

# Control Design and Simulation Module

June 2008, 371894C-01

Simulation is a process that involves using software to recreate and analyze the behavior of dynamic systems. You use the simulation process to lower product development costs by accelerating product development. You also use the simulation process to provide insight into the behavior of dynamic systems you cannot replicate conveniently in the laboratory. For example, simulating a jet engine saves time, labor, and money compared to building, testing, and rebuilding an actual jet engine. You can use the LabVIEW Control Design and Simulation Module to simulate a dynamic system or a component of a dynamic system. For example, you can simulate only the plant while using hardware for the controller, actuators, and sensors.

If you are new to the Control Design and Simulation Module, consider completing the [Getting Started with Simulation](#) tutorial.

In addition to the topics contained in this help file, the [LabVIEW Control Design User Manual](#) contains information about using LabVIEW to design, analyze, and deploy controllers for dynamic systems.

The following table describes the tasks you can perform with the Control Design and Simulation Module and the components you use for these tasks.

Task	Component
Design, analyze, and deploy controllers for dynamic system models	<a href="#">Control Design</a> VIs and functions. You also can use the <a href="#">Control Design MathScript functions</a> to design and analyze controllers.
Configure simulation parameters, including the ordinary differential equation (ODE) solver, and define the simulation as part of a LabVIEW block diagram	<a href="#">Simulation Loop</a>
<ul style="list-style-type: none"><li>• <a href="#">Build</a>, simulate, and deploy <a href="#">dynamic system models</a>, including models developed with the</li></ul>	<a href="#">Simulation</a> functions

<p>LabVIEW System Identification Toolkit or the Control Design and Simulation Module.</p> <ul style="list-style-type: none"> <li>• <a href="#">Execute offline, Rapid Control Prototype (RCP), and Hardware-in-the-Loop (HIL) configurations</a></li> <li>• <a href="#">Generate and combine</a> input and feedback signals</li> <li>• <a href="#">Collect</a> and <a href="#">display</a> simulation data</li> </ul>	
<p><a href="#">Trim and linearize</a> a nonlinear dynamic system model</p>	<p><a href="#">Trim &amp; Linearize</a> VIs; <a href="#">Linearize Subsystem</a> dialog box</p>
<p><a href="#">Determine the optimal parameters</a> for a dynamic system model, given a set of constraints</p>	<p><a href="#">Optimal Design</a> VIs</p>
<p><a href="#">Convert</a> a model developed in The MathWorks, Inc. Simulink® application software into LabVIEW block diagram code</p>	<p><a href="#">Simulation Model Converter</a></p>

## Version 8.6 Features (Control Design and Simulation Module)

Refer to the [LabVIEW 8.6 Features and Changes](#) topic for information about new features in LabVIEW 8.6.

Refer to the readme\_ControlandSim.html file, located in the labview\readme directory, for a complete list of new features and changes, information about upgrade and compatibility issues specific to different versions of the Control Design and Simulation Module, and information about known issues with the Control Design and Simulation Module 8.6.

## Using Simulation Subsystems Outside a Simulation Loop

You now can place a simulation subsystem on a block diagram [outside a Simulation Loop](#). If you run a simulation subsystem on a block diagram outside a Simulation Loop, the simulation subsystem executes one step of the ordinary differential equation solver each time the simulation subsystem is called.

## Placing Constraints on an MPC Controller

Use the [CD Create MPC Controller](#) VI to create an MPC controller model with constraints defined using either the dual optimization or barrier function method. Use the [CD Update MPC Window](#) VI to provide setpoint and disturbance profiles to the MPC controller during implementation.

You also can use the [CD Set MPC Controller](#) VI to update specified parameters, including any constraints, of an MPC controller at run time.

Refer to the [LabVIEW Control Design User Manual](#) for more information about setting constraints for an MPC controller.

## Implementation Palette Changes

The **Control Design»Implementation** palette has the following changes:

- The [CD Discrete Stochastic State-Space](#) function replaces the CD Discrete Stochastic State-Space (External) function and the CD Discrete Stochastic State-Space (Internal) function. Use the CD Discrete Stochastic State-Space function outside a Simulation Loop. Use the new [Discrete Stochastic State-Space](#) function inside a Simulation Loop.
- The [CD Discrete Observer](#) function replaces the CD Predictive Observer function, the CD Current Observer Corrector VI, and the CD Current Observer Predictor VI. The CD Current Observer Corrector VI and the CD Current Observer Predictor VI no longer are on the palette but still work in VIs you created in a previous version of the Control Design and Simulation Module. Use the CD Discrete Observer function outside a Simulation Loop. Use the new [Discrete Observer](#) function inside a Simulation Loop.
- The [CD Discrete Kalman Filter](#) function replaces the CD Discrete Recursive Kalman Corrector VI and the CD Discrete Recursive Kalman Predictor VI. The CD Discrete Recursive Kalman Corrector VI and the CD Discrete Recursive Kalman Predictor VI no longer are on the palette but still work in VIs you created in a previous version of the Control Design and Simulation Module. Use the CD Discrete Kalman Filter function outside a Simulation Loop. Use the new [Discrete Kalman Filter](#) function inside a Simulation Loop.

The **Simulation»CD Implementation** palette has been removed. The changes to the **Control Design»Implementation** palette also apply to the **Simulation»CD Implementation** palette. Additionally, the CD Continuous Observer function and the CD Continuous Recursive Kalman Filter function now are on the **Simulation»Continuous Linear Systems** palette.

## **LabVIEW Control Design Assistant 3.0**

Use the interactive LabVIEW Control Design Assistant to develop models that reflect the behavior of single-input single-output (SISO) systems. Using the Control Design Assistant, you can load or create a model of a plant, analyze the time or frequency response, and then synthesize a controller. The Control Design Assistant has windows in which you can immediately see the mathematical equation and graphical representation that describe the model. You also can view the response data and the configuration of the controller. Select **Tools»Control Design and Simulation»Launch Control Design Assistant** to launch the Control Design Assistant from LabVIEW.

## MathScript Node on the Simulation Diagram

You now can place a [MathScript Node](#) directly on a simulation diagram.

## Related Documentation (Control Design and Simulation Module)

The following documents contain information that you might find helpful as you use the LabVIEW Control Design and Simulation Module.

- [Getting Started with LabVIEW](#)—This manual contains an in-depth introduction to LabVIEW, including several tutorials that showcase LabVIEW features.
- [LabVIEW Fundamentals](#)—This manual provides information about LabVIEW programming concepts, techniques, features, VIs, and functions you can use to create many types of applications.
- [LabVIEW Control Design User Manual](#)—This manual contains information about using LabVIEW to design, analyze, and deploy controllers for dynamic systems.
- [Getting Started with the LabVIEW Real-Time Module](#)—This manual introduces the concepts necessary to [create real-time simulations](#).
- [Real-Time Execution Trace Toolkit](#) documentation.
- *NI-CAN Hardware and Software Manual*
- *NI-DAQmx Help*
- *LabVIEW Control Design and Simulation Module Readme*—Use this file to learn important last-minute information, including installation and upgrade issues, compatibility issues, changes from the previous version, and known issues with the Control Design and Simulation Module. Open this readme by selecting **Start»All Programs»National Instruments»LabVIEW»Readme** and opening `readme_ControlandSim.html` or by navigating to the `labview\readme` directory and opening `readme_ControlandSim.html`.
- LabVIEW Control Design and Simulation Module example VIs—Refer to the `labview\examples\Control and Simulation` directory for example VIs that demonstrate common tasks using the Control Design and Simulation Module. You also can access these VIs by selecting **Help»Find Examples** and selecting **Toolkits and Modules»Control and Simulation** in the [NI Example Finder](#) window.

- [Additional LabVIEW documentation](#).

You must have Adobe Reader 6.0.1 or later installed to view or [search](#) the PDF versions of these manuals.

Refer to the [Adobe Systems Incorporated Web site](#) to download Acrobat Reader. Refer to the [National Instruments Product Manuals Library](#) for updated documentation resources.

You must [install the PDFs](#) to access them from this help system.



**Note** The following resources offer useful background information on the general concepts discussed in this documentation. These resources are provided for general informational purposes only and are not affiliated, sponsored, or endorsed by National Instruments. The content of these resources is not a representation of, may not correspond to, and does not imply current or future functionality in the Control Design and Simulation Module or any other National Instruments product.

- Åström, K., and T. Hågglund. 1995. *PID controllers: theory, design, and tuning*. 2d ed. ISA.
- Balbis, Luisella. 2006. Predictive control tool kit. *UKACC control, 2006. Mini symposia*: 87–96.
- Bertsekas, Dimitri P. 1999. *Nonlinear Programming*. 2d ed. Belmont, MA: Athena Scientific.
- Dorf, R. C., and R. H. Bishop. 2001. *Modern control systems*. 9th ed. Upper Saddle River, NJ: Prentice Hall.
- Franklin, G. F., J. D. Powell, and A. Emami-Naeini. 2002. *Feedback control of dynamic systems*. 4th ed. Upper Saddle River, NJ: Prentice Hall.
- Franklin, G. F., J. D. Powell, and M. L. Workman. 1998. *Digital control of dynamic systems*. 3d ed. Menlo Park, CA: Addison Wesley Longman, Inc.
- Kuo, Benjamin C. 1992. *Digital Control Systems*. 2d ed. Ft. Worth: Saunders College.
- Nise, Norman S. 2000. *Control systems engineering*. 3d ed. New York: John Wiley & Sons, Inc.
- Ogata, Katsuhiko. 1995. *Discrete-time control systems*. 2d ed. Englewood Cliffs, N.J.: Prentice Hall.

- Ogata, Katsuhiko. 2001. *Modern control engineering*. 4th ed. Upper Saddle River, NJ: Prentice Hall.
- Zhou, Kemin, and John C. Doyle. 1998. *Essentials of robust control*. Upper Saddle River, NJ: Prentice Hall.

The following books contain information about the [ordinary differential equation](#) (ODE) solvers the Control Design and Simulation Module uses.

- Ascher, U. M., and L. R. Petzold. 1998. *Computer methods for ordinary differential equations and differential-algebraic equations*. Philadelphia: Society for Industrial and Applied Mathematics.
- Shampine, Lawrence F. 1994. *Numerical solution of ordinary differential equations*. New York: Chapman & Hall, Inc.

# Ordinary Differential Equation Solvers (Control Design and Simulation Module)

Because many [dynamic system models](#) consist of differential equations, you must [solve these differential equations](#) to observe the behavior of the simulated system. LabVIEW includes [several](#) ordinary differential equation (ODE) solvers that solve these equations. Before you simulate a dynamic system model, you must [specify and configure the ODE solver](#) for that simulation.

ODE solvers use methods to approximate the solution to a differential equation. The ODE solvers implement these methods in a variety of ways, each with various strengths and weaknesses. Defining characteristics of an ODE solver include the following qualities:

- [Accuracy or order](#)
- Stability
- Computational speed
- Use of a [fixed time step size](#) versus a variable time step size
- Use of a [single step](#) versus multiple steps
- Suitability for [stiff problems](#)

# Considerations for Embedded Targets (Control Design and Simulation Module)

If you install LabVIEW for a particular embedded target, you can use the LabVIEW Control Design and Simulation Module to develop and execute simulations on that target. Refer to the documentation for the target you purchased for information about developing and executing VIs on that target.

When developing a simulation for an embedded target, take the following factors into consideration:

- You cannot use the [Trim & Linearize](#) VIs or [Optimal Design](#) VIs when developing on an embedded target. Therefore, LabVIEW does not display these palettes when you are in an embedded context.
- You can use a [Conditional Disable](#) structure to enable real-world I/O when the simulation is on the embedded target and simulated I/O when the same simulation is running on a Windows computer. Using this method, you do not have to modify the VI manually to use different I/O code when you switch targets.
- Use as few Simulation functions as possible. Combine as many sequential transformations as possible into a single Simulation function. For example, instead of using three [State-Space](#) functions to describe a controller model, combine those functions into a single State-Space function.



**Note** You can use the [Model Interconnection](#) VIs to combine dynamic system models.

- Consider the speed/memory trade-off when using simulation subsystems. Using fewer simulation subsystems might increase execution speed but might require you to duplicate code elsewhere on the simulation diagram.
- Consider using [fixed step-size ordinary differential equation \(ODE\) solvers](#), that is, the solvers not marked **(variable)**. Variable step-size ODE solvers introduce computational overhead when changing step sizes.



**Note** The Control Design and Simulation Module does not guarantee accuracy when using the BDF or Rosenbrock

ODE solvers on embedded targets. If these solvers appear to produce inaccurate results on an embedded target, choose another ODE solver.

# Glossary (Control Design and Simulation Module)

A B C D E F H I L M N O P R S T

[LabVIEW Glossary](#)

## A

Ackermann A technique for placing poles in a system model. Use the [CD Ackermann](#) VI to implement this technique.

actuator A physical device that applies the control action to the plant.

auto-covariance A measure of how closely a value of a stochastic process, such as noise, varies with the subsequent value of that process.

## B

balanced system model A system model with identical controllability and observability diagonal Grammians. Use the [CD Balance State-Space Model \(Diagonal\)](#) VI and the [CD Balance State-Space Model \(Grammians\)](#) VI to balance a state-space system model.

Bode plot A plot that shows the gain and phase margins of a system model for a common frequency range. Bode plots show how close a system model is to instability. Use the [CD Bode](#) VI to create a Bode plot for a system model.

## C

- CGD Common Graph Description. The format the [Simulation Model Converter](#) uses to store each system, subsystem, block, and line from a model developed in The MathWorks, Inc. Simulink® simulation environment.
- constraints **See** [inequality constraints](#).
- continuous model A [dynamic system model](#) that represents real-world signals, which vary continuously with time. You characterize a continuous model by differential equations. **See also** [discrete model](#).
- controller A device that regulates the operation of a [dynamic system](#).
- cost The scalar value that results from solving a [cost function](#).
- cost function A performance measure you want to minimize when [designing optimal parameters](#). A cost function is a functional equation that maps a set of points in a time series to a single scalar value.

## D

design	See <a href="#">optimal design</a> .
direct feedthrough	A <a href="#">relationship</a> between a function input and a function output in which the function uses the input at the current step to calculate the output at the current step. <b>See also</b> <a href="#">indirect feedthrough</a> .
discrete model	A <a href="#">dynamic system model</a> that represents signals that are sampled at discontinuous intervals in time. You characterize a discrete model by difference equations. <b>See also</b> <a href="#">continuous model</a> .
distributed parameter model	A <a href="#">physical model</a> that you can describe with partial differential equations. <b>See also</b> <a href="#">lumped parameter model</a> .
dynamic system	A system whose behavior varies with time.
dynamic system model	A differential or difference equation that represents the behavior of all or part of a <a href="#">dynamic system</a> .

## E

empirical modeling A modeling technique in which you use experimental data to define a [dynamic system model](#). **See also** [physical modeling](#).

## F

feedback cycle A [cycle](#) in which data flow originates from an output of a function or subsystem and terminates as an input of the same function or subsystem. **See also** [indirect feedthrough](#).

## H

HIL Hardware-in-the-loop. A [simulation configuration](#) in which you test a controller implementation with a simulated system. **See also** [RCP](#).

## I

- indirect feedthrough A [relationship](#) between a function input and a function output in which the function does not use the input at the current step to compute the output at the current step. **See also** [direct feedthrough](#).
- inequality constraints Any restrictions you place on how the optimal design process determines optimal parameter values. You can [define inequality constraints](#) for the control action, the output, the rate of change of the control action, and the rate of change of the output.
- Input Node A collection of input terminals attached to the [Simulation Loop](#). Use the Input Node to [configure simulation parameters programmatically](#). **See also** [Output Node](#).

## L

- linear model A [dynamic system model](#) that obeys the principles of superposition and homogeneity. **See also** [nonlinear model](#).
- linearize A [procedure](#) that approximates the behavior of a nonlinear model. **See also** [trim](#).
- lumped parameter model A [physical model](#) you can describe with an ordinary differential equation. **See also** [distributed parameter model](#).

## M

model **See** [dynamic system model](#).

## N

nonlinear model A [dynamic system model](#) that does not obey the principles of superposition or homogeneity. **See also** [linear model](#).

## O

- offline A [simulation configuration](#) in which you use software to simulate the controller and the system you want to control. No hardware is involved in an offline simulation.
- optimal design The process of [selecting parameter values that maximize a measure of performance](#).
- Output Node An output terminal on the [Simulation Loop](#). Use the Output Node to view any errors the simulation diagram generates. **See also** [Input Node](#).

## P

parameter bounds Any [restrictions you place](#) on possible parameter values during the optimal design process. The optimal design process does not consider any parameter values outside the bounds you define.

parameter design **See** [optimal design](#).

parameter mesh A [set of points](#) that defines the distribution patterns of sets of parameter values and the number of sets to generate.

period The amount of time in which a [discrete linear Simulation function](#) must complete.

physical modeling A modeling technique in which you use the laws of physics to define a [dynamic system model](#). **See also** [empirical modeling](#).

plant A [dynamic system](#) whose behavior you want to observe, replicate, or manipulate.

## R

RCP Rapid control prototype. A [simulation configuration](#) in which you test plant hardware with a software model of the controller. **See also [HIL](#).**

## S

- simulation diagram A LabVIEW diagram that allows you to use [Simulation functions](#) within a [Simulation Loop](#) or [simulation subsystem](#). A simulation diagram, like other LabVIEW diagrams, has the following semantic properties: The order of operations is not completely specified by the user. The order of operations is implied by data interdependencies. A function can execute only after all necessary inputs have become available. Outputs are generated after a function completes execution.
- Simulation Loop The structure that executes the simulation diagram over multiple time steps.
- skew The amount of time by which you want to delay the execution of a [discrete linear Simulation function](#).
- SQP Sequential Quadratic Programming. A general-purpose [numerical optimization algorithm](#).
- subsystem A section of a simulation diagram you represent with a single icon instead of multiple Simulation functions and wires.

## T

time-invariant model     A [dynamic system model](#) whose parameters do not change with time.

time-variant model     A [dynamic system model](#) whose parameters change with time.

trim     A [procedure](#) that searches for the values of states and inputs that produce output and/or state derivative conditions you specify. **See also** [linearize](#).

# Building and Configuring Simulations (Control Design and Simulation Module)

Use the LabVIEW Control Design and Simulation Module to build a simulation diagram, which graphically displays a [dynamic system model](#) in LabVIEW. You build and execute a simulation diagram by placing [Simulation](#) functions and other LabVIEW VIs and structures inside the [Simulation Loop](#). The simulation diagram then uses an [ordinary differential equation](#) (ODE) solver to compute the behavior of the dynamic system model.

The simulation diagram supports standard LabVIEW [debugging techniques](#). You can use execution highlighting, breakpoints, probes, custom probes, and single-stepping on the simulation diagram.

 **(Windows)** To view related topics, click the **Locate** button, shown at left, in the toolbar at the top of this window. The *LabVIEW Help* highlights this topic in the **Contents** tab so you can navigate the related topics.

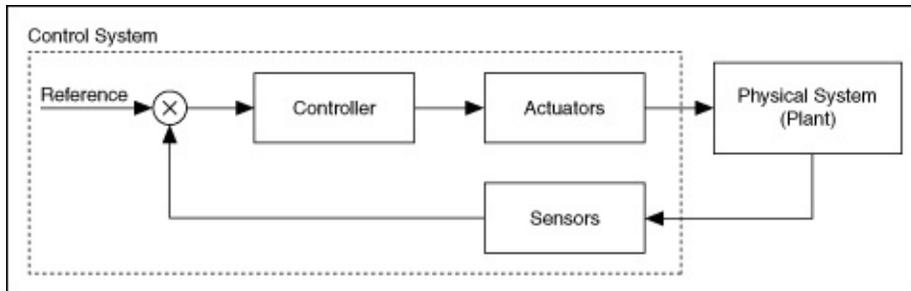
## Concepts

Use this book to learn about concepts in the LabVIEW Control Design and Simulation Module. Refer to the [How-To](#) book for step-by-step instructions for using the Control Design and Simulation Module.

 **(Windows)** To view related topics, click the **Locate** button, shown at left, in the toolbar at the top of this window. The *LabVIEW Help* highlights this topic in the **Contents** tab so you can navigate the related topics.

# Understanding Dynamic System Models (Control Design and Simulation Module)

A dynamic system model is a mathematical representation of the dynamics between the inputs and outputs of a dynamic system. You generally represent dynamic system models with differential or difference equations. The following figure shows a sample dynamic system.



The dynamic system in the previous figure represents a closed-loop system, also known as a feedback system. In closed-loop systems, the controller monitors the output of the plant and adjusts the actuators to achieve a specified response.

You can use physical laws or experimental data to develop a dynamic system model. The following sections describe features of both the physical modeling and the empirical modeling techniques.

## Physical Models

The laws of physics define the physical model of a system. The following sections describe various classifications and features of physical models.

### Lumped versus Distributed Parameter Models

If you can use an ordinary differential equation to describe a physical system, the resulting model is a lumped parameter model. If you can use a partial differential equation to describe a system, the resulting model is a distributed parameter model.

### Linear versus Nonlinear Models

Dynamic system models are either linear or nonlinear. A linear model obeys the principle of superposition and homogeneity. The following equations are true for linear models.

$$y_1 = f(u_1)$$

$$y_2 = f(u_2)$$

$$f(u_1 + u_2) = f(u_1) + f(u_2) = y_1 + y_2$$

$$f(au_1) = af(u_1) = ay_1$$

where  $u_1$  and  $u_2$  are the system inputs,  $y_1$  and  $y_2$  are the system outputs, and  $a$  is a constant.

Conversely, nonlinear models do not obey the principles of superposition or homogeneity. [Nonlinear effects](#) in real-world systems include [saturation](#), [dead-zone](#), [friction](#), [backlash](#), and [quantization](#) effects; [relays](#); [switches](#); and [rate limiters](#). Many real-world systems are nonlinear, though you can [linearize](#) nonlinear models to simplify a design or analysis procedure.

### Time-Variant versus Time-Invariant Models

Dynamic system models are either time-variant or time-invariant. The parameters of a time-variant model change with time. For example, you can use a time-variant model to describe the mass of an automobile. As fuel burns, the mass of the vehicle changes with time.

Conversely, the parameters of a time-invariant model do not change with time. For an example of a time-invariant model, consider a simple robot. Generally, the dynamic characteristics of robots do not change over short

periods of time.

### **Continuous versus Discrete Models**

Dynamic system models are either continuous or discrete. Both continuous and discrete system models can be linear or nonlinear and time-invariant or time-variant. Continuous models describe how the behavior of a system varies continuously with time, which means you can obtain the properties of a system at any certain moment from the continuous model. Discrete models describe the behavior of a system at separate time instants, which means you cannot obtain the behavior of the system between any two sampling points.

Continuous system models are analog. You derive continuous models of a physical system from differential equations of the system. The coefficients of continuous models have clear physical meanings. For example, you can derive the continuous transfer function of a resistor-capacitor (RC) circuit if you know the details of the circuit. The coefficients of the continuous transfer function are the functions of R and C in the circuit. You use continuous models if you need to match the coefficients of a model to some physical components in the system.

Discrete system models are digital. You derive discrete models of a physical system from difference equations or by converting continuous models to discrete models. In computer-based applications, signals and operations are digital. Therefore, you can use discrete models to implement a digital controller or to simulate the behavior of a physical system at discrete instants. You also can use discrete models in the accurate model-based design of a discrete controller for a plant.

## Empirical Models

Empirical models use data gathered from experiments to define the mathematical model of a system. To some degree, physical models are empirical because you experimentally determine certain constants used to develop the model. A variety of empirical modeling methods exist. One method of empirical modeling uses tables of experimental data that represent the system you want to model. Another method for developing models uses system identification methods. System identification methods use measured data to create differential or difference equation representations that model the data.



**Note** You can use the LabVIEW System Identification Toolkit to construct models by using system identification methods.

## Linear Model Forms

You can use the Control Design and Simulation Module to represent [continuous](#) and [discrete](#) linear models in the following three forms:

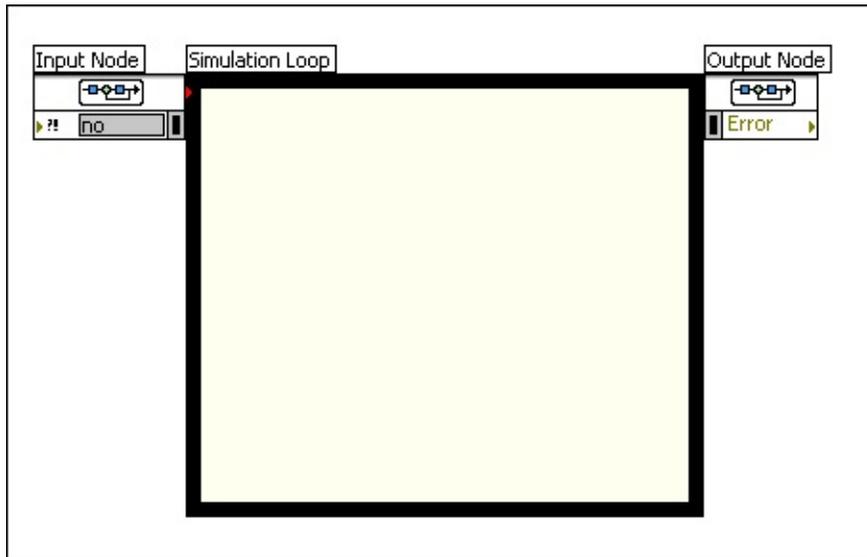
- **Transfer Function**—These models use polynomial functions to define the relationship between the inputs and outputs of a dynamic system. You analyze transfer function models in the frequency domain.
- **Zero-Pole-Gain**—These models are transfer function models that you rewrite to show the gain and the locations of the zeros and poles of the dynamic system. You analyze zero-pole-gain models in the frequency domain.
- **State-Space**—These models represent the dynamic system in terms of physical states. Continuous state-space models use first-order differential equations to describe the dynamic system, whereas discrete state-space models use first-order difference equations. You analyze state-space models in the time domain.

## **SISO vs. MIMO Models**

When you configure a continuous or discrete transfer function, zero-pole-gain, or state-space function, you can use the [configuration dialog box](#) to specify whether a function is single-input single-output (SISO) or multiple-input multiple-output (MIMO). SISO models have only one input and one output, whereas MIMO models have two or more inputs or outputs. You make this choice by selecting the appropriate option from the **Polymorphic instance** pull-down menu.

# Configuring Simulation Parameters (Control Design and Simulation Module)

The [Simulation Loop](#), shown in the following figure, contains the parameters that define the behavior of the simulation.



You can configure these simulation parameters by using the following two methods:

- Using the [Configure Simulation Parameters](#) dialog box
- Wiring values to the Input Node

You also can use a combination of these two methods in the same simulation diagram. However, values that you programmatically configure override any equivalent settings that you make in the **Configure Simulation Parameters** dialog box.



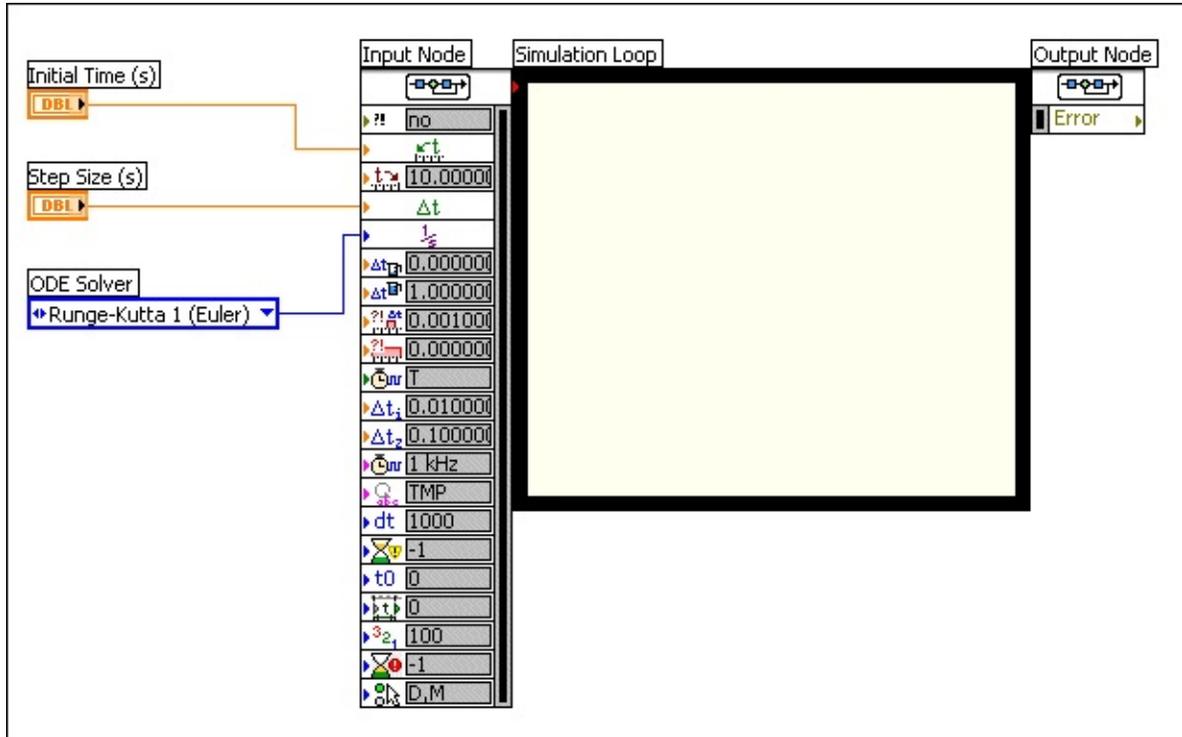
**Tip** Use the Output Node to view any errors that occur during the execution of the Simulation Loop. Use the [Get Simulation Parameters](#) function to display the parameters you configure for the simulation.

The Simulation Loop is a version of the [Timed Loop](#). Both loops can iterate deterministically according to a period and priority you define. However, the Simulation Loop also executes the simulation diagram according to the [ordinary differential equation](#) (ODE) solver you choose. The inside of the Simulation Loop also has a pale yellow background to distinguish the simulation diagram from the LabVIEW block diagram.

Similar to [other loops and structures](#), you can use tunnels to pass data in and out of the Simulation Loop.

## Configuring Simulation Parameters Programmatically

The following figure shows how you configure a simulation diagram programmatically.

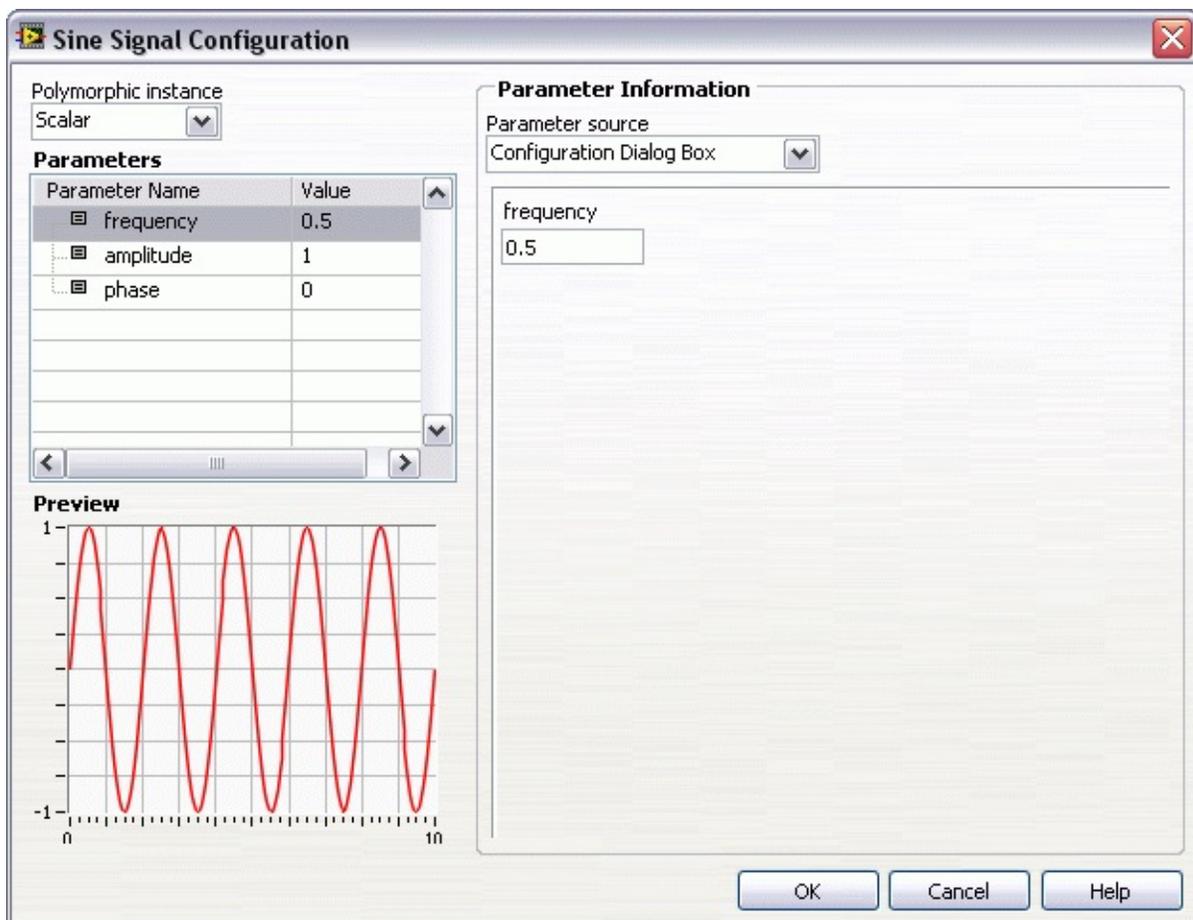


The previous figure shows how the gray boxes on the Input Node display any values that you configure in the **Configure Simulation Parameters** dialog box. Values that you configure programmatically do not have gray boxes.

## Using Simulation Functions and VIs (Control Design and Simulation Module)

The LabVIEW Control Design and Simulation Module includes both [functions and VIs](#). The [Simulation](#) functions are the elements that comprise a simulation model. Use these functions to perform tasks such as defining [dynamic system models](#), [generating](#) and [combining](#) input signals, and [analyzing](#) and [displaying](#) simulation data.

You can configure most Simulation functions using the configuration dialog box of that function. After you place a Simulation function on the simulation diagram, double-click that function to launch its configuration dialog box. You also can launch this dialog box by right-clicking the Simulation function and selecting **Configuration** from the shortcut menu. For example, the following figure shows the configuration dialog box for the [Sine Signal](#) function.



The **Parameters** section lists all the parameters that you can configure

for the Sine Signal function. When you select a parameter from the **Parameters** section, the **Parameter Information** section displays a control you can use to set the value of that parameter.

Use the **Parameter source** control to specify the source of the parameter value. If you select **Configuration Dialog Box**, LabVIEW removes that input from the simulation diagram. You then must set the value for this parameter in the configuration dialog box. If you select **Terminal**, LabVIEW displays an input terminal for that parameter on the simulation diagram, and you can wire values to this input to configure the Simulation function.

The parameters you specify for a Simulation function are unique to that function. If you create multiple instances of the same function, you can set different parameter values for each instance.



**Note** Discrete simulation functions have additional parameters you use to configure the period and skew of the function.

In addition to the Simulation functions, the Control Design and Simulation Module includes VIs. Use these VIs to perform tasks indirectly related to the simulation, such as trimming and linearizing a dynamic system model or designing optimal parameters for a dynamic system model.

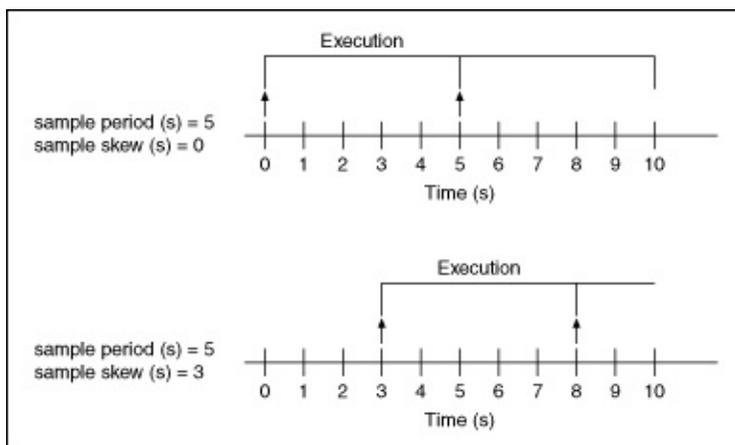
## Dynamic Simulation Functions

The following Simulation functions are dynamic elements that depend on the [ordinary differential equation](#) (ODE) solver you [specify](#).

- [Integrator](#)
- [Transfer Function](#)
- [Zero-Pole-Gain](#)
- [State-Space](#)

## Configuring Discrete Simulation Functions (Control Design and Simulation Module)

The [Discrete Linear Systems](#) functions have a **sample period (s)** parameter and a **sample skew (s)** parameter. These parameters are located in the configuration dialog box of that function. The **sample period (s)** parameter sets the length of the step size of that function. The **sample skew (s)** parameter delays the execution of that function. The following figure shows how these two parameters affect the execution of a discrete Simulation function.



The **sample period (s)** of a discrete function must be a multiple of the discrete step size of the simulation. To configure this overall discrete step size, double-click the Input Node of the [Simulation Loop](#) to launch the [Configure Simulation Parameters](#) dialog box. On the [Simulation Parameters](#) page, you can enter the **Discrete Step Size (s)** or [automatically configure the discrete step size of the simulation](#).



**Note** If you enter a value of  $-1$  for the **sample period (s)** parameter of a Simulation function, that function inherits the same step size as defined in the [Configure Simulation Parameters](#) dialog box.

# **Generating, Collecting, and Displaying Simulation Data (Control Design and Simulation Module)**

The LabVIEW Control Design and Simulation Module includes several functions you use to generate, collect, and display simulation data. The following sections provide information about using these functions.

## Generating and Combining Signals

Use the [Signal Generation](#) functions to generate many different types of signals, including [sine](#), [ramp](#), [step](#), [pulse](#), and [chirp](#) signals. These functions are useful when you want to see how a dynamic system responds to a particular type of input. For example, the [Step Signal](#) function generates a step signal, which you commonly use to test controller performance. Each Signal Generation function has configuration options you can use to fit the needs of a particular situation. Use the [Signal Arithmetic](#) functions to add, subtract, multiply, and divide signals.

## Collecting and Indexing Simulation Data

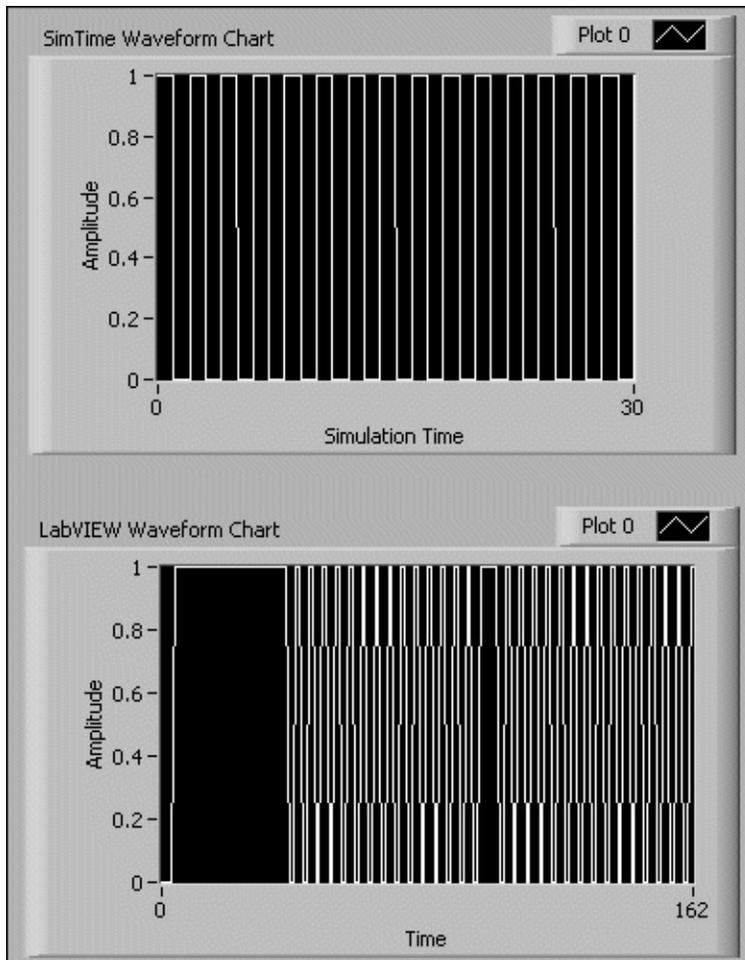
To store all or part of a signal history over the entire simulation for later analysis, use the [Collector](#) function. This function stores values in an array, similar to the auto-indexing output tunnel of a [For Loop](#). However, whereas LabVIEW indexes For Loop arrays by the loop iteration, the Control Design and Simulation Module indexes Collector arrays by simulation time. Therefore, after the simulation finishes, you can see the values that correspond with certain points in time.

The opposite of the Collector function is the [Indexer](#) function, which takes an array of data and returns the correct value based on the simulation time. For example, you can define an arbitrary signal as an array of timestamps and values at each timestamp. You then wire this array to the **Input** input of an Indexer function. When you run the simulation, this function returns the correct array value at the correct time. If you do not define a value for a specific time, this function [linearly interpolates](#) the expected result according to several options you can specify. The Indexer function operates similarly to the auto-indexing input tunnel of a For Loop, except LabVIEW indexes For Loop arrays by loop iteration instead of simulation time.

## Displaying Simulation Data

The [SimTime Waveform](#) function and [Buffer XY Graph](#) function operate similarly to the LabVIEW [Waveform Chart](#) and [XY Graph](#) objects. However, whereas LabVIEW displays the loop iteration on the x-axis of these objects, the Simulation versions of these functions properly display simulation time on the x-axis. This distinction is important when you are using a [variable step-size ordinary differential equation](#) (ODE) solver. In this situation, the simulation might not return a value at every loop iteration, due to changes in the step size. However, the LabVIEW Waveform Chart still plots a value every loop iteration. The SimTime Waveform function corrects this behavior and properly displays unevenly-spaced values on the x-axis.

The following figure shows how a **SimTime Waveform Chart** and a **LabVIEW Waveform Chart** display the output of the [Pulse Signal](#) function when simulated for 30 seconds using a variable step-size ODE solver.



Notice the irregularities in the **LabVIEW Waveform Chart** when compared to the **SimTime Waveform Chart**.



**Note** When you place a SimTime Waveform function or Buffer XY Graph function on the simulation diagram, LabVIEW automatically creates a chart or graph object connected to the function output.

## Storing Precalculated Data in Lookup Tables

Lookup tables are useful for defining sets of experimental data, such as the result of a function, and then retrieving that data without calculating the function. If you store the data in a lookup table, the simulation does not have to compute the function every time step. Instead, you call the lookup table to obtain the appropriate value. In this situation, you improve computation performance by reducing the need to calculate functions at each iteration of the [Simulation Loop](#).

A lookup table consists of two data sets: a set of table values and a corresponding set of data values. When you specify an input value, the lookup table matches that input value to a table value and returns the appropriate data value.

For example, consider a lookup table with table values of [0 1 5] and data values of [4 2 8]. If you specify an input value of 0, the lookup table returns 4. If you specify an input value of 5, the lookup table returns 8. You also can configure how the lookup table operates if you specify an input value that does not exist as a table value. For example, you can configure a lookup table to [interpolate or extrapolate](#) the appropriate data value from the available table values. Use the **Method** parameter of the [Lookup Tables](#) functions to define this behavior.

This example uses a [one-dimensional lookup table](#); however, the Control Design and Simulation Module also includes functions that implement [two-](#) and [three-dimensional](#) lookup tables.

## Transferring Data Between Simulation Loop Iterations

Use the [Memory](#) function to transfer the value of a signal from one iteration of the Simulation Loop to the next. This function behaves similarly to a [shift register](#) you can place on a [While Loop](#). This function is polymorphic and accepts any data type you wire to the **Initial Value** input. To implement a fixed-time delay, use the [Discrete Unit Delay](#) function.

## Changing Function Icon Styles (Control Design and Simulation Module)

You can change the icon style of a [Simulation](#) function on the simulation diagram. Right-click a Simulation function and select **Icon Style** from the shortcut menu to display the following options:

- **Static**—Displays the Simulation function as a standard VI.
- **Dynamic**—Displays the Simulation function as an object that you can resize. Dynamic icons also display a preview of their contents. For example, a Sine Signal function with a dynamic icon displays a sine wave with the frequency, amplitude, and phase that you configure.
- **Text Only**—Displays the Simulation function as a list of parameter values.
- **Express**—Displays the Simulation function with a list of parameters below the icon. You can resize the parameter list to display more inputs and outputs. This icon style also shows parameter values directly on the simulation diagram.

# **Determining Feedthrough Behavior and Defining Feedback Cycles (Control Design and Simulation Module)**

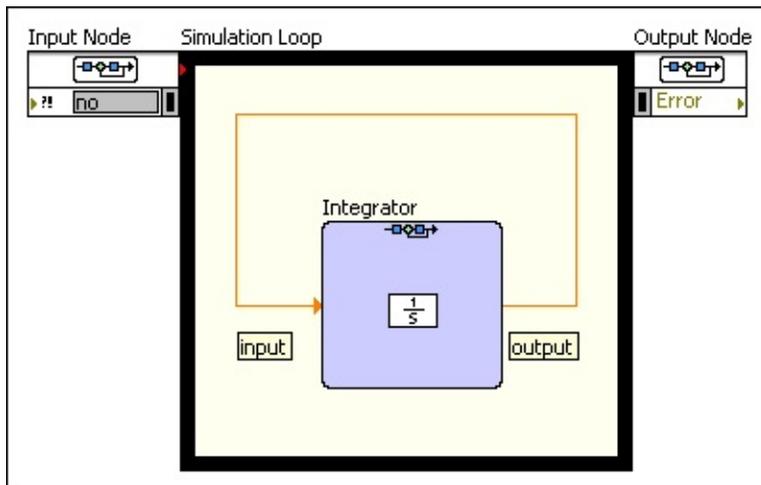
The relationship between a function input and output defines the feedthrough behavior of that I/O pair. An I/O pair can have indirect, direct, or parameter-dependent feedthrough behavior. If an I/O pair has indirect feedthrough behavior, you can create a feedback cycle between that input and output. An I/O pair with direct feedthrough behavior does not allow a feedback cycle. The following sections provide more information about these behaviors.

## Indirect Feedthrough and Feedback Cycles

Whereas LabVIEW VIs execute only after receiving the value of all inputs to that VI, many [Simulation](#) functions can execute without receiving the value of certain inputs. Consider a Simulation function with input  $u$  and output  $y$ . At any time step, if the function does not require the value of  $u$  to compute the value of  $y$ ,  $u$  has indirect feedthrough to  $y$ .

When indirect feedthrough exists between  $u$  and  $y$ , you can create a feedback cycle between these parameters. In a feedback cycle, the value of  $y$  at time  $t$  relies on the value of  $u$  at time  $t - dt$ ,  $t - dt_2$ , and so on.

For example, the **input** parameter of the Integrator function has indirect feedthrough to the **output** parameter. You can create a feedback cycle between this input and output. The following figure shows this behavior:



You can use one or more Simulation functions and other LabVIEW functions in a feedback cycle as long as at least one Simulation function in the feedback cycle has indirect feedthrough behavior. The indirect feedthrough function can start the data flow by executing the function output at the current step before receiving an input from the cycle at the current step. Therefore, the input at the current step and the output at the current step must not depend on each other directly in at least one function in the cycle.

The following Simulation functions have at least one I/O pair with indirect feedthrough.

- [Discrete Kalman Filter](#)
- [Discrete Observer](#)

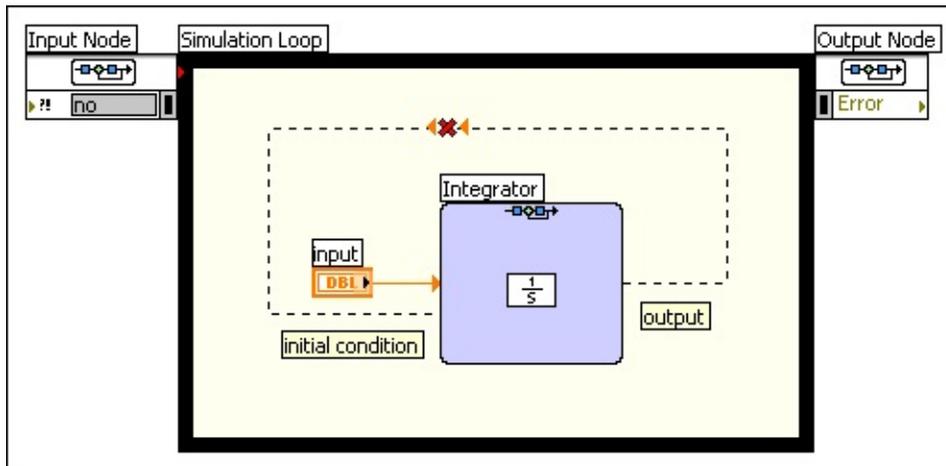
- [Discrete Stochastic State-Space](#)
- [Discrete Unit Delay](#)
- [Integrator](#)
- [Memory](#)
- [Transport Delay](#)

## Direct Feedthrough

If a function output  $y$  requires an input  $u$  in order to execute,  $u$  has direct feedthrough to  $y$ . You cannot create a feedback cycle between inputs and outputs with direct feedthrough.

For example, the **initial condition** parameter of the Integrator function has direct feedthrough behavior to the output parameter. This function requires a value for the **initial condition** parameter in order to calculate the **output** parameter. Other functions, such as [Friction](#), require the values of all inputs in order to execute.

If you attempt to create a feedback cycle between an input and output with direct feedthrough, the wire appears broken. The following figure shows this behavior:



Notice the difference between the previous figure and the figure showing the feedback cycle.

## Parameter-Dependent Feedthrough

Several functions have feedthrough behavior that depends on how you configure the parameters of that function. For example, consider the Transfer Function function. The feedthrough behavior of this function depends on the order of the numerator and denominator polynomial equations you specify. The following Simulation functions have at least one I/O pair with parameter-dependent feedthrough.

- [Transfer Function](#)
- [Zero-Pole-Gain](#)
- [State-Space](#)
- [Discrete Filter](#)
- [Discrete Integrator](#)
- [Discrete Transfer Function](#)
- [Discrete Zero-Pole-Gain](#)
- [Discrete State-Space](#)

If you use the configuration dialog box to define the parameters of these functions, such as the numerator and denominator of a transfer function, LabVIEW automatically determines the appropriate feedthrough behavior and displays this choice in the **Feedthrough** pull-down menu. However, if you use block diagram terminals to define the parameters of these functions, you must set the feedthrough behavior manually.

All Simulation functions not mentioned in this section or in the [Indirect Feedthrough and Feedback Cycles](#) section have direct feedthrough.

## Placing LabVIEW VIs, Functions, and Structures on the Simulation Diagram (Control Design and Simulation Module)

You can use a majority of LabVIEW VIs and functions to describe a [dynamic system model](#). However, you cannot place certain structures, such as the [While Loop](#), [For Loop](#), or [Event structure](#), directly on the simulation diagram. Instead, you can place these structures in a [subVI](#), which you then place on the simulation diagram.

By default, the Control Design and Simulation Module executes VIs and Express VIs as continuous functions. You can change this behavior by using the [SubVI Node Setup](#) dialog box. To launch this dialog box, right-click the object and select **SubVI Node Setup** from the shortcut menu. You can configure a VI to execute at only major time steps of the ODE solver, at both major and minor time steps of the ODE solver, as a discrete function, or at initialization of the simulation diagram.

## Using Case Structures on the Simulation Diagram

You can place a [Case structure](#) directly on the simulation diagram. The value you wire to the selector terminal determines which model to evaluate. If an input-output pair on any subdiagram of the case structure contains [direct feedthrough](#), you cannot create a [feedback cycle](#) between that input and output.

National Instruments recommends you use a [fixed step-size ordinary differential equation](#) (ODE) solver when using a Case structure on the simulation diagram.



**Note** You cannot place [front panel terminals](#) inside a Case structure on a simulation diagram.

## How-To

This book contains step-by-step instructions and other information that might be useful as you use the LabVIEW Control Design and Simulation Module. Refer to the [Concepts](#) book to learn about related concepts.

 **(Windows)** To view related topics, click the **Locate** button, shown at left, in the toolbar at the top of this window. The *LabVIEW Help* highlights this topic in the **Contents** tab so you can navigate the related topics.

## Modularizing the Simulation Diagram (Control Design and Simulation Module)

The LabVIEW Control Design and Simulation Module supports [simulation subsystems](#), which you use to modularize and encapsulate portions of the simulation diagram. You [create simulation subsystems](#) similarly to creating subVIs.



**(Windows)** To view related topics, click the **Locate** button, shown at left, in the toolbar at the top of this window. The *LabVIEW Help* highlights this topic in the **Contents** tab so you can navigate the related topics.

## Types of Simulation Subsystems (Control Design and Simulation Module)

Simulation subsystems provide a way to modularize simulation diagram code. By combining several functions into a subsystem, you reduce the amount of space needed on the simulation diagram, making the simulation easier to navigate visually. Simulation subsystems also are useful for validating, distributing, and reusing portions of the simulation diagram.

You can use many [Simulation](#) VIs and functions only on a simulation diagram, such as within a [Simulation Loop](#). Because simulation subsystems modularize simulation diagram code, you can use the Simulation VIs and functions in simulation subsystems as well. The entire block diagram of a simulation subsystem is pale yellow like the inside of a Simulation Loop. When you use Simulation VIs and functions in a simulation subsystem, you place the VIs and functions directly on the subsystem block diagram rather than within a Simulation Loop.

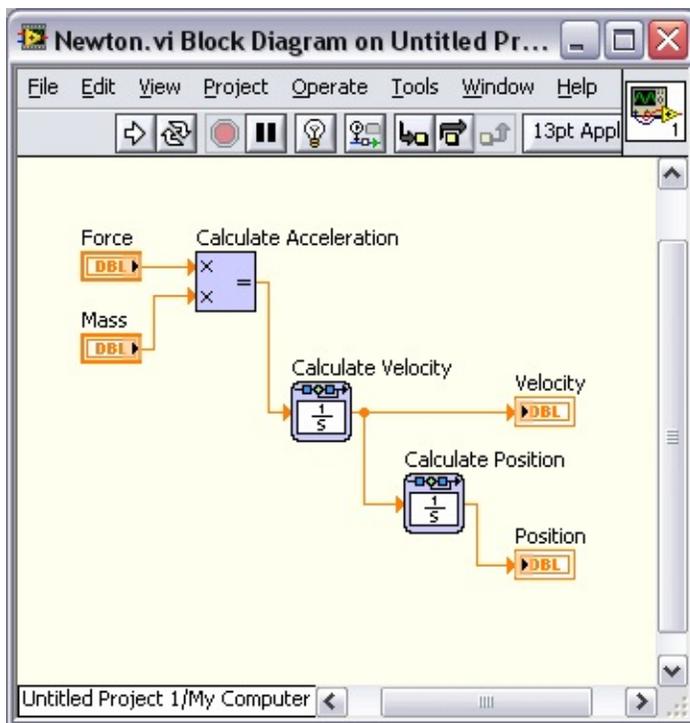
You can run simulation subsystems as [stand-alone VIs](#), [within a Simulation Loop or another simulation subsystem](#), or [on a block diagram outside a Simulation Loop](#).

## Running a Subsystem as a Stand-Alone VI

When you run a simulation subsystem as a stand-alone VI, you configure the simulation parameters by selecting **Operate»Configure Simulation Parameters** to launch the [Configure Simulation Parameters](#) dialog box. You also can configure the execution and appearance of the subsystem by selecting **File»VI Properties**.

When running a simulation subsystem as a stand-alone VI, you can use standard LabVIEW [debugging](#) techniques, such as [execution highlighting](#), [breakpoints](#), [probes](#), and [single-stepping](#). You cannot use these techniques on a subsystem that is within another Simulation Loop. You also cannot step into the subsystem. However, you can set a breakpoint on the entire subsystem by right-clicking the subsystem and selecting **Breakpoint»Set Breakpoint** from the shortcut menu. You also can use a probe or a custom probe to monitor the subsystem output.

The following figure shows the simulation diagram of a simulation subsystem `Newton.vi`, which obtains the position of a mass by using Newton's Second Law of Motion.



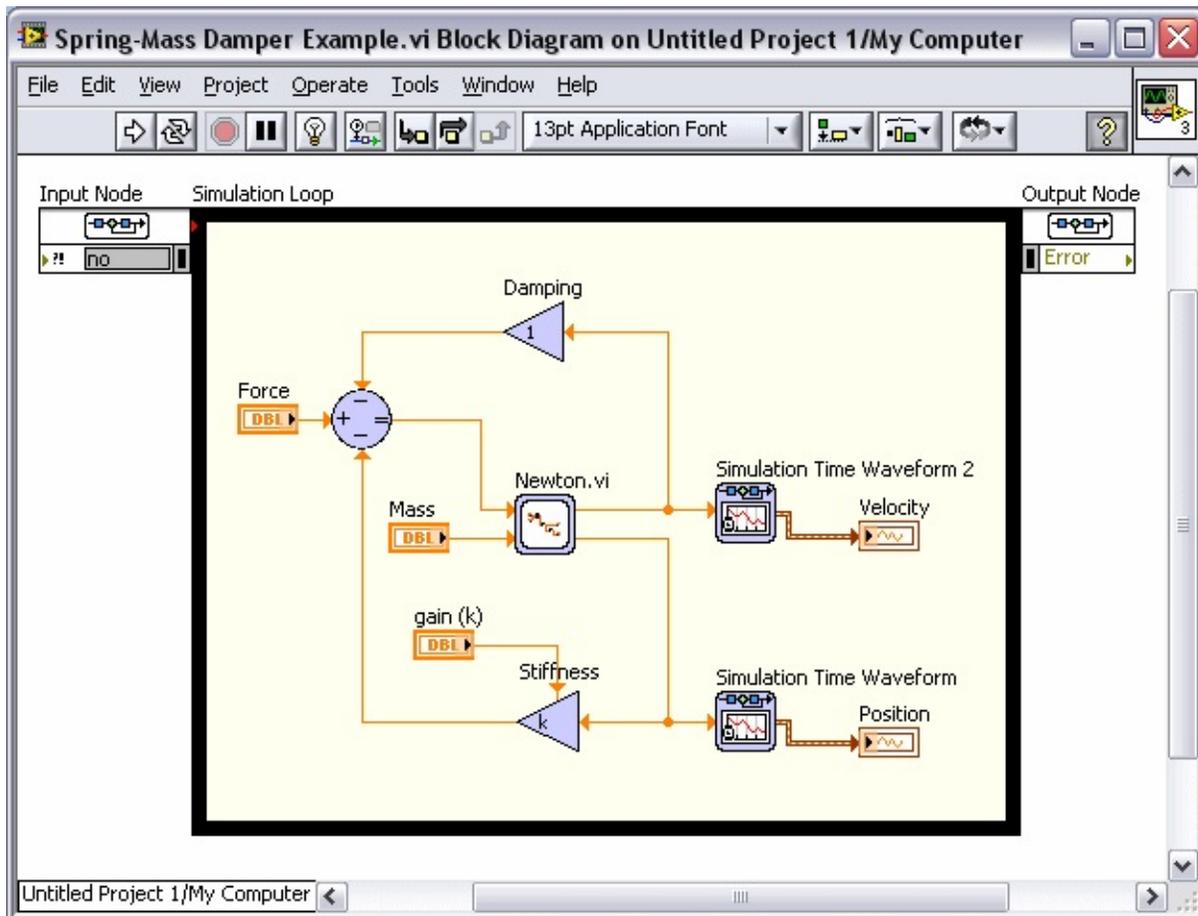
In the previous figure, this subsystem has front panel controls and indicators, so you can run this subsystem by clicking the **Run** button. Because `Newton.vi` does not have a Simulation Loop, you configure the

parameters of this subsystem by selecting **Operate»Configure Simulation Parameters**.

## Running a Subsystem in a Simulation Loop

If you run a simulation subsystem inside a Simulation Loop, the simulation subsystem inherits the parameters from the Simulation Loop.

The following figure shows Newton.vi included within the Simulation Loop of another simulation diagram.



In the previous figure, the parameters of the Simulation Loop override any parameters you configured specifically for Newton.vi.



**Note** You create this subsystem and this VI in the [Getting Started with Simulation](#) tutorial.

## Running a Subsystem Outside a Simulation Loop

If you run a simulation subsystem on a block diagram outside a Simulation Loop, the simulation subsystem executes one step of the [ordinary differential equation](#) (ODE) solver each time the simulation subsystem is called. For example, if you place the simulation subsystem in a [Timed Loop](#), one step of the ODE solver executes at each iteration of the loop. You can use only fixed step-size ODE solvers for a simulation subsystem outside a Simulation Loop. Specify the time step using the **Step Size (s)** configuration option of the subsystem.



**Tip** If you create an application with a simulation subsystem inside a Timed Loop and then deploy the application to a real-time target, set the **Step Size (s)** of the subsystem equal to the **period** of the Timed Loop.

When you place a simulation subsystem on a block diagram outside a Simulation Loop, the icon of the simulation subsystem appears in the **Express style** by default. You can wire values to the parameters of the simulation subsystem to configure the simulation subsystem programmatically.



**Note** The **Static** and **Dynamic** icon styles are disabled for subsystems outside a Simulation Loop.

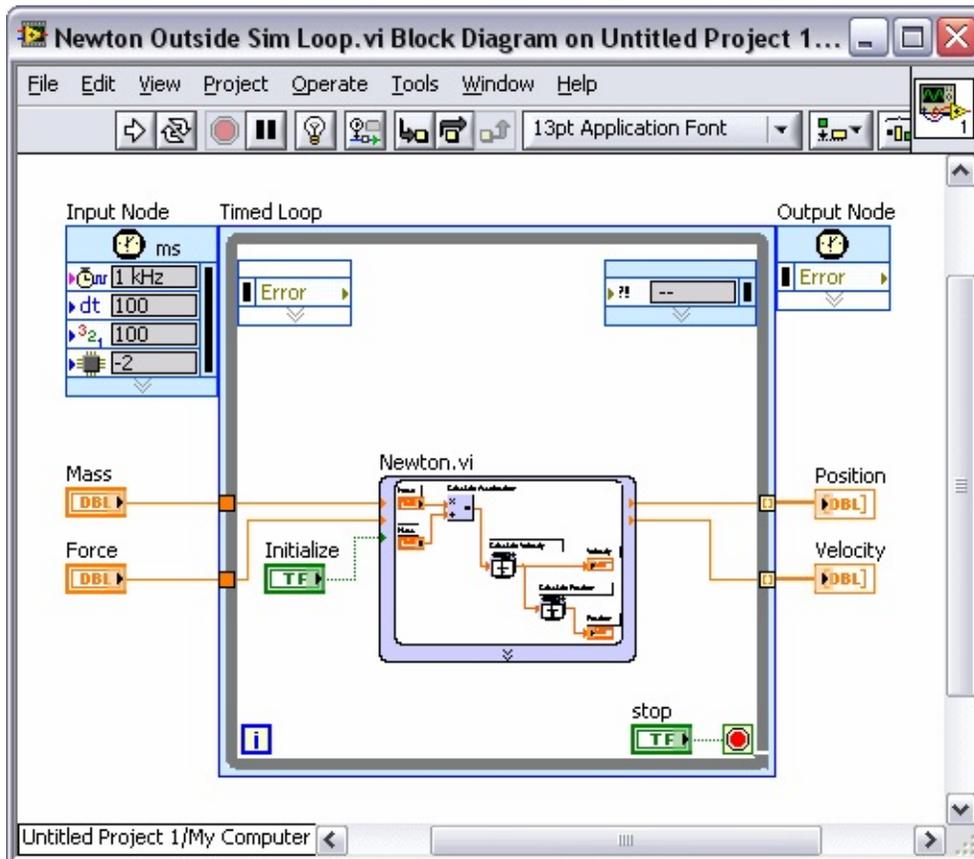
You also can configure the parameters of the simulation subsystem interactively. Double-click the simulation subsystem to launch the configuration dialog box of that simulation subsystem. In this configuration dialog box, you can configure the specific parameters of the simulation subsystem as well as the following general simulation parameters:

- **Initial Time (s)**—Specifies the time at which to start the ODE solver.
- **ODE Solver**—Specifies the type of ODE solver the simulation uses.
- **Nan/Inf Check**—Specifies that you want the Control Design and Simulation Module to check the simulation values for [not a number](#) (NaN) and infinite (Inf) values. The Control Design and Simulation Module returns an error if it encounters a NaN or Inf value.

- **Step Size (s)**—Specifies the interval, in seconds, between the times at which the ODE solver evaluates the model and updates the model output.
- **Discrete Step Size (s)**—Specifies the base time step size, in seconds, for the simulation.
- **Auto Discrete Time**—Automatically calculates the **Discrete Step Size (s)**.

These parameters are identical to parameters you can configure in the **Configure Simulation Parameters** dialog box of a Simulation Loop. In the configuration dialog box of a simulation subsystem, these parameters appear as sub-items of the **Simulation Parameters** item in the **Parameters** tree. If the simulation subsystem already contains a parameter with the same name as one of the general simulation parameters listed above, LabVIEW appends the general simulation parameter name with an underscore.

The following figure shows Newton.vi on a block diagram outside a Simulation Loop.



In the previous figure, the Timed Loop determines when Newton.vi executes the next step of the ODE solver. Because Newton.vi is outside a Simulation Loop, you can configure the simulation parameters, including the ODE solver to use, by double-clicking the simulation subsystem.

If you run a simulation subsystem outside a Simulation Loop, you can reinitialize the simulation subsystem by setting the **Initialize** input of the simulation subsystem to TRUE. This input restarts the ODE solver from the specified **Initial Time (s)**. The **Initialize** input is available for a simulation subsystem only when the subsystem is not inside a Simulation Loop or another simulation subsystem.

## Polymorphic Subsystems

If you create one or more subsystems that perform the same operation on different data types, you can package those subsystems together to create a polymorphic subsystem. A polymorphic subsystem is a [single VI](#) that points to one or more subsystems, called instances. Each instance accepts a different data type for a single input or output terminal. LabVIEW automatically selects the correct instance based on the input data type.

For example, one subsystem could operate on a double-precision floating point number, while another subsystem performs the same operation on a 16-bit integer. Instead of placing both subsystems on the simulation diagram, you can [create a polymorphic subsystem](#) that automatically chooses the correct instance.

For a polymorphic subsystem to work, each instance of the polymorphic subsystem must be a simulation subsystem. You cannot create a polymorphic subsystem with both VIs and subsystems as instances. Also, each subsystem must have an identical [connector pane pattern](#). Additionally, the names of corresponding input parameters for each instance must be identical.

# Trimming and Linearizing Nonlinear Models (Control Design and Simulation Module)

Many real-world [dynamic system models](#) are [nonlinear](#). If you want to design a controller for a nonlinear model, you must first linearize that model. Linearizing a nonlinear model involves approximating the behavior of that model around an operating point. The operating point is the set of the inputs and states of the model. When you linearize a model, the result is a [linear time-invariant](#) (LTI) [state-space](#) model.



**Note** You can design a controller for LTI models using the [Control Design](#) VIs and functions.

Trimming a model involves searching for values of model inputs and states that satisfy any conditions you specify. For example, you can specify that the model outputs or state derivatives must have a certain value. Trimming the model using these conditions returns the values of the inputs and states that, when given to the model, produce the outputs and state derivatives you specified. You also can trim a model to determine an operating point about which you linearize the model.

The LabVIEW Control Design and Simulation Module provides several methods for trimming and linearizing models. You can [interactively trim and linearize a model](#). You also can [programmatically trim](#) and [programmatically linearize](#) a model.



**Note** The above methods operate on continuous [simulation subsystems](#). Therefore, you must [create a simulation subsystem](#) that represents the model before trimming or linearizing that model.



**(Windows)** To view related topics, click the **Locate** button, shown at left, in the toolbar at the top of this window. The *LabVIEW Help* highlights this topic in the **Contents** tab so you can navigate the related topics.

## Executing Simulations in Real Time (Control Design and Simulation Module)

You can use the LabVIEW Control Design and Simulation Module with the LabVIEW Real-Time Module and various National Instruments real-time (RT) targets to implement simulations and controllers in real time with real-world inputs and outputs. For example, you can combine this software and hardware to design and implement a rapid control prototype (RCP) or hardware-in-the-loop (HIL) configuration. You [configure the timing Simulation Loop](#) according to the needs of the simulation.

The Control Design and Simulation Module can execute VIs on hardware targets running the real-time operating system (RTOS) of the Ardenne Phar Lap [Embedded Tool Suite](#) (ETS) or Wind River VxWorks.



**Note** The Control Design and Simulation Module supports only ETS and VxWorks targets with at least 32 MB of RAM.

The Real-Time Module includes the [Getting Started with the LabVIEW Real-Time Module](#) manual, which introduces the concepts necessary to create real-time applications. The Real-Time Module documentation also includes topics about [organizing and managing projects](#), [creating deterministic applications](#), and [sharing data in deterministic applications](#).



**(Windows)** To view related topics, click the **Locate** button, shown at left, in the toolbar at the top of this window. The *LabVIEW Help* highlights this topic in the **Contents** tab so you can navigate the related topics.

## **Offline, RCP, and HIL Configurations (Control Design and Simulation Module)**

The following sections provide an overview of the process you might use to simulate a dynamic system. The following sections also describe examples of offline simulations, rapid control prototyping (RCP) configurations, and hardware-in-the-loop (HIL) configurations.

## Offline Simulation

An offline simulation is one that is not connected to any hardware. You use the LabVIEW Control Design and Simulation Module to simulate all parts of the dynamic system, including the controller, the system you want to control, and any inputs or outputs. The following figure represents a simulation of an offline control system.

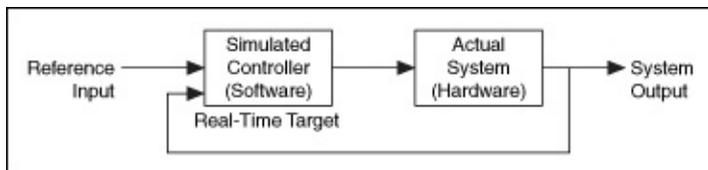


If you are running an offline simulation on a Windows computer, National Instruments recommends you place a checkmark in the **Synchronize Loop to Timing Source** checkbox on the **Timing Parameters** page of the [Configure Simulation Parameters](#) dialog box for optimal performance.

## Rapid Control Prototype Configuration

An RCP configuration replaces the simulated system with an actual system. Use this configuration to test multiple controller algorithms without building the controller again every time you make a change. In this situation, the simulated controller is connected to hardware actuators and hardware sensors. To convert an offline simulation to an RCP configuration, remove the system model from the simulation. Replace the system input with an output from a hardware device, and replace the system output with an input from a hardware device.

The following figure represents an RCP configuration.



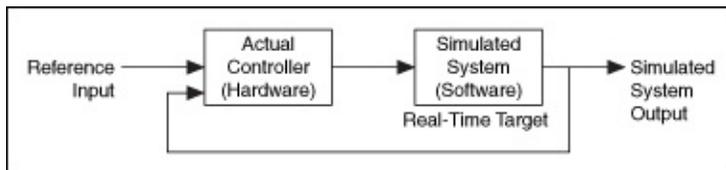
In the previous figure, the simulated controller uses National Instruments real-time hardware, such as a DAQ device, to send data to the hardware system.

## Hardware-in-the-Loop Configuration

A HIL configuration involves the actual controller providing input to a simulated system. Use this configuration to test a controller on a system without actually having that system available. For example, if you were testing an engine control unit (ECU) for a car, you could test the ECU without having to build the car multiple times. HIL configurations also are useful for testing controllers under extreme conditions that you cannot replicate conveniently in a laboratory.

To convert an offline simulation to a HIL configuration, remove the controller model from the simulation. Replace the controller input with an output from a hardware device, and replace the controller output with an input from a hardware device. The result is similar to the RCP configuration, except with the controller model, not the system model, replaced with physical hardware inputs and outputs.

The following figure shows a HIL configuration of the example in the previous figure.



In the previous figure, the controller uses National Instruments real-time hardware, such as a DAQ device, to send data to the simulated system.

# Deterministic ODE Solvers (Control Design and Simulation Module)

Running a simulation or controller in real time means that the simulation time must equal the wall-clock time at each point that the simulation or controller interacts with the real world. Generally, these physical interaction points correspond to the sampling points of the input and output hardware. Therefore, at each sampling time, the simulation time must equal the wall-clock time.

To meet the real-time deadline, you can configure the LabVIEW Control Design and Simulation Module to execute deterministically by [placing a strict upper bound on the execution time of the Simulation Loop](#). National Instruments also recommends you [configure the Simulation Loop](#) to use a [fixed step-size ordinary differential equation](#) (ODE) solver. These ODE solvers are deterministic, which ensures that block diagram code running at each time step meets the deadlines imposed by the timing of the hardware inputs and outputs.

The Control Design and Simulation Module includes the following deterministic [ODE solvers](#):

- Runge-Kutta 1
- Runge Kutta 2
- Runge-Kutta 3
- Runge-Kutta 4
- Discrete States Only

Variable step-size ODE solvers are not appropriate for real-time applications because these solvers adjust the step size based on the estimated error of the solution. This adjustment requires additional computational resources, which can interfere with the timing requirements of a real-time application.

## Considerations for ETS Targets (Control Design and Simulation Module)

If you are executing a simulation on an Ardence Phar Lap Embedded Tool Suite (ETS) target, National Instruments recommends you let the LabVIEW Control Design and Simulation Module calculate the necessary value of the simulation period. To automatically calculate the period, place a checkmark in the **Auto Period** checkbox, which is located on the **Timing Parameters** page of the **Configure Simulation Parameters** dialog box.

When you follow this procedure, other tasks can continue to execute when the simulation is not scheduled to execute.

## LabVIEW Projects and Shared Variables (Control Design and Simulation Module)

To execute a simulation on a real-time (RT) target, you must create a [project](#). A project provides a way to manage VIs, RT targets, dependencies, build specifications, and other files related to the project. Use the [Project Explorer](#) window, available by selecting **File»New Project**, to manage the contents of a project.

Another component of a real-time simulation is the [shared variable](#). You [create shared variables](#) to simplify the process of sharing live data between the Windows computer and the RT target.

For more information about the LabVIEW project and the shared variable, including tutorials that introduce you to these concepts, refer to the *Getting Started with the LabVIEW Real-Time Module* document, located in the labview\manuals directory. If you have not installed the LabVIEW Real-Time Module, you can access this document at [ni.com/manuals](http://ni.com/manuals).

## Optimizing Design Parameters (Control Design and Simulation Module)

One important application of simulating dynamic system models is using the simulation to determine parameter values that maximize some measure of performance. The LabVIEW Control Design and Simulation Module includes the [SIM Optimal Design](#) VI, which you can use to obtain parameters that minimize a cost function while satisfying constraints on a dynamic system. You can use this VI with both linear and nonlinear systems, although the Control Design and Simulation Module includes pre-defined options for only linear systems.

Design problems can range from designing physical elements, such as springs, to designing more abstract elements such as controllers or digital filters. Correspondingly, performance specifications might range from simple mechanical limits on outputs to more sophisticated requirements such as frequency domain norms for controlled systems.

For example, when designing a suspension system for a car, you must select a stiffness constant for a spring and a damping constant for a dissipative element. The goal is to find a parameter set that provides maximum comfort. This [optimal parameter set](#) corresponds to a performance measure, such as the average deviation of the passenger from a desired height as the car travels down the road. You use parameter design to determine this optimal parameter set while taking into account the dynamics of the system and the expected operating conditions and disturbances.

You can use several techniques to determine this parameter set. For some problems, you might be able to compute the optimum analytically. However, analytical solutions typically are difficult or impossible to compute. In such cases, you can use numerical optimization instead. A powerful and general purpose numerical optimization algorithm is [Sequential Quadratic Programming](#) (SQP). The SIM Optimal Design VI uses this algorithm. This VI provides domain-specific functions you can use to perform parameter optimization for design purposes. Specifically, you can use this VI to determine optimal parameters from finite-horizon time-domain dynamics simulations.

The following expressions define the nonlinear optimization problem.

$$\min(J(p))$$

$$h_l \leq H(p) \leq h_u$$

$$p_l \leq p \leq p_u$$

where  $p$  is a parameter value,  $J(p)$  is a cost function, and  $H(p)$  is a set of constraints. The objective of the SQP algorithm is to minimize  $J(p)$  and satisfy the constraints defined by  $h_l \leq H(p) \leq h_u$  while keeping  $p$  within specified minimum and maximum values.

Designing a system using the SQP algorithm involves the following steps:

1. **Constructing** the dynamic system model and specifying the component of that model for which you want to find optimal parameter values.
2. **Defining** a performance measure, also known as a cost function, you want to minimize.
3. **Defining** any constraints on the dynamic system that any feasible parameter values must satisfy.
4. **Defining** minimum and maximum values for each parameter.
5. **Defining** a set of initial parameter values and an initial parameters mesh, which generates additional sets of initial parameter values.
6. **Executing** the SQP algorithm, using the information you specified in steps 1 through 5, by running the SIM Optimal Design VI.

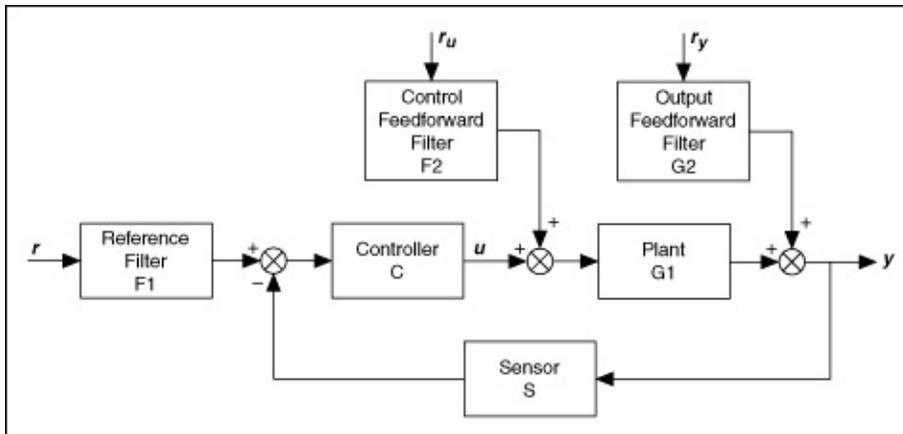
The cost function, inequality constraints, and component to optimize make up the **Problem Specification** parameter of the SIM Optimal Design VI. For each option, you can choose from pre-defined types or specify a customized version.



**Note** The Control Design and Simulation Module includes a **case study** that determines the optimal parameters for a proportional-integral-derivative (PID) controller.

# Constructing the Dynamic System Model (Control Design and Simulation Module)

By default, the [SIM Optimal Design](#) VI computes optimal design parameters for a proportional-integral-derivative (PID) controller placed in a closed-loop dynamic system. The following figure shows this controller and the dynamic system structure.



$F1$ ,  $F2$ ,  $C$ ,  $G1$ ,  $G2$ , and  $S$  consist of transfer functions and associated information, such as delays and sampling time. You can use the [SIM Construct Default System](#) VI to construct these transfer functions and specify reference input signals  $r$ ,  $r_u$ , and  $r_y$ . This VI returns the necessary dynamic system information in the **System Data** output, which you then can wire to the **System Data** input of the SIM Optimal Design VI. The SIM Optimal Design VI then excites the system using the defined inputs and obtains the time response.

Use the **System response type** parameter of the SIM Optimal Design VI to specify if you want this VI to return optimal parameter values for  $C$ ,  $F1$ , or  $F2$ . By default,  $C$  is a parallel PID controller defined by the following equation:

$$U(s) = K_p e + \frac{K_I}{s} + \frac{K_D s}{\alpha s + 1}$$

where  $\alpha = 0.01$ ,  $U$  is the control action,  $s$  is the Laplace variable, and  $K_p$ ,  $K_I$ , and  $K_D$  are the proportional, integral, and derivative gains, respectively.

You also can define a custom type of system response data you want to optimize by using VI templates. To access these templates, select

**File»New** to launch the **New** dialog box. Then select **VI»From Template»Simulation»Optimal Design** from the **Create New** tree. Double-click **System Response (Modify Controller Only)** to modify only the structure of the controller. Double-click **System Response (General)** to define a new dynamic system structure.

If you define a new dynamic system structure, the block diagram code you write must generate the output vector **y** and the time vector **Time**. The code also must generate the control action vector **u** unless the optimization problem does not require a control action. For example, if you use the SIM Optimal Design VI to design the physical parameters of a mechanism, you do not need to specify a control action. In this situation, ensure the cost function and inequality constraints you specify do not take a control action into account.

## Defining a Cost Function (Control Design and Simulation Module)

A cost function is the performance measure you want to minimize. Examples of cost include total power consumption, integrated error, and deviation from a reference value of a signal. The cost function is a functional equation, which maps a set of points in a time series to a single scalar value. This scalar value is the cost.

Use the **Cost type** parameter of the [SIM Optimal Design](#) VI to specify the type of cost function you want this VI to minimize. The LabVIEW Control Design and Simulation Module includes the following types of cost functions:

- **IE**—A cost function that integrates the error.
- **IAE**—A cost function that integrates the absolute value of the error.
- **ISE**—A cost function that integrates the square of the error.
- **ITAE**—A cost function that integrates the time multiplied by the absolute value of the error.
- **ITE**—A cost function that integrates the time multiplied by the error.
- **ITSE**—A cost function that integrates the time multiplied by the square of the error.
- **ISTE**—A cost function that integrates the square of the time multiplied by the square of the error.
- **LQ**—A linear quadratic cost function.
- **Sum of Variances**—A cost function based on the variance of the error multiplied by the variance of the control action.

You also can define a custom cost function using a VI template. To load this template, in the [New](#) dialog box, select **VI»From Template»Simulation»Optimal Design»Compute Cost** from the **Create New** tree.

The block diagram of this template contains several parameters including the control action  $u$ , the dynamic system output  $y$ , an array of input signals, and a time series vector. You also can specify any weights on any part of the cost function.

After you define these parameters, you can write LabVIEW block diagram code to manipulate the parameters according to the cost function. For example, the following equation defines the IE cost function.

$$J_{IE} = \sum_i \int_0^T e_i(t) dt \cong \sum_i \sum_{n=0}^N (\Delta t \cdot e_i(n))$$

where  $e(t)$  is the measured error,  $N$  is the total number of samples in the time response,  $n$  is the current time response sample, and  $i$  is the index of the current output.

You can view the VI that implements this cost function in the `labview\vi.lib\Simulation\Optimization Based Design\Cost` directory.



**Note** If you create a cost function VI that does not take the control action into account, do not delete the **u** parameter from the block diagram. Deleting this parameter breaks the [connector pane structure](#) on which the SIM Optimal Design VI depends. Instead, leave the parameter unwired.

After you save the custom cost function as a VI, you must specify the location of the custom function in the **Problem Specification** parameter of the SIM Optimal Design VI. Select **User defined** for the **Cost type** parameter and specify the path to the VI in the **File path user defined custom cost calculation** path control.

## Defining Inequality Constraints (Control Design and Simulation Module)

Inequality constraints represent trade-offs implicit in the problem specification. For example, you might be able to remove errors in a control loop by applying a very large control action. However, the necessary control action might be impossible to achieve in the real world. If you specify constraints on the control action before executing the [Sequential Quadratic Programming](#) (SQP) algorithm, you can eliminate optimal values that require an unfeasible control action.



**Note** Constraints add a great deal of complexity to the optimization problem. If possible, minimize the number of constraints before executing the SQP algorithm. One strategy to minimize the number of constraints involves first finding optimal values with no constraints, then gradually adding constraints and determining the least amount of constraints required for the dynamic system.

Because the [optimization problem](#) is based on a finite-horizon time-domain simulation, you specify the inequality constraints as envelopes that bound the time response of the control action and the output. You also can place inequality constraint envelopes on the rate of change of the control action and the rate of change of the output.

These envelopes are piecewise linear curves that specify the upper and lower limits on a signal during the simulation. The [SIM Optimal Design VI](#) then calculates  $H(p)$  as the minimum and maximum distance of the time series points from these envelopes.

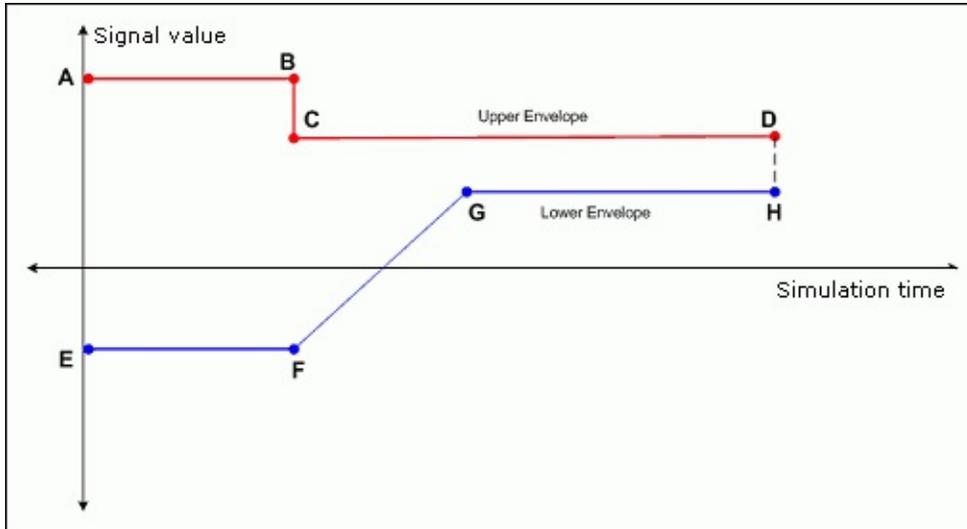
Use the **Inequality Constraints** parameter of the SIM Optimal Design VI to define these envelopes. This parameter specifies the upper and lower constraint envelopes on the following four areas of the dynamic system: the control action, the output, the rate of change of the control action, and the rate of change of the output.



**Note** You can use the Graphically Specify Inequality Constraints VI, located in the `labview\examples\Control and Simulation\Simulation\Optimal Control Design\Graphically Specify Inequality Constraints` directory, to draw the upper and lower envelopes. This VI returns a set of points you then can wire to the

## Inequality Constraints parameter.

For example, consider an output  $y_i(t)$  constrained by envelopes, as shown in the following figure.



A, B, C, and D are points that define the upper envelope  $UE_i(t)$ , and E, F, G, and H are points that define the lower envelope  $LE_i(t)$ . The SIM Optimal Design VI then constrains  $y_i(t)$  to the following relationship:

$$LE_i(t) < y_i(t) < UE_i(t)$$

This VI encodes this constraint by computing the clearance between the output and each envelope. The upper clearance  $UC_i$  is defined as  $\max (UE_i(t) - y_i(t))$ . The lower clearance  $LC_i$  is defined as  $\max (y_i(t) - LE_i(t))$ . These clearances clarify that the constraints must be positive, as the following relationships show:

$$-\varepsilon < UC_i < \infty$$

$$-\varepsilon < LC_i < \infty$$

where  $\varepsilon = 1E^{-21}$ .

You also can place constraints on the rate of change of control actions and outputs. If at least five points are available, this VI computes these rates of change using the following equation:

$$\dot{f}(t) = \frac{f(t-2h) - 8f(t-h) + 8f(t+h) - f(t+2h)}{12h}$$

where  $t$  is the simulation time,  $h$  is the space between time steps, and  $f(t)$

is an output or control action signal.

At boundaries, or if fewer than five points are available, this VI uses the following equations instead:

$$\dot{f}(t) = \frac{f(t+h) - f(t-h)}{2h}$$

or

$$\dot{f}(t) = \frac{f(t+h) - f(t)}{h}$$

You can use a VI template to specify custom calculations for implementing the inequality constraints. To load this template, in the [New](#) dialog box, select **VI»From Template»Simulation»Optimal Design»Compute Inequality Constraints** from the **Create New** tree. To see an example of how to define and manipulate these parameters, open the SIMopt Compute Inequality Constraints (Default) VI, located in the labview\vi.lib\Simulation\Optimization Based Design\Constraints directory. This VI implements the inequality constraints the previous equations specified.

After you save the custom inequality constraint calculations as a VI, you must specify the location of the custom function in the **Problem Specification** parameter of the SIM Optimal Design VI. Select **User defined** for the **Inequality constraints type** parameter and specify the path to the VI in the **File path user defined inequality constraints** path control.

## Defining Parameter Bounds (Control Design and Simulation Module)

Parameter bounds are constraints on parameter values being [optimized](#). For example, while searching for the best value of a spring constant, you might know that springs are available only with certain physical properties. In this case, you can specify that the parameter  $k$  must stay within minimum and maximum values. Parameter bounds are important because these bounds define the parameter space in which the [Sequential Quadratic Programming](#) (SQP) algorithm searches for optimal values.

Use the **Parameter Bounds** parameter of the [SIM Optimal Design](#) VI to specify minimum and maximum values for each parameter.

## Defining Initial Parameter Values and a Mesh (Control Design and Simulation Module)

After you define the parameter space using the [minimum and maximum values](#) of each parameter, you must specify the initial values of each parameter. These initial parameter values determine where the [Sequential Quadratic Programming](#) (SQP) algorithm begins the search for optimal values. However, if you choose only a single initial set of initial values, the SQP algorithm might return [local optimal values](#). Local optimal values are values that minimize the cost function within only a subset of parameter space. Local optimal values are not the true solution to the SQP algorithm because the true optimal values might exist outside the parameter space the algorithm searched.

To mitigate this problem, you can execute the SQP algorithm several times, using a different set of initial parameter values each time. If you use a large enough range of initial parameter values within the given parameter space, you can be relatively confident that the SQP algorithm finds the [global optimal values](#).

You can implement this strategy by defining an initial parameters mesh. The initial parameters mesh defines the distribution pattern of these sets of initial values and the total number of initial value sets to generate. You can choose from four patterns depending on the needs of the problem: **Uniform grid**, **Uniform random**, **Quasirandom**, and **Random walk**.

Each pattern has unique characteristics and strengths. For example, the simplest possible option is the uniform grid, which generates a specified number of equally-spaced locations in the parameter space. However, the uniform random and quasirandom options often provide better coverage of the parameter space while using a fewer number of points than the uniform grid option. The random walk option biases the search to explore close to the initial values but eventually explores a larger region of parameter space. This option is useful if you think a particular parameter space contains the optimal values and you want to focus on a certain region of that space, such as the center.

Use the **Initial Parameters** parameter of the [SIM Optimal Design](#) VI to specify initial parameter values. Use the **Initial Parameters Mesh** parameter of this VI to define an initial parameters mesh.

## Executing the SQP Algorithm (Control Design and Simulation Module)

The [SIM Optimal Design](#) VI uses an internal simulation diagram to obtain the finite-horizon time-domain response of the dynamic system model. Use the **Solver Parameters** parameter of this VI to configure the simulation. You also can configure the [Sequential Quadratic Programming](#) (SQP) algorithm using the **beginning state**, **cn0 settings**, and **stopping criteria** parameters.

This VI returns the following information:

- **Signals**—The finite-horizon time-response data for the output and control action of the dynamic system, evaluated at each point specified in the **Optimal parameters** array.
- **Design parameters**—The set of parameter values that minimize the specified [cost function](#). These values are the optimal parameter values.
- **Design cost**—The result of the specified cost function if you apply the values from the **Design parameters** array.
- **Optimal parameters**—A list of possible optimal parameter values. Each column of this array corresponds to one parameter you specified in the **Parameter Bounds** array. Each row of this array corresponds to one execution of the SQP algorithm.
- **Optimal costs**—The results of the specified cost function that correspond to each row of the **Optimal parameters** array.

The SQP algorithm takes as long to execute as the product of the number of function evaluations and the run time of the simulation. If you specify only one set of initial parameter values, the algorithm must solve, on average, between 30 and 200 functions. The front panel of the SIM Optimal Design VI includes a **Current Data** page that you can use to monitor the progress of the algorithm as the VI runs. This page updates each time the SQP algorithm executes from one set of initial parameter values.

The **Optimal Design Parameters** page of this VI also includes the **Best Parameters (Infeasible Constraints)** and **Best cost (Infeasible constraints)** parameters. These parameters return optimal parameter values and the associated cost function result with no constraints. This

information can be useful when revising the constraint envelopes.

If the dynamic system has constraints and the SQP algorithm does not return feasible optimal values, try ensuring that the specified cost function remains constant when parameter values are outside the feasible range.

This method helps you set reasonable **parameter bounds**. Additionally, reducing system discontinuities helps the SQP algorithm execute precisely. You can use several methods to reduce discontinuities, for example, avoiding saturation effects and rate limiters in the system model.

## Using the Simulation Model Converter (Control Design and Simulation Module)

You can use the Simulation Model Converter to convert a .mdl file, developed in The MathWorks, Inc. Simulink® simulation environment, into a LabVIEW VI that consists of a simulation diagram containing LabVIEW functions, wires, and simulation subsystems corresponding to the contents of the .mdl file. As part of the conversion process, the Simulation Model Converter uses the MathWorks, Inc. MATLAB® application software and the Simulink application software to compile the .mdl file and execute any .m files that you specify in the dialog box. If the MATLAB software or the Simulink software is not installed on the same computer as the LabVIEW Control Design and Simulation Module, the results of the conversion might be less accurate.



**Note** The Simulation Model Converter cannot convert diagrams developed with The MathWorks, Inc. Stateflow® application software or other Simulink blocksets.



**(Windows)** To view related topics, click the **Locate** button, shown at left, in the toolbar at the top of this window. The *LabVIEW Help* highlights this topic in the **Contents** tab so you can navigate the related topics.

## Common Warnings (Control Design and Simulation Module)

If the Simulation Model Converter cannot find a value for a parameter in the .mdl file it is converting, LabVIEW displays a warning. In these cases, the Simulation Model Converter uses the default value of the parameter in the corresponding LabVIEW function.



**Note** In some cases, the Simulation Model Converter cannot find a value for a parameter because the parameter contains an expression instead of a constant value. If the MathWorks, Inc. MATLAB® software is installed on the computer, the Simulation Model Converter attempts to evaluate the MATLAB software expressions in the .mdl file prior to converting the file. If the Simulation Model Converter successfully evaluates the expression, the Simulation Model Converter uses the result of that evaluation as the parameter value and does not produce a warning.

The Simulation Model Converter cannot fully convert all functions of every model to LabVIEW block diagram code. If the Simulation Model Converter encounters a **block it cannot convert**, you receive a warning. In these cases, the Simulation Model Converter creates a placeholder simulation subsystem. You must **create a simulation subsystem** using a LabVIEW VI to accomplish the same functionality as the block to replace this placeholder simulation subsystem.

Because LabVIEW is strict about data types, the converted simulation subsystem might have broken wires. In this case, add block diagram code to convert between converted data types.

# Unsupported Blocks (Control Design and Simulation Module)

The [Simulation Model Converter](#) dialog box cannot convert the following blocks used in The MathWorks, Inc Simulink® application software. In these cases, the Simulation Model Converter creates a placeholder simulation subsystem. You must [create a simulation subsystem](#) using a LabVIEW VI to accomplish the same functionality as the block to replace this placeholder simulation subsystem.

- Algebraic Constraint
- Atomic Subsystem
- Band-Limited White Noise
- Configurable Subsystem
- Enabled
- Enabled and Triggered
- For Subsystem
- From Workspace
- Function-Call
- Function-Call Generator
- If
- If Action Subsystem
- Interpolation (n-D) using PreLook-Up
- Look-Up Table (n-D)
- Memory
- Merge
- Model Info
- PreLook-Up Index Search
- Probe
- Random Number
- Rate Transition
- Repeating Sequence
- S-Function
- S-Function Builder
- Switch Case Action Subsystem

- SwitchCase
- To Workspace
- Triggered
- While Iterator Subsystem

# Additional Important Information (Control Design and Simulation Module)

[Trademarks](#)

[Patents](#)

## Trademarks

National Instruments, NI, ni.com, and LabVIEW are trademarks of National Instruments Corporation. Refer to the *Terms of Use* section on [ni.com/legal](http://ni.com/legal) for more information about [National Instruments trademarks](#).

MATLAB®, Stateflow®, and Simulink® are registered trademarks of The MathWorks, Inc.

Other product and company names mentioned herein are trademarks or trade names of their respective companies.

Members of the National Instruments Alliance Partner Program are business entities independent from National Instruments and have no agency, partnership, or joint-venture relationship with National Instruments.

## Patents

For patents covering National Instruments products, refer to the appropriate location: **Help»Patents** in your software, the patents.txt file on your media, or [ni.com/patents](http://ni.com/patents).

You are only permitted to use this product in accordance with the accompanying license agreement. All rights not expressly granted to you in the license agreement accompanying the product are reserved to NI. Further, and without limiting the foregoing, no license or any right of any kind (whether by express license, implied license, the doctrine of exhaustion or otherwise) is granted under any NI patents.