LabVIEW™

Simulation Module User Manual

April 2004 Edition
Part Number 371013A-01

Worldwide Technical Support and Product Information

`ni.com`

National Instruments Corporate Headquarters

11500 North Mopac Expressway   Austin, Texas 78759-3504   USA   Tel: 512 683 0100

Worldwide Offices

Australia 1800 300 800, Austria 43 0 662 45 79 90 0, Belgium 32 0 2 757 00 20, Brazil 55 11 3262 3599,
Canada (Calgary) 403 274 9391, Canada (Ottawa) 613 233 5949, Canada (Québec) 450 510 3055,
Canada (Toronto) 905 785 0085, Canada (Vancouver) 514 685 7530, China 86 21 6555 7838,
Czech Republic 420 224 235 774, Denmark 45 45 76 26 00, Finland 385 0 9 725 725 11,
France 33 0 1 48 14 24 24, Germany 49 0 89 741 31 30, Greece 30 2 10 42 96 427, India 91 80 51190000,
Israel 972 0 3 6393737, Italy 39 02 413091, Japan 81 3 5472 2970, Korea 82 02 3451 3400,
Malaysia 603 9131 0918, Mexico 001 800 010 0793, Netherlands 31 0 348 433 466,
New Zealand 0800 553 322, Norway 47 0 66 90 76 60, Poland 48 22 3390150, Portugal 351 210 311 210,
Russia 7 095 783 68 51, Singapore 65 6226 5886, Slovenia 386 3 425 4200, South Africa 27 0 11 805 8197,
Spain 34 91 640 0085, Sweden 46 0 8 587 895 00, Switzerland 41 56 200 51 51, Taiwan 886 2 2528 7227,
Thailand 662 992 7519, United Kingdom 44 0 1635 523545

For further support information, refer to the *Technical Support and Professional Services* appendix. To comment on the documentation, send email to `techpubs@ni.com`.

Important Information

Contents

About This Manual

Conventions vii
Related Documentation viii

Chapter 1 Introduction to Modeling

Modeling Dynamic Systems 1-1
Determining Model Complexity 1-2
Developing the Model 1-2
Physical Modeling 1-2
Lumped versus Distributed Parameter Models 1-2
Linear versus Nonlinear Models 1-2
Time-Variant versus Time-Invariant Models 1-3
Continuous versus Discrete Models 1-3
Empirical Modeling 1-3
Introduction to Ordinary Differential Equation Solvers 1-3
Introduction to Rapid Control Prototyping and Hardware-in-the-Loop Configurations 1-4

Chapter 2 Simulation Environment

Simulation Diagram 2-1
Simulation Loop 2-1
LabVIEW VIs, Functions, and Structures 2-2
Feedback Cycles 2-3
Setting Simulation Parameters 2-5
Simulation Time Parameters 2-7
Continuous Solver Parameters 2-7
Discrete Solver Parameters 2-8
Setting Timing Parameters 2-9
LabVIEW Real-Time Module for ETS Targets 2-9
LabVIEW Real-Time Module for RTX Targets 2-9
Off-Line Simulations 2-10
Simulation Functions 2-10
Discrete Systems Functions 2-12
Displaying Dynamic Content on Expandable Nodes 2-12
Flipping Function Direction 2-13
Programmatically Stopping a Simulation 2-14

Simulation Subsystems 2-14
Creating a Simulation Subsystem 2-14
Stand-Alone Subsystems 2-15
Subsystems within a Simulation Diagram 2-15
Linearizing a Subsystem 2-16
Simulation Debugging 2-17

Chapter 3 Real-Time Applications

Determinism 3-1
Case Study: Rapid Control Prototype and Hardware-in-the-Loop Simulation 3-2
Off-Line Simulation 3-2
Rapid Control Prototype Implementation 3-3
Hardware-in-the-Loop Implementation 3-4
Discrete Behavior 3-4

Chapter 4 Ordinary Differential Equation Solvers

Simulation Discontinuities 4-2
ODE Solver Accuracy and Order 4-2
Variable Step-Size versus Fixed Step-Size ODE Solvers 4-3
Single-Step versus Multi-Step ODE Solvers 4-3
Stiff ODE Solvers 4-4
LabVIEW Simulation ODE Solvers 4-4

Chapter 5 Simulink Translator

Converting Simulink Models into LabVIEW Code 5-1
Common Warnings 5-2

Appendix A Technical Support and Professional Services

Glossary

About This Manual

The *LabVIEW Simulation Module User Manual* describes how to use the simulation environment, Simulation functions, and simulation subsystems. Refer to the *LabVIEW Help* for more information about specific Simulation Module palettes, functions, and dialog box options.

Use this manual to learn how to use the LabVIEW Simulation Module in real-time applications and how to use the Simulink Translator to convert Simulink® model (`.mdl`) files into LabVIEW VIs. This manual also describes factors to consider when you develop a model and factors to consider when you select an ordinary differential equation (ODE) solver to use for a simulation.

Conventions

The following conventions appear in this manual:

»	The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box.
📝	This icon denotes a note, which alerts you to important information.
bold	Bold text denotes items that you must select or click in the software, such as menu items and dialog box options. Bold text also denotes parameter names; dialog box names; and pages, sections, and components of dialog boxes.
italic	Italic text denotes variables, emphasis, or a cross reference. This font also denotes text that is a placeholder for a word or value that you must supply.
`monospace`	Text in this font denotes text or characters that you should enter from the keyboard. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames, and extensions.

Related Documentation

The following documents contain information that you might find helpful as you read this manual:

- *LabVIEW Control Design Toolkit User Manual*
- *LabVIEW Execution Trace Toolkit User Guide*
- *LabVIEW Help*
- *LabVIEW Real-Time Module User Manual*
- *LabVIEW User Manual*
- *NI-CAN Hardware and Software Manual*
- *NI-DAQmx Help*

1

Introduction to Modeling

With the LabVIEW Simulation Module, you can investigate the time-dependent behavior of complex engineering systems. Use the Simulation Module to model and simulate any systems that differential and/or difference equations can characterize. Such systems include, but are not limited to, mechanical, electrical, fluid, and thermodynamic engineering systems. The Simulation Module provides analysis functions for both linear and nonlinear systems.

With the Simulation Module, you can create simulations to lower product development costs by accelerating product development, providing higher quality, and reducing associated risk. You also can create simulations to provide insight into the behavior of systems that you cannot replicate conveniently in the laboratory.

Note This document is not intended to provide a comprehensive discussion of modeling. Refer to the following books for more information about modeling: *Modern Control Systems*[1], *Feedback Control of Dynamic Systems*[2], *Digital Control of Dynamic Systems*[3], *Control Systems Engineering*[4], and *Modern Control Engineering*[5].

Modeling Dynamic Systems

A model is a set of equations that characterizes the behavior of a system. A dynamic system is a system whose behavior varies with time.

Consider the following issues when you develop a dynamic system model.

[1] Dorf, Richard C., and Robert H. Bishop. *Modern Control Systems*, 9th ed. Upper Saddle River, NJ: Prentice-Hall, Inc., 2001.

[2] Franklin, Gene F., J. David Powell, and Abbas Emami-Naeini. *Feedback Control of Dynamic Systems*, 4th ed. Upper Saddle River, NJ: Prentice Hall, 2002.

[3] Franklin, Gene F., J. David Powell, and Michael L. Workman. *Digital Control of Dynamic Systems*, 3rd ed. Menlo Park, CA: Addison Wesley Longman, Inc., 1998.

[4] Nise, Norman S. *Control Systems Engineering*, 3rd ed. New York: John Wiley & Sons, Inc., 2000.

[5] Ogata, Katsuhiko. *Modern Control Engineering*, 4th ed. Upper Saddle River, NJ: Prentice-Hall, Inc. 2001.

Determining Model Complexity

When you develop a system model, you must decide what to include in the model and how complex the model will be. For example, if you are developing a model of a vehicle shock system, you might take into account engine vibration or passenger movement in the vehicle. However, because these effects are minimal, you do not need to include them in the model. By limiting the complexity of the model, you can reduce model development and validation effort.

Developing the Model

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

Physical Modeling

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 partial differential equations to describe a system, the resulting model is a distributed parameter model.

Linear versus Nonlinear Models

A linear model obeys the principle of superposition. The following equations are true for linear models:

$$y_1 = f(x_1)$$

$$y_2 = f(x_2)$$

$$Y = f(x_1 + x_2) = y_1 + y_2$$

A differential equation is linear if the coefficients are constant or change only with the independent variable. Often this independent variable is time.

A nonlinear model does not obey the principle of superposition. Nonlinear effects that are observed in real-world systems include, but are not limited to, saturation, dead-zone, friction, backlash, and quantization effects;

relays; switches; and rate limiters. All real-world systems are nonlinear, though you can linearize the model to simplify a design or analysis task.

Time-Variant versus Time-Invariant Models

Dynamic models are time-variant or time-invariant. If the parameters of a model do not change with time, then the model is a time-invariant model. If the parameters change with time, then the model is a time-variant model. For example, you can use a time-variant model to describe an automobile. As fuel burns, the mass of the vehicle changes with time.

Continuous versus Discrete Models

Dynamic models can be either continuous or discrete. Continuous models represent real-world signals that vary continuously with time. For example, a model that describes the orbital motion of a satellite is a continuous model. You can use differential equations to describe continuous systems. Discrete models represent signals that are sampled in time at discrete intervals. For example, a model that controls the altitude of the satellite is a discrete model. You can use difference equations to describe discrete systems. In either case, the equations that describe the system can be linear or nonlinear and time-variant or time-invariant.

Empirical Modeling

Empirical models use data gathered from experiments to define the mathematical model of a system. To some degree, even physical models are empirical because you determine experimentally 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. You can use LabVIEW System Identification Toolkit VIs to create models using system identification methods. If you have the System Identification Toolkit installed, refer to the *LabVIEW Help* for more information about these VIs.

Introduction to Ordinary Differential Equation Solvers

You can use a system of differential equations to mathematically describe dynamic systems. You must solve these equations to observe the behavior of the simulated system. The Simulation Module includes ODE solvers, which can solve these equations.

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 its accuracy (or order), stability, use of a fixed time step versus a variable time step, and use of a single step versus multiple steps.

Difficulties can arise when you solve a differential equation in a system whose dynamics are described by widely differing time constants. Such a system is known as a stiff system. You must give stiff systems special consideration when you choose an integration method. Refer to Chapter 4, *Ordinary Differential Equation Solvers*, for more information about ODE solvers.

Introduction to Rapid Control Prototyping and Hardware-in-the-Loop Configurations

Design and test engineers use rapid control prototyping (RCP) and hardware-in-the-loop (HIL) to evaluate and validate system components while they develop complex control systems. In an RCP configuration, you test a control algorithm with the actual physical system the algorithm controls. The control algorithm runs as part of the software model on real-time prototyping hardware. You can verify the control algorithm with the actual physical plant before deploying the algorithm to the actual controller hardware. In an HIL configuration, you test a controller implementation, with the actual controller hardware, using software models to simulate the plant. In both the RCP and HIL configurations, the physical components tested respond to the simulated signals as though they were operating in the real system.

Replacing parts of the system with computers running software simulations reduces the size and complexity of the application, increases the flexibility and rate of running many different tests and test scenarios, and reduces system development costs. For example, with RCP, you can evaluate control logic without incurring the costs associated with generating target code and using controller hardware. Refer to Chapter 3, *Real-Time Applications*, for more information about RCP and HIL.

2 Simulation Environment

You can use the simulation environment within LabVIEW to design system models for simulation in typical block diagram form. As you build a simulation model, notice that the simulation diagram looks different than other LabVIEW diagrams. The simulation diagram has a distinct yellow background, and Simulation function icons have purple backgrounds, which distinguish them from other LabVIEW features.

Simulation Diagram

The simulation diagram graphically displays the simulation model that an application must evaluate at each time step of the simulation. You place a dynamic element and other logical or arithmetic code on the simulation diagram to define the model the ODE solver evaluates at each time step.

Note Dynamic elements include the following Simulation functions: Integrator, State Space, Transfer Function, Zero-Pole-Gain, Discrete Integrator, Discrete State Space, Discrete Transfer Function, and Discrete Zero-Pole-Gain.

The simulation diagram uses an ODE solver to compute the behavior of a model over time. Given the behavior of the model at time *t* – *dt*, the ODE solver determines the behavior and outputs of the model at time *t*, where *dt* is the time step of the ODE solver. Depending on the type of ODE solver the simulation diagram uses, the ODE solver might evaluate a simulation diagram several times during a single time step.

Simulation Loop

Because simulations naturally run as loops that iterate over multiple time steps, the Simulation Module includes the Simulation Loop. The Simulation Loop, shown in Figure 2-1, executes the simulation diagram until it reaches the simulation final time or until the Halt Simulation function programmatically stops the execution.

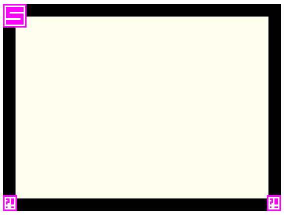


Figure 2-1. Simulation Loop

You must place all Simulation functions, except the Set Diagram Parameters function, and all simulation subsystems within a Simulation Loop or within another simulation subsystem. Refer to the *Simulation Functions* section for information about these functions.

You can wire the Set Diagram Parameters function to the optional parameters terminal of the Simulation Loop, shown at left, to programmatically configure simulation parameters for the enclosed simulation diagram. If the parameters terminal is unwired, you can right-click the border of the Simulation Loop and select **Configure Simulation Parameters** from the shortcut menu to configure the simulation parameters using the **Simulation Parameters** dialog box. Refer to the *Setting Simulation Parameters* section for more information about setting these parameters.

The Simulation Loop has **error in** and **error out** terminals, which send error information through the simulation diagram. If the **error in** terminal detects an error, the simulation diagram returns the error information in the **error out** terminal and does not evaluate the simulation. If an error occurs while the Simulation Loop is executing, the simulation stops running, and the error information is output in the **error out** terminal. Refer to the *Simulation Debugging* section for more information about how to resolve simulation errors.

LabVIEW VIs, Functions, and Structures

You can use a majority of LabVIEW VIs and functions to describe a model. However, you cannot place certain structures, such as the Case structure, While Loop, For Loop, Event structure, or the Sequence structures, directly on the simulation diagram. Instead, you can place these structures in a subVI and then place the subVI on a simulation diagram.

Note You can load models you create using the LabVIEW Control Design VIs into the simulation diagram. Refer to the *LabVIEW Control Design Toolkit User Manual* for more information about control design models.

When you place a VI on a simulation diagram, you can specify whether to run that VI as a discrete node. By running the VI as a discrete node, you ensure that VI executes only once per fixed time period. This fixed time period equals the discrete time step of the simulation multiplied by the **Sample Rate Divisor** for the node. When you run a VI as a discrete node, a red 'D' appears on the VI icon, shown at left. Right-click the VI on the simulation diagram and select **SubVI Node Setup** from the shortcut menu to display the **SubVI Node Setup** dialog box. You can use this dialog box to specify whether to run the VI as a discrete node. If you place a checkmark in the **Run as Discrete Node** checkbox, you must specify the **Sample Rate Divisor** for the node. Configure this divisor as you would in a discrete system. Refer to the *Simulation Subsystems* section for more information about the **Sample Rate Divisor**.

Note The **Run as Discrete Node** option is enabled by default with a **Sample Rate Divisor** of `1`.

Feedback Cycles

The relationship between the inputs and outputs of a function defines the feedthrough behavior of that function. An input has direct feedthrough to an output if the function uses the input at the current step to compute the output at the current step. An input has indirect feedthrough to an output if the function does not use the input at the current step to compute the output at the current step. The indirect feedthrough function uses the input from the previous step or steps to compute the output at the current step.

A function on a LabVIEW simulation diagram might require only a subset of its inputs at the current step to compute any given output at the current step. A function on the simulation diagram returns a given output as soon as it has received the inputs that have direct feedthrough to the output, regardless of whether the function has received the inputs that have indirect feedthrough to the output. Therefore, on a simulation diagram, you can create a cycle in which data flow originates from an output of a function or subsystem that has indirect feedthrough behavior and terminates as an input of the same function or subsystem. This flow of data is known as a feedback cycle. In a feedback cycle, the output of the indirect feedthrough function or subsystem at time t is a function of the input to the same function or subsystem at time $t - dt$, $t - dt_2$, and so on.

You can use one or more Simulation functions as well as 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 its 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.

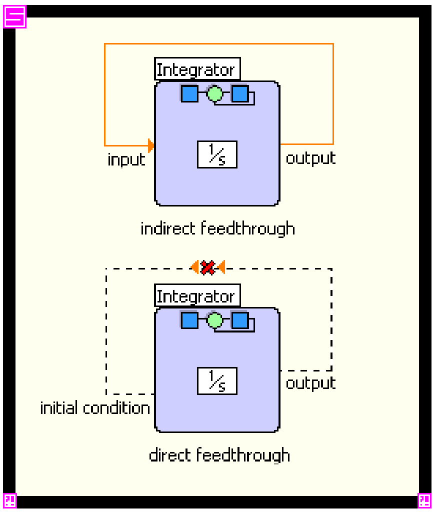


Figure 2-2. Feedback Cycles

Various Simulation functions, such as the Integrator function, have indirect feedthrough behavior and, therefore, allow feedback cycles. LabVIEW automatically determines the type of feedthrough behavior that exists between inputs and outputs. If you attempt to wire an output to an input that has direct feedthrough to that output, the wire is broken. In the top Integrator function shown in Figure 2-2, **output** is wired to **input**. Because **input**, in this case, does not have direct feedthrough to **output**, LabVIEW allows the feedback cycle. In the bottom Integrator function, **output** is wired to the **initial condition** input. Because **initial condition** has direct feedthrough to **output**, LabVIEW does not allow this feedback cycle and the wire appears broken.

Note The wires on the simulation diagram use arrows to indicate the direction of data flow. These arrows help you identify feedback cycles on the simulation diagram by showing data flow direction.

Feedthrough behavior differs from function to function. The following Simulation functions have indirect feedthrough behavior:

- Integrator
- Transport Delay
- Discrete Unit Delay

For other Simulation functions, the parameter values you specify determine the feedthrough behavior. The following Simulation functions have parameter-dependent feedthrough behavior:

- State Space
- Transfer Function
- Zero-Pole-Gain
- Discrete Filter
- Discrete Integrator
- Discrete State Space
- Discrete Transfer Function
- Discrete Zero-Pole-Gain

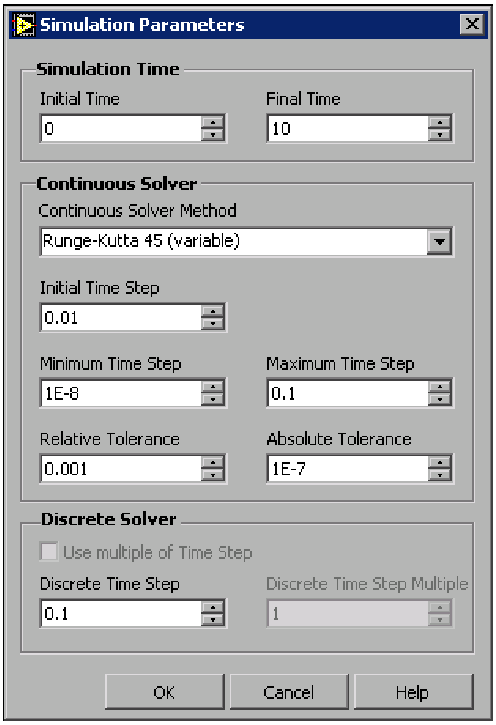
Refer to the *LabVIEW Help* for information about which parameters determine the feedthrough behavior of these functions. All other Simulation functions have direct feedthrough behavior. Refer to the *LabVIEW Help* for information about the direct or indirect feedthrough behavior of individual Simulation functions.

Setting Simulation Parameters

The simulation parameters determine the way the simulation diagram executes. Any subsystem you place on a simulation diagram inherits the simulation parameters from the parent simulation diagram. Refer to the *Simulation Subsystems* section for more information about how simulation parameters operate in subsystems.

To set the simulation parameters for the simulation diagram at edit time, you can use the **Simulation Parameters** dialog box. To set the simulation parameters programmatically, use the Set Diagram Parameters function.

The **Simulation Parameters** dialog box, shown in Figure 2-3, contains options for setting the duration of the simulation and the parameters of the ODE solver. To access the **Simulation Parameters** dialog box, right-click the border of the Simulation Loop and select **Configure Simulation Parameters** from the shortcut menu. When you run a stand-alone simulation subsystem, you can access the **Simulation Parameters** dialog box by selecting **Operate»Configure Simulation Parameters** from the pull-down menu.



Simulation Time
Initial Time: 0
Final Time: 10

Continuous Solver
Continuous Solver Method: Runge-Kutta 45 (variable)
Initial Time Step: 0.01
Minimum Time Step: 1E-8
Maximum Time Step: 0.1
Relative Tolerance: 0.001
Absolute Tolerance: 1E-7

Discrete Solver
Use multiple of Time Step
Discrete Time Step: 0.1
Discrete Time Step Multiple: 1

Figure 2-3. Simulation Parameters Dialog Box

Use the Set Diagram Parameters function to set the values of the simulation parameters programmatically.

Note Do not select **Create»Control** or **Create»Constant** directly from the parameters terminal to set the simulation parameters. If you create a control or constant directly from the parameters terminal, the control or constant does not explicitly list all of the simulation parameters you can set.

Place the Set Diagram Parameters function outside the Simulation Loop to the left of the parameters terminal and wire the **simulation parameters out** output of this function to the parameters terminal. Then, right-click the Set Diagram Parameters function and select **Create»Control** or **Create» Constant** from the shortcut menu to create a set of **simulation parameters** controls on the front panel. Figure 2-4 shows how to wire the Set Diagram

Parameters function to create a control you can use to configure the simulation parameters.

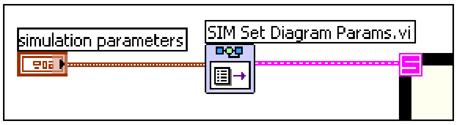


Figure 2-4. Wiring the Set Diagram Parameters Function

Note If you wire the Set Diagram Parameters function to the parameters terminal, the configuration data the function specifies—including the default values—override the static configuration you set using the **Simulation Parameters** dialog box.

Simulation Time Parameters

When LabVIEW executes a simulation, an ODE solver evaluates the simulation diagram over multiple time steps beginning at the time you specify in the **Initial Time** parameter and ending at the time you specify in the **Final Time** parameter.

Continuous Solver Parameters

Use the **Continuous Solver Method** parameter to specify the type of ODE solver LabVIEW uses to evaluate the simulation diagram. The type of ODE solver you select determines what other simulation parameters you can set. If you specify a fixed step-size ODE solver method, you can set the **Time Step**, in units of seconds. The **Time Step** is the interval between the times at which the ODE solver evaluates the model and updates the model output.

If you select a variable step-size ODE solver method, you can specify the **Initial Time Step**, which is the time step size for the first time step of the simulation diagram evaluation. A variable step-size ODE solver dynamically adjusts subsequent time steps based upon what you specify in the **Relative Tolerance** and **Absolute Tolerance** parameters. You can use the **Minimum Time Step** and **Maximum Time Step** parameters to specify the smallest and largest time step sizes the variable step-size ODE solver can use to evaluate the simulation diagram.

Note Methods whose names are followed by the word *(variable)* in the **Continuous Solver Method** list are variable step-size ODE solver methods. All other methods in the list are fixed step-size ODE solver methods.

The time step size affects how accurate a simulation is and how fast the simulation runs. A simulation with a large time step runs faster than a simulation with a smaller time step. However, a simulation with a large time step is less accurate than a simulation with a smaller time step. Refer to Chapter 4, *Ordinary Differential Equation Solvers*, for more information about ODE solvers and time steps.

Discrete Solver Parameters

The **Discrete Time Step** is the base time interval, in units of seconds, for which the ODE solver evaluates the discrete functions and updates the function outputs. However, the ODE solver might not evaluate a discrete function each **Discrete Time Step**. Rather, the ODE solver evaluates the discrete function every *n* discrete time steps, where *n* is the **sample rate divisor** you specify for that function. Refer to the *Discrete Systems Functions* section for more information about how LabVIEW determines when to update the output of a discrete function.

The type of ODE solver you select determines how you can specify the **Discrete Time Step**. If you select a fixed step-size ODE solver method, you can use the **Use multiple of Time Step** option, which is enabled by default, to set the **Discrete Time Step**. When this option is enabled, LabVIEW multiplies the **Time Step** by the **Discrete Time Step Multiple** to calculate the **Discrete Time Step**. The **Use multiple of Time Step** option is useful because, for fixed step-size ODE solvers, the **Discrete Time Step** must be an integer multiple of the **Time Step**.

If you remove the checkmark from the **Use multiple of Time Step** checkbox and specify the **Discrete Time Step** directly, LabVIEW verifies that the **Discrete Time Step** is an integer multiple of the **Time Step** for fixed step-size ODE solvers. If you attempt to specify a **Discrete Time Step** that is not an integer multiple of the **Time Step**, LabVIEW launches a **Configuration Error** dialog box when you click **OK** to close the **Simulation Parameters** dialog box. You can select **Modify Configuration** or **Continue Anyway** from the **Configuration Error** dialog box.

If you select a variable step-size ODE solver, the **Discrete Time Step** is not required to be an integer multiple of the **Time Step**. Therefore, the **Use multiple of Time Step** option is not available. You must specify the **Discrete Time Step** directly. The **Use multiple of Time Step** option is also unavailable if you configure the simulation parameters programmatically, regardless of the ODE solver method you select.

Setting Timing Parameters

Right-click the Simulation Loop and select **Configure Simulation Timing Parameters** from the shortcut menu to display the **Loop Configuration** dialog box. Use this dialog box to configure the simulation timing parameters.

Note The simulation timing parameters are most accurate in a LabVIEW Real-Time application. If you do not run an application on a real-time system, other processes on the operating system can interrupt the execution of the Simulation Loop.

LabVIEW Real-Time Module for ETS Targets

If you are executing a simulation on a real-time operating system, National Instruments recommends you set the **Period**, in milliseconds/microseconds (the unit depends on the timing source), to the same value as the **Time Step** of the simulation model. The Simulation Loop uses the **Period** to achieve wall-clock time.

LabVIEW Real-Time Module for RTX Targets

You must set the **Period** to `0` to avoid receiving a run-time error when you select the RTX target as the execution target for a simulation diagram or subsystem. If you select the RTX target as the execution target for a top-level simulation diagram or subsystem with a non-zero **Period**, LabVIEW returns a run-time error if you attempt to run the simulation diagram or subsystem. If the simulation diagram or subsystem is embedded in a subVI, LabVIEW returns a run-time error if you attempt to run the subVI.

Complete the following steps to avoid these errors when you run a simulation diagram or subsystem that is downloaded to the RTX target:

1. Open the **Loop Configuration** dialog box and set the **Period** to `0`.
2. Place a subVI that contains a Wait Until Next ms Multiple function on the simulation diagram or subsystem and set the **millisecond multiple** accordingly.

This procedure allows other tasks to continue to execute when the simulation is not scheduled to execute. Refer to the *LabVIEW Real-Time Module User Manual* for more information about using the LabVIEW Real-Time Module for RTX Targets.

Off-Line Simulations

If you are executing an off-line simulation on a Windows operating system, National Instruments recommends you set the **Period** to `0` for optimal performance. Refer to the *Loop Configuration Dialog Box (Simulation Module)* topic in the *LabVIEW Help* for more information about setting the timing parameters for the Simulation Loop.

Simulation Functions

Use the Simulation functions and other LabVIEW functions to build a simulation model. You must place all Simulation functions, except the Set Diagram Parameters function, on the simulation diagram, either inside a Simulation Loop or in a simulation subsystem. Refer to *Simulation Subsystems* for information about placing functions in a simulation subsystem.

Many Simulation functions have a configuration dialog box that you can use to view and set the function parameters. Double-click a Simulation function to display its configuration dialog box. For example, Figure 2-5 displays the configuration dialog box for the Sine Wave function.

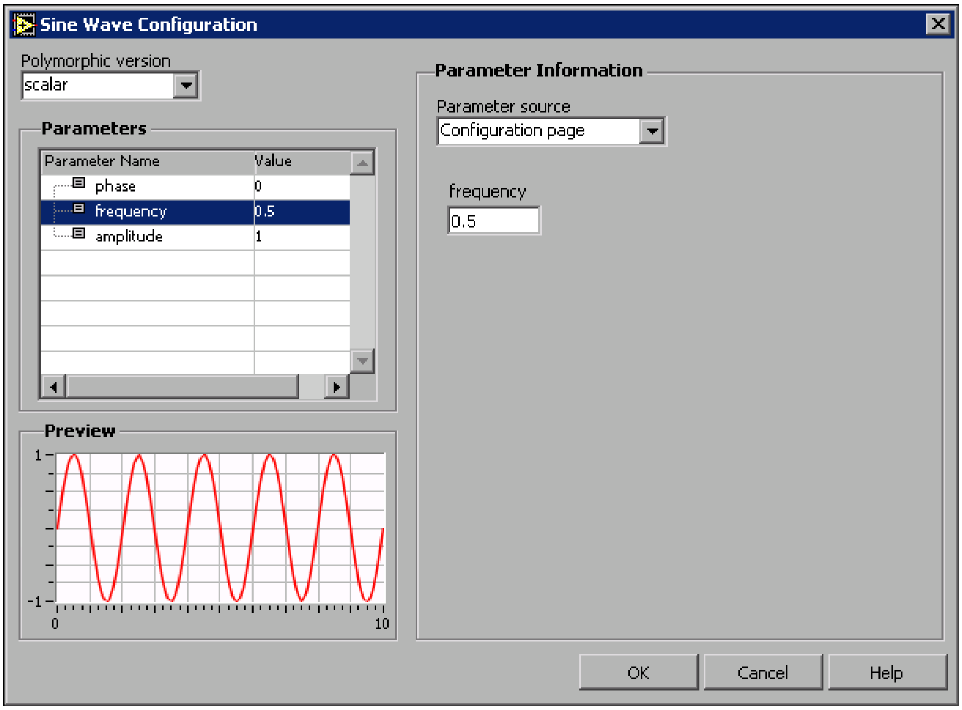


Figure 2-5. Configuration Dialog Box

The left side of the configuration dialog box lists all the parameters that you can configure for the Sine Wave function. When you select a parameter from the **Parameters** table, the **Parameter Information** section displays a control you can use to set the value of that parameter. You can use the **Parameter source** control to specify the source of the parameter value. If you select **Configuration page** as the **Parameter source**, LabVIEW uses the parameter value you specify in the configuration dialog box. If you select **Terminal**, LabVIEW uses the parameter value you wire to the input terminal on the simulation diagram.

Note If you specify **Configuration page** as the **Parameter source** for a parameter, an input terminal does not exist on the function icon for that parameter.

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 of the Simulation function. If you copy and paste a Simulation function, the copy of the function retains the parameter values of the original function.

Discrete Systems Functions

The sample rate of a discrete function must be a multiple of the discrete time step of the system. All discrete functions have a **sample rate divisor** parameter on the configuration dialog box. This parameter specifies the multiple of the discrete time step that LabVIEW uses to determine the sample rate of the discrete function. If the **sample rate divisor** is 1, the discrete function updates its output every discrete time step. If the **sample rate divisor** is *n*, the discrete function updates its output every *n* discrete time steps.

Note Different discrete functions can run at different sample rates in the Simulation Loop.

Displaying Dynamic Content on Expandable Nodes

Simulation function icons, called expandable nodes, can display dynamic content on the simulation diagram. The configuration of the function you specify in the configuration dialog box determines the expandable node display. For example, the Signal Generator function graphically displays a preview of the signal it will generate. If you change the **signal type** parameter of this function, the signal the icon displays changes. Figure 2-6 shows the expandable node for the Signal Generator function. The Signal Generator function on the left shows the icon for the function when you set the **signal type** to **Sine**. The Signal Generator function on the right shows the icon for the same function when you set **signal type** to **Sawtooth**.

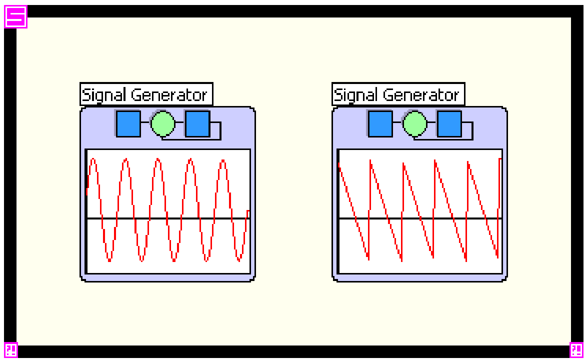


Figure 2-6. Displaying Dynamic Content on an Expandable Node

Note A function displays dynamic content only if it uses the dynamic icon style. Right-click an icon and select **Icon Style»Dynamic** from the shortcut menu to specify this icon style. **Dynamic** is the default icon style for Simulation functions.

If an expandable node uses the dynamic icon style, you can change the size of that expandable node on the simulation diagram. For example, you can resize the Signal Generator node on the simulation diagram to view more of the graph. Figure 2-7 displays the original icon and the resized icon.

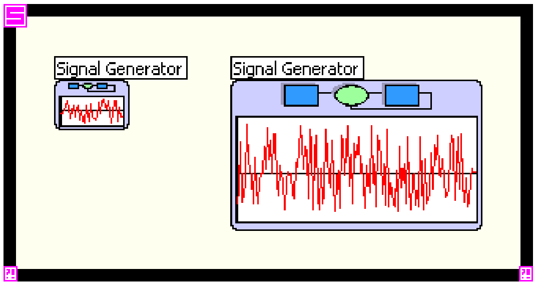


Figure 2-7. Resizing Expandable Nodes

Flipping Function Direction

Because feedback cycles are an inherent part of a simulation model, feedback loops are common. You can flip Simulation functions horizontally to better display the data flow of a looped system, as shown in Figure 2-8.

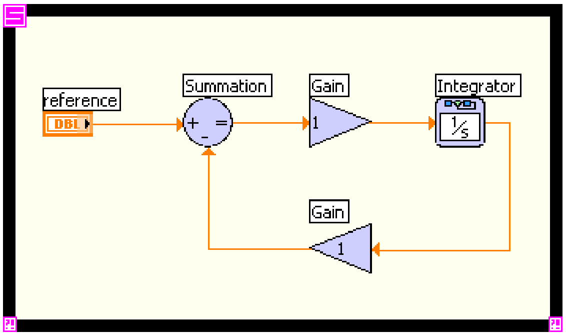


Figure 2-8. Horizontally Flipping the Gain Function

To flip a Simulation function, right-click the icon and select **Reverse Terminals** from the shortcut menu.

Programmatically Stopping a Simulation

To stop the simulation programmatically, use the Halt Simulation function. Place the Halt Simulation function on the simulation diagram and wire a Boolean control to the **Halt?** input. If the **Halt?** Boolean control is TRUE, the function stops the simulation after the current time step. You also can place a Halt Simulation function in a simulation subsystem to stop the execution of the parent simulation diagram. The Halt Simulation function operates like the conditional terminal on a While Loop. However, you can place more than one Halt Simulation function on the simulation diagram. With multiple Halt Simulation functions, you can stop the simulation from various points in a simulation diagram or subsystem.

Simulation Subsystems

Creating simulation models using the Simulation Module can require a large amount of space on the block diagram. To reduce the amount of space required to create a simulation diagram, you can convert a section of that simulation diagram into a simulation subsystem.

A simulation subsystem, like a subVI, is a type of LabVIEW VI. A function in a simulation subsystem can execute when that function receives all the inputs that function uses, even if the subsystem has not received all the inputs the subsystem uses. A simulation subsystem stores a feedthrough mapping from all inputs to all outputs. If an indirect feedthrough function is in the data flow from a subsystem input to a subsystem output, that is, if a dependency does not exist between a subsystem input at the current step and a subsystem output at the current step, LabVIEW allows you to use the subsystem as an indirect feedthrough function in a feedback cycle. Refer to the *Feedback Cycles* section for more information.

Creating a Simulation Subsystem

You can create a simulation subsystem by selecting a section of a simulation diagram and selecting **Edit»Create Simulation Subsystem** from the pull-down menu. LabVIEW replaces the nodes in the selected section in the simulation diagram with a single node that represents the simulation subsystem.

Note If you select **Edit»Create SubVI** after highlighting nodes on a simulation diagram, LabVIEW creates a subVI from only the non-Simulation function nodes in the selection.

You also can create a simulation subsystem using the **New** dialog box, which you can access by selecting **File»New** from the pull-down menu. Select **Other Document Types»Simulation Subsystem** from the **Create new** list.

Stand-Alone Subsystems

When you execute the simulation subsystem as a stand-alone VI, the simulation diagram executes as if it were contained within a Simulation Loop. This behavior enables you to edit the simulation model without having to place a Simulation Loop on the block diagram.

Simulation subsystems store default simulation parameters that LabVIEW uses when executing the subsystem as a stand-alone VI. You can use the **Simulation Parameters** dialog box to configure the simulation parameters for the subsystem. Select **Operate»Configure Simulation Parameters** from the pull-down menu to access this dialog box from a subsystem. The parameter settings for the subsystem are valid only when the subsystem runs as a stand-alone VI. When you place the subsystem on a simulation diagram, the subsystem inherits the parameter values from the parent simulation diagram. Refer to the *Setting Simulation Parameters* section for more information about simulation parameters.

You can configure the simulation timing parameters for a stand-alone subsystem by selecting **Operate»Configure Simulation Timing Parameters** from the pull-down menu. When you place the subsystem on a simulation diagram, the subsystem inherits the timing parameters of the parent simulation diagram. Refer to the *Setting Timing Parameters* section for more information about simulation timing.

Subsystems within a Simulation Diagram

If you do not run the simulation subsystem as a stand-alone VI, you can place the simulation subsystem on a simulation diagram. LabVIEW executes a simulation subsystem like other Simulation functions on the simulation diagram.

Note The settings you specify in the **Execution Properties** page for the simulation subsystem do not affect the execution of the parent simulation diagram when you place the subsystem on a simulation diagram. Select **File»VI Properties** and select **Execution** from the pull-down menu to display the **Execution Properties** page.

When you place a simulation subsystem on a simulation diagram, LabVIEW creates a special simulation subsystem node. To open the front panel of the simulation subsystem, right-click the simulation subsystem node and select **Open Subsystem** from the shortcut menu. The simulation subsystem node, like Simulation function nodes, is resizable and dynamically displays the simulation diagram of the subsystem.

Like a Simulation function, the simulation subsystem has a configuration dialog box, which you can use to specify sources and values for each of the input parameters wired to the simulation subsystem. Double-click the simulation subsystem node to access this dialog box. The control connection type for each parameter on the subsystem connector pane determines the default source of the parameter in the configuration dialog box. The initial value for the parameter is the default value of the control on the subsystem simulation diagram that corresponds to that parameter. The following list describes the default sources of parameters as they correspond to the control connection types.

- If the connection is required, the parameter is available only as a terminal on the node. The parameter is not visible in the configuration dialog box.
- If the connection is recommended, the initial source of the parameter is the terminal. You also have the option to configure the parameter using the configuration dialog box.
- If the connection is optional, the initial source of the parameter is the configuration dialog box. You also have the option to configure the parameter using a terminal on the node.

Note You can configure the connector pane for a simulation subsystem the same way you configure the connector pane for a subVI.

Linearizing a Subsystem

You can use the **Linearize Subsystem** dialog box to generate a linear time-invariant state-space model from the subsystem model. You can use this linear state-space model for control design. Select **Tools»Simulation Tools»Linearize Subsystem** from the pull-down menu to display this dialog box. Refer to the *LabVIEW Help* for more information about the **Linearize Subsystem** dialog box.

Simulation Debugging

LabVIEW performs syntax checking on the simulation diagram. While you edit the simulation diagram, LabVIEW notifies you if there is a data type problem or an invalid feedback cycle.

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

You cannot use execution highlighting, breakpoints, probes, or single-stepping in a simulation subsystem if that subsystem is part of a simulation diagram. You also cannot step into a subsystem. You can set a breakpoint on the entire subsystem by right-clicking the subsystem and selecting **Set Breakpoint** from the shortcut menu. You also can use a probe or a custom probe to monitor the subsystem output.

Note If you run the subsystem in stand-alone mode, you can debug it the same way you can debug a simulation diagram.

Refer to the *LabVIEW User Manual* for information about LabVIEW debugging techniques.

3

Real-Time Applications

The Simulation Module, in conjunction with the LabVIEW Real-Time Module and National Instruments hardware, allows you to implement simulations and controllers in real time with real-world inputs and outputs. This chapter provides an overview of a real-time application and describes a case study that involves the rapid control prototype (RCP) and hardware-in-the-loop (HIL) implementations.

Refer to the *LabVIEW Real-Time Module User Manual* for more information about creating real-time applications using the LabVIEW Real-Time (RT) Module. You can use the RT Module to get data into and out of the real-time VI. You also can use tools such as the RT Communication Wizard and the LabVIEW Execution Trace Tool to develop and fine-tune an application. Refer to the *LabVIEW Execution Trace Toolkit User Guide* for more information about the LabVIEW Execution Trace Tool.

Refer to the documentation for the type of I/O you are using for more information about I/O programming in the Real-Time Module. For example, refer to the *NI-DAQmx Help* for information about analog, digital, and timing I/O. Refer to the *NI-CAN Hardware and Software Manual* for information about CAN I/O.

Determinism

Running a simulation or controller in real time means that the simulation time must equal the wall-clock time at each point at which 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. Thus, at each sampling time, the simulation time must equal the wall-clock time.

To meet the real-time deadline, the software implementing the simulation or controller must execute deterministically. That is, there is a strict upper bound on the execution time of the software. Running a model in real time requires that you use deterministic algorithms in the time-critical portion of the application. Running the model in real time ensures that code running at each time step meets the deadlines imposed by the timing of the hardware inputs and outputs. Some ODE solvers, including the variable step-size ODE solvers, are inherently non-deterministic. You cannot use these non-deterministic ODE solvers in a real-time application.

To meet determinism requirements, you must use an ODE solver designed for determinism, which includes the Runge-Kutta 1–4 ODE solvers. These ODE solvers are all fixed time-step algorithms, which are appropriate for fixed sampling-rate inputs and outputs.

All of the discrete ODE solvers have an inherently fixed time step size and are inherently deterministic. Therefore, the discrete ODE solvers are appropriate for real-time implementation. The Discrete Systems functions use the discrete ODE solvers in their implementation. ODE solver determinism is important only when you use continuous dynamic functions, such as the Integrator, State Space, Transfer Function, and Zero-Pole-Gain functions.

Case Study: Rapid Control Prototype and Hardware-in-the-Loop Simulation

This section provides an overview of the process you might use to create a controller. It describes an off-line simulation that contains both the plant and the controller. It also describes how to modify this off-line simulation to create an RCP, followed by an HIL, simulation.

Off-Line Simulation

The starting point is the off-line simulation of the full system. The simulation diagram in Figure 3-1 represents a simple control system. The system contains a controller, a model of the plant, and a front panel control that represents the set point or reference signal.

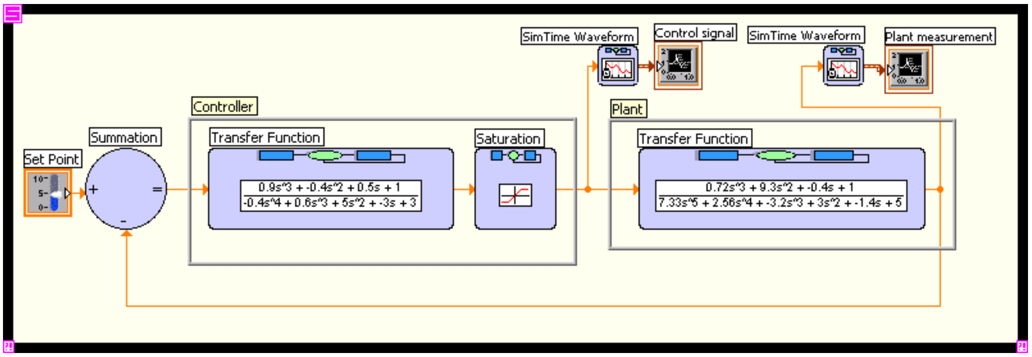


Figure 3-1. Full System Simulation

Rapid Control Prototype Implementation

You might want to prototype the controller implementations on real-time hardware with the controller outputs connected to real actuators and the controller inputs connected to real sensors. This prototyping method is often referred to as rapid control prototyping.

To convert the full simulation to a rapid control prototype, remove the plant model from the simulation. Replace the plant input with a hardware output and replace the plant output with a hardware input.

Note The inputs and outputs can be analog or digital signals, timing signals such as encoders, CAN signals, and so on.

Figure 3-2 shows the block diagram of the RCP implementation.

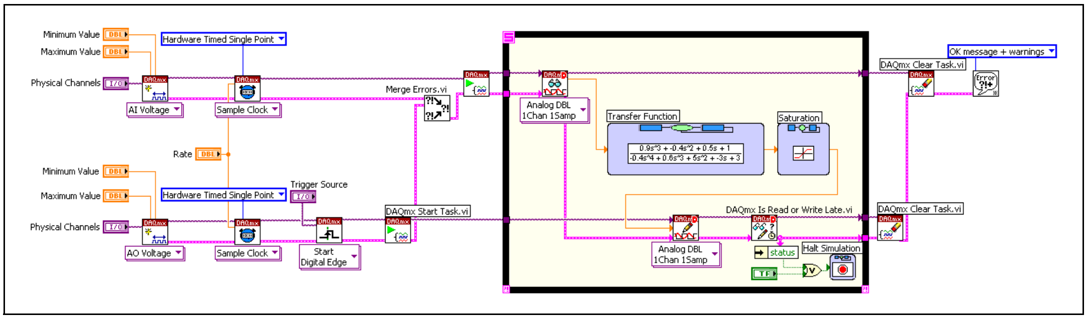


Figure 3-2. Rapid Control Prototype Implementation

In Figure 3-2, notice that the block diagram code to the left of the Simulation Loop is the NI-DAQmx setup code. Inside the Simulation Loop, the DAQmx Read (Analog DBL 1Chan 1Samp) VI receives an analog value from the data acquisition device, and the DAQmx Write (Analog DBL 1Chan 1Samp) VI returns an analog value to the device. Refer to the *NI-DAQmx Help* for more information about single-point hardware-timed I/O.

Hardware-in-the-Loop Implementation

You might want to test a controller implementation versus a plant simulation. In these HIL simulations, the plant model and its associated inputs and outputs must occur in real time to provide the most accurate and reliable controller testing.

You can convert the full simulation of the system to an HIL simulation in a manner similar to the RCP implementation. You must remove the controller from the full off-line simulation and then replace the plant model input with a hardware input and the plant model output with a hardware output. The result, shown in Figure 3-3, is a system similar to the RCP implementation, except with the controller model, not the plant model, replaced with physical hardware inputs and outputs.

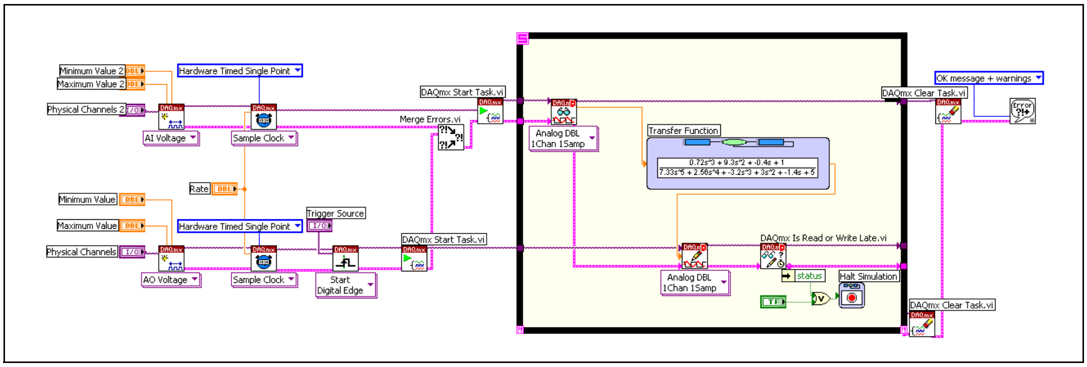


Figure 3-3. Hardware-in-the-Loop Implementation

Discrete Behavior

Notice the discrete nature of the I/O calls in the simulation diagram. The I/O executes only once per fixed time step of the continuous Integrator function. Refer to the *LabVIEW VIs, Functions, and Structures* section of Chapter 2, *Simulation Environment*, for more information about configuring the discrete behavior of LabVIEW VIs you place on a simulation diagram.

4

Ordinary Differential Equation Solvers

To compute the behavior of a continuous model over time, the Simulation Module must solve the following initial value problem:

$$\frac{dy}{dt} = f(t, y)$$

$$y(t_0) = y_0$$

y represents the outputs of the Integrator functions, y_0 represents the initial conditions for those Integrator functions, and $f(t, y)$ represents the other (non-integrating) functions on the simulation diagram.

The Simulation Module provides a number of solution methods for this problem. Each of the ODE solvers approximates the behavior of the model at time $t + dt$ based on the behavior of the model from t_0 to t. The quantity dt is the step size of the ODE solver, and the interval from t to $t + dt$ is one time step taken by the ODE solver. All ODE solvers might need to evaluate the model diagram multiple times to compute accurate values for time $t + dt$. This means that when you are debugging a simulation, you might notice data flowing through the simulation diagram several times before LabVIEW updates the indicators and graphs with their values for the end of the time step.

It is useful to understand the various strengths and weaknesses of the ODE solvers when simulating different types of models so you can determine the appropriate ODE solver to use for an application. Refer to the following books for more information about ODE solvers: *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*[1] and *Numerical Solution of Ordinary Differential Equations*[2].

[1] Ascher, Uri M., and Linda R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. Philadelphia: Society for Industrial and Applied Mathematics, 1998.

[2] Shampine, Lawrence F. *Numerical Solution of Ordinary Differential Equations*. New York: Chapman & Hall, Inc., 1994.

Simulation Discontinuities

In general, the Simulation Module ODE solvers assume that all simulation diagram signals and their derivatives are continuous throughout any time step. To get the most accurate solution possible, the ODE solver must stop and restart whenever it encounters a discontinuity. Therefore, the presence of many discontinuities in a simulation limits the maximum step size that an ODE solver can take. This influences which ODE solver you should choose.

LabVIEW already accounts for discontinuities introduced by the Nonlinear Systems and Discrete Systems functions the Simulation Module provides. However, if a model contains continuous-mode subVIs whose outputs, or derivatives of outputs, are not continuous, then the model must reset the ODE solver at the points of discontinuity. You can add a Zero Cross Detect function or wire the **reset** terminal on an Integrator function to ensure the model resets the ODE solver correctly.

ODE Solver Accuracy and Order

To measure the accuracy of a simulation, you can measure the error introduced into the solution per time step, which is known as the local error, and measure the maximum difference between the computed solution and the exact solution, which is known as the global error. The amount the error changes when you vary the step size depends on the order of the ODE solver you use. If you reduce the step size of an ODE solver by a factor of λ, then the local error is reduced by approximately λ^{n+1}, where n is the order of the ODE solver. Depending on the simulation, the global error might be reduced by approximately λ^n. For example, if you run a model once with a step size of 0.1 and then run the model again with a step size of 0.05, the solution a first-order ODE solver gives might be approximately twice as accurate in the second run as in the first run. The solution a fourth-order ODE solver gives might be approximately 16 times more accurate in the second run than in the first run.

In general, with a higher order ODE solver, you can use fewer time steps and larger step sizes to get the accuracy you need, thus decreasing the effects of round-off in the solution and potentially reducing the amount of time needed to compute the solution. However, this accuracy might come with a higher computational cost per step. If the step size is limited by factors other than error, such as discontinuities, you might get the accuracy you need more efficiently by using a lower order ODE solver.

Variable Step-Size versus Fixed Step-Size ODE Solvers

Some of the ODE solvers the Simulation Module provides estimate the error introduced by the ODE solver at each time step. These ODE solvers then adjust the step size throughout the simulation to ensure that this per-step error remains at a given relative and absolute tolerance. More specifically, for each integrator variable *y,* the ODE solver varies the step size to control the error according to the following approximation:

$$\text{per-step error} \approx |y| * \text{relative tolerance} + \text{absolute tolerance}$$

These variable step-size ODE solvers can take small time steps when the simulation variables vary rapidly and can take larger time steps when the simulation variables vary more slowly, which increases computational efficiency.

However, variable step-size ODE solvers are not appropriate for deterministic real-time applications because the computational overhead of taking a time step varies over the course of a simulation. Therefore, the Simulation Module also provides fixed step-size, deterministic ODE solvers. These ODE solvers do not estimate the local per-step error and maintain a fixed step size throughout a simulation.

Single-Step versus Multi-Step ODE Solvers

The Simulation Module single-step ODE solvers approximate the behavior of the model at time $t + dt$ by taking into account only the behavior of the model at time t. In contrast, multi-step ODE solvers approximate the behavior at the end of the time step by taking into account the model behavior at a number of previous time steps.

To achieve high order accuracy, single-step ODE solvers might need to evaluate the model diagram more often per time step than multi-step ODE solvers. Therefore, single-step ODE solvers might incur a higher computational cost per step. Because of this, multi-step ODE solvers might be able to compute an accurate solution more efficiently, in some cases, than single-step ODE solvers. However, a certain amount of work is required to start up a multi-step ODE solver, which must be incurred each time an Integrator function is reset or a discontinuity is encountered. In simulations in which the ODE solver is reset often, it might be more efficient to use a single-step ODE solver.

Stiff ODE Solvers

Certain problems, especially those with transients that vary much more quickly than the problem solution, can be difficult to solve numerically without using special ODE solver techniques. These problems are stiff problems. When you solve stiff problems without a stiff ODE solver, you might notice that variable step-size ODE solvers take smaller and smaller time steps until they can no longer make progress on the simulation. You also might notice an inaccurate and rapidly growing oscillatory solution no matter how small a step size you use. In this situation, you can use a stiff ODE solver to get a more accurate solution.

LabVIEW Simulation ODE Solvers

Use the **Simulation Parameters** dialog box or Set Diagram Parameters function to specify the ODE solver to use to evaluate the simulation diagram. The Simulation Module provides the following ODE solvers:

- **Runge-Kutta 1 (Euler)**—This is a fixed step-size, single-step explicit Runge-Kutta ODE solver of first order.
- **Runge-Kutta 2**—This is a fixed step-size, single-step explicit Runge-Kutta ODE solver of second order.
- **Runge-Kutta 3**—This is a fixed step-size, single-step explicit Runge-Kutta ODE solver of third order.
- **Runge-Kutta 4**—This is a fixed step-size, single-step explicit Runge-Kutta ODE solver of fourth order.
- **Runge-Kutta 23**—This is a variable step-size, single-step explicit Runge-Kutta ODE solver of third order.
- **Runge-Kutta 45**—This is a variable step-size, single-step explicit Runge-Kutta ODE solver of fifth order, which uses the Dormand-Prince coefficients.
- **BDF**—This is a variable step-size, variable order (orders 1 through 5) implementation of the multi-step Gear’s Method. This method is adequate for moderately stiff problems.
- **Adams-Moulton**—This is a variable step-size, variable order (orders 1 through 12) implementation of the Adams-Moulton predictor-corrector pair in predict-evaluate-correct-evaluate (PECE) mode.

Refer to the *Setting Simulation Parameters* section in Chapter 2, *Simulation Environment*, for more information about specifying an ODE solver.

5

Simulink Translator

You can use the Simulink Translator to convert a Simulink model (`.mdl`) file into a LabVIEW VI that contains a simulation diagram. The `.mdl` file contains a hierarchical definition of the system and the blocks, lines, and subsystems within the system.

Note The Simulink Translator cannot convert Stateflow® diagrams or other Simulink blocksets.

The Simulink Translator converts the `.mdl` file into a top-level VI that consists of a simulation diagram containing LabVIEW functions, wires, and subsystems corresponding to the contents of the `.mdl` file.

Converting Simulink Models into LabVIEW Code

Select **Tools»Simulation Tools»Simulink Translator** from the pull-down menu to display the **Simulink Translator** dialog box. Refer to the *LabVIEW Help* for information about specific **Simulink Translator** dialog box options.

The Simulink Translator reads and parses the `.mdl` file for model information such as timing and ODE solver settings.

Note If MathWorks MATLAB® and Simulink are installed on the computer, the Simulink Translator automatically executes any `.m` files included in the `.mdl` file before it parses the `.mdl` file.

The Simulink Translator parses and stores each system, subsystem, block, and line in Common Graph Description (CGD) format, which is XML-based. The Simulink Translator uses the CGD file to generate a LabVIEW subsystem for each Simulink subsystem in order. Converting in order ensures that the Simulink Translator generates every subsystem that has a parent simulation diagram before it generates the parent simulation diagram. The Simulink Translator converts all blocks into one or more LabVIEW functions or subsystems. The Simulink Translator also converts lines into wires that connect terminals on the LabVIEW functions and subsystems.

Common Warnings

If the Simulink Translator cannot find a value for a parameter in the `.mdl` file it is converting, you receive a warning. In these cases, the Simulink Translator uses the default value of the parameter in the corresponding LabVIEW function.

Note In some cases, the Simulink Translator cannot find a value for a parameter because the parameter contains an expression instead of a constant value. If MATLAB is installed on the computer, the Simulink Translator attempts to evaluate these MATLAB expressions in the `.mdl` file prior to translating the file. If the Simulink Translator successfully evaluates the expression, the Simulink Translator uses the result of that evaluation as the parameter value and does not produce a warning.

The Simulink Translator cannot fully convert all elements of every Simulink model to LabVIEW block diagram code. If the Simulink Translator encounters a block it cannot convert, you receive a warning. In these cases, the Simulink Translator creates a placeholder subsystem. You must create a subsystem with the same functionality as the block to replace this placeholder subsystem. Refer to the *LabVIEW Help* for a list of the Simulink blocks the Simulink Translator cannot convert.

A Technical Support and Professional Services

Visit the following sections of the National Instruments Web site at `ni.com` for technical support and professional services:

- **Support**—Online technical support resources at `ni.com/support` include the following:
  - **Self-Help Resources**—For immediate answers and solutions, visit the award-winning National Instruments Web site for software drivers and updates, a searchable KnowledgeBase, product manuals, step-by-step troubleshooting wizards, thousands of example programs, tutorials, application notes, instrument drivers, and so on.
  - **Free Technical Support**—All registered users receive free Basic Service, which includes access to hundreds of Application Engineers worldwide in the NI Developer Exchange at `ni.com/exchange`. National Instruments Application Engineers make sure every question receives an answer.
- **Training and Certification**—Visit `ni.com/training` for self-paced training, eLearning virtual classrooms, interactive CDs, and Certification program information. You also can register for instructor-led, hands-on courses at locations around the world.
- **System Integration**—If you have time constraints, limited in-house technical resources, or other project challenges, NI Alliance Program members can help. To learn more, call your local NI office or visit `ni.com/alliance`.

If you searched `ni.com` and could not find the answers you need, contact your local office or NI corporate headquarters. Phone numbers for our worldwide offices are listed at the front of this manual. You also can visit the Worldwide Offices section of `ni.com/niglobal` to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

Glossary

B

BDF	Backwards difference formula. Also known as Gear's Method.

C

CGD	Common Graph Description. The format the Simulink Translator uses to store each system, subsystem, block, and line from the Simulink model.
continuous model	Dynamic system model used to represent real-world signals that vary continuously with time. A continuous model is characterized by differential equations.

D

direct feedthrough	Relationship 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.
discrete model	Dynamic system model used to represent signals that are sampled in time at discrete intervals. A discrete model is characterized by difference equations.
distributed parameter model	Physical model that can be described by partial differential equations.
dynamic system	System whose behavior varies with time.

E

empirical modeling	Modeling technique in which you use experimental data to define a system model.

F

feedback cycle	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.

G

global error	Maximum difference between the solution the function computes and the exact solution.

H

HIL	Hardware-in-the-loop. A simulation configuration in which you test a controller implementation with a software model of the plant.

I

indirect feedthrough	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.

L

linear model	Model that obeys the principle of superposition.
local error	Error introduced into the solution per time step.
lumped parameter model	Physical model described by an ordinary differential equation.

M

multi-step ODE solver	ODE solver that approximates the behavior of a model at time $t + dt$ by taking into account the behavior of the model at a number of previous time steps.

N

nonlinear model	Model that does not obey the principle of superposition.

O

ODE	Ordinary differential equation.
order	ODE solver characteristic that determines how much the error amount changes when you vary the step size.

P

parameters terminal	Optional Simulation Loop terminal used to configure a cluster of simulation parameters for the enclosed diagram.
PECE	Predict-evaluate-correct-evaluate.
physical modeling	Modeling technique in which you use the laws of physics to define a system model.

R

RCP	Rapid control prototype. A simulation configuration in which you test plant hardware with a software model of the controller.

S

simulation diagram	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 may only execute after all necessary inputs have become available. • Outputs are generated after a function completes execution.
Simulation Loop	Loop that executes the simulation diagram over multiple time steps.

single-step ODE solver	ODE solver that approximates the behavior of a model at time $t + dt$ by taking into account only the behavior of the model at time t.
step size	Size of the interval of one time step.
stiff ODE solver	ODE solver used to evaluate a stiff model.
stiff system	System whose dynamics are described by widely varying time constants.

T

time step	Interval from t to $t + dt$.
time-invariant model	Model whose parameters do not change with time.
time-variant model	Model whose parameters change with time.

V

variable step-size ODE solver	ODE solver that adjusts the step size throughout the simulation to ensure that the per-step error remains at a given relative and absolute tolerance.