization is TRUNCATE, then the value is truncated to 0.00011001.

4. The parameter used to specify the overflow properties of fixed-point mathematical operations is labeled OverflowHandler or OvflwType. Both of these are an enumerated type. The OverflowHandler parameter has four options: wrapped, saturate, zero_saturate, or warning. The OvflwType parameter has two options: wrapped or saturate. The OvflwType parameter is used only by fixed-point components in the Numeric Synthesizable DSP library. The OverflowHandler is used by all other fixed-point components. The overflow parameter specifies the overflow characteristic to use when the result of a fixed-point operation cannot fit into the precision specified.

Consider a fixed-point ramp data source (RampFix) as shown in the following figure. It has a step size of 0.2, initial value of 0, output precision of 2.14, with round type quantization.
Understanding Parameters

When the OverflowHandler is set to wrapped, the following data display results:

Figure 4-2. Simulation Plot with Overflow Handler Set to Wrapped

Note that as a 2's complement signal, the maximum value for a 2.14 precision with rounding is nearly 1.9 and the minimum value is nearly −2.0. There are actually more decimal places in these values due to the quantization of the step size. This maximum and minimum is obtained by first converting the step size of 0.2 into its fixed-point form with 2.14 precision. This becomes the step size for the fixed-point ramp accumulation. The signal begins at zero, and increments by the fixed-point...
binary representation of 0.2 with each sample. When the maximum value is reached, the output wraps to the minimum value for the given precision and quantization.

When the above example uses the truncate type of quantization the following data display results:

![Figure 4-3. Simulation Plot with Truncate Quantization](image)

Note that as a 2's complement signal, the maximum value for a 2.14 precision with truncation is 2.0, and the minimum value is -1.9. The signal begins at zero, and increments by 0.2 with each sample. When the maximum value is reached, the output wraps to the minimum value for the given precision.

With truncation used, but with overflow set to saturate, the following data display results:
Note that when the ramp rises to 2.0, it stays constant at that level. With truncation used, but with overflow set to zero_saturate, the following data display results:
Note that when the ramp rises to 2.0, it resets to the value of zero and continues to rise.

Again with truncation used, but with overflow set to warning the following data display results:

![Simulation Plot with Truncate Quantization and Overflow Handler Set to Warning](image)

Note that the saturate characteristic is used. Additionally, the warning mode results in ReportOverflow (explained below) being set to REPORT. This reports the number of overflows at the end of the simulation.
Understanding Parameters

5. The parameter used to specify whether overflow is reported is labeled ReportOverflow. This parameter is an enumerated type with two options: REPORT and DONT_REPORT. Consider the preceding overflow displays. For each of the cases, when the ReportOverflow parameter is set to REPORT a warning message is displayed in the Simulation/Synthesis Messages window after simulation. In the previous simulation for the zero_saturate data display, the warning is:

1: R1: experienced overflow in 9 out of 102 fixed-point calculations checked (8.8%)

When you click this message, the RampFix component with the name R1 is highlighted in the schematic window.

When the ReportOverflow parameter is set to DONT_REPORT, this warning message does not appear.

6. The parameter used to specify whether a component is to use the input signal with its arriving precision, or whether this signal is to be cast into another component’s specific precision is labeled UseArrivingPrecision. This parameter is an enumerated type with two options: NO or YES. This parameter is used with the InputPrecision parameter. When UseArrivingPrecision = NO, the input signal is cast to the precision specified by InputPrecision. Otherwise, the input signal’s precision is used.

String Parameters

String parameters are assigned a text value that may include any alpha-numeric symbol, including spaces and other punctuation symbols. If a double-quote symbol (") is to be used, it must be used with two such sequential symbols (""") and will be interpreted as only a single, double-quote symbol.

Filename Parameters

Filename parameters are assigned a filename value that may include the file path name and environmental variables such as ~/ $HOME, $HPEESOF_DIR, or others. If no path name is provided, the current project data subdirectory is the assumed path for the file.
Array Parameters

When defining arrays of integers, floating-point (real) numbers, complex numbers, fixed-point numbers, or strings, the basic syntax is a simple list separated by spaces, as shown in the following example:

1 2 3 4 5

defines an integer array with five elements. Repetition can be indicated using the following syntax, value\( [n] \), as demonstrated in the following example:

1 2 3\([10]\) 4 5

where \( n \) is an integer. This example has ten instances of the value 3. An array or portion of an array can be input from a file using the symbol “\(<\)" as shown in the following example:

1 2 < filename 3 4

Here the first two elements of the array will be 1 and 2, the next elements will be read from file filename, and the last two elements will be 3 and 4. This latter capability can be used in combination with the WaveForm component to read a signal from a file.

When defining complex array values, the basic syntax is

\((1.2, 2.5)\) \((-2.3, 1.3)\) \((5.6, -1.4)\) \((2.8, 3.4)\)

where all entered complex values are ordered pairs of real and imaginary values of complex numbers enclosed in parentheses and separated by commas. Optional spaces may follow each comma. Ordered pairs must be separated by a space. All entries must be numbers.

When defining string array values, the basic syntax is

"Button 1" "Button 2" "Button 3"

where all entered string values are enclosed in double quote symbols.

Another way to define arrays is to separate values with commas, and enclose the set of values in curly braces. For example:

\{1, 2, 3, 4, 5\}

Complex numbers must be specified using the format a+j*b. For example:

\{1.2+j*2.5, -2.3+j*1.3, 5.6-j*1.4, 2.8+j*3.4\}
Understanding Parameters

However, using the “<filename” method inside curly braces to include values is not supported.

**Reading Array Parameter Values From Files**

The values of all array parameter types can be read from a file. The syntax for this is to use the symbol “<” as in the following example:

```
<filename
```

or

```
1.2 2.6 <filename 2.8 6.4
```

If the filename has no path specified, the project data directory is used. Otherwise, the filename should typically contain the full pathname to the file. Any references to environment variables or home directories are substituted to generate a complete path name. All values in the filename must be numeric values for the numeric array types (integerarray, realarray, fixedpointarray, complexarray), and must be string values for the string array type. The contents of the file are read and spliced into the parameter expression and re-parsed. File inputs can be very useful for array parameters which may require a large amount of data. Other expressions may come before or after the “<filename” syntax (any white space that appears after the < character is ignored). Within the file, comment lines containing a leading pound (#) symbol are ignored by the file parser.
Parameters With Optimization and Swept Attributes

Many component parameters may have associated attributes that are used during nominal optimization. Within the Component dialog box, any parameter of type real, fixed point, integer, or enumerated, may also be optimized for design performance. A complex value may be optimized by optimizing its real and/or imaginary parts.

Parameters of type Complex, Precision, Array, String, or Filename may be optimized or swept by creating a string that references optimized or swept variables. To reference an optimized variable, the variable must be defined in a VAR (Variables and Equations) component with the Standard entry mode and with Optimization/Statistic Setup enabled.

In a similar manner to optimization attributes, there can also be parameters with swept attributes.

For more information on optimization in Agilent Ptolemy, refer to Chapter 8, Using Nominal Optimization in this manual. For more information on sweeping parameters in Agilent Ptolemy, refer to Chapter 7, Performing Parameter Sweeps in this manual.
Understanding Parameters
Chapter 5: Using Data Types

This chapter reviews some basic material on Data Types that was introduced in Chapter 3, Data, Controllers, Sinks, and Components, but then goes into more detail.

Agilent Ptolemy uses different data types such as integer, fixed-point, floating-point (real), and complex in scalar or matrix forms. In Agilent Ptolemy documentation there are numerous references to data and signal types. When data is presented versus an independent variable such as time, the data can be thought of as a signal. Regardless of the terminology, data or signals consist of packets of information that are passed from one component to another.

Representation of Data Types

Agilent Ptolemy schematics contain component stems with different colors and thicknesses. Each component input and output pin has an associated data type, and each type is represented in the component symbol by use of a color code and a thickness of stem. Additionally, each component stem may have single or multiple arrowheads. Table 5-1 describes stem color and thickness in Signal Processing schematics.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Stem Color</th>
<th>Stem Thickness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalar Fixed Point</td>
<td>Magenta</td>
<td>Thin</td>
</tr>
<tr>
<td>Scalar Floating Point (Real)</td>
<td>Blue</td>
<td>Thin</td>
</tr>
<tr>
<td>Scalar Integer</td>
<td>Orange</td>
<td>Thin</td>
</tr>
<tr>
<td>Scalar Complex</td>
<td>Green</td>
<td>Thin</td>
</tr>
<tr>
<td>Matrix Fixed Point</td>
<td>Magenta</td>
<td>Thick</td>
</tr>
<tr>
<td>Matrix Floating Point (Real)</td>
<td>Blue</td>
<td>Thick</td>
</tr>
<tr>
<td>Matrix Integer</td>
<td>Orange</td>
<td>Thick</td>
</tr>
<tr>
<td>Matrix Complex</td>
<td>Green</td>
<td>Thick</td>
</tr>
<tr>
<td>Timed</td>
<td>Black</td>
<td>Thin</td>
</tr>
<tr>
<td>AnyType</td>
<td>Red</td>
<td>Thin</td>
</tr>
</tbody>
</table>
Using Data Types

**Stem Thickness**

*Figure 5-1* illustrates stem thickness:

*Figure 5-1. Matrix Data (Thick Lines) vs. Scalar Data (Thin Lines)*
Single and Multiple Arrowheads

Agilent Ptolemy uses block diagram schematics to enter information for simulation, which implies that all signals flowing between components are directional. Therefore, each input or output stem has arrowheads indicating the signal flow direction. This is not the case in circuit schematics where signals (wires) are generally bidirectional.

The signal flow is indicated by a single arrowhead. While single arrowhead stems carry only one distinct signal, double arrowhead stems can carry any number of independent signals or data. Figure 5-2 shows the difference between single and multiple arrowheads. In this figure, the input of the multiplier component is a single multiple input carrying data from any number of inputs.

Figure 5-2. Single and Multiple Arrowheads

Data Types Defined

Presently, there are two general signal types used in Agilent Ptolemy, numeric and timed. The numeric type has several subtypes, such as fixed-point, real, scalar, and matrix. Numeric signals have sequential numbers as the independent variable. Timed signals have time as the independent variable and are derived from complex data. Timed signals have additional attributes.

Typically, numeric data is used for algorithmic development in the baseband portion of a communication system. Timed signals are used to simulate the signal in the modulation channel, as well as for cosimulation with certain Advanced Design System circuit simulators.
Using Data Types

Numeric Scalar Data

Numeric scalar data is defined as follows:

- **int** single, integer value (signed value defined with a 32-bit value)
- **fixed** single, fixed-point value with the following properties and operation attributes:
  - precision defined using \( x.y \) or \( y/w \) where
    - \( x \) = bits to the left of the decimal point
    - \( y \) = bits to the right of the decimal point
  - \( w = x + y \) = total bit width, 1 to 255
  - arithmetic type
    - two's complement (with sign bit included in \( x \))
    - unsigned
  - quantization type
    - truncate, round
  - overflow type
    - saturate, saturate to zero, wrapped
- **floating-point (real)** double precision floating-point (real) number
- **complex** pair of double precision floating-point (real) number for real and imaginary parts

Numeric Matrix Data

All matrix data is defined as a two-dimensional array (rows, columns) of either int, fixed, floating-point (real), or complex values. All matrix data types are indicated by thick stems, in contrast with the thin stems used for scalar data types.
**Timed Data**

Agilent Ptolemy supports timed data. This signal is derived from complex data and includes additional attributes. The timed signal packet includes five members

\[ i(t), q(t), \text{flavor}, F_c \text{ and } t \]

where \( i(t) \) and \( q(t) \) are the timed signal in phase and quadrature components, \( \text{flavor} \) indicates the representation of a modulated signal, \( F_c \) is the carrier (or characterization) frequency, and \( t \) is the time.

There are two equivalent representations (flavors) of a timed signal:

- complex envelope (ComplexEnv) \( v(t) \)
- real baseband (BaseBand) \( V(t) \)

RF signals that are represented in the ComplexEnv flavor \( v(t) \) together with \( F_c \) can be converted to the real BaseBand flavor \( V(t) \) as:

\[
V(t) = \text{Re}\left\{v(t)e^{j2\pi F_c t}\right\}
\]
Conversion of Data Types

What Happens During Conversion?

We introduced this topic in Chapter 3. In this section, we go into more detail. Most conversions do what you expect. For example, when converting from lower precision to higher precision data types, such as integer to floating-point (real), no data is lost; only the format is changed.

When converting from higher precision to lower precision data types, such as floating-point (real) to integer, the outcome is governed by your computer’s math rounding rules.

Whether you manually place a converter, or the simulator “splices” in a converter, the conversion process is the same. It is similar to the casting operation used in C or C++ languages. If the conversion from A to B requires more information (integer to floating-point (real), floating-point (real) to complex, etc.) the “obvious” thing happens. For example, conversion from floating-point (real) to complex is done by setting the imaginary part of the complex number equal to 0.0. However if the conversion involves loss of information (complex to double, double to integer, etc.), a set of rules are followed that are in most cases very simple and intuitive.

Numeric Scalar and Matrix Conversions

Table 5-2 outlines the rules regarding scalar conversions among numeric data types:

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complex</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td>round mag</td>
</tr>
<tr>
<td>Real</td>
<td></td>
</tr>
<tr>
<td>Fixed</td>
<td></td>
</tr>
</tbody>
</table>

Note that mag in preceding table means the magnitude of the complex number $C = a + jb$, which is equal to

$$\sqrt{a^2 + b^2}.$$
For automatic conversion (when no converter is explicitly used) to the Fixed data type, the resulting fixed-point number has the default length of 32 bits and a precision of the minimum number of integer bits needed for a two's complement representation. For example, the integer 5 is converted to the fixed point number 0101.0000000000000000000000000000 (precision “4.28”), whereas the floating-point (real) number 3.375 is converted to 011.01100000000000000000000000000 (precision “3.29”). If this is not the behavior you want, you must explicitly use a converter.

For matrix conversions, the above operations hold for all entries in the matrix.

**Timed Data Conversions**

You can convert between timed and scalar numeric data types by placing one of the following converters and supplying the parameters as needed:

- Timed to Complex or Complex to Timed
- Timed to Float (real) or Float (real) to Timed
- Timed to Fixed or Fixed to Timed
- Timed to Integer or Integer to Timed

Given the Timed data type \{i(t), q(t), flavor, Fc and t\}, the conversions between input and output of a converter are summarized below:

**Timed To Float (Real)**

If x is the input and y is the output for the TimedToFloat converter, then:

\[ y[n] = \begin{cases} 
  i(t) \cos(2\pi F_c t) - q(t) \sin(2\pi F_c t) & \text{when flavor = ComplexEnv} \\
  i(t) & \text{when flavor = BaseBand} 
\end{cases} \]
Using Data Types

**Float (Real) To Timed**

The FloatToTimed converter has one specific parameter, TStep. If \( x \) is the input and \( y \) is the output for this converter, then the \( y(t) \) packet has the following parts:

- \( i(t) = x[n*TStep] \)
- \( q(t) = 0.0 \)
- \( F_c = 0.0 \)
- \( \text{flavor} = \text{BaseBand} \)

**Timed To Complex**

The Timed To Complex converter has no parameters. If \( x \) is the input and \( y \) is the output for this converter, then:

\[
\begin{align*}
y[n] &= i(t) +jq(t) \\
&\quad \text{when flavor} = \text{ComplexEnv} \\
y[n] &= i(t) + j 0.0 \\
&\quad \text{when flavor} = \text{BaseBand}
\end{align*}
\]
### Complex To Timed

The Complex To Timed converter has two specific parameters, TStep and FCarrier. If \( x \) is the input and \( y \) is the output for this converter, then the \( y(t) \) packet has the following parts:

- \( i(t) = \text{Real}\{x[n*TStep]\} \)
- \( q(t) = \text{Imag}\{x[n*TStep]\} \)
- \( F_c = \text{FCarrier} \)
- \( \text{flavor} = \text{ComplexEnv} \)

### Rules and Exceptions

The converter devices are in general not reciprocal, i.e., putting two converters with opposite functionality back-to-back does not necessarily recover the original signal.

Based on the three categories of numeric scalar, numeric matrix, and timed data types discussed above, the following rules should be considered:

- The conversion between numeric scalar and numeric matrix types are done by explicitly placing Pack and UnPack components. No automatic conversion is performed between these two categories.
- The conversion between numeric scalar and timed data is done by placing the appropriate converters. Automatic conversion between these two categories is allowed (see details below).
- There is no direct conversion between numeric matrix and timed data types.

Figure 5-3 summarizes the conversion among data types.
Using Data Types

Automatic or Manual Data Type Conversion

If the output of component A and the input of component B is the same (which means they are represented by the same color), the data is simply copied from A to B. If the output of component A and the input of component B is different, a conversion is necessary.

Although the software will automatically convert dissimilar data types, such as complex to floating-point (real), you may want to manually place an appropriate converter (from the Signal Converters library) in your schematic. This acts as a visual reminder that a conversion is taking place, and also helps you decode error messages that may arise. Automatic conversion means that an appropriate converter is “spliced in” behind the scenes and is not shown on the schematic.

Automatic conversion is allowed among scalar data types and among matrix data types, but not between scalar and matrix data types.
Allowed and Disallowed Automatic Conversions

Automatic conversion is available among all numeric scalar types. The same is true for matrix types. Figure 5-4 summarizes the allowed conversions.

![Figure 5-4. Automatic Conversion Among Numeric Scalar and Matrix Types](image)

With one exception (complex to timed), automatic conversion between timed and numeric scalar types is also supported, as depicted below in Figure 5-5.

![Figure 5-5. Automatic Conversion of Timed and Scalar Numeric Types](image)

Automatic conversion from Complex to Timed is not supported because the information about the carrier frequency must be supplied by the user. You need to place a ComplexToTimed converter and enter the appropriate parameters if such a conversion is needed.
Using Data Types

Also note that automatic conversion of Float (real) to Timed, Fixed to Timed, Integer to Timed, or Complex to Timed is only possible when there is at least one component in the design defining the TStep.

When a scalar pin is directly connected to a matrix pin (or vice versa), without a Pack or Unpack converter, an error message is generated.

In the Numeric Matrix Library, there are four converters, such as Pack_M and PackCx_M, that are used to “pack” scalar data into matrix data. Similarly, there are four converters, such as UnPk_M and UnPkCx_M, that “unpack” the data (back to scalar). There is no automatic conversion between scalar and matrix data (or vice versa). You must place the converters where needed in your design.
Chapter 6: Understanding File Formats

Real, complex, and string array data can be used with component parameters of type real array, complex array, or string array, respectively. Real array data can be used as input with the ReadFile component. Real, complex, floating-point (real) matrix, fixed matrix, complex matrix, and integer matrix can be used as output from the Printer component.

In the following seven examples of Agilent Ptolemy file formats (from real array data through complex matrix data), the examples are drawn from the code and include the “#” symbol, which denotes a comment on that line.

Real Array Data

# Template for Agilent Ptolemy real data
# Each number separated by newlines
1
0
0

Complex Array Data

# Template for Agilent Ptolemy complex data
# Each complex value, (real, imag), separated by newlines
(1.0, 0.0)
(0.0, 0.0)
(0.0, 0.0)

String Array Data

# Template for Agilent Ptolemy string data
# Each string value enclosed on double-quote marks, "", and separated by newlines
"text 1"
"text 2"
"text 3"
Understanding File Formats

Real Matrix Data

# Template for Agilent Ptolemy real matrix data
# Each matrix data set separately listed with brackets around each row and matrix
# Each matrix row separated by newlines
[
  [1.2, -2, 2]
  [-2, 2.25, -2]
  [2.5, -2.1, 3.2]
  [-3.5, 2.4, -1.3]
  [2.2, -2.4, 3.8]
  [-2.5, 2, -2.6]
]

Fixed-Point Matrix Data

# Template for Agilent Ptolemy fixed-point matrix data
# Each matrix data set separately listed with brackets around each row and matrix
# Each matrix row separated by newlines
[
  [1.2, -2, 2]
  [-2, 2.25, -2]
  [2.5, -2.1, 3.2]
  [-3.5, 2.25, -1.25]
  [2.2, -2.5, 3.5]
  [-2.5, 2, -2.5]
]

Integer Matrix Data

# Template for Agilent Ptolemy integer matrix data
# Each matrix data set separately listed with brackets around each row and matrix
# Each matrix row separated by newlines
[
  [1, -2, 2]
  [-2, 2, -2]
  [2, -2, 3]
  [-3, 2, -1]
  [2, -2, 3]
  [-2, 2, -2]
]
Complex Matrix Data

# Template for Agilent Ptolemy complex matrix data
# Each matrix data set separately listed with brackets around each row and matrix
# Each matrix row separated by newlines
[[ 11.0+0.0j, 12.0+0.0j, 13.0+0.0j ]
 [ 21.0+0.0j, 22.0+0.0j, 23.0+0.0j ]]

SPW (.ascoSIG and .sig) File Formats

SPW format data files can be read by the system simulator by specifying them as input files in a TimeDataFile component. They can also be written by the simulator by specifying them as output files in a TimeDataWrite component. The binary format .sig file has the same ASCII header information as the .ascoSIG file but the data is stored as a pointer in binary format.

1. The SPW version 3.0 data file format must be used.
2. Comments can only be included on the one line following the $USER_COMMENT statement.
3. The TimeDataFile source can read a real double-format or complex double-format SPW data file. To read an SPW format file, the appropriate .ascoSIG or .sig extension must be specified with the filename.
Understanding File Formats

**Real Double Data Format Example .ascsig File**

```
$SIGNAL_FILE 9
$USER_COMMENT

$COMMON_INFO
SPW Version = 3.0

Sampling Frequency = 1
Starting Time = 0
$DATA_INFO
Number of points = 6
Signal Type = Double
$DATA
1.000000000000000000000
1.000000000000000000000
-1.000000000000000000000
-1.000000000000000000000
1.000000000000000000000
1.000000000000000000000
END

(Note: There is no space between the sign and the number, e.g., -1.0.)
```
Complex double data format example .ascsig file

$SIGNAL_FILE 9
$USER_COMMENT

$COMMON_INFO
SPW Version = 3.0

Sampling Frequency = 1
Starting Time = 0

$DATA_INFO
Number of points = 10
Signal Type = Double
Complex Format = Real_Imag

$DATA
1.000000000000000000000+j1.000000000000000000000
1.000000000000000000000+j1.000000000000000000000
-1.000000000000000000000+j1.000000000000000000000
-1.000000000000000000000+j1.000000000000000000000
-1.000000000000000000000+j1.000000000000000000000
1.000000000000000000000+j1.000000000000000000000
1.000000000000000000000+j1.000000000000000000000
-1.000000000000000000000+j1.000000000000000000000
-1.000000000000000000000+j1.000000000000000000000
-1.000000000000000000000+j1.000000000000000000000

END

(Note: There is no space between the sign and the number, e.g., -1.0.)

Time-Domain Waveform Data (.tim) File, MDIF ASCII Format

The general .tim file format is:

BEGIN TIMEDATA
# T ( SEC V R xx )
% t v
<data line>
.
.
.<data line>
END
Guidelines for .tim files

An exclamation point ( ! ) at the beginning of a line makes it a comment line; characters following ! are ignored by the program.

TIMEDATA data block is required.

- The BEGIN statement:
  
  BEGIN TIMEDATA ! Begin time-domain waveform data

- Option line:
  
  # T ( [SEC/MSEC/USEC/NSEC/PSEC] V/MV/DATA R xx )

  where
  
  # = delimiter that tells the program you are specifying these parameters
  T = time
  SEC/MSEC/USEC/NSEC/PSEC = seconds/milliseconds/microseconds/nanoseconds/picoseconds
  V/MV/DATA = volts/millivolts/data
  R = reference resistance, default 50.0
  xx = user-specified value for reference resistance

  (Note that values R and xx are for documentation purposes and are not used by the simulator.)

- Format line:
  
  % t v

  where
  
  % = delimiter that tells the program you are specifying these parameters
  t = time
  v = voltage

- TIMEDATA data requirements are:
  
  - A value for t=0 is not required.
  - The signal is assumed to be time periodic with time period equal to maximum time minus minimum time.
  - The DATA option line value specifies that voltage values are to be interpreted as discrete voltage levels with no interpolation between consecutive voltage points, with each voltage level held constant until the next voltage point.
Figure 6-1 shows two .tim example files that result in a time periodic voltage versus time with time period 32 µsec. Example 1 is interpreted as a piecewise linear voltage description; example 2 is interpreted as a voltage state description.

**Time-Domain Waveform Data (.bintim) File in Binary Format**

The general .bintim file format is:

```
NUMBER OF DATA XX1
BEGIN TIMEDATA
  # T ( SEC V R XX)
  % T V
  <binary data block>
```

For binary (.bintim) the line NUMBER OF DATA is added as the first file line to specify the number of time-voltage pairs, XX1. The begin, option, and format lines follow the same rules as for the .tim file. And, there is no END line.

**Agilent Standard Data Format (.dat) Files**

The .dat file is a signal file form used with the Agilent 89400 and 89600 series of test instruments (vector modulation generators/analyzers). Refer to the Agilent Standard Data Format Utilities User’s Guide, Agilent Part No. 5061-8056.
### Example 1
```
BEGIN TIMEDATA
# T ( USEC V R 50. )
% t v
0.0 -1.0
2.0  1.0
4.0  2.0
8.0  3.0
10.0  3.0
14.0  0.0
18.0 -1.0
24.0 -2.0
28.0  0.0
32.0 -1.0
END
```

### Example 2
```
BEGIN TIMEDATA
# T ( USEC DATA R 50. )
% t v
0.0 -1.0
2.0  1.0
4.0  2.0
8.0  3.0
10.0  3.0
14.0  0.0
18.0 -1.0
24.0 -2.0
28.0  0.0
32.0 -1.0
END
```

---

Figure 6-1. Example .tim Files
Chapter 7: Performing Parameter Sweeps

This chapter discusses performing parameter sweeps with Agilent Ptolemy. This capability works in a similar manner to Advance Design System's Analog/RF Systems simulators. In this chapter, we will describe this capability using signal processing examples and point out some things to be aware of when performing sweeps with Agilent Ptolemy.

Introduction

Parameter sweeps are a quick way to conduct a series of simulations while varying a parameter and displaying the output on one plot.

For example, you could analyze a bit error rate (BER) measurement while sweeping the amount of noise added to the design.

Note  In many of Advanced Design System's circuit simulators, sweeps of individual parameters (such as frequency) can be performed from within many of the simulator dialog boxes themselves. However, in signal processing, a Parameter Sweep component must be used.
Performing Parameter Sweeps

Sweeping Parameters Using the ParamSweep Component

To sweep parameters, you need to place and specify parameters for the ParamSweep component. Optionally, you can place a VAR component (variables and equations) to aid in defining terms. Both ParamSweep and VAR components are found in the Controllers component library.

To learn how to use parameter sweeps, we will build a simple design, as shown in Figure 7-1. In this design, we have placed a Waveform source, a Gain component (from the Common Components library), and a Numeric Sink (from the Sinks library). Default values are used for each. We have also placed the required Data Flow controller and a ParamSweep component. The parameter we will sweep is the gain of the Gain component.

![Diagram of a simple design with a Waveform source, a Gain component, and a Numeric Sink, along with a Data Flow controller and a ParamSweep component.](image)

**Figure 7-1. Sweeping a Gain Component**
1. Double-click the **ParamSweep** component to display the Component Parameters dialog box, as shown next.

2. With the Sweep tab displayed (default), in the Parameter to sweep field, enter **G1.Gain**. G1 is the instance name of the Gain component. A period separates this name from the parameter we want to sweep (Gain). With this syntax, you can set up any parameter of any component for sweeping. Additionally, this syntax is hierarchical. If the parameter Gain was in a subnetwork called A, you would use A.G1.Gain.

3. Leave the Sweep Type field set to **Linear**. Other choices are single point and log.

4. With the Start/Stop option button selected, enter the following parameters:
   - Start = 1
   - Stop = 5
   - Step-size = 2
   - Num. of pts. = 3 (this field is calculated by the program)
Performing Parameter Sweeps

5. Now switch to the **Simulations** tab.

6. In the Simulations to perform field, enter **DF** next to Simulation 1. If this field is left blank, the simulation will not be successful. Here you are telling the program to use the Data Flow controller specified by the instance name DF as the simulator. On some occasions, you may have other simulation controllers, each with its own instance name.

7. Click **OK** to accept your changes and dismiss the dialog box.

8. Double-click the Data Flow controller to edit its parameters.

9. Change the Stop parameter to **10.0**.

10. Choose **Simulate > Simulate**.

11. When the simulation is finished, choose **Window > New Data Display**.

12. Place a rectangular plot, add N1 from your dataset, and choose **OK**. Your result should indicate the waveform multiplied by three different Gain values and look similar to the one shown in the next figure.
Sweeping Parameters Using the VAR Component

An alternate way to conduct parameter sweeps is to place a VAR component. A VAR component is used to define variables and equations, and you place one in addition to the ParamSweep component.

The following procedure uses the same design as before, but this time we will place VAR component to help perform the parameter sweep.

Procedure

1. Place a VAR component from the Controllers library anywhere in the schematic.

2. Edit the Gain component so that Gain=XYZ (instead of the default of Gain=1.0).

3. Edit the VAR component so that XYZ=1.0.

4. Edit the ParamSweep component so that SweepVar="XYZ" and SimInstanceName[1]="DF". Leave the rest of the ParamSweep parameters unchanged.
Performing Parameter Sweeps

In this method, you are connecting the Gain parameters with the sweep variable XYZ. Any reference to XYZ in this design would be swept. Further, you may want to use VAR components to define other relationships in a design and add another line to define the parameter to be swept. The use of VAR components allows you a flexible way of building complicated sweep relationships. However, for a simple parameter sweep, it is easiest to use only the ParamSweep component.

Using a Sweep Plan

It is possible to combine sweeps of several parameters or several ranges of one parameter into a single sweep plan. This plan of multiple parameter sweeps is controlled by placing a Sweep Plan in your schematic.

Procedure

1. Place a Sweep Plan component from the Controllers library anywhere in the schematic.
2. Edit the Parameter Sweep component so that the Use sweep plan check box is selected, as shown below. The Sweep tab must be selected before you can check this box.

3. Double-click the Sweep Plan component to edit its parameters.

4. Enter two ranges of gain steps. First, in the Start/Stop field repeat the range that was entered for the single parameter sweep:
   - Start = 1
   - Stop = 5
   - Step-size = 2

5. Choose **Apply**.

6. Next, change the Start/Stop field to:
   - Start = 10
   - Stop = 14
   - Step-size = 2
Performing Parameter Sweeps

7. Choose the **Add** button (left side). This adds our new range to the sweep plan, as shown below.

![Sweep Plan dialog box](image)

8. Click **OK**.

It is possible to place multiple Sweep Plan components that control simulation in a chain of events. To do this, you place additional Sweep Plan components and choose the **Next Sweep Plan** check box (in the lower-center of the Sweep Plan dialog box).

When the simulation is finished, your Data Display window should look similar to the following:

The Gain parameter has now been swept over two ranges.

Note  The placement of sweep components within a design does not affect the order in which parameters are swept. Similarly, the order in which the sweeps are automatically numbered does not determine the order in which they are executed. The order of execution is determined by the order in which one sweep calls another, as determined by the value of the SweepPlan. The simulation component calls the first sweep plan to be conducted, whatever it is named.
Performing Parameter Sweeps

**Sweeping Various Parameter Types**

If you want to sweep a real, integer, or fixed-point parameter type, the procedure is similar to the example just presented.

If you want to sweep complex, precision, array, string, or filename parameter types, you must use a VAR (Variables and equations) component to define the swept variable. You then embed a variable from the VAR in the string of the component parameter value. The reason for this procedure is the simulator only sweeps numbers and these parameter types are strings that are interpreted by the simulator.

An example of sweeping a filename is the case of 10 files, myfile1.dat through myfile10.dat each containing filter coefficients. You might want to sweep a range of these files.

**Note** Earlier versions of Advanced Design System (1.0 and 1.1) required the use of sprintf and strcat functions to reference strings. While no longer needed for complex, precision, or array types, designs built with these functions will still work.

**Sweeping a Complex Waveform Component**

This example, shown in Figure 7-2, shows how to sweep a parameter type that is represented by a string.
The basic parameter sweep set up is similar to our earlier example. Here we use a VAR (Variables and equations—symbol Var Eqn) component (from the Controllers library or palette) to define the swept variable called X. Let's zoom in on the WaveformCx component, which produces a complex waveform.

Figure 7-2. Sweeping a Complex Waveform Component's Value
Performing Parameter Sweeps

The parameter we want to sweep is the imaginary part of the first complex entry in an array of two complex numbers. Since complex arrays are handled as strings in Agilent Ptolemy, we sweep the imaginary part as follows:

Value="(1+j*X)(2+j*4)"

The string "(1-j*X)(2-j*4)" tells the software to evaluate the mathematical expression for variable \(X\) and convert it to a string.

Now if we sweep the variable \(X\) from 0 to 3, we obtain the following result for the magnitude of the output:
Performing Multidimensional Sweeps

Sometimes you need to sweep two or more variables and observe the composite results. An example is analyzing the bit error rate (BER) of a communication system for two modulation schemes at three different power levels. Here, for each of two modulation formats $X$, there are three power levels $Y$. You set up this type of simulation using two Parameter Sweep components.

An example of another multidimensional sweep design is shown Figure 7-3:

![Figure 7-3. Multidimensional Sweep Example Using a DownSample Component](image)

In this example, a RampFloat is used as a source for a DownSample component, with the results stored in a Numeric Sink. The parameters Factor=$X$ and Phase=$Y$ of the DownSample component are swept simultaneously. $X$ is swept from 10 to 20 in steps of 2. $Y$ is swept from 0 to 5 in steps of 5. There are a total of $6 \times 2$ traces in the resulting Data Display window, shown Figure 7-4:
Performing Parameter Sweeps

Figure 7-4. Simulation Results of Multidimensional Sweep Example
Each line is associated with a downsampling factor (X) for a given phase (Y).
Chapter 8: Using Nominal Optimization

This chapter describes using Nominal Optimization with Agilent Ptolemy. Nominal optimization, also called performance optimization, employs iterative simulation to achieve user-specified goals by automatically varying specific simulation design parameter values over user-specified ranges. For example, you could optimize the gain of a carrier recovery loop to achieve a desired lock time and residual loop error or you could optimize a fixed-point bit-width parameter in a DSP design.

This capability generally works the same way as in Advance Design System’s Analog/RF Systems simulators. The Nominal Optimization procedure is found in the manual Tuning, Optimization, and Statistical Design in “Performing Nominal Optimization”.

In this chapter we will present a DSP example on optimizing bit-width and describe some additional information on signal processing parameter types to be aware of when performing optimization with Agilent Ptolemy.
Using Nominal Optimization

## Optimizing Various Parameter Types

If you want to optimize a real, integer, or fixed-point parameter type, the procedure is similar to standard nominal optimization.

If you want to optimize complex, precision, array, string, or filename parameter types, you must use a VAR component to define the optimizable variable. You then embed a variable from the VAR in the string of the component parameter value. The reason for this procedure is the simulator only sweeps numbers and these parameter types are strings that are interpreted by the simulator.

To reference an optimized variable for parameter types that use strings, the variable is defined in a VAR (Variables and equations—symbol Var Eqn) component with the Standard entry mode and Optimization/Statistic Setup enabled. Once defined, this variable may be used as a component parameter, as shown in Example 2, below. If you type this variable on the schematic, it must be enclosed in quotes (" "). If you enter it in the Component dialog box, quotes are automatically added.

An example of optimizing a filename is the case of ten files, myfile1.dat through myfile10.dat, each containing filter coefficients. You might want to conduct an optimization based on a range of these files.

---

**Note** Earlier versions of Advanced Design System (1.0 and 1.1) required the use of sprintf and strcat functions to reference strings. While no longer needed for complex, precision, or array types, designs built with these functions will still work.
Optimizing Input and Output Bit Width

This example shows how to set up a simple fixed-point bit-width parameter optimization. To learn how to do this optimization, we will build a simple design, as shown in Figure 8-1. You can copy it from the examples/Tutorial directory. The project file is dspopt_prj and the design is simpleopt2. Or, you can quickly build this example or just follow along from this manual.

Figure 8-1. Optimizing Input and Output Precision of a Gainfix Component

The value of the input and output precision of Gainfix in component G1 is optimized to achieve a "dc" data value of 0.2 in the sample stored in the numeric sink. The goal is to represent the gain with the minimum number of bits possible.
Using Nominal Optimization

The design consists of the following:

- A Fixed Constant Output (ConstFix) source component (from the Numeric Sources library), with the Level parameter set to 1.0.
- An Integer Constant Output (ConstInt) source component (from the Numeric Sources library), with the Level parameter set to \( D \), as explained in the section “Setting Up the ConstInt, Second Sink, and Second Goal Components” on page 8-6.
- A GainSyn component, with \( \text{Gain}=0.2 \) and \( \text{GainPrecision}="W.D" \).
- A Numeric Sink, with default values accepted.
- A second Numeric Sink, with default values accepted.
- A Data Flow controller, with default values accepted, except set Default Stop to 0.
- A VAR component, set up as described in the next section.
- An Optim component, set up as described in the section “Setting Optimization Job Parameters” in the Tuning, Optimization, and Statistical Design manual. Use the default values, except set MaxIters = 30, and \( P = 2 \). The default optimizer type is the Random optimizer.
- A Goal component, for the expression \( N1 \).
- A second Goal component, for the expression \( N2 \), set up as described in the section “Setting Up the ConstInt, Second Sink, and Second Goal Components” on page 8-6.

The value of 0.2 cannot be exactly represented with only one or two fractional bits (bits to the right of the decimal point). Without optimization, too many bits (such as 16) might be used. While the number 0.2 would be represented very accurately, the extra bits could be wasteful in the final implementation.

By studying this simple example, you can learn the procedure you would use to solve real-world design problems, such as optimizing the bit width of an FIR filter.
Setting Up the VAR Component

Let’s zoom in on the VAR component:

\[
\begin{align*}
\text{VAR} & \text{ Eqn} \\
\text{VAR2} & \\
W &= 2 \\
D &= 1 \text{ opt\{discrete 1 to 16 by 1\} }
\end{align*}
\]

Figure 8-2. VAR Component Parameters

Double-click the VAR component in your schematic to display its dialog box. The following is a description of each parameter we have to set so that the VAR component matches Figure 8-2:

**Note** The labels, such as W or D, are user-defined variable names and could be anything you wanted.

1. Enter \( W = 2 \). W is the number of bits to the left of the decimal point, including the sign bit.
2. Enter \( D = 1 \). D is the number of bits to the right of the decimal point. 1 is our nominal value before optimization. This should be your best estimate for the nominal value.
3. After you enter \( D = 1 \), choose the Optimization/Statistics Setup button.
4. From the Optimization Status drop-down list, select Enabled.
5. From the Type drop-down list, select Discrete.
6. In the Minimum Value field, enter 1.
7. In the Maximum Value field, enter 16.
8. In the Step Value field, enter 1.

These last four steps tell the program to optimize using only the discrete values of 1 through 16 in steps of 1.
9. Click OK to return to the main Variables and Equations dialog box.
10. When done, click OK.
Using Nominal Optimization

**Setting Up the ConstInt, Second Sink, and Second Goal Components**

Optimizing bit width requires placing a second source, ConstInt; a second Numeric Sink; and a second Goal component. Wire the second ConstInt to the second Numeric Sink. The purpose of these secondary components is to create a second optimization goal, which the program will attempt while meeting the criteria of the first optimization goal. Do the following:

1. Edit the ConstInt Component to set Level=D. D is the number of bits to the right of the decimal point, that you defined in the VAR component.
2. Accept the defaults for the second Numeric Sink, N2.
3. Edit the parameters for the second Goal component, OptimGoal2, as follows:
   - Expr = “N2”
   - SimInstanceName = “DF1”
   - Min = 0
   - Max = 0
   - Weight = 1
   - RangeVar = Index
   - RangeMin = 0
   - RangeMax = 0

**Weight Parameter**

Keep in mind that the Weight parameter weighs the importance of one goal to the other goal(s). Generally, the first goal may be more important, such as when it meets a performance specification, such as frequency response. The second goal (in this example, bit width) is weighted less. Because the error function of the first goal is small compared to the second goal, the Weight of the first goal is set to 1e9.

**Completing the Optimization**

You are now ready to complete the optimization.

The remainder of the procedure for completing and running the optimization for parameter types such as precision or string, as described in this section, are the same as for any optimization. To review these procedures, refer to “Specifying Component Parameters for Optimization” in the Tuning, Optimization, and Statistical Design manual for details.
Chapter 9: Theory of Operation

Agilent Ptolemy provides signal processing simulation for Advanced Design System's specialized design environments. Each of these specialized design environments capture a model of computation, called a domain, that has been optimized to simulate a subset of the communication signal path. Advanced Design System domains that are part of Agilent Ptolemy, or can cosimulate with Agilent Ptolemy are:

<table>
<thead>
<tr>
<th>Domain</th>
<th>Simulation Technology</th>
<th>Controller</th>
<th>Application Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous Dataflow (SDF)</td>
<td>Numeric dataflow</td>
<td>Data Flow</td>
<td>Synchronous multirate signal processing simulation</td>
</tr>
<tr>
<td>Timed Synchronous Dataflow (TSDF)</td>
<td>Timed dataflow</td>
<td>Data Flow</td>
<td>Baseband and RF functional simulation (e.g., antenna and propagation models, timed sources)</td>
</tr>
<tr>
<td>Circuit Envelope</td>
<td>Time- and frequency-domain analog</td>
<td>Envelope</td>
<td>Complex RF simulation</td>
</tr>
<tr>
<td>Transient</td>
<td>Time-domain analog</td>
<td>Transient</td>
<td>Baseband analog simulation</td>
</tr>
</tbody>
</table>

In Agilent Ptolemy, a complex system is specified as a hierarchical composition (nested tree structure) of simpler circuits. Each subnetwork is modeled by a domain. A subnetwork can internally use a different domain than that of its parent. In mixing domains, the key is to ensure that at the interface, the child subnetwork obeys the semantics of the parent domain.

Thus, the key concept in Agilent Ptolemy is to mix models of computation, implementation languages, and design styles, rather than trying to develop one, all-encompassing technique. The rationale is that specialized design techniques are more useful to the system-level designer, and more amenable to a high-quality, high-level synthesis of hardware and software. In the next sections we will describe the SDF and TSDF. For general documentation on the Circuit Envelope and Transient simulators refer to the Circuit Envelope and RF Transient/Convolution Simulation chapters in the Circuit Simulation manual. For information on cosimulation with Agilent Ptolemy and these circuit simulators, refer to Chapter 11, Cosimulation with Analog/RF Systems.
Synchronous Dataflow (SDF)

SDF is a special case of the dataflow model of computation, which was developed by Dennis\(^1\). The specialization of the model of computation is to those dataflow graphs where the flow of control is completely predictable at compile time. It is a good match for synchronous signal processing systems, those with sample rates that are rational multiples of one another.

The SDF domain is suitable for fixed and adaptive digital filtering, in the time or frequency domains. It naturally supports multirate applications, and its rich component library includes polyphase FIR filters.

Advanced Design System's Examples directories contain numerous application examples which rely on SDF semantics. To view these examples, chose File > Example Project, a dialog box appears. Select the DSP/ dsp_demos_prj directory for one group of SDF examples.

Synchronous dataflow (SDF) is a data-driven, statically scheduled domain in Agilent Ptolemy. It is a direct implementation of the techniques given by Lee\(^2,3\). Data-driven means that the availability of data at the inputs of a component enables it. Components without any inputs are always enabled. Statically scheduled means that the firing order of the components is periodic and determined once, during the start-up phase. It is a simulation domain, but the model of computation is the same as that used for bit-true simulation of synthesizable hardware used by the DSP Synthesis tool. A number of different schedulers have been developed for this model of computation.
Basic Dataflow Terminology

The SDF dataflow model is equivalent to the computation graph model of Karp and Miller. In the terminology of the dataflow literature, components are called actors. An invocation of a component is called a firing. The signal carried along the arc connecting the blocks are made of individual packets of data called tokens. In a digital signal processing system, a sequence of tokens might represent a sequence of samples of a speech signal or a sequence of frames in a video sequence.

Note Some Agilent Ptolemy terminology is different from UCB Ptolemy terminology. For example, a component in Advanced Design System is called a star in UCB Ptolemy and an arc is a wire. Refer to the Glossary for more information.

When an actor fires, it consumes some number of tokens from its input arcs, and produces some number of output tokens. In synchronous dataflow, these numbers remain constant throughout the execution of the system. It is for this reason that this model of computation is suitable for synchronous signal processing systems, but not for asynchronous systems. The fact that the firing pattern is determined statically is both a strength and a weakness of this domain. It means that long runs can be very efficient, a fact that is heavily exploited in Agilent Ptolemy and the Agilent DSP Synthesis tool. But it also means that data-dependent flow of control is not allowed. This would require dynamically changing firing patterns.

Balancing Production and Consumption of Tokens

Each port of each SDF component has an attribute that specifies the number of tokens consumed (for inputs) or the number of tokens produced (for outputs). When you connect an output to an input with an arc, the number of tokens produced on the arc by the source component may not be the same as the number of tokens consumed from that arc by the destination component. To maintain a balanced system, the scheduler must fire the source and destination components with different frequency.

Figure 9-1. Simple Connection of SDF Components Illustrates Balance Equations

Constructing a Schedule
Consider a simple connection between three components, as shown in Figure 9-1. The symbols adjacent to the ports, such as $N_{A_1}$, represent the number of tokens consumed or produced by that port when the component fires. For many signal processing components, these numbers are simply one, indicating that only a single token is consumed or produced when the component fires. But there are three basic circumstances in which these numbers differ from one:

- Vector processing in the SDF domain can be accomplished by consuming and producing multiple tokens on a single firing. For example, a component that computes a fast Fourier transform (FFT) will typically consume and produce $2^M$ samples when it fires, where $M$ is some integer. Examples of vector processing components that work this way are FFT_Cx, Average, and FIR. This behavior is quite different from the matrix components, which operate on tokens where each individual token represents a matrix.

- In multirate signal processing systems, a component may consume $M$ samples and produce $N$, thus achieving a sampling rate conversion of $N/M$. For example, the FIR component can optionally perform such a sampling rate conversion, and with an appropriate choice of filter coefficients, can interpolate between samples. Other components that perform sample rate conversion include UpSample, DownSample, and Chop.

- Multiple signals can be merged using components such as Commutator or a single signal can be split into subsignals at a lower sample rate using the Distributor component.

To be able to handle these circumstances, the scheduler first associates a simple balance equation with each connection in the graph. For the graph in Figure 9-1, the balance equations are

\[
\begin{align*}
  r_A N_{A_1} &= r_C N_{C_1} \\
  r_A N_{A_2} &= r_B N_{B_1} \\
  r_B N_{B_2} &= r_C N_{C_2}
\end{align*}
\]

This is a set of three simultaneous equations in three unknowns. The unknowns, $r_A$, $r_B$, and $r_C$ are the repetitions of each actor that are required to maintain balance on each arc. The first task of the scheduler is to find the smallest non-zero integer solution for these repetitions. It is proven in Lee¹ that such a solution exists and is unique for every SDF graph that is “consistent,” as defined below.

---

How Schedulers Work in Agilent Ptolemy

Synchronous Dataflow (SDF) is a restricted version of dataflow in which the number of data produced or consumed by a component per invocation is known at compile time. As the name implies, SDF can be used to model synchronous signal processing algorithms. In these algorithms, all of the sampling rates are rationally related to one another.

An SDF scheduler takes a simulation design and determines the sequence or order of invocation of each component. The simulator will simulate the design according to the schedule generated by the scheduler. Agilent Ptolemy provides three user-selectable schedulers for any given simulation:

• Classical Scheduler
• Cluster Loop Scheduler
• Acyclic Loop Scheduler
• Multithreaded Scheduler

The trade-offs between them are typically the time needed to generate the schedule, the memory usage of storing the schedule data structure, and the memory usage of buffers for data collection.

Classical Scheduler

This is the classic scheduler from UC Berkeley Ptolemy, which tries to minimize the buffer sizes. It tries to put off the firing of any block as long as possible. A component firing is deferred until none of the components that feeds data to it can be fired. It usually takes longer to generate the schedule than other schedulers, and has a large schedule size, but it uses less memory buffers across components. This scheduler is good for uni-rate designs.

Cluster Loop Scheduler

This scheduler generates single-appearance schedules that take less time than the Classical scheduler. The advantage of a single-appearance schedule is that it significantly reduces schedule size. Each component appears once in the schedule and possibly with a loop factor. However, the buffer memory usage will be increased by the loop factor. Most of the increased buffer memory usage can be reduced by clustering components and limiting the buffer increases to only between clusters. This scheduler is good for multirate designs such as wireless 2.5/3G designs.
Theory of Operation

**Acyclic Loop Scheduler**

This scheduler generates single-appearance looped schedules for acyclic graphs to optimizing both the schedule and buffer sizes by using a “recursive partitioning by minimum cuts” heuristic. More information about the algorithms and heuristics can be found in chapters 6 and 7 of Software Synthesis from Dataflow Graphs (http://ptolemy.eecs.berkeley.edu/~murthy/book.html) by Shuvra S. Bhattacharyya, Praveen K. Murthy, and Edward A. Lee, published by Kluwer Academic Publishers, Norwood, MA, 1996. This scheduler is good for multirate designs without cycles.

**Multithreaded Scheduler**

This Scheduler is based on the Cluster Looped Scheduler which utilizes multithreading techniques to achieve speedup on multiprocessor machines. It tries to extract parallelism from Synchronous Dataflow graphs and distributes the load among multiple threads. The actual speedup depends on the complexity of designs and the number of processors used. However, speedup does not scale linearly to the number of processors. For more information refer to “Multithreaded Synchronous Dataflow” on page 9-13.

**Iterations in SDF**

At each SDF iteration, each component is fired the minimum number of times to satisfy the balance equations.

Suppose for example that component B in Figure 9-1 is an FFT_Cx component with its parameters set so that it will consume 128 samples and produce 128 samples. Suppose further, that component A produces exactly one sample on each output, and component C consumes one sample from each input. In summary,

\[
N_{A1} = N_{A2} = N_{C1} = N_{C2} = 1
\]

\[
N_{B1} = N_{B2} = 128.
\]

The balance equations become

\[
\begin{align*}
  r_A &= r_C \\
  r_A &= 128r_B \\
  128r_B &= r_C.
\end{align*}
\]
The smallest integer solution is

\[
\begin{align*}
   r_A &= r_C = 128 \\
   r_B &= 1.
\end{align*}
\]

Hence, each iteration of the system includes one firing of the FFT\_Cx component and 128 firings each of components A and B.

**Inconsistency**

It is not always possible to solve the balance equations. Suppose that in Figure 9-1 we have

\[
\begin{align*}
   \psi_{A1} &= N_{A2} = N_{C1} = N_{C2} = N_{B1} = 1 \\
   N_{B2} &= 2.
\end{align*}
\]

In this case, the balance equations have no non-zero solution. The problem with this system is that there is no sequence of firings that can be repeated indefinitely with bounded memory. If we fire A, B, C in sequence, a single token will be left over on the arc between B and C. If we repeat this sequence, two tokens will be left over. Such a system is said to be inconsistent, and is flagged as an error. The SDF scheduler will refuse to run it.

**Deadlocks**

Figure 9-2. A Deadlocked SDF System

While scheduling a system, it is possible that we are not able to complete all of the firings specified in the repetition vector because none of the remaining components are enabled. Such a system is said to be a deadlock state. Figure 9-2 shows a system in such a state. The repetitions vector for this system is:

\[
\begin{align*}
   r_A &= 1 \\
   r_B &= 1.
\end{align*}
\]
Theory of Operation

Thus this system is consistent; however, neither A nor B are enabled because each is waiting for one token from the other. The way to resolve this deadlock is to introduce an initial token on one of the two arcs in the figure. This initial token is known as a delay.

The Delay component is indicated by a component with a diamond that is connected to an arc. The delay has a single parameter, the number of samples of delay to be introduced. In the SDF domain, a delay with parameter equal to one is simply an initial token on an arc. This initial token may enable a component, assuming that the destination component for the delay arc requires one token in order to fire. To avoid deadlock, all feedback loops must have delays. The SDF scheduler will flag an error if it finds a loop with no delays. For most token types, the initial value of a delay will be zero.

By default, a delay has a zero value. To specify a specific value for the initial token, use the InitDelay token.

There are a number of specialized components in Agilent Ptolemy that add a delay on an arc, such as DelayRF, VcDelayRF, ShiftRegPPSyn, ShiftRegPSSyn, ShiftRegSPSyn, and CounterSyn. Delay_M is used for matrices.

Deadlock Resolution

In Agilent Ptolemy, there is an optional deadlock resolution algorithm that can determine where Delay components need to be inserted. If the user desires, the delay components can be automatically spliced in.

For the Data Flow controller, an option item called Deadlock Management is listed. There are three options:

- Report deadlock. The Agilent Ptolemy simulator will simply report deadlocks if the schedule has failed because of a deadlock.

- Identify deadlocked loops. A new algorithm is turned on to locate where the problem occurs. By using this algorithm, Agilent Ptolemy highlights each loop that has deadlocked.

- Resolve deadlock by inserting tokens. Agilent Ptolemy will resolve the deadlock automatically by inserting delays. This option must be used with care. In general, there are many places Agilent Ptolemy can insert a delay to break a deadlock. Each of these cases can lead to different simulation results.
Timed Synchronous Dataflow (TSDF)

TSDF is an extension of SDF described in the previous section. TSDF adds a Timed data type, which is described in Chapter 5, Using Data Types. For each token of the Timed type, both a time step and a carrier frequency must be resolved.

Advanced Design System's Examples directories contain numerous application examples that rely on TSDF semantics. To view these examples, choose File > Example Project, a dialog box appears. Select the DSP/ModemTimed_prj directory for one group of TSDF examples.

You should note that the Agilent Ptolemy (Pt) simulation time domain signals are different from those used for Circuit Envelope (CE) and Transient (T) simulation.

• For CE, the simulation time step is set by the simulation controller and is constant for the duration of the simulation. This is global for all components simulated in the design.

• For T, the simulation maximum time step is set by the simulation controller, but the actual simulation time step may vary for the duration of the simulation. This simulation time step is global for all components simulated in the design.

• For Pt, the simulation time step is not global and the input signals for different components may have a different time step. However, the simulation time step at each node of a design, once set at time=0, remains constant for the duration of the simulation. The time step is initially set by the time domain signal sources or numeric to timed converters. The time step in the data flow graph may be further changed due to upsample (decreases the time step by the upsampling factor) and/or down sample (increases the time step by the downsample factor) components. The time step associated with signals at the inputs of the timed sinks may or may not be the same as the one that was initiated by the timed sources or numeric to timed converters. This is dependant on any up or down sampling that may have occurred in the signal flow graph.

• For CE and T, all time domain data sent to the ADS Data Display has the same global time value.

• For Pt, each timed sink that sends data to the ADS Data Display has time values that are specific to the individual sink and may or may not be the same as the time values associated with data from other timed sinks.
Theory of Operation

- For CE, a time domain signal may have more than one carrier frequency associated with it concurrently. The carrier frequencies in the simulation are harmonically related to the frequencies defined in the CE controller and their translated values that result from nonlinear devices.

- For Pt, a time domain signal has only one characterization frequency associated with it. This characterization frequency is also typically the signal carrier frequency. However, the term “carrier frequency” is more typically used to mean the RF frequency at which signal information content is centered. A signal has one characterization frequency, but the signal represented may have information content centered at one or more carrier frequencies. This would occur when several RF bandpass signals at different carrier frequencies are combined to form one total composite RF signal containing the full information of the multiple carriers.

Example

Two RF signals and their summation:

\[
\begin{align*}
A_{rf} &= A_i \cos(w_a t) - A_q \sin(w_a t) \text{ with carrier frequency } w_a \\
B_{rf} &= B_i \cos(w_b t) - B_q \sin(w_b t) \text{ with carrier frequency } w_b \\
\text{The summation of these two signals, } C_{rf}, \text{ can be represented at one} \\
& \text{characterization frequency, } w_c, \text{ as follows:} \\
C_{rf} &= C_i \cos(w_c t) - C_q \sin(w_c t) \text{ with carrier frequency } w_c \\
& \text{where} \\
w_c &= \max(w_a, w_b) \\
C_i &= A_i + B_i \cos((w_a-w_b)t) + B_q \sin((w_a-w_b)t) \text{ (assuming } w_a > w_b) \\
C_q &= A_q - B_i \sin((w_a-w_b)t) + B_q \cos((w_a-w_b)t) \text{ (assuming } w_a > w_b)
\end{align*}
\]

Time Step Resolution

In TSDF, each Timed arc has an associated time step. This time step specifies the time between each sample. Thus the sampling frequency for the envelope of a Timed arc is 1/time step.

The sampling frequency is propagated over the entire graph, including both Timed and numeric arcs. To calculate a time step, the SDF input and output numbers of tokens consumed/produced are used.

For any given SDF or TSDF component, the sampling frequency of the component is defined as the sampling frequency on any input (or output) divided by the consumption (or production) SDF parameter on that port. After a sampling frequency is derived for a given component, it is propagated to every port by multiplying the
component's rate with the SDF parameter of the port. A sample rate inconsistency error message is returned if inconsistent sample rates are derived.

**Carrier Frequency Resolution**

Each Timed arc in a timed dataflow system has an associated carrier frequency ($F_c$). These $F_c$ values are used when a conversion occurs between Timed and other data types, as well as by the Timed components.

The $F_c$ has either a numerical value, which is greater than or equal to zero ($F_c \geq 0.0$), or is undefined ($F_c = \text{UNDEFINED}$). All Timed ports have an associated $F_c \geq 0.0$. Non-timed ports have an UNDEFINED $F_c$.

During the simulation, all the $F_c$ values associated with all of Timed ports are resolved by the simulator. The resolution algorithm begins by propagating the $F_c$ specified by the user in the Timed sources parameter $F_{carrier}$ until all ports have their associated $F_c$. At times, the user may have specified incompatible carrier frequencies, and Agilent Ptolemy will return an error message.

In the feedforward designs, the algorithm will converge quickly to a unique solution. In the designs with feedback, the algorithm takes additional steps to resolve the carrier frequency at all pins.

For feedback paths, a default $F_c$ is assigned by the simulator. This default $F_c$ is then propagated until the $F_c$ converges on the feedback path. This $F_c$ is occasionally non-unique. To specify a unique value, use the SetFc Timed component.

**Input/Output Resistance**

Resistors can be used with timed components. Resistors provide a means to support analog/RF component signal processing. They provide definition of analog/RF input and output resistance, additive resistive Gaussian thermal (Johnson) noise, and power-level definition for time-domain signals.

Though resistors are circuit components, they are used in the data flow graph by defining their inputs from the outputs of connected TSDF components and their outputs at connected TSDF component inputs.
The above figure shows two TSDF components, T1 and T2, with an interconnected series resistor, R1, at the output of T1 and a shunt resistor, R2, at the input of T2. Such interconnected resistors are collected and replaced with the appropriate signal transformation model that includes time-domain signal scaling and additive thermal noise.

Resistors contribute additive thermal noise (kT\*B) to signals when the specified resistance temperature (R\text{Temp}) is greater than absolute zero (-273.16 degrees Celsius) where:

\[
\begin{align*}
k &= \text{Boltzmann's constant} \\
T &= \text{temperature in Kelvin} \\
B &= \text{simulation frequency bandwidth} \\
1/2\ T\text{Step}: \text{if signal is a timed baseband signal} \\
1\ T\text{Step}: \text{if signal is a timed complex envelope signal}
\end{align*}
\]
Multithreaded Synchronous Dataflow

The Multithreaded scheduler is targeted for multiprocessors workstations. It can significantly reduce running time of multi-rate SDF simulations. The basic idea behind the scheduler is to realize the potential parallelism that exists in an SDF graph and transform the efficient uniprocessor hierarchical cluster looped schedule into a multithreaded parallel schedule for simulation environment.

Parallelism with Clustering

The main idea behind Clustering Looped Scheduling is to limit the explosion of nodes in an SDF schedule while minimize the buffer memory requirement. Figure 9-4 shows a chain-structured multi-rate SDF graph. Using the classical scheduling to minimize the buffer usage, a component will fire when their input requirement is fulfilled. The schedule will use the minimum amount of buffer $4+1+4=9$ but the schedule will consist of 41 nodes, ABCABCDABC..., before it repeats again.

![Figure 9-4.](image)

Using Looped Scheduling, the schedule becomes $(9A)(12B)(12C)(8D)$ which greatly reduces the nodes into four where each has a loop factor. Unfortunately, the buffer memory requirement has gone up to $36+12+24=72$. With the help of clustering, the schedule can be transformed into $(3(3A)(4B))(4(3C)(2D))$, grouping A and B into cluster $\alpha$, C and D into cluster $\beta$. The resultant memory requirement is $12+12+6=30$, not as good as the minimal but less than half of Looped Scheduling.
Now if we apply $\alpha$ and $\beta$ using pipelining on dual processors machine, two threads can run independently, one thread fires 21 components and the other 20 components before they synchronize at the end of the time slot. Let us assume that each component firing consumes 2 cycles, and thread synchronization takes 10 cycles. A single processor will finish the schedule in 82 cycles. In the 2 processor scenario, it will take $\text{max}(42,40)+10=52$ cycles to finish the schedule. Furthermore, one can do schedule unfolding to increase the workload before the synchronization of each thread to generate higher throughput. The drawback with schedule unfolding is the proportional increase in memory buffer requirement.

Note: Data dependency issues and exact algorithms to increase parallelism are beyond the scope of this manual and will not be discussed here.

Memory Overhead

The classic trade off between memory and speed happens in Multithreaded SDF scheduling. When loop unfolding is in effect (applicable to small designs only), the buffer size increase is proportional to the number unfolding. This is in contrast to pipelining of clusters, where buffer size is doubled. Furthermore, only the buffers which bridge across clusters, are doubled and buffers connecting components within a single cluster remain the same size.
Sample Results

A subset of WLAN and 3GPP designs results are provided here. All designs are shipped as example designs. Results are gathered by running the designs on a dual processor PC (specifications of PC) with both the Cluster Looped and Multithreaded Scheduler:

<table>
<thead>
<tr>
<th>Wireless Lan</th>
<th>WCDMA</th>
<th>Seconds</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>WLAN_80211a_RxNonAdjCh_12Mbps.dsn</td>
<td>5090</td>
<td>3512</td>
<td>1.45</td>
</tr>
<tr>
<td>UE_Rx_In_Band_Blocking.dsn</td>
<td>2701</td>
<td>1406</td>
<td>1.92</td>
</tr>
<tr>
<td>WLAN_80211a_RxAdjCh_9Mbps.dsn</td>
<td>2634</td>
<td>2058</td>
<td>1.28</td>
</tr>
<tr>
<td>WLAN_80211a_RxNonAdjCh_48Mbps.dsn</td>
<td>2230</td>
<td>1006</td>
<td>2.22</td>
</tr>
<tr>
<td>WLAN_80211a_RxSensitivity_6Mbps.dsn</td>
<td>1762</td>
<td>1949</td>
<td>0.90</td>
</tr>
<tr>
<td>WLAN_80211a_ChannelCoding.dsn</td>
<td>1567</td>
<td>1229</td>
<td>1.28</td>
</tr>
<tr>
<td>WLAN_80211a_RxAdjCh_18Mbps.dsn</td>
<td>1485</td>
<td>755</td>
<td>1.97</td>
</tr>
<tr>
<td>WLAN_80211a_GI.dsn</td>
<td>1261</td>
<td>1138</td>
<td>1.11</td>
</tr>
<tr>
<td>UE_Rx_ACS.dsn</td>
<td>1245</td>
<td>500</td>
<td>2.49</td>
</tr>
<tr>
<td>WLAN_80211a_RxAdjCh_36Mbps.dsn</td>
<td>976</td>
<td>517</td>
<td>1.89</td>
</tr>
<tr>
<td>UE_Rx_MaxLevel.dsn</td>
<td>823</td>
<td>329</td>
<td>2.50</td>
</tr>
<tr>
<td>UE_Rx_MaxLevel.dsn</td>
<td>815</td>
<td>333</td>
<td>2.45</td>
</tr>
<tr>
<td>WLAN_80211a_RxSensitivity_24Mbps.dsn</td>
<td>826</td>
<td>429</td>
<td>1.46</td>
</tr>
<tr>
<td>WLAN_80211a_TxEVM_Turbo.dsn</td>
<td>617</td>
<td>360</td>
<td>1.71</td>
</tr>
</tbody>
</table>

Average Speedup 1.76

The simulation controls associated with the above designs are modified such that the simulation time is normalized to fall between 10 mins and an hour and a half. The average speedup for a two processors machine is around 1.76x.
Theory of Operation

The next table of designs indicate a set which takes a much shorter time (3 - 10 mins). The idea was to observe the speedup among designs that take less time to simulate.

<table>
<thead>
<tr>
<th>Design</th>
<th>Seconds</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wireless Lan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WLAN_80211a_TxEVM.dsn</td>
<td>578</td>
<td>437</td>
</tr>
<tr>
<td>UpLk_1DPCCH_5DPDCH.dsn</td>
<td>519</td>
<td>298</td>
</tr>
<tr>
<td>UpLk_1DPCCH_6DPDCH.dsn</td>
<td>510</td>
<td>294</td>
</tr>
<tr>
<td>DnLk_DPCH.dsn</td>
<td>495</td>
<td>243</td>
</tr>
<tr>
<td>UpLk_1DPCCH_3DPDCH.dsn</td>
<td>489</td>
<td>289</td>
</tr>
<tr>
<td>DnLk_TestModel.dsn</td>
<td>488</td>
<td>250</td>
</tr>
<tr>
<td>UpLk_1DPCCH.dsn</td>
<td>484</td>
<td>244</td>
</tr>
<tr>
<td>DnLk_PCCPCH_SCH.dsn</td>
<td>482</td>
<td>241</td>
</tr>
<tr>
<td>UpLk_1DPCCH_4DPDCH.dsn</td>
<td>481</td>
<td>287</td>
</tr>
<tr>
<td>UpLk_1DPCCH_1DPDCH.dsn</td>
<td>473</td>
<td>273</td>
</tr>
<tr>
<td>WLAN_80211a_RxSensitivity_54Mbps.dsn</td>
<td>402</td>
<td>284</td>
</tr>
<tr>
<td>UpLk_1DPCCH_2DPDCH.dsn</td>
<td>349</td>
<td>214</td>
</tr>
<tr>
<td>WLAN_80211a_TxSpectrum.dsn</td>
<td>301</td>
<td>197</td>
</tr>
<tr>
<td>3GPPFDD_UE_Tx_12_2.dsn</td>
<td>274</td>
<td>246</td>
</tr>
<tr>
<td>UE_Rx_RefLevel_PhyBER.dsn</td>
<td>249</td>
<td>149</td>
</tr>
<tr>
<td><strong>Average Speedup</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

As seen here, shorter simulation time gives a slightly smaller speedup. This is due to the overhead of the Multithread scheduler. Again, speedup depends on the parallelism of a design.

Reentrancy

Due to reentrancy issues, some simulations may not demonstrate a desired speedup improvement. For example, Analog/RF cosimulation limitations allow only one processor to perform execution in Analog/RF portion. Multithreading will only be on for the Ptolemy portion.
Thread Safe Programming

In order to satisfy reentrancy, some components have serial regions to assure proper multithreading simulations. Custom models can usually be made thread-safe by not relying on static data and the communication between models with global data.

It is not always possible to avoid using this type data structure when designing a model. When it is used, you will be responsible for any data synchronization that is necessary.

The Netscape Portable Runtime Library, http://www.mozilla.org/projects/nspr/, is a free public library for multithread programming that is shipped with ADS. The following is an example on how to guard against a static variable.

```c
Defstar {
    Name { myStar }
    Domain { SDF }

    hinclude { "prlock.h" }

    protected {
        static int sharedData;
        static PRLock* sampleLock;
    }

    code {
        PRLock* SDFmyStar::sampleLock = PR_NewLock();
    }

    go {
        PR_Lock(sampleLock);
        sharedData++;
        PR_Unlock(sampleLock);
    }
}
```

First, prlock.h is included so we can use the Lock functions provided by NSPR. Assume that we have a variable, which is shared by multiple instances of SDF myStar, called sharedData. In order to make the model thread safe, we declare another static variable sampleLock for guarding the access of sharedData. In the go method, we create a critical section that guarantees only one thread at a time will go into the critical section, accessing the variable sharedData.
Note  The current version of Tcl/Tk library shipped with ADS is not thread-safe. Therefore, designs with Tcl/Tk components are not allowed to be used with the Multithreaded Scheduler.

References


Chapter 10: Introduction to MATLAB Cosimulation

The MATLAB models provide an interface between Agilent Ptolemy and MATLAB, a numeric computation and visualization environment from The MathWorks, Inc. Each MATLAB script-interpreting model can contain a MATLAB function, command, statement, or several statements. A MATLAB library-importing model can contain a .m script file or a precompiled shared library. Agilent Ptolemy handles the conversion of data to and from MATLAB. You should expect substantial speed improvement using library-importing models over script-interpreting models.

Setting Up MATLAB

Agilent Ptolemy Matlab Cosimulation requires MATLAB, and supports MATLAB versions 6.1 (Release 12.1) and 6.5 (Release 13) or higher on all platforms.

Under all platforms, set the MATLAB configuration variable in hpads.cfg to the root of your MATLAB installation. For example, you might set it to

```
MATLAB=/usr/local/matlab
```
on a UNIX platform, or

```
MATLAB=c:/matlab
```
on a Windows platform. For more details on the hpads.cfg file, see the Advanced Design System installation documentation for your platform.

Under UNIX, if the command to invoke MATLAB is not `matlab`, set the MATLABCMD configuration variable to the correct command. For example, you might set it to

```
MATLABCMD="matlab -c /path/to/license/file"
```
so that matlab correctly finds its license file. Most people won’t need to set the MATLABCMD variable. The variable’s setting is ignored under Windows.

Finally, under Windows, you will need to install MATLAB COM Automation (ActiveX) in the Windows registry. To install the entries, run:

```
matlab /Regserver
```
from the command line. MATLAB will register itself and remain running and minimized. At that point, you should exit MATLAB. You need only do this once. For
more details, refer to “COM and DDE Support” in the MATLAB External Interfaces/API documentation.

If a MATLAB model is run and MATLAB has not been set up correctly, then Agilent Ptolemy will report an error. All MATLAB models in a simulation send their commands to the same MATLAB process.

In order to generate a MATLAB shared library using MATLAB library-importing components, MATLAB Compiler 3.0 or above with a valid license must be installed on the system. Make sure that $MATLAB/bin/$platform and $MATLAB/extern/lib/$platform are in the library search path. Use the following names for each platform:

<table>
<thead>
<tr>
<th>Platform</th>
<th>Library Path Variable</th>
<th>$platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>HP-UX</td>
<td>SHLIB_PATH</td>
<td>hpux</td>
</tr>
<tr>
<td>Sun</td>
<td>LD_LIBRARY_PATH</td>
<td>sol2</td>
</tr>
<tr>
<td>Windows</td>
<td>PATH</td>
<td>win32</td>
</tr>
</tbody>
</table>

An example of setting the path on HP-UX with C-Shell is

   setenv SHLIB_PATH $MATLAB/bin/hpux:$MATLAB/extern/lib/hpux

Currently, MATLAB shared library is supported on all ADS-supported platforms except Linux. To verify installation, run:

   mcc

from the command line. Usage of mcc and command line options will be displayed if the MATLAB Compiler has been set up correctly.
Simulating with MATLAB (Script-Interpreting)

MATLAB distinguishes between complex matrices and floating-point (real) matrices. Agilent Ptolemy distinguishes between models that produce data and models that don’t. As a consequence, there are six MATLAB models in Agilent Ptolemy, two producing floating-point (real) matrices, two producing complex-valued matrices and two, sinks, that produce nothing. All the models can accept any number of inputs provided that the inputs have the same data type, floating-point (real) or complex.

The models are:

<table>
<thead>
<tr>
<th>Model</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matlab_M</td>
<td>Evaluates a MATLAB expression and outputs the result as floating-point (real) matrices.</td>
</tr>
<tr>
<td>MatlabCx_M</td>
<td>Evaluates a MATLAB expression and outputs the result as complex-valued matrices.</td>
</tr>
<tr>
<td>MatlabSink</td>
<td>Evaluates a MATLAB expression a fixed number of times.</td>
</tr>
<tr>
<td>MatlabF_M</td>
<td>Evaluates a MATLAB script file and outputs the results as floating-point (real) matrices.</td>
</tr>
<tr>
<td>MatlabFCx_M</td>
<td>Evaluates a MATLAB script file and outputs the results as complex-valued matrices.</td>
</tr>
<tr>
<td>MatlabSinkF</td>
<td>Evaluates a MATLAB script file a fixed number of times.</td>
</tr>
</tbody>
</table>

The models all use a common MATLAB engine interface that is managed by a base MATLAB model. The base model does not have any inputs or outputs. It provides methods for starting and killing a MATLAB process, evaluating MATLAB commands, managing MATLAB figures, changing directories in MATLAB, and passing Agilent Ptolemy matrices into and out of MATLAB. Currently, the base model supports only 2-D real and complex-valued matrices.

The MATLAB interpreter’s working directory is set to the ScriptDirectory parameter, if it is given. Any custom MATLAB models will be searched from there, and any output files will be written there.

Figures generated by a MATLAB model are managed according to the value of the DeleteOldFigures parameter. If this parameter is YES, the MATLAB model will close any MATLAB plots or graphics when it is destroyed. If NO, the figures must be manually closed.
Writing Functions for the MATLAB Models
(Script-Interpreting)

There are several ways in which MATLAB commands can be specified in the MATLAB models in the MatlabFunction parameter.

If only a MATLAB function name is given for this parameter, the function is applied to the inputs in order. The function's outputs are sent to the model's outputs.

For example, specifying `eig` means to perform the eigendecomposition of the input. The function will be called to produce one or two outputs, according to how many output ports there are. If there is a mismatch in the number of inputs and outputs between the Agilent Ptolemy model and the MATLAB function, then an error will be reported by MATLAB.

You may also explicitly specify how the inputs are to be passed to a MATLAB function and how the outputs are taken from the MATLAB function. For example, consider a two-input, two-output MATLAB model to perform a generalized eigendecomposition. The command

```
[output#2, output#1] = eig( input#2, input#1 )
```

says to perform the generalized eigendecomposition on the two-input matrices, place the generalized eigenvectors on output#2, and the eigenvalues (as a diagonal matrix) on output#1. Before this command is sent to MATLAB, the pound characters “#” are replaced with the underscore character “_” because the pound character is illegal in a MATLAB variable name.

The MATLAB models also allow a sequence of commands to be evaluated. Continuing with the previous example, we can plot the eigenvalues on a graph after taking the generalized eigendecomposition:

```
[output#2, output#1] = eig( input#2, input#1 ); plot( output#1 )
```

When entering such a collection of commands in Agilent Ptolemy, both commands appear on the same line without a new line after the semicolon. In this way, very complicated MATLAB commands can be built up. We can make the plot of eigenvalues always appear in the same plot without interfering with other plots generated by other MATLAB models with this function. (New lines are inserted after the semicolons to improve readability.)
[output#2, output#1] = eig(input#2, input#1);
if ( exist(’myEigFig’) == 0 ) myEigFig = figure; end;
figure(myEigFig);
plot(output#1);

The parameters MatlabSetup and MatlabWriteUp are called during the model's begin and wrap-up procedures. During each of these procedures, there is no data passing into or out of the model.

Because the same MATLAB interpreter is used for the entire simulation, variables are preserved from iteration to iteration. For example, the output of a Matlab_M model with settings:

MatlabSetUp = “x=ones(2;1)”
MatlabFunction = “output#1=x(2)/x(1); x=[x(2),sum(x)];”

will converge on the golden mean. It is impossible, however, to share variables between different MATLAB components. Such a simulation would be non-deterministic.

Simulating with MATLAB (Library-Importing)

Two models that allow cosimulation between MATLAB and ADS through a native shared library provide substantial speed improvement over script interpreting. It utilizes MATLAB Compiler to convert .m script files to shared libraries. It also allows sharing libraries without exposing source files. You should be able to import a MATLAB .m script file directly without any code change.

The models are:

<table>
<thead>
<tr>
<th>Model</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MatlabLibLink</td>
<td>Evaluates a MATLAB expression and outputs the result as floating-point (real) matrices.</td>
</tr>
<tr>
<td>MatlabLibLinkCx</td>
<td>Evaluates a MATLAB expression and outputs the result as complex-valued matrices.</td>
</tr>
</tbody>
</table>

The models accept the .m script file for matrices evaluation and an optional .m script file for passing setup parameters to the MATLAB environment. You can switch between scripting and shared library modes by changing the mode parameter. Due to the limitation of the MATLAB Compiler, no figures are supported in the shared library mode. Also, the Linux platform is not supported in shared library mode.
Introduction to MATLAB Cosimulation

The shared libraries are compiled in the project/data directory. By default, libraries are also imported from the project/data directory. Make sure that the current directory ".", is in the library search path (i.e., LD_LIBRARY_PATH on Sun, SHLIB_PATH on HP-UX, and PATH on Windows). You can also set the library search path to a specified directory where pre-compiled libraries are located.

Examples

We recommend studying the MATLAB examples in the Advanced Design System examples directories. From the ../examples/DSP directory, choose the dsp_demos_prj project. The Sombrero network demonstrates how to use each of the three MATLAB models effectively. Sombrero is a simple example that plots a SINC function. For an explanation about this example, see the following Sombrero documentation:

../examples/dsp/Sombrero.html

In ADS, open the following example project to see an example of how to call a MATLAB.m file:

../examples/DSP/MATLABlink_prj project

Channel_Estimate.dsn demonstrates Script-Interpreting with the use of Matlab_M.

Channel_Estimate2.dsn demonstrates Library-Importing with the use of MatlabLibLink.
Chapter 11: Cosimulation with Analog/RF Systems

Simulation of behavioral DSP designs along with analog/RF circuit designs is critical to the success of the integrated components, devices, and subsystems used in today's wireless applications. The need to verify the impact of real-world analog/RF issues on the DSP algorithms and vice versa in a tightly integrated environment is highly desirable.

For designs of low complexity, it is possible to use separate simulators for the signal processing and analog/RF portions and then integrate the results. However, today's state-of-the-art designs using a mix of analog/RF and dedicated on-chip DSP blocks require high levels of integration at the two-environment boundary. Advanced Design System cosimulation between signal processing and circuits addresses this need. Agilent Ptolemy provides the signal processing simulation, while the analog/RF simulation is provided by either the Circuit Envelope or High-Frequency SPICE (Transient) simulators.

Other types of cosimulation include placing MATLAB components or HDL blocks in a signal processing simulation. This chapter describes cosimulation with Analog/RF Systems.

---

**Note** For information on A/RF cosimulation with Cadence refer to the RFIC Dynamic Link documentation located in the Design Flow area of the ADS Documentation set. See the chapter “Running a DSP and Analog/RF Cosimulation with RFIC Dynamic Link”.

---

*Figure 11-1* shows a mixture of RF circuitry and DSP components. The Advanced Design System provides a variety of analog/RF circuit simulators, including Linear, Harmonic Balance, Circuit Envelope, High-Frequency SPICE, and Convolution. For signal processing simulation, Agilent Ptolemy is used. Only circuits simulated with either Circuit Envelope or High-Frequency SPICE can be instantiated as a subnetwork and included in a signal processing schematic. These circuit blocks can then be simulated along with signal processing components. The steps needed for cosimulation are outlined next.
Cosimulation with Analog/RF Systems

![Diagram of cosimulation](#)

**Figure 11-1. Cosimulation: Different Design Portions Simulated by Different Simulators in the Same Schematic**

**Note** Circuit Envelope and High-Frequency SPICE simulators are included with some, but not all, Advanced Design System suites.
Setting Up the Analog/RF Circuit Schematic

The following general steps are used to create circuit designs for cosimulation.

1. In the analog/RF circuit Schematic window, create a circuit schematic that includes a simulation component for either Circuit Envelope (called ENV) or High-Frequency SPICE simulation (called TRAN).

2. Generally, use Circuit Envelope for an RF simulation and High-Frequency SPICE (transient) for a baseband simulation.

3. Do not use both Envelope and Transient simulators in the same design. Use the deactivate and activate commands if you want to keep both controllers in your design.

4. Add ports to your design.

5. Save your design.

In the above figure, a diode rectifier is set to be simulated with the Circuit Envelope simulator. Next, we will place this subnetwork in the signal processing schematic, where it will be represented as a block.

Figure 11-2. Diode Rectifier—Circuit Design Used in Cosimulation
Cosimulation with Analog/RF Systems

Setting Up the Signal Processing Schematic

To create signal processing designs for cosimulation:

1. To place the circuit subnetwork(s) you have already created in the signal processing schematic, choose Component > Component Library. Your opened projects are listed at the top of the list. Circuit projects have (A/RF) at the end.

2. Choose the circuit design you want and place it in your signal processing schematic.

3. Add the signal processing components.

4. Add the signal processing controller(s).

5. Connect the circuit design to the signal processing components.

6. For cosimulation with the Circuit Envelope simulator, see Circuit Simulation Controllers, later in this section.

7. If your circuit subnetworks have feedback loops between them, see Feedback Loops, later in this section.

8. If the input signal into the circuit subnetwork is not of type Timed, see Numeric-to-Timed Converters, below.

9. Start the simulation.
Circuit Simulation Controllers

As stated earlier, Agilent Ptolemy can cosimulate with only the Circuit Envelope or High-Frequency SPICE simulators. Any circuit simulation control components other than ENV or TRAN (such as for harmonic balance or S-parameter simulation) are ignored in the cosimulation from the signal processing schematic.

Numeric-to-Timed Converters

Both Circuit Envelope and Transient simulators deal with time-domain signals. Therefore, signal processing components connected to the circuit subnetwork need to be of type Timed. If the input component (connecting the signal processing components to the circuit) produces numeric data, place the appropriate numeric-to-timed converter (such as float-to-timed or complex-to-timed) in your schematic. These components are found in the Signal Converters library, and assure that the input into the circuit subnetwork is in the time domain. Refer to “Time Converters” on page 11-15 for more information on this topic.

Enabling Automatic Verification Modeling (Fast Cosimulation)

Automatic Verification Modeling is a simulation mode that can significantly accelerate formerly lengthy cosimulations of Analog/RF circuits. You can enable Automatic Verification Modeling in the Circuit Envelope Simulation Controller. When enabled, this mode characterizes an analog subcircuit into a behavior model, then the model is used to predict the response of the subcircuit at each time point. For details about Automatic Verification Modeling, see the documentation Simulation > Circuit Envelope Simulation > Automatic Verification Modeling (Fast Cosimulation) Overview.

The following steps demonstrate how to enable the fast cosimulation mode using PtolemyDocExamples/ AVM_prj from the examples directory:

1. In the ADS Main window, choose File > Copy Project. This opens the Copy Project dialog box.
Cosimulation with Analog/RF Systems

**Note**  On UNIX platforms, you cannot work directly in the Advanced Design System Examples directories. You must copy the example project to a directory for which you have write permission. On PC platforms, you can work in the Examples directories; however, it's better to copy examples to a working directory.

2. In the From Project area, click **Example Directory**, then click **Browse**. The Copy From File Browse dialog box showing the Examples directories.

3. Select the PtolemyDocExamples directory. Click **Filter**.

4. In the Files list, select AVM_prj. Click **OK**.

5. In the To Project area, click **Startup Directory** or **Working Directory** to select the destination directory for the copied project. Click **Browse** to select another directory.

6. Click **Copy Project Hierarchy** to enable this option which ensures all appropriate directories and files are copied.

7. Click **OK** to copy the project and close the dialog box.

8. In the ADS Main window, choose **File > Open Project** to open the Open Project dialog box appears. In the Directories list, select the directory containing the copied project. In the Files list, double-click AVM_prj.

9. In the Main window, choose **File > Open Design** to open the Open Design dialog box. In the Designs list, double-click TestMixer.dsn.

10. In the Schematic window, select DUT_Mix_sub, and push into its hierarchy, then push into the DUT_Mixer hierarchy as shown in the following figure.

11. In the DUT_Mixer, double-click the Envelope simulation controller to open its setup dialog. Select the Cosim tab, and click **Enable AVM (Fast Cosim)** to enable the mode.

**Hint** To enable AVM (Fast Cosim) directly on the schematic, click the Display tab on the Circuit Envelope setup dialog. Enable the **ABM_Mode** parameter. On the schematic, set **ABM_Mode=yes** to enable the mode; set **ABM_Mode=no** to disable the mode.
Figure 11-4. Pushing into TestMixer.dsn Hierarchy to Enable AVM (Fast Cosim)
Clustering of Circuit Subnetworks

Clustering is the process of defining the boundaries of the signal processing and analog/RF simulators. Initially, this boundary is defined by circuit schematics, where you define the circuit subnetworks and then make an instance of those on the Signal Processing schematic. However, there is a bit more to clustering than what is on the two schematics.

Circuit subnetworks directly connected in the Signal Processing schematic are automatically clustered by the program and treated as one circuit subnetwork, as shown in Figure 11-5. Therefore, use only one circuit simulation control component in either of the two (or more) directly connected subnetworks.

Figure 11-5. Connected Subnetworks Treated as One
Connected Circuit Subnetworks

For example, what happens when two circuit subnetworks defined on two different circuit schematics are connected on a Signal Processing schematic? The two circuit subnetworks are clustered into one by the program. This should not concern the user, since this would be done transparently. However, if each of these two circuit subnetworks use their own simulation controller, then the circuit engine would not know which one to choose for simulation. This would result in an error message.

Connected Resistors

Another aspect of clustering is when circuit components available on the Signal Processing schematic (resistors in the first release of Advanced Design System) are connected to a circuit subnetwork. In this case, such resistors will be absorbed into the circuit subnetwork during the clustering and will be simulated by the circuit engine as part of circuit subnetwork.
Feedback Loops

Circuit subnetworks that form a feedback loop via signal processing components require a delay component in the feedback loop to facilitate the signal processing simulation scheduling, as shown in Figure 11-6 and Figure 11-7. If such a delay is not present, an error message will be issued. To have the program automatically insert the delay, you need to edit the parameters for the Data Flow controller. To do this, double-click the controller and choose the Options tab and then Resolve deadlock by inserting tokens from the Deadlock Management drop-down list. For more information about deadlocks, refer to “Deadlocks” on page 9-7 in Chapter 9, Theory of Operation.

Figure 11-6. Feedback Loop Before Delay Added by Program

Figure 11-7. Feedback Loop After Delay Added by Program (Delay Not Shown on Schematic)
Named Connections and Measurements in Circuit Designs

Named connections and measurements included in the circuit schematic design (such as for a voltage) are ignored in cosimulation. The only results you get from a cosimulation are obtained from the Signal Processing schematic using Sink components or Interactive Components and Displays items.

Circuit Envelope Specific Rules

The output of the Circuit Envelope simulator is a collection of time waveforms, each at a different fundamental frequency. You need to select the waveform you want by specifying this fundamental frequency. You do this by choosing the EnvOutSelector or EnvOutShort component from the Circuit Cosimulation library. Refer to “EnvOutSelector and EnvOutShort Components” on page 11-16 for more information. You should place this component at all circuit subnetwork output ports in the Signal Processing schematic.

Circuit Envelope simulation requires that many parameters be set up in the circuit schematic. For more information, refer to Advanced Design System’s Circuit Envelope Simulation documentation. For cosimulation, the key parameter is the Step parameter. This is the time step used by the simulator, and can be set equal to or less than the time step at the connecting port in the signal processing schematic design. Other important parameters for cosimulation (especially nonlinear designs) are MaxOrder, Freq[], and Order[]. Make sure that the OutFreq parameter specified at the EnvOutSelector is among the fundamental frequency or harmonics specified by the Circuit Envelope controller.

To enable or disable the addition of noise from the Circuit Envelope components, use the Turn on all noise parameter in the Env Params tab. For cosimulation, the Nonlinear noise button, the Small-signal button, and their corresponding tab pages should not be used as they are not used during cosimulation. Note that explicit noise sources, such as V_Noise, are always on. The Turn on all noise enables or disables only the noise generated by non-source models, such as the resistors, lossy transmission lines, and transistors.

The amplitude of these noise sources is determined by the Circuit Envelope bandwidth, which is determined by 1/(Time Step). Normally, this Circuit Envelope timestep would be the same timestep as that used by Ptolemy, so the total noise bandwidths are the same. However, for designs where the Ptolemy waveform is
changing too rapidly for the analog simulation and the user has reduced the Circuit Envelope timestep, then the Circuit Envelope noise bandwidth also increases. Depending on the circuit behavior, this broader bandwidth noise may appear at the circuit output, where it is now effectively sampled by the EnvOut element. This sampling will then alias all the higher bandwidth circuit noise, and result in a higher noise density within the Ptolemy noise bandwidth. If this is not the desired simulation behavior, then a filter may need to be placed at the circuit output. Of course, this will also filter any signals being passed back to the Ptolemy simulation.

### Transient Simulation Specific Rules

When cosimulation with Agilent Ptolemy and the Transient simulator is required, the circuit schematic must have a Transient controller (also called a simulation component). No explicit user setting is required for the Transient controller. That is, the default parameters will work for cosimulation. However, the Transient controller’s Freq [x] parameter is required whenever there are any frequency-dependent sources. The Freq [x] parameter specifies the fundamental frequency.

### Nested Simulation Approach

Advanced Design System cosimulation is based on a nested simulation approach. In this use model, you first create your circuit designs on the circuit schematic. This circuit design can be tested using appropriate circuit sources and measurements with either the Circuit Envelope or High-Frequency SPICE simulators. Once the circuit design has been verified, ports to be interfaced with the signal processing design are identified and placed. Next, you place an instance of the design on the Signal Processing schematic and connect it to the other blocks. The combined schematic design can now be simulated.

### Signal Processing Model of the Circuit Network

Agilent Ptolemy uses a data flow simulation approach, and this simulation is controlled using the Data Flow Controller component. To understand this chapter, you need to be aware that this simulation is based on invoking a schedule. A schedule tells the simulator engine to “fire” components in a certain order and with a certain frequency. A simulation is typically a repetition of a schedule many times.
From the Data Flow engine perspective, a circuit subnetwork on the Signal Processing schematic is just a component with a certain number of input and output ports. This circuit subnetwork is part of the schedule determined by the Data Flow engine. It would be fired just like any other component according to the schedule, and as many times as required. Every time the circuit subnetwork is fired, the circuit simulator (designated by the simulation controller on the circuit schematic) continues to carry on the simulation based on the input it receives from the signal processing interface. Once the circuit simulator is finished with its analysis, it passes the simulation results back to the signal processing interface. This cycle repeats as many times as the scheduler requires. The duration of the circuit simulation each time it is invoked is determined by the time step provided by the connecting signal processing component at the input interface to the circuit subnetwork.

**Circuit Model of the Signal Processing Network**

From the circuit simulator engine point of view, the signal processing input interface is viewed as an ideal (0 ohm impedance) source. The more ports at the input interface to the circuit, the more ideal sources there will be feeding the circuit subnetwork. At the output interface of the circuit, there would be a node where the results are shared with the connecting signal processing component.

**Interface Issues**

At the interface boundary of the signal processing and analog/RF circuit simulators, there needs to be an exchange of information. The semantics and fundamentals of simulation in the two application areas are quite different and therefore, you need to understand these differences for proper use. The following sections outline the most important aspects of this interface.

**Time Step**

Time samples for signal processing are one fixed time step apart. However, both Envelope and Transient simulators define the time step in the simulation controller with various options.

The Transient Simulator controller component has several parameters, including Start time, Stop time, Min time step, and Max time step (see the Time Setup tab). In addition, the Integration tab contains a time step control method parameter with
Cosimulation with Analog/RF Systems

Fixed, Iteration Count, and Truncation Error options. For more information, refer to the Transient/ Convolution Simulation documentation.

For cosimulation with the Transient simulator, keep in mind one key issue: The Transient simulator may need time steps smaller than Agilent Ptolemy’s Time Step to satisfy its own setup requirements. In addition, the Transient simulator, when needed, will take additional time steps to match the time points in the signal processing simulation. Only time steps that match the signal processing time points will be passed on to Agilent Ptolemy.

For all practical purposes, the only parameter that may concern the cosimulation user is the Max time step. The rest of the parameters in the Transient Simulation control component can be left at default values.

For the Circuit Envelope simulator, the time step parameter in the ENV Simulation controller component should be set equal to or less than the Time Step at the signal-processing-to-circuit interface.

Depending on the Time Step value you set, the simulator will set the internal Circuit Envelope time step to either the Ptolemy time step value, or to an integer sub-multiple of this value. Time step values less than the Ptolemy time step value are sometimes required to achieve the desired accuracy, due to rapidly changing signals and the integration required for capacitive and inductive components. If the Circuit Envelope value is less than one-tenth of the Ptolemy time step value, a warning is generated to alert you to this so you can avoid inadvertent small time step values that might unnecessarily be slowing down the cosimulation.

The Stop time for the simulation is determined by the Signal Processing’s Data Flow controller and/or Sinks. The Circuit Envelope Stop time does not affect the duration of the cosimulation. The Stop Time value is used in a few models, and, ideally, should be set to reflect an approximate stop time range. As an example, for explicit Analog/RF Noise sources that have a user-specified baseband frequency response, this stop time value is used to determine the maximum duration of the pre-computed random noise sequence. Presently, the stop time is limited to 128K times the time step value. If the cosimulation runs longer than this, then this noise data is repeated.
Delays in Feedback Loops

As stated earlier, Data Flow simulation requires that a Delay component exist in the feedback loops for proper activation of the schedule. Circuit subnetworks that form a feedback loop, for this same reason require a delay component in their path. Typically, a Delay_Rf component in such feedback loops will suffice. If such a delay does not exist, Agilent Ptolemy will report a deadlock by default.

Time Converters

The common signal being exchanged between signal processing Data Flow components and the circuit simulators (Circuit Envelope and Transient) is a time-domain signal. All three engines, hence, deal with the notion of time step.

The signal entering the circuit subnetwork should be Timed. The Transient simulator deals only with real-baseband time-domain signals while Circuit Envelope can handle both baseband and complex envelope timed signals.

If the signal entering into the circuit subnetwork is not Timed (that is, the signal is Numeric), you should place a FloatToTimed, FixedToTimed, IntToTimed, or CxToTimed converter to accommodate the conversion. Although Agilent Ptolemy will place appropriate converters when they do not exist, it is always a good practice to explicitly place and connect these converters in your design. This will ensure that the input parameters into the circuit subnetwork are correct, as well as helping to debug possible errors that might occur.

Carrier Frequency

In the case of cosimulation with the Circuit Envelope simulator, the timed signal entering the circuit subnetwork is typically a carrier-modulated timed signal. This means that timed data has an $F_c$ field that is passed to the Circuit Envelope simulator, which is needed by the simulator. The Circuit Envelope simulator, depending on a particular design, will generate a number of time-domain waveforms, each associated with a carrier (harmonic) frequency. Since Agilent Ptolemy supports only one carrier frequency at each node, you need to select which one of the waveforms you desire in the signal processing portion of the design. This is done by placing a Circuit-Envelope specific component described next.
EnvOutSelector and EnvOutShort Components

When cosimulating with the Circuit Envelope simulator, additional information is needed for proper cosimulation. This is done by connecting an EnvOutSelector or EnvOutShort component (from the Circuit Cosimulation library in the Signal Processing schematic) to each output port of the subnetwork design.

The EnvOutSelector component acts as an open, blocking everything connected to its output from loading the circuit. If such loading is desired, use the EnvOutShort component. The EnvOutShort component acts as a short and therefore loads the circuit with the connecting Signal Processing components.

The EnvOutSelector and EnvOutShort components have a parameter called OutFreq. OutFreq specifies which waveform is selected from the time-domain waveforms at the output of the Circuit Envelope simulator. OutFreq has the following options:

- Lowpass—selects the time-varying DC component.
- Bandpass—(default) lets you specify any frequency.
- Allpass—forms the composite (baseband) signal.

One or more EnvOutSelector components can be connected to each output port of a circuit subnetwork, refer to Figure 11-8, below. This means that all waveforms generated by the Circuit Envelope simulator can be accessed in the Signal Processing schematic.

**Figure 11-8.** EnvOutSelector Components at each Circuit Subnetwork Output Port
If either the EnvOutSelector or EnvOutShort component is used with a design simulated by the Transient simulator, their effect is to be either an open or a short, respectively. Other than this, they do not affect transient cosimulation designs, so this component can be left in place without any impact on the cosimulation with the Transient simulator.

**Snapping Rule**

In the Bandpass option of the OutFreq parameter, you can type in the desired fundamental whose time waveform you are interested in. If the frequency you specify does not exist in the list of fundamentals, the interface program will search and snap to the nearest fundamental. Presently, anything within 0.01% of a fundamental will be snapped to that fundamental frequency. If the frequency specified in the Bandpass option of OutFreq is not within 0.01% of the fundamental, a default value of 100 MHz will be used and a warning message issued.

**Preparing Wireless Test Bench Designs for RFDE**

This feature enables a System designer to make a DSP system design, created with Ptolemy in ADS, available to an RFIC designer as a Wireless Test Bench (WTB) model. The RFIC designer can then validate and verify the RFIC design or Device Under Test (DUT).

Currently, RFIC designers use traditional measurements such as spectrum, TOI, etc. to validate their designs. They cannot successfully perform more complex system measurements such as EVM, BER, etc. Empirical methods are used to map the Analog/RF measurements to get the corresponding system measurements.

RFIC designs used in wireless systems are required to be qualified against the standard specifications. Qualifying designs involves complex system measurements. Using traditional measurements can produce errors. The Wireless Test Bench is a mechanism by which a System designer can make system measurements available to an RFIC designer to validate the RFIC designs used in wireless systems.
Creating a Wireless Test Bench Design

Wireless Test Bench designs can be created using Ptolemy components in ADS. The design setup is similar to setting up other Analog/RF designs for cosimulation.

The design must be created using a DSP schematic using only DSP components. The WTB design can be split between two parts.

The first part of the WTB design is the group of components that generate right kind of signal to stimulate the RFIC or DUT. Add ports at the end of this part of the design. These ports will act as WTB model output and DUT input when used in RFDE. Name the ports appropriately.

The second part of the WTB design is the group of components that process the DUT output and perform different system measurements using Ptolemy Sink components. Add ports at the input side of this part of the design. These ports will act as WTB model input and DUT output. Name the ports appropriately. For each input port connect the correct Envelope Selector (See EnvOutSelector or EnvOutShort) component to select the correct frequency to be extracted from the Circuit Envelope analysis of the DUT for system measurement.

You must also add a DF controller to the design which uses the correct setup of scheduler type, deadlock manager, and other parameters.

Create a symbol that corresponds to this design. For each port in the design, correct the directional attribute. Select the port directions to be input or output only. Inout type of ports are not supported.

Create the WTB design parameters that the RFIC designer can use to modify the WTB design’s configuration, using the File > Design Parameters in the ADS schematic window.

Setting the Units for WTB Design Parameters

The System designer can use the list of units provided in the Design Parameter dialog box to correctly specify the unit used for a parameter. If the parameter value is a unit that is not part of the list, such as percent, and the System designer wants to communicate the units to the RFIC designer, then the unit can be placed within parentheses at the end of the parameter’s description text.
Categorizing WTB Design Parameters

WTB designs support a feature that enables a person using the WTB design to categorize the parameters. It is helpful for the System designer to group the parameters in a way that makes it easier for the RFIC designer.

To categorize the list of parameters, define a string type of parameter at the beginning of the group of parameters that belong in a category. The parameter must have a description field specified that contains a very short description of the category. The parameter’s default value must be set to Category to identify this parameter as a categorization parameter. Also, the parameter’s attributes must be set such that it will not be netlisted and not displayed.

The group of parameters will appear in different tabs in the Wireless Test Bench Setup dialog in RFDE. The first group of parameters are considered Required Parameters and will appear in a separate widget along with the ports in the top half of the Wireless Test Bench Setup dialog box. These parameters are considered as parameters that must have value and cannot be left blank. All the subsequent groups of parameters will be considered as optional parameters for modification.

Information Parameters

The System designer can insert various information parameters that provide brief documentation for a particular aspect of the WTB design, such as instructions related to timing setup. To create such a parameter, define a string type of parameter and set its attribute such that it will not be netlisted and not displayed on the schematic. Leave the parameter’s default value blank.

Exporting the WTB Design to RFDE

After creating a WTB design as described in “Creating a Wireless Test Bench Design” on page 11-18, you must export the design so it can be used in RFDE. To export the WTB design, save it and select Tools > Export WTB Design to RFDE in the schematic window containing the design. An Export Status/Error/Warning dialog box appears showing the exporting progress. A temporary design window will open where the WTB design will be instantiated before the export starts.

A successful export process generates the files <WTBname>_TB.il and <WTBname>_TB.net. These files are saved with the exported design under the project directory ..._prj/hptolemy/wtb.
WTB Design User Interface Attributes

Each WTB design has a category to which it belongs. The category name is the project name from which it was exported without the _prj extension. The WTB design name is same as the corresponding design name.

The Wireless Test Bench Setup dialog in RFDE contains a pull-down list of categories. Each category selection updates another pull-down list containing WTB models exported from a single project.

Following the WTB model selection pull-down list is the Required Parameters widget that contains the ports that can be connected to the DUT, along with the first group of parameters created by the system designer. If no categories are created, all of the parameters are included as part of this widget. The items in this widget cannot be left blank, or unspecified, and must contain some value. To aid WTB model and DUT port connection, a button is provided, per port, that takes the RFIC designer to the schematic window to select a pin on the DUT.

Following the Required Parameters widget is a widget containing tabs for all subsequent parameter groups. The tab name is the description specified for the categorization parameter. These parameters are optional parameters for modification.

Verifying a WTB Design in ADS

To verify the WTB design in ADS use the following steps:

1. Create a DUT in ADS. Add an Envelope controller inside this DUT.
2. Open a new DSP schematic window.
3. Add an instance of the WTB design along with the instance of the DUT.
4. Connect them together.
5. Add a WTB controller. For details about the WTB controller, see “WTB Controller” on page 3-16.
6. Simulate this design and generate the dataset to verify the results.

Important Do not use more than one unconnected DUT design or Envelope controller during WTB design verification in ADS.
Creating a Results Display for WTB Designs

A WTB design can have complex data that the RFIC designer will not know how to interpret. To help simplify data analysis, the System designer must use the following steps:

1. System designer must use a simple DUT in ADS to create a dataset, as mentioned above.
2. Open a new DDS window. Add one or more new pages and name them appropriately.
3. Leave the page 1 untouched and blank. This page is used for other Analog/RF automatic plots.
5. Save this DDS file as a template by choosing File > Save as Template on the DDS window. Select the User category to save the template.
6. The template file is saved under $HOME/hpeesof/circuit/templates.
7. Copy the template file to _prj/hptolemy/templates.
8. Place an OutputOption component in the WTB design's schematic window.
9. Add the data display template name, created above, to the list of names. More than one data display template is allowed for a given WTB design.
10. Re-export the WTB design.

Now, when this WTB design is used in an RFDE simulation, the templates named in the OutputOption component will be inserted in the DDS window that appears at the end of the simulation.

The OutputOption component can be placed at any level of hierarchy in the WTB design. For more details, see “OutputOption Controller” on page 3-15.
Cosimulation with Analog/RF Systems

**Circuit Envelope Parameters**

There are some parameters that must be defined as WTB design parameters to ensure compliance with Circuit Envelope requirements within the WTB design’s simulation setup. Those parameters are:

**CE_TimeStep**

The WTB design must have a parameter named `CE_TimeStep`, which enables the RFIC designer to set the Circuit Envelope time step. If this parameter is not added to a WTB design, the time step will always be set to 0.01usec.

**FSource**

The WTB design can optionally have a parameter named `FSource` to enable the RFIC designer to set the source carrier frequency. This parameter will also be automatically added to the Fundamental Tones parameters of Circuit Envelope after the WTB design is set up in RFDE. This frequency must not be deleted from the Fundamental Tones to ensure correct behavior.

**FMeasurement**

The WTB design can optionally have a parameter named `FMeasurement` to enable the RFIC designer to set the measurement carrier frequency. This parameter will also be automatically added to the Fundamental Tones parameters of Circuit Envelope after the WTB design is set up in RFDE. This frequency must not be deleted from the Fundamental Tones to ensure correct behavior.
Wireless Test Bench Design Examples

For examples on how to create a WTB design for export to RFDE, look at the designs listed with the example projects in the following table. These projects are contained in the WLAN, TD-SCDMA, and 3GPP W-CDMA Design Libraries which can be installed by using the custom installation option.

<table>
<thead>
<tr>
<th>Example Project</th>
<th>Design</th>
<th>License Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDSCDMA_Export_prj</td>
<td>TDSCDMA_WTB_Export</td>
<td>mdl_tdscdma</td>
</tr>
<tr>
<td>WCDMA3G_Export_prj</td>
<td>WCDMA3G_WTB_Export</td>
<td>mdl_wcdma3g</td>
</tr>
<tr>
<td>WLAN_Export_prj</td>
<td>WLAN_WTB_Export</td>
<td>mdl_wlan</td>
</tr>
</tbody>
</table>

Each design has instances of the pre-configured WTB model components. Push into the components to view the design. These designs should be used as examples to create WTB designs for exporting to RFDE.

**Note**  RangeCheck components used in the pre-configured WTB designs are created specifically for those WTB designs. Do not use the RangeCheck components when you create your own WTB designs.
Cosimulation with Analog/RF Systems

Troubleshooting Common Problems

Although the cosimulation use model is intuitive, there are a few pitfalls you should know about to avoid errors:

1. At present in Advanced Design System, only the Transient and Circuit Envelope circuit simulators can cosimulate with Agilent Ptolemy. Other circuit simulation controllers on the analog/RF schematic (such as S-parameter or AC) will be ignored in cosimulation.

2. Directly connected circuit subnetworks placed as instances on Signal Processing schematics are clustered together and hence should be considered as one circuit subnetwork. This means that if each of these subnetworks has their own circuit simulation controller, an error message will be issued. To avoid such problems you can either:
   - Deactivate all controllers but one on the circuit schematics.
   - Connect a signal processing component between the two circuit subnetworks, thereby preventing the two subnetworks from being clustered into one.

3. Resistor components that are part of hierarchical designs of timed components in the Signal Processing schematic will be absorbed into connecting circuit subnetworks by the program. If the EnvOutSelector component is used, the absorbed resistors will not load the circuit, since the circuit model is an open. If the EnvOutShort component is used, the absorbed resistors will load the circuit, and the results will be different by a scale factor.

4. Since resistors that are part of timed subnetworks are absorbed into connecting circuit subnetworks, you should avoid placing a sink or any other signal processing component directly at this port, when cosimulating with Circuit Envelope. If placed, an error message is issued, requiring an EnvOutSelector component to be placed. The reason for this condition is the fact that the sink now constitutes an output port from the perspective of the circuit subnetwork.

5. Writing a VAR in the analog/RF subnetwork to the dataset cannot be done using the Output tab on the Circuit Envelope Controller’s setup dialog box, nor by using OutVar in the Data Flow Controller. It can be done by using the Output tab option in the top-level DF controller, or defining the VAR in the top-level DSP design.
Cosimulation Example

To illustrate cosimulation, we will use an example called RectifierCosim_prj.

Copying and Opening the Project

1. From the Main window, choose File > Copy Project. A dialog box appears.

   Note On UNIX platforms, you will not be able to work in the Advanced Design System Examples directories. You must copy the example project to a directory for which you have write permission. On PC platforms, while you can work in the Examples directories if you want, it's still a good idea to copy the examples to another directory.

2. In the From Project field, click the Examples Directory button, and then the Browse button. The File Browse dialog box appears with the Example directories listed.

3. Select the /Com_Sys directory.

4. Select RectifierCosim_prj from the list of files in the Files field.

5. In the To Project field, click the Startup Directory or Working Directory button (depending on where you want to copy the project to) or choose the Browse button if you want to select another directory.

6. Choose Copy Project Hierarchy. This ensures that all the appropriate directories and files will be copied.

7. Click OK to copy the project and close the dialog box.

8. From the Main window, choose File > Open Project. When the Open Project dialog box appears, select <the directory you copied the example to> in the Directories field, then double-click RectifierCosim_prj in the Files field.
Rectifier Schematic

The top level design RectifierCx_Tutorial, as shown in Figure 11-9, generates a complex modulated signal and sends that signal into a simple rectifier circuit.

Figure 11-9. RectifierCosim Project Top-Level Schematic

Two identical instances of this rectifier circuit are created on a circuit schematic, one with a TRAN controller (rct_Tran), Figure 11-10, and the other with an ENV controller (rct_Env), Figure 11-11.
To view the circuit subnetworks from the top-level design, choose View > Push Into Hierarchy or click the Push Into Hierarchy button (down arrow) from the toolbar.

The circuit design in both rct Tran and rct Env is a simple diode with a shunt parallel RC attached to its output. Note the placement of ports and that rct Env has an Circuit Envelope controller, while rct Tran a Transient controller. The two circuit designs are then placed on the Signal Processing schematic.
Cosimulation with Analog/RF Systems

On the Signal Processing schematic, the generation of complex modulated signals is accomplished via a RectCx component that generates a periodic pulse with a complex amplitude. This complex pulse is then fed into a CxToTimed component that effectively upconverts the signal. The TStep is set to 0.01 μsec and the carrier is at 11 MHz. This modulated RF signal is then split into two branches by a SplitterRF component and fed into the rct_Tran and rct_Env circuit subnetworks. In addition, there are TkPlot components (which display the simulation results) attached to the output of RectCx and CxToTimed to monitor the signal before simulation.

Since the TStep of the signal entering circuit design is at 0.01 μsec, the MaxTimeStep on the Transient controller is set to the same value. This value should always be smaller than or equal to the Signal Processing TStep. The other Time setup parameters, such as Start time and Stop time, are ignored in the Transient cosimulation. Note that the output of rct_Tran is directly fed into a TkPlot without an interface component.

Similarly for the Circuit Envelope simulator, the Step parameter of the simulation controller should be set less than or equal to the Signal Processing TStep. Other parameters of interest are Freq[], Order[], and MaxOrder, which specify the fundamentals and related harmonics to be analyzed. In this example, the fundamental of interest is Freq[1] = 11 MHz and the MaxOrder and Order[1] are set to 5. Note also that Freq[0] is the dc term that is always available.

Typically, there is only one EnvOutSelector component attached to the Circuit Envelope subnetwork output, but in this example we have used three to show the different signals that can be selected from the Circuit Envelope output. Specifically, the OutFreq parameter is set to the Bandpass, Allpass and Lowpass options in the three instances. When the Bandpass option is selected, the dialog box changes so you can enter the desired fundamental frequency. In this case, OutFreq is set to 11 MHz. The output of EnvOutSelectors are then fed into three interactive TkPlot display components.

When we simulate this design, 6 TkPlot windows pop up, as shown next.
The plot Complex Bits displays the magnitude of the complex periodic pulse and the plot Rectifier Input depicts the pulse-modulated signal at 11 MHz.

The plot Transient Output is the rectified version of the modulated signal (note that there are no negative components in the signal), where the value of the time constant
Cosimulation with Analog/RF Systems

(RC) determines the degree of pulse fall-off. Note also that this output is a real-baseband signal and includes all the harmonics.

The allpass, lowpass, and bandpass output plots are shown next.
Chapter 12: Using Interactive Controls and Displays

Agilent Ptolemy contains a library of 19 Interactive Control and Display components. These components provide real-time simulation input control and animated plots of your simulation results. Interactive Displays represent one of two general use models for viewing simulation results.

In contrast, the main Advanced Design System method to display simulation results is to save your simulation data and open a Data Display window to view the results. You can set up your schematic for both methods, by placing both an Interactive Control and Display component and a Sink. The following table describes the two basic use models for displaying results:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interactive Controls</td>
<td>Animated displays of simulation results and interactive control items. Data</td>
</tr>
<tr>
<td>and Displays</td>
<td>is not saved. No post processing. Display starts during simulation.</td>
</tr>
<tr>
<td>Data Display Window</td>
<td>Data is saved after simulation is complete. You open a Data Display window,</td>
</tr>
<tr>
<td></td>
<td>choose a plot type, parameters to plot, and have many data manipulation</td>
</tr>
<tr>
<td></td>
<td>options.</td>
</tr>
</tbody>
</table>

The Interactive Control and Display components are a quick and easy way to display your simulation results. They also give you a way to interactively change parameters while your simulation is running and display animated plots.

The Interactive Controls and Displays were derived from a scripting language called Tool Command Language (Tcl) and the Tool Kit attached to Tcl (Tk). These were originally developed at the University of California at Berkeley. Therefore, the abbreviations Tcl or Tk appear in the component names.

**Note** To use the Interactive Controls and Displays library components (such as TkPlot) with Advanced Design System's tune mode, you must dismiss the Interactive Controls and Displays component between each tune with its pop-up dialog box.
Using Interactive Controls and Displays

**Additional Resources**

Tcl is a language that was created to be easily embedded into applications. Tk is a graphical toolkit that makes creating user interfaces easier. Both were created by John Ousterhout while he was a professor at U.C. Berkeley. If you want to explore this language further to write your own applications, here is a list of resources:


The following web site is a good place to start looking for information on Tcl/Tk:

http://www.yahoo.com/Computers_and_Internet/programming_and_development/languages/tcl_tk/
List of Interactive Control and Display Components

Table 12-2 shows the Interactive Controls and Displays library.

Table 12-2. Interactive Controls and Displays Library

<table>
<thead>
<tr>
<th>Descriptive Name</th>
<th>Component Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interactive Slider</td>
<td>TkSlider</td>
</tr>
<tr>
<td>Plot Inputs versus Time</td>
<td>TkPlot</td>
</tr>
<tr>
<td>Display History of Input Values</td>
<td>TkText</td>
</tr>
<tr>
<td>Input Values Display</td>
<td>TkShowValues</td>
</tr>
<tr>
<td>Plot Y versus X Inputs</td>
<td>TkXYPlot</td>
</tr>
<tr>
<td>Bar Chart Display</td>
<td>TkBarGraph</td>
</tr>
<tr>
<td>Interactive Complex LMS Adaptive Filter</td>
<td>LMS_CxTkPlot</td>
</tr>
<tr>
<td>Interactive LMS Adaptive Filter</td>
<td>LMS_TkPlot</td>
</tr>
<tr>
<td>Interactive Buttons</td>
<td>TkButtons</td>
</tr>
<tr>
<td>Conditional Breakpoint</td>
<td>TkBreakPt</td>
</tr>
<tr>
<td>Bar Meters Display</td>
<td>TkMeter</td>
</tr>
<tr>
<td>Booleans Display</td>
<td>TkShowBooleans</td>
</tr>
<tr>
<td>Baseband Equivalent Channel</td>
<td>TkBasebandEquivChannel</td>
</tr>
<tr>
<td>Invoke Tcl Script</td>
<td>TclScript</td>
</tr>
<tr>
<td>Eye Diagram</td>
<td>TkEye</td>
</tr>
<tr>
<td>IQ Constellation Diagram</td>
<td>TkConstellation</td>
</tr>
<tr>
<td>Histogram Diagram</td>
<td>TkHistogram</td>
</tr>
<tr>
<td>Display rms value of input IQ signal</td>
<td>TkLQrms</td>
</tr>
<tr>
<td>Signal Power Display in dBm</td>
<td>TkPower</td>
</tr>
</tbody>
</table>

In this chapter, you will find:

- An introduction to the Interactive Control and Display components.
- Cross-references to the Signal Processing Components manual, which contains a description of the parameters associated with each component.
- References to application examples shipped with Advanced Design System that contain or illustrate various Interactive Control and Display components.
TkSlider and TkPlot Components Example

The following figure is taken from the ../examples/DSP/dsp_demos_prj project. It is called EYE. If you want to run this example, copy the project from the examples directory to a directory for which you have write permission, using the File > Copy Project command.

The EYE schematic uses an Interactive Control component called TkSlider to adjust the amount of noise added to the pulse stream. You then can instantly see the simulation results change in an animated display provided by the TkPlot component. TkPlot simply plots one input on the Y-axis versus time, or sample number, on the X-axis.

![Figure 12-1. Schematic Using TkSlider and TkPlot Components](image1)

When you begin simulation of a design using TkSlider, the dialog box shown in Figure 12-2 is displayed.

![Figure 12-2. TkSlider is Interactively Controlled by This Dialog Box](image2)
This dialog box is called the Agilent Ptolemy Control Panel. At a minimum, the Pause and Quit buttons are shown. One of the parameters for TKSlider is called PutInControlPanel. If the default of Yes is accepted, the slider is put in the Control Panel. (If you choose No for this parameter, a separate box will be displayed for each slider.)

With the slider control all the way to the left, 0.0 noise is added and the resulting clean eye-diagram plot is shown below.

![Figure 12-3. Eye Diagram Plot with Noise at 0.0](image)

The slider control can be adjusted, as shown in the next figure.

![Figure 12-4. Slider Control Moved to Add Noise](image)

If you move the slider to the right so that the Noise Magnitude is 0.37, the animated plot changes to display the poor eye diagram shown in Figure 12-5.
Using Interactive Controls and Displays

![Eye Diagram Plot with Noise at 0.37](image)

**Figure 12-5. Eye Diagram Plot with Noise at 0.37**

More or less noise can be added and the results will be shown immediately.

For a complete parameter description, see the Signal Processing Components manual in "Interactive Controls and Displays" > “TkSlider (Interactive Slider)” or “TkPlot (Plot Inputs versus Time)".
Placing Multiple TkSlider Components

Placing multiple TKSlider components in a schematic will result in multiple sliders in the Control Panel, as shown below. You can type a label for each by editing the parameters on-screen or double-clicking the component to bring up the Component Parameters dialog box.

Figure 12-6. Control Panel with Multiple TkSliders
Using Interactive Controls and Displays

**TkText and TkShowValues**

The TkText component simply displays the input values in text form, as shown below. This is an input history.

![Text Display](image)

The TkShowValues component works in a similar manner, except only one value is shown at a time (rather than a history). This value is displayed in the lower part of the Agilent Ptolemy Control Panel, as shown below.

![Agilent Ptolemy Control Panel](image)

For a complete parameter description, see the Signal Processing Components manual in “Interactive Controls and Displays” > “TkText (Display History of Input Values)” or “TkShowValues (Input Values Display)”. 

12-8   TkText and TkShowValues
TkXYPlot Component

The following figure is taken from the `./examples/DSP/dsp_demos_prj` project. It is called EQ_16QAM. If you want to run this example, copy the project from the examples directory to a directory for which you have write permission, using the File > Copy Project command.

Several TkXYPlot components are placed in this design to show animated results where two inputs, X versus Y, are to be plotted. In this example the TkXYPlot components are used to display constellation diagrams. The equalized constellation diagram is shown in Figure 12-8.

![Figure 12-7. Equalized 16 QAM System with Multipath and Phase Noise.](image)

For a complete parameter description, see the Signal Processing Components manual in “Interactive Controls and Displays” > “TkXYPlot (Plot Y versus X Inputs)”.

---

TkXYPlot Component 12-9
Figure 12-8. TkXYPlot for Equalized Constellation Diagram
TkBarGraph

The TkBarGraph component displays input (multiple anytype) data in a bar graph format. Different inputs are assigned different colors, up to 12, then the colors are repeated. A TkBarGraph example is shown in Figure 12-9.

![TkBarGraph Display](image)

Figure 12-9. TkBarGraph Display

For a complete parameter description, see the Signal Processing Components manual in “Interactive Controls and Displays” > “TkBarGraph (Bar Chart Display)”.
LMS Adaptive Filter Components

There are two LMS adaptive filter components in the Interactive Controls and Displays library: LMS_CxTkPlot and LMS_TkPlot. They are the same except that the former expects complex data and the later expects real data.

An example of this component can be found in the same project we have just used: ../examples/DSP/dsp_demos_prj project. It is called EQ_16QAM, and is shown in Figure 12-10.

Both LMS components implement an adaptive filter using the least-mean square algorithm, also know as the stochastic-gradient algorithm.

The size of the LMS filter is determined by the number of coefficients in the Taps parameter; the default gives an 8th-order, linear-phase lowpass filter. LMS supports decimation, but not interpolation.

The filter coefficients can be specified directly or read from a file. To load filter coefficients from a file, replace the default coefficients with the string <filename> (use an absolute path name for the filename to allow the filter to work as expected regardless of the directory where the simulation process actually runs).

When used correctly, this LMS adaptive filter will adapt to try to minimize the mean-squared error of the signal at its error input. The output of the filter should be compared to (subtracted from) some reference signal to produce an error signal. That error signal should be fed back to the error input. The ErrorDelay parameter must equal the total number of delays in the path from the output of the filter back to the error input. This ensures correct alignment of the adaptation algorithm. The number of delays must be greater than 0 or the simulation will deadlock.

If the SaveTapsFile string is non-null, a file will be created with the name given by that string, and the final tap values will be stored there after the run has completed.
Figure 12-10. LMS Adaptive Filter (Complex) Component in the 16 QAM System
The plot generated from this design upon simulation is shown in Figure 12-11.

Figure 12-11. Plot Resulting From LMS_CxTkPlot FilterTaps Parameter
For a complete parameter description, see the Signal Processing Components manual in “Interactive Controls and Displays” > “LMS_CxTkPlot (Interactive Complex LMS Adaptive Filter)” or “LMS_TkPlot (Interactive LMS Adaptive Filter)”.

LMS Adaptive Filter Components 12-13
Using Interactive Controls and Displays

**TkButtons**

Like TkSlider, TkButtons produces an output. The data type of the output is multiple anytype. TkButtons outputs 0.0 unless the corresponding button is pushed, when the output becomes the value assigned in the parameter Value. You can assign your own identifiers using strings for the corresponding parameter.

For a complete parameter description, see the Signal Processing Components manual in “Interactive Controls and Displays” > “TkButtons (Interactive Buttons)”.

**TkBreakPt**

TkBreakPt allows you to pause or stop a simulation. Its principal use is to help debug simulations. You can stop the simulation based on a condition of the model’s inputs. The parameter syntax is described in the Signal Processing Components manual in “Interactive Controls and Displays” > “TkBreakPt (Conditional Breakpoint)”.
TkMeter

TkMeter dynamically displays the value of any number of input signals on a set of bar meters. The input type is anytype. These values are displayed in the lower part of the Agilent Ptolemy Control Panel, as shown below.

For a complete parameter description, see the Signal Processing Components manual in “Interactive Controls and Displays” > “TkMeter (Bar Meters Display)”.

TkShowBooleans

The TkShowBooleans component works in a similar manner to TkShowValues, except that a Boolean value of zero (false) or non-zero (true) is shown. This value is displayed in the lower part of the Agilent Ptolemy Control Panel, as shown below.

For a complete parameter description, see the Signal Processing Components manual in “Interactive Controls and Displays” > “TkShowBooleans (Booleans Display)”.

---

TkMeter  12-15
Baseband Equivalent Channel

The TkBasebandEquivChannel component is a subnetwork as shown in Figure 12-12. It models a baseband equivalent channel with linear distortion, frequency offset, phase jitter, and additive white Gaussian noise. You can dynamically set many of the channel model parameters. TkBasebandEquivChannel accepts complex data and outputs the input signal plus distortions.

To model linear distortion, such as intersymbol interference, the input signal is passed through a complex FIR filter with the taps set by LinearDistortionTaps. The frequency offset distortion is set by the Freq. Offset slider control. Similarly, the phase jitter amplitude (peak-to-peak, in degrees) is set by the Phase Jitter slider control while the phase jitter frequency is set by the Phase Jitter Frequency Hz parameter. The phase of both the frequency offset and the phase jitter can be reset with the Reset Phase control button. The amplitude of the added complex white Gaussian noise is set by the Noise Power slider control.

Figure 12-12. TkBasebandEquivChannel Component Subnetwork

For a complete parameter description, see the Signal Processing Components manual in “Interactive Controls and Displays” > “TkBasebandEquivChannel (Baseband Equivalent Channel)”.

12-16  Baseband Equivalent Channel
Invoke Tcl Script

TclScript reads a file containing Tcl commands. It can be used in a variety of ways, including using Tk to animate or control a simulation. A number of procedures and global variables will have been defined for use by the Tcl script by the time it is sourced. These enable the script to read the inputs to the component or set output values. The Tcl script can optionally define a procedure to be called by Agilent Ptolemy for every simulation of the component.

The parameter syntax is described in the Signal Processing Components manual in “Interactive Controls and Displays” > “TclScript (Invoke Tcl Script)”. 

TkEye, TkConstellation, TkHistogram, TkIQRms, and TkPower

These TclTk components generate an eye diagram, an IQ constellation diagram, a histogram diagram, a display of the rms value of the IQ input signal, and a signal power display in dBm, respectively. For more information on these components, refer to Interactive Controls and Displays in the Signal Processing Components manual.
Using Interactive Controls and Displays
Index

A
Advanced Design System
approach to cosimulation, 11-12
Agilent Ptolemy
arrowheads in schematic, 5-3
data types, conversion, 5-6
data types, representation, 5-1
Interactive Controls and Displays,
compared to sinks, 12-1
overview versus UC Berkeley Ptolemy, 1-2
references, 9-18
terminology versus UCB Ptolemy, 1-3
theory of operation, 9-1
array parameters, 4-17
arrowheads in schematic, 5-3
Automatic Verification Modeling (Fast
Cosimulation), 11-5

B
Baseband Equivalent Channel, 12-16

C
carrier frequency resolution, 9-11
Circuit Envelope simulator
selecting waveform for cosimulation, 11-15
circuit subnetworks, consequences of
clustering, 11-9
clustering
consequences of, 11-9
defined for cosimulation, 11-8
simulating resistors, 11-9
complex-valued parameters, 4-9
cosimulation
Automatic Verification Modeling, 11-5
cosimulating, consequences of, 11-9
clustering, defined, 11-8
element of, 11-25
element results, 11-29
fast, 11-5
nested simulation approach, 11-12
resolving deadlocks, 11-10
selecting circuit envelope waveform, 11-15
time step sizes, 11-14
troubleshooting common problems, 11-24
using EnvOutSelector, 11-16
using EnvOutShort, 11-16
using time converters, 11-15
using time steps, 11-13

D
data types
conversion of, 5-6
conversion, automatic or manual, 5-10
numeric matrix, 5-4
numeric scalar, 5-4
representation, 5-1
timed data, 5-5
dataflow terminology, 9-3
deadlocks, 9-7
resolving for cosimulation, 11-10
DF (DataFlow) Controller, 3-6

E
EnvOutSelector, using for cosimulation,
11-16
EnvOutShort, using for cosimulation, 11-16

F
fast cosimulation, 11-5
filename
parameters, 4-16

H
homogeneous synchronous dataflow, 1-5

I
Interactive Control and Display components
table, 12-3
Invoke Tcl Script, 12-17

L
LMS Adaptive Filter, 12-12

M
MATLAB
cosimulation, 10-1
elements, 10-6
setting up, 10-1
simulating with, 10-3
writing function for, 10-3
multithreading, 9-6
nominal optimization

See optimization, 8-1

optimization

bit width example, 8-3
using various parameter types, 8-2
OutputOption Controller, 3-15

parameter expressions, 4-9
parameters
for fixed-point components, 4-10
with optimization attributes, 4-19
ParamSweep component, 7-2
performance optimization
See optimization, 8-1

resistance, input/output, 9-11
resistors
cosimulation with clustering, 11-9

signals, time domain, 9-9
string parameters, 4-16
Sweep Plan, 7-6
sweeping parameters, 7-1
performing multidimensional sweeps, 7-13
procedure, 7-3
using the VAR component, 7-5
using various parameter types, 7-6, 7-10
Synchronous Dataflow, 9-2

time converters
placing for cosimulation, 11-15
time step resolution, 9-10
time steps
size for cosimulation, 11-14
using for cosimulation, 11-13
Timed Synchronous Dataflow, 9-9
TkBarGraph, 12-11
TkBreakPt, 12-14
TkButtons, 12-14
TkMeter, 12-15
TkPlot, 12-4

TkShowBooleans, 12-15
TkShowValues, 12-8
TkSlider, 12-4
TkText, 12-8
TkXYPlot, 12-9
Transient simulator
time step size for cosimulation, 11-14

value types, 4-1
complex array, entering, 4-5
complex, entering, 4-2
entering, 4-1
enumerated type, entering, 4-6
filename, entering, 4-3
fixed point array, entering, 4-4
fixed point, entering, 4-2
integer array, entering, 4-3
integer, entering, 4-1
precision, entering, 4-2
real array, entering, 4-4
real, entering, 4-1
string array, entering, 4-6
string, entering, 4-2
table of, 4-1

Wireless Test Bench
exporting designs to RFDE, 11-17
WTB Controller, 3-16