LabVIEW™ Real-Time 2: Architecting Embedded Systems Exercises

Course Software Version 2012
November 2012 Edition
Part Number 325585B-01

Copyright
© 2010–2012 National Instruments Corporation. All rights reserved.
Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.
National Instruments respects the intellectual property of others, and we ask our users to do the same. NI software is protected by copyright and other intellectual property laws. Where NI software may be used to reproduce software or other materials belonging to others, you may use NI software only to reproduce materials that you may reproduce in accordance with the terms of any applicable license or other legal restriction.

End-User License Agreements and Third-Party Legal Notices
You can find end-user license agreements (EULAs) and third-party legal notices in the following locations:
• Notices are located in the <National Instruments>_Legal Information and <National Instruments> directories.
• EULAs are located in the <National Instruments>_Shared\MDF\Legal\license directory.
• Review <National Instruments>_Legal Information.txt for more information on including legal information in installers built with NI products.

Trademarks
LabVIEW, National Instruments, NI, ni.com, the National Instruments corporate logo, and the Eagle logo are trademarks of National Instruments Corporation. Refer to the Trademark Information at ni.com/trademarks for other National Instruments trademarks.
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/technology, refer to the appropriate location: Help>Patents in your software, the patents.txt file on your media, or the National Instruments Patent Notice at ni.com/patents.
Worldwide Technical Support and Product Information
ni.com

Worldwide Offices
Visit 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.

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

To comment on National Instruments documentation, refer to the National Instruments Web site at ni.com/info and enter the Info Code feedback.
Contents

Student Guide
A. NI Certification........................................................................................................................................................................................vii
B. Course Description ..................................................................................................................................................................................viii
C. What You Need to Get Started...............................................................................................................................................................viii
D. Installing the Course Software ................................................................................................................................................................ix
E. Course Goals ...........................................................................................................................................................................................ix
F. Course Conventions.................................................................................................................................................................................x

Lesson 1
Real-Time Application Design Flow
Exercise 1-1 Configure the RT System and Review Temperature Chamber Project .................................................................1-2

Lesson 2
Documenting Your Design
Exercise 2-1 Data Communication Diagram ........................................................................................................................................2-2

Lesson 3
LabVIEW RT Processes and Inter-process Communication
Exercise 3-1 Create a Deterministic Temperature Control Loop ........................................................................................................3-2
Exercise 3-2 Create a User Interface VI that Sends Messages Using Queues .........................................................................................3-7
Exercise 3-3 (Self-Study) Inter-Process Communication Using RT FIFO Functions ...........................................................................3-26

Lesson 4
LabVIEW RT Network Communication
Exercise 4-1 Add Network Communication .......................................................................................................................................4-2

Lesson 5
Managing Memory and Monitoring System Health
Exercise 5-1 System Health and Memory Monitoring .......................................................................................................................5-2
## Lesson 6
### Reliability
<table>
<thead>
<tr>
<th>Exercise 6-1</th>
<th>Adding a Safe State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exercise 6-2</td>
<td>Central Error Handling</td>
</tr>
<tr>
<td>Exercise 6-3</td>
<td>Adding a Watchdog</td>
</tr>
</tbody>
</table>

## Lesson 7
### Debugging, Benchmarking, and Testing
<table>
<thead>
<tr>
<th>Exercise 7-1</th>
<th>(Self-Study) Creating a Time Budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exercise 7-2</td>
<td>(Self-Study) Real-Time Execution Trace Toolkit</td>
</tr>
<tr>
<td>Exercise 7-3</td>
<td>(Self-Study) Using Flags in the Real-Time Execution Trace Toolkit</td>
</tr>
</tbody>
</table>
LabVIEW RT Processes and Inter-process Communication

Complete the exercises in this lesson to reinforce the concepts you learn in Lesson 3 of LabVIEW Real-Time 2.

Exercises

- Exercise 3-1 Create a Deterministic Temperature Control Loop
- Exercise 3-2 Create a User Interface VI that Sends Messages Using Queues
- Exercise 3-3 (Self-Study) Inter-Process Communication Using RT FIFO Functions
Exercise 3-1  Create a Deterministic Temperature Control Loop

Goal
Create a high-priority deterministic control loop that uses a proportional-integral-derivative (PID) algorithm to control the temperature of a temperature chamber using a lamp and fan. You should be able to set a desired temperature setpoint on the front panel, and adjust the PID gains to tweak the controller while it is running. This controller uses the RIO Scan Interface (RSI) programming mode to interface with I/O.

Scenario
Currently, the PID algorithm is the most common control algorithm used in the industry. Often, people use PID to control processes that include heating and cooling systems, fluid level monitoring, flow control, and pressure control. In PID control, you must specify a process variable and a setpoint. The process variable is the system parameter you want to control, such as temperature, pressure, or flow rate, and the setpoint is the desired value for the parameter you are controlling. A PID controller determines a controller output value, such as the heater power or valve position. The controller applies the controller output value to the system, which in turn drives the process variable toward the setpoint value.

In this exercise you use the RIO Scan Interface (RSI) to communicate with I/O within your control loop, as shown in Figure 3-1. RSI technology allows single-point I/O access up to rates of a few hundred hertz without the need to write FPGA code or an RT to FPGA interface. When the controller is accessing I/O via the scan interface, module I/O is automatically read from the modules and placed in a current value table on the CompactRIO controller.

Figure 3-1. Simple System Communication Diagram
Implementation
1. If not already open, open Temperature Chamber.lvproj from the <Exercises>\LabVIEW Real-Time 2\Course Project directory.
2. In the Project Explorer window, expand the Real-Time target.
3. Right-click the cRIO-9074»RT Loops virtual folder and select New»VI from the shortcut menu.
4. Save the VI as Temperature Control.vi in the <Exercises>\LabVIEW Real-Time 2\Course Project\RT Loops directory.
5. On the block diagram, place down a Timed Loop.

Tip Press <Ctrl-Space> to open the Quick Drop dialog box and quickly find the VI, function, or structure you are looking for by name.
6. Complete the block diagram as shown in Figure 3-2 to create the temperature control loop.

**Figure 3-2. Temperature Control Block Diagram**

1. Timed Loop—Double-click the Timed Loop input node. Set the Loop Timing Source to Synchronize to Scan Engine. The Timed Loop will execute at the end of each scan of the NI Scan Engine. The period between iterations corresponds to the Scan Period setting in the CompactRIO Properties window, which is 10 ms by default.

2. PID VI—This implements a PID controller using a PID algorithm.

3. PID Gains control—Right-click the PID gains input of the PID VI and select Create » Control.

4. Temperature I/O variable—Drag cRIO-9074 » Chassis » Mod1 » Temperature from the Project Explorer window to the block diagram.

5. Current Temperature indicator—Right-click the Temperature I/O variable and select Create » Indicator. Rename the control Current Temperature.

6. Setpoint control—Right-click the setpoint input of the PID VI and select Create » Control.

7. Output Range constant—Right-click the output range input of the PID VI and select Create » Constant. Change the values as shown in Figure 3-2.

8. Unbundle By Name—Wire the Output Range constant to the Unbundle By Name function and set the Unbundle By Name terminal to output high.

9. Fan I/O variable—Drag cRIO-9074 » Chassis » Mod2 » Fan from the Project Explorer window to the block diagram.

10. Lamp I/O variable—Drag cRIO-9074 » Chassis » Mod2 » Lamp from the Project Explorer window to the block diagram.
7. Arrange the front panel of the Temperature Control VI, as shown in Figure 3-3.

Figure 3-3. Temperature Control Front Panel

8. Edit the Temperature Control VI Icon as shown in Figure 3-4.

Figure 3-4. Temperature Control VI Icon

9. Save the VI.
Testing
1. Run the VI.
2. Look at the Current Temperature indicator for the current temperature in the temperature chamber.
3. Set PID Gains control to the following values:
   - Proportional gain = 15
   - Integral time = 0.05
   - Derivative time = 0
4. Set the Setpoint control to a value three or four degrees higher than the current temperature.
5. Verify that the lamp intensity increases, and the fan intensity decreases.
6. Notice what happens as the temperature approaches the setpoint temperature, and as it reaches the setpoint temperature.
7. Now set the Setpoint control to a value a few degrees lower than the current temperature.
8. Verify that the fan intensity increases, and the lamp intensity decreases.
9. Notice what happens as the temperature approaches the setpoint.
10. Click the Stop button.
11. Verify that both the lamp and the fan turn off.
12. Save and close the VI.

End of Exercise 3-1
Exercise 3-2  Create a User Interface VI that Sends Messages Using Queues

Goal
In this exercise you will begin creating the User Interface VI which will run on your host computer. The User Interface VI sends messages using the Queue functions for inter-process communication. This exercise uses a Producer/Consumer with Events design pattern. The Producer or Event Handler loop uses an Event Structure to quickly respond to user interface events, and passes data to the Consumer or UI Message loop using queues, as shown in Figure 3-5. By the end of the exercise you are able to run the User Interface VI and respond to user interface events.

Description
Queues are the recommended inter-process communication mechanism for command- or message-based communication between two non-deterministic processes because they are flexible, easy to use, and allow you to transfer buffered data. When sending messages, you might need to send multiple data types within a single packet. Queues support the Cluster data type, which you can use to bundle together different data types, both fixed size and variable size such as strings. You can use queues on both real-time and Windows operating systems.

Implementation
1. If not already open, open the Temperature Chamber.lvproj from the <Exercises>\LabVIEW Real-Time 2\Course Project directory.
2. Open User Interface.vi from the Project Explorer window.
3. Place two While Loops and an Event structure on the block diagram as shown in Figure 3-6.
1 **Event Handling Loop**—Place down a While Loop and label it as *Event Handling Loop*. Place an Event structure inside this While Loop. The Event Handling Loop quickly handles the events from the user interface front panel and sends messages to the UI Message Loop.

2 **UI Message Loop**—Place down a While Loop and label it as *UI Message Loop*. The UI Message Loop receives messages and executes the corresponding actions, such as connecting to the RT target and updating the control configuration.
4. Add a queue to the block diagram as shown in Figure 3-7.

**Figure 3-7. Adding the Queue to the User Interface VI**

1. Obtain Queue—Returns a reference to a queue. Right-click the max queue size input and select Create¬»Constant. Set the constant to 100.

2. UI Data.ctl type def constant—Drag **My Computer»Controls»UI Data.ctl** from the Project Explorer window to the block diagram. The UI Data.ctl is a type def and defines the data type of the queue. The UI Data.ctl is a cluster containing a string element named Message and a variant element named Data.

3. Enqueue Element—Adds an element to the back of the queue.

4. Bundle By Name—Use this function to create an element of the data type defined by UI Data.ctl. Wire the UI Data.ctl constant to the input cluster output of the Bundle By Name function. Wire the output cluster output of the Bundle By Name function to the element input of the Enqueue Element function. Set the Bundle By Name item to Message. Right-click the input terminal and select Create¬»Constant. Set the string constant to Initialize.

5. Flat Sequence Structure—Use this structure to ensure that the Connected indicator sets back to FALSE each time the User Interface VI starts running.

6. Connected? local variable—Right-click the Connected? indicator to the right side of the While Loops and select Create¬»Local Variable. Place the Connected? local variable in the Flat Sequence Structure. Right-click the input of the local variable and select Create¬»Constant to wire a FALSE constant to the local variable.

7. Error Constant—Right-click the error in input of the Obtain Queue function and select Create¬»Constant. Drag the error constant into the Flat Sequence structure.
Event Handling Loop
The Event Handling Loop quickly handles the events from the user interface front panel and sends messages to the UI Message Loop.

1. Right-click the Event structure and select **Edit Events Handled by This Case**.
   - In the Edit Events dialog box, select **Controls»Exit** in the Event Sources section.
   - Select **Value Change** in the Events section and click **OK**.

2. Modify the block diagram as shown in Figure 3-8 to add the Exit event case.

   **Figure 3-8. Exit Event Case Block Diagram**

   1 Exit control—Drag the **Exit** control from the left side of the block diagram to the "Exit":Value Change case.
   2 UI Data.ctl type def constant—Drag **My Computer»Controls»UI Data.ctl** from the Project Explorer window to the block diagram.
   3 Bundle By Name—Use this function to create an element of the data type defined by UI Data.ctl. Wire the UI Data.ctl constant to the input cluster input of the Bundle By Name function. Set the Bundle By Name item to **Message**. Right-click the input terminal and select **Create»Constant**. Set the string constant to **Exit**.
   4 Enqueue Element function—Adds an element to the back of the queue.

3. Right-click the border of the Event structure and select **Duplicate Event Case**.
   - In the Edit Events dialog box, select **Controls»Connect to Controller** in the Event Sources section.
   - Select **Value Change** in the Events section and click **OK**.
4. Modify the block diagram as shown in Figure 3-9 to add the Connect to Controller case.

**Figure 3-9. Connect to Controller Event Case**

- Connect to Controller control—Delete the **Exit 2** control and drag the **Connect to Controller** control from the left side of the block diagram into the Event structure.
- String Constant—Modify the string constant to **Connect**.

5. Right-click the Event structure and select **Duplicate Event Case**.
   - In the Edit Events dialog box, select **Controls»Run Control** in the Event Sources section.
   - Select **Value Change** in the Events section and click **OK**.
6. Modify the block diagram as shown in Figure 3-10 to add the Run Control case:

**Figure 3-10. Run Control Event Case**

1. Run Control—Delete the **Connect to Controller 2** control and drag the **Run Control** control from the left side of the block diagram into the Event structure.
2. String constant—Modify the string constant to **Run Control**.

7. Right-click the Event structure and select **Duplicate Event Case**.
   - In the Edit Events dialog box, select **Controls»Stop Control** in the Event Sources section.
   - Select **Value Change** in the **Events** section and click **OK**.
8. Modify the block diagram as shown in Figure 3-11 to add the Stop Control case.

**Figure 3-11. Stop Control Event Case**

1. Stop Control—Delete the Run Control 2 control and drag the Stop Control control from the left side of the block diagram into the Event structure.
2. String constant—Modify the string constant to Stop Control.

9. Right-click the Event structure and select **Duplicate Event Case**.
   - In the Edit Events dialog box, select **Controls»Update Control Configuration** in the Event Sources section.
   - Select **Value Change** in the Events section and click **OK**.
10. Modify the block diagram as shown in Figure 3-12 to add the Update Control Configuration case:

**Figure 3-12. Update Control Configuration Event Case**

1. Update Control Configuration control—Delete the Stop Control 2 control and drag the Update Control Configuration control from the left side of the block diagram into the Event structure.
2. String constant—Modify the string constant to Update Control Configuration.
3. Bundle By Name—Resize the Bundle By Name function to show the Data item, which has a variant data type.
4. Control Configurations control—Drag the Control Configurations control from the left side of the block diagram into the Event case.
5. To Variant function—This function converts any LabVIEW data type to variant data type.

11. Right-click the Event structure and select Duplicate Event Case.

   - In the Edit Events dialog box, select Controls.Channel 0 Setpoint in the Event Sources section.
   - Select Value Change in the Events section and click OK.
12. Modify the block diagram as shown in Figure 3-13 to add the Channel 0 Setpoint case:

**Figure 3-13. Channel 0 Setpoint Event Case**

1. **String constant**—Modify the string constant to **Update Setpoints**.
2. **Channel 0 Setpoint control**—Delete the **Control Configurations** control and drag the **Channel 0 Setpoint** control from the left side of the block diagram to this event case. Wire the **Channel 0 Setpoint** control to the To Variant function.
13. Modify the block diagram, as shown in Figure 3-14, to add a Case structure to handle error conditions:

Figure 3-14. Adding the Case Structure No Error Case

1. Case structure—Place down a Case structure. Wire the error cluster from the Event structure to the Selector Terminal of the Case structure.

2. True Constant—Place a True Constant in the “Exit”:Value Change event. Wire the True Constant through the No Error case of the Case structure to the Loop Condition terminal.

3. Use Default If Unwired tunnel—Right-click the tunnel and verify that Use Default If Unwired is selected.
14. Go to the Error case of the Case structure and modify the block diagram, as shown in Figure 3-15, to handle errors.

**Tip** You can select portion of the code from the “Exit” Value Change event case of the Event structure and <Ctrl>-drag a copy of the code to the Error case.

**Figure 3-15.** Event Structure Error Case

1. **Error case**—If an error occurs, this case adds an Exit message to the queue.
2. **True Constant**—If an error occurs, the True Constant in this case will exit the While Loop.

**UI Message Loop**

The UI Message Loop receives messages and executes the corresponding actions, such as connecting to the RT target and updating the control configuration.
1. Modify the UI Message Loop on the block diagram, as shown in Figure 3-16:

**Figure 3-16. UI Message Loop Default Case**

1. **Dequeue Element**—This function dequeues the data enqueued from the Event Handling While Loop. Place a Dequeue function inside the UI Message While Loop. Branch the queue out and error out outputs of the Enqueue Element function located outside the UI Message Loop to the corresponding inputs of the Dequeue Element function.

2. **Unbundle By Name**—Resize this function to show two elements.

3. **Case Structure**—Wire the Message terminal of the Unbundle by Name function to the Case Selector terminal.

4. **Case Selector Label**—Rename the "False", Default case to Default.
2. Create the Initialize case as shown in Figure 3-17.

**Figure 3-17. UI Message Loop Initialize Case**

1. Case structure—Go to the True case. Rename the True case as Initialize.
2. System Status indicator—Drag the System Status indicator from the right side of the block diagram to the UI Message While Loop.
3. String Constant—Wire an empty string constant to the System Status indicator.
4. Right-click the tunnel and select **Use Default if Unwired**.

3. Right-click the Case structure and select **Duplicate Case** from the shortcut menu.
4. Modify the new case as shown in Figure 3-18.

**Figure 3-18. UI Message Loop Connect Case**

1. Case selector label—Rename the new case **Connect**.
2. String constant—Change the string constant to **Connect**.

**Note** You modify these cases in later exercises.

5. Right-click the Case structure and select **Duplicate Case** from the shortcut menu.
6. Modify the Case structure as shown in Figure 3-19.

**Figure 3-19. UI Message Loop Update Control Configuration Case**

1. **Case selector label**—Rename the new case as “Update Control Configuration”, “Update Setpoints”.

2. **String Constant**—Change the String Constant to “Update Control Configuration”.

7. Right-click the Case structure and select **Duplicate Case** from the shortcut menu.
8. Modify the case structure as shown in Figure 3-20.

**Figure 3-20. UI Message Loop Run Control Case**

1. **Case selector label**—Rename the new case as "Run Control", "Stop Control".

2. **String constant**—Change the string constant to Run Control/Stop Control.

9. Right-click the Case structure and select **Duplicate Case** from the shortcut menu.
10. Modify the Case structure as shown in Figure 3-21.

**Figure 3-21. UI Message Loop Exit Case**

1. Case selector label—Rename the new case as **Exit**.
2. Release Queue—This function releases the reference to the queue.
3. True Constant—Wire the True constant to the Loop Condition terminal of the While Loop. Right-click the tunnel and select **Use Default If Unwired**.
11. Add error handling to the block diagram as shown in Figure 3-22.

**Figure 3-22.** User Interface Error Handling

1. Merge Errors
2. Simple Error Handler VI

12. Save the User Interface VI.
Testing
1. Run the User Interface VI.
2. Click **Connect** and verify that the message in the System Status indicator is **Connect**.
3. Click **Run Control, Stop Control, and Update Control Configurations** buttons and verify the messages they produce.
4. Click **Exit** to stop the application.
5. Save the VI and project.

End of Exercise 3-2
Exercise 3-3  (Self-Study) Inter-Process Communication Using RT FIFO Functions

Goal
To dynamically control an RT FIFO using RT FIFO functions for inter-process communication.

Scenario
In this exercise, you create a VI that runs on the RT target that requires transferring numeric data between the deterministic writer loop and non-deterministic reader loop, and the VI must not lose any numeric data. Therefore, you must use an RT FIFO to write and read data deterministically, and you must buffer the transfer of numeric data.

As a general rule, the RT FIFO size should be the period of the reader loop divided by the period of the writer loop multiplied by 1.1. However, this VI allows the user to configure the period for each loop using front panel controls, so the proper size of the RT FIFO cannot be determined until the VI is running.

Therefore, you should use the RT FIFO functions to transfer waveform data between the deterministic writer loop and non-deterministic reader loop because the RT FIFO functions allow you to programmatically create an RT FIFO and dynamically configure the RT FIFO buffer size at while the VI is running.

Implementation

Project Explorer
1. Open <Exercises>\LabVIEW Real-Time 2\RT FIFO Functions\RT FIFO Functions.lvproj.

2. Configure the CompactRIO target.
   - Right-click the RT CompactRIO target in the Project Explorer and select Properties.
   - In the General category, set the IP address to the IP address of your cRIO-9074.

3. Explore the shared variables used in this project.
   - Double-click each of the shared variables located in the cRIO-9074\Communication Shared Library.lvlib library to view the properties.
   - Notice in the Shared Variable Properties window that both shared variables have Variable Type set to Single Process. These shared variables can only be accessed locally by VIs on the RT target.
   - Notice in the RT FIFO category of the Shared Variable Properties window that both shared variables have a single element RT FIFO enabled. This configures the shared variables to write and read their latest value deterministically.
     - The RT FIFO Functions VI only needs to write and read the latest value of the Stop and Waveform Type shared variables.
   - Close the Shared Variable Property windows when finished.
Initialization
Create and configure an RT FIFO before the deterministic Timed Loop and non-deterministic While Loop begin executing.

1. Open the RT FIFO Functions VI from the Project Explorer.

2. Modify the block diagram, as shown in Figure 3-23, to programmatically create an RT FIFO and dynamically set the RT FIFO buffer size based on the values of the Reader Period and Writer Period controls using the following items:

   - Divide function
   - Multiply function
   - Round Toward +Infinity function

---

Figure 3-23. RT FIFO Functions VI Block Diagram—Create and Configure RT FIFO
- RT FIFO Create function
  - Place a numeric constant on the block diagram. Right-click the numeric constant and select Representation»DBL. Wire the numeric constant to the type input of the RT FIFO Create function.

- Create the remaining constants shown in Figure 3-23 by right-clicking each corresponding input and selecting Create»Constant.

- Complete the wiring in the block diagram as shown in Figure 3-23.

Deterministic Writer Loop

In this deterministic application, you separate deterministic tasks from non-deterministic tasks and place the deterministic tasks in the Timed Loop with the highest priority in an RT target VI to ensure those tasks receive enough processor resources. This Timed Loop will be the deterministic loop in this application and will perform the deterministic task of generating waveform data.

1. Modify the deterministic Timed Loop, as shown in Figure 3-24, using the following items:

   Figure 3-24. RT FIFO Functions VI Block Diagram—Deterministic Loop

- Waveform Type shared variable

- Waveform Generator VI—Place the Waveform Generator VI on the block diagram inside the Deterministic Timed Loop. This VI is located in the Project Files virtual folder in the Project Explorer.

- RT FIFO Write function—Writes an element to the RT FIFO. If left unwired, the timeout input is 0 ms. If this function tries to write an element to the RT FIFO when the RT FIFO is already full, the timed out? output terminal outputs true.

- Unbundle By Name—Extracts the status boolean data from the error cluster.
Stop shared variable

Or function—If an error has occurred or the Stop shared variable outputs a true value, then stop the Timed Loop.

Non-deterministic Reader Loop
In this deterministic application, you will separate non-deterministic tasks from deterministic tasks and place them in a normal priority While Loop which has a lower priority than the deterministic Timed Loop to ensure the non-deterministic tasks do not preempt the deterministic tasks. The While Loop containing the non-deterministic task of plotting waveform data will be the non-deterministic loop in this application.

1. Modify the non-deterministic reader While Loop, as shown in Figure 3-25, to read elements from the RT FIFO using the following items:

   **Figure 3-25. RT FIFO Functions VI—Non-deterministic Loop**

   - To Unsigned Byte Integer function—This function converts the Waveform Type control enum data type into a unsigned 8-bit integer data type.
   - Waveform Type shared variable
     - Right-click the variable and select **Access Mode»Write**.
   - While Loop—Place a While Loop inside the non-deterministic While Loop. This While Loop will read from the Result Data RT FIFO until the RT FIFO is empty.
   - RT FIFO Read
- Case structure—Place the Result Data indicator in the False case. Leave the True case empty. The Result Data indicator will only update when the RT FIFO is not empty.

- Stop shared variable
  - Right-click the variable and select Access Mode»Write.

- Unbundle By Name—Extracts the status boolean data from the error cluster. If an error has occurred or the Stop button has been pressed, then stop the While Loop.

- Or function

- Complete the wiring in the block diagram as shown in Figure 3-25.
Shutdown

After the deterministic Timed Loop and non-deterministic While Loop stop executing, delete the RT FIFO and handle the errors.

- Delete the RT FIFO and handle errors as shown in Figure 3-26 using the following items:

  Figure 3-26. RT FIFO Functions VI Block Diagram—Shutdown

- Merge Errors function
- RT FIFO Delete
- Simple Error Handler VI

2. In the Project Explorer, select **File»Save All (this Project).**
Test
1. Configure the front panel controls to the following values:
   - Reader Period: 100
   - Writer Period: 10
   - Waveform Type: Sine
2. Run the VI.
   The VI dynamically creates an RT FIFO that can hold 11 elements. Notice that the Overflow indicator stays unlit while the VI is running.

   **Note** As a general rule, the RT FIFO size should be the period of the reader loop divided by the period of the writer loop multiplied by 1.1. The calculation for the current reader and writer loops is (100 ms / 10 ms) × 1.1 which equals 11. Therefore, a safe buffer size for the Result Data RT FIFO is 11.
3. Stop the VI.
4. Configure the front panel controls to the following values:
   - Reader Period: 100
   - Writer Period: 50
5. Run the VI.
   This time the VI dynamically creates an RT FIFO that can hold 3 elements. Notice that the Overflow indicator stays unlit while the VI is running.
6. Stop the VI.
7. Configure the front panel controls to the following values:
   - Reader Period: 1000
   - Writer Period: 10
8. Run the VI.
   This time the VI dynamically creates an RT FIFO that can hold 110 elements. Notice that the Overflow indicator stays unlit while the VI is running.

9. Stop the VI.

End of Exercise 3-3