Contents

CHAPTER 13
Using the LabVIEW for CompactRIO Sample Projects

LabVIEW 2012 and later provides several fully functioning project templates and sample projects to use as starting points for your own applications. All of the sample projects are built on existing LabVIEW project templates, like the Simple State Machine or the Queued Message Handler. You can spend less time developing your own application by using one of the sample projects that already has a lot of the application written for you. Using a sample project also ensures that the design your application is based on is an architecture recommended by National Instruments.

LabVIEW 2012 and later includes three sample projects that were written for CompactRIO and NI Single-Board RIO targets:

- LabVIEW FPGA Control on CompactRIO
- LabVIEW Real-Time Control on CompactRIO (NI Scan Engine)
- LabVIEW FPGA Waveform Acquisition and Logging
- LabVIEW 2013 and later includes an additional sample project designed for SCADA systems:
- LabVIEW Supervisory Control and Data Acquisition
- LabVIEW Real-Time Sequencer Sample Project

Overview of Sample Projects

This document examines the architecture of the LabVIEW FPGA Control on CompactRIO sample project, but the architectural drawings for the other sample projects are included below as a reference.

LabVIEW FPGA Control on CompactRIO Sample Project

The LabVIEW FPGA Control on CompactRIO sample project implements deterministic, hardware-based control of a plant. The control algorithm, which was written with the LabVIEW FPGA Module, runs on the FPGA inside the CompactRIO device. You send commands and setpoint changes to the FPGA from the user interface, running on a desktop computer, by way of the real-time controller in the device. This controller also monitors the status of the application, such as CPU load and memory usage. This sample project has the following features:

- **High-performance control**—The control loop can run faster than 10 kHz and features four control algorithms operating in parallel, all with minimal jitter.
- **Hardware-based control**—Running the control algorithm and safety logic on the FPGA provides maximum reliability.
- **User interface with headless option**—The user interface VI interacts with the CompactRIO device and displays data. This VI can connect and disconnect from the device at any time without affecting the control loop.
- **Error handling**—The application reports and logs all errors from the CompactRIO device and then shuts down on critical errors.

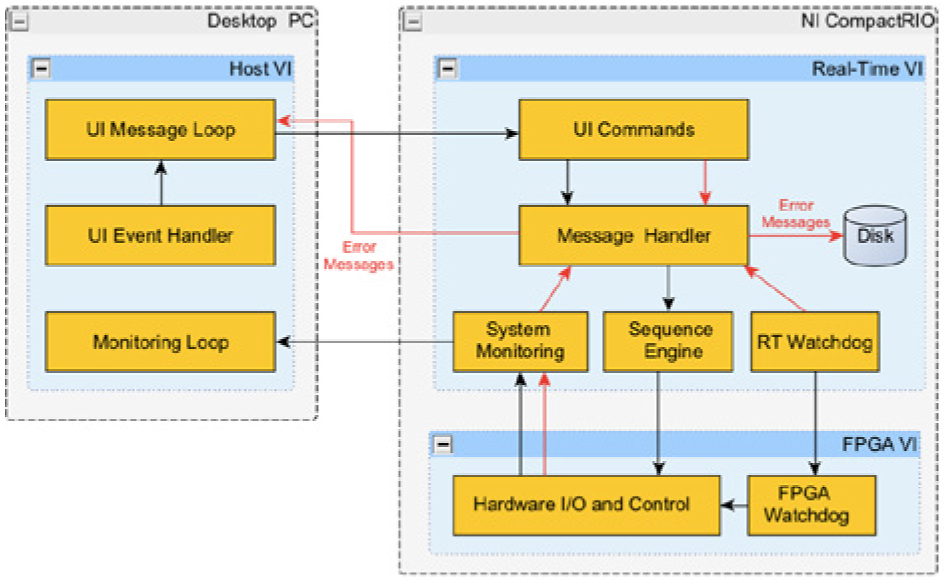


Figure 13.1. LabVIEW FPGA Control on CompactRIO Sample Project Architecture

LabVIEW Real-Time Sequencer on CompactRIO Sample Project

The LabVIEW Real-Time Sequencer on CompactRIO sample project is similar to the LabVIEW FPGA Control on CompactRIO sample project, but it implements a user-customizable control sequence to perform the control.

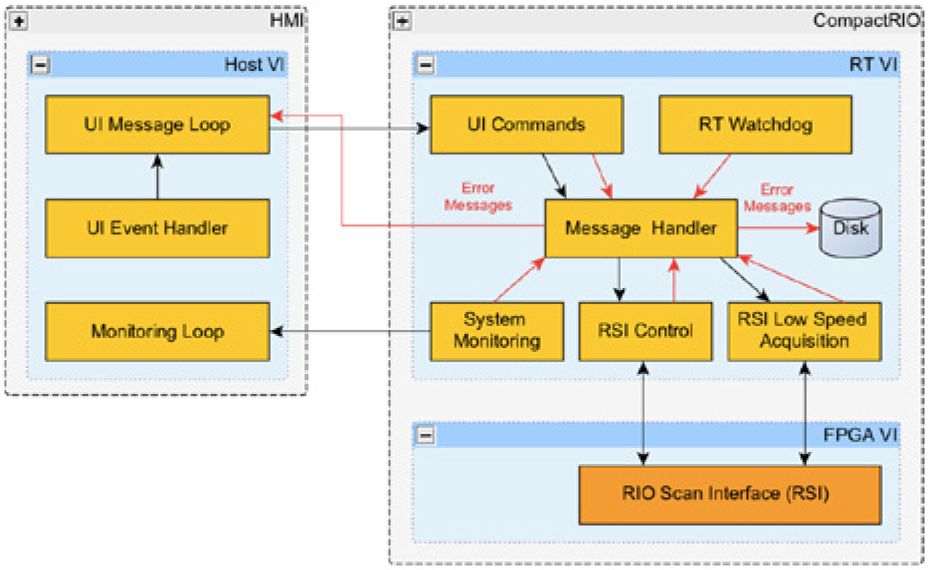


Figure 13.2. LabVIEW Real-Time Sequencer on CompactRIO Sample Project Architecture

LabVIEW Waveform Acquisition and Logging on CompactRIO Sample Project

The LabVIEW Waveform Acquisition and Logging on CompactRIO sample project acquires continuous waveform data from C Series I/O modules and logs it to disk on the real-time controller. This sample project has the following features:

- **Headless operation with optional user interface**—The user interface VI interacts with the CompactRIO device and displays data. This VI can connect and disconnect from the device at any time without affecting the acquisition and logging loop.

- **Triggered data logging**—The real-time VI logs acquired data to disk as TDMS files when a trigger condition is met. This sample project also manages the amount of disk space being used.

- **Error handling**—The application reports and logs all errors from the CompactRIO device, shutting down on any critical error.

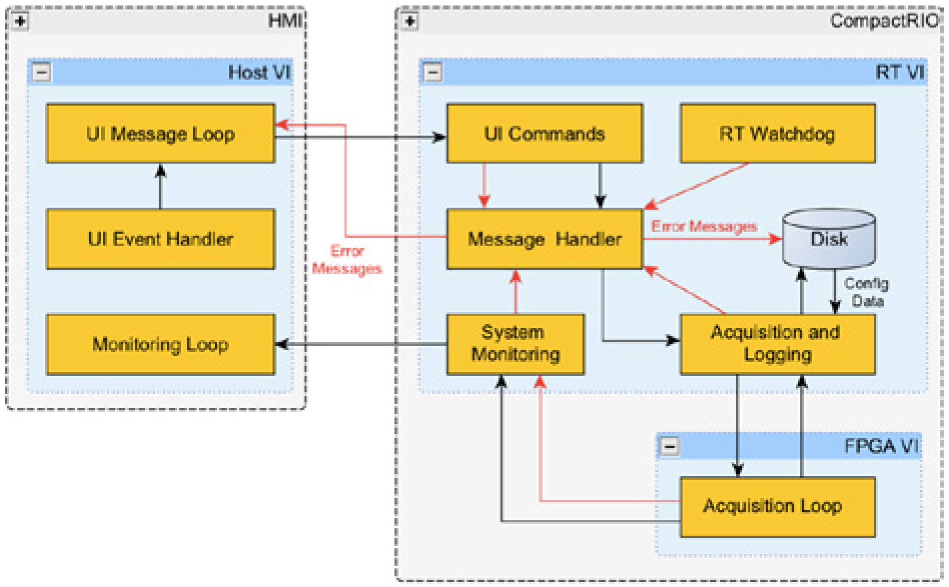


Figure 13.3. LabVIEW Waveform Acquisition and Logging Sample Project Architecture

LabVIEW Real-Time Control on CompactRIO (RIO Scan Interface) Sample Project

The LabVIEW Real-Time Control on CompactRIO (RIO Scan Interface) sample project implements deterministic, software-based control of a plant. This sample project uses the NI RIO Scan Interface, which is an alternative to programming with the LabVIEW FPGA Module, to perform I/O on the FPGA. The RIO Scan Interface programming mode is available on most CompactRIO and NI Single-Board RIO devices and is sufficient for applications that require single-point access to I/O at rates of a few hundred hertz. This sample project has the following features:

- **Deterministic, software-based control**—The real-time controller executes parallel control algorithms that run at different rates.

- **RIO Scan Interface**—The RIO Scan Interface provides single-point updates to I/O variables at rates up to a few hundred hertz without requiring FPGA programming.

- **User interface**—The user interface VI interacts with the CompactRIO device and displays data. This VI can disconnect from the device and reconnect at any time without affecting the control loop.

- **Error handling**—The application reports and logs all errors from the CompactRIO device and shuts down on critical errors.

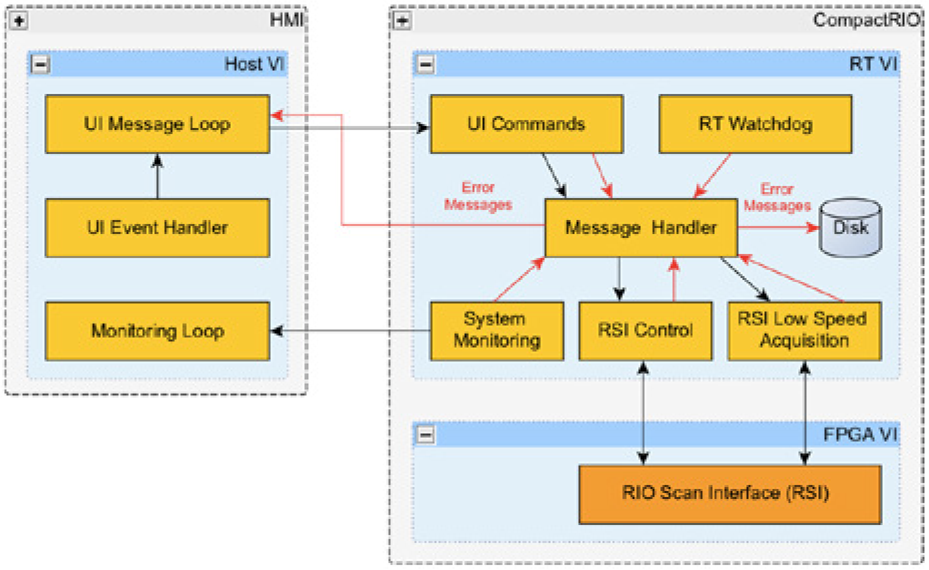


Figure 13.4. LabVIEW Real-Time Control (RIO Scan Interface) on CompactRIO Sample Project Architecture

LabVIEW Supervisory Control and Data Acquisition System Sample Project

The LabVIEW Supervisory Control and Data Acquisition System sample project demonstrates how to implement a supervisory control and data acquisition (SCADA) system with scalable architecture for building systems with many I/O points. This sample project has the following features:

- SCADA server—The SCADA server manages I/O points in the system, logs data and alarms in the historical database, and regularly archives data from the logging database to the archive database.
- Administrator console—The administrator console allows the system administrator to configure and manage the SCADA server.
- Client—The client allows the operators to monitor the system status and I/O points. The operator can view the values of the I/O points, alarms, and historical trends in different detailed views.
- Simulated CompactRIO system and simulated PLC-based system—A simulated CompactRIO system and a simulated PLC-based system are included in this sample project. The systems demonstrate how to connect to third-party devices. You can replace the simulated systems with a real CompactRIO system and a real PLC-based system.
- Scalability—You can use this sample project as a starting point to scale to a large system by adding a large number of I/O points or adding many subsystems in this sample project.

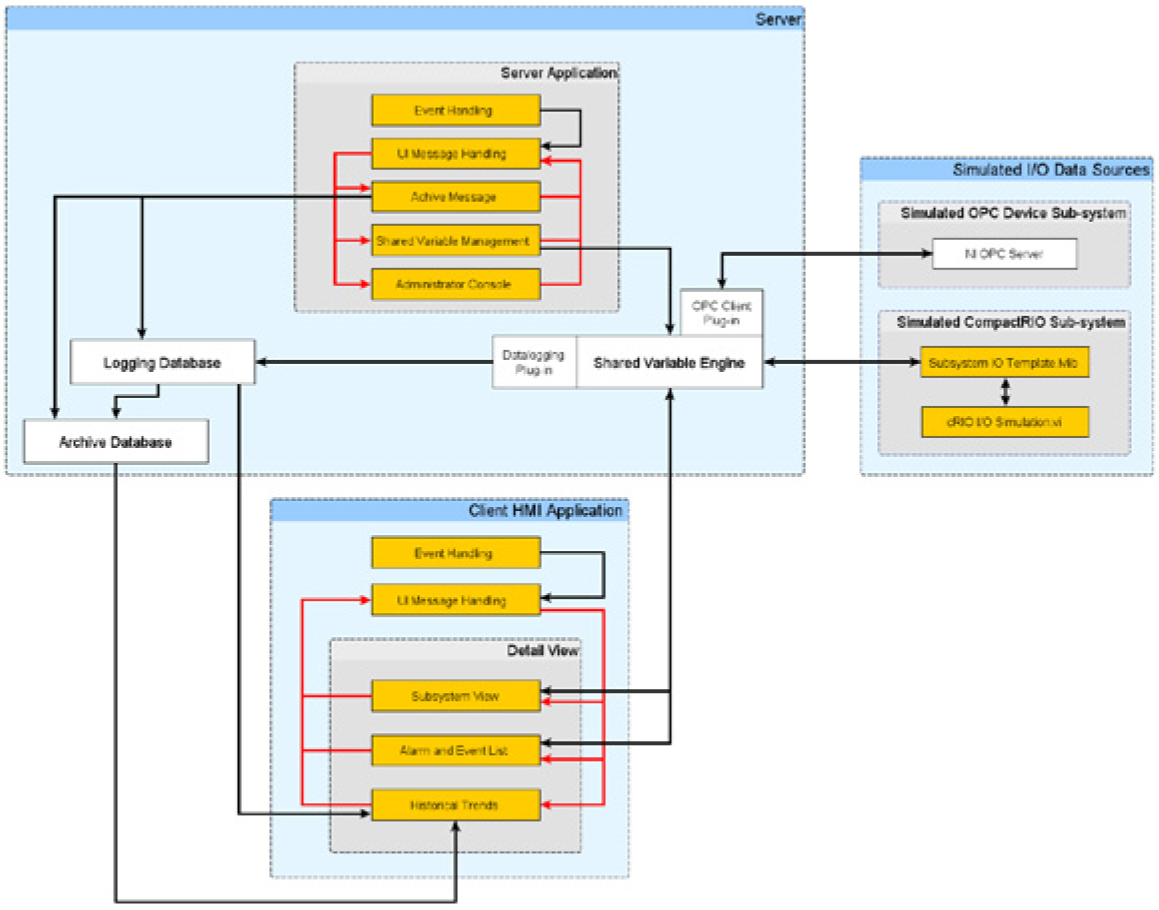


Figure 13.5. LabVIEW Supervisory Control and Data Acquisition Sample Project Architecture

Walk-Through of the LabVIEW FPGA Control on CompactRIO Sample Project

You can see from the communication diagram in Figure 13.1 that this sample project has VIs running on three different targets: the desktop PC, the CompactRIO real-time target, and the CompactRIO FPGA target. The FPGA target features two tasks running in parallel: one executes the control loop and one executes the Watchdog Loop. On the real-time target, tasks are receiving commands from the desktop, monitoring the health of the real-time system, handling messages from all the other tasks, and monitoring the Watchdog Loop. The desktop PC features tasks responsible for sending and receiving commands over the network to the real-time target, handling actions on the user interface, and updating the user interface with data from the real-time target.

Note: The LabVIEW FPGA Control on CompactRIO sample project works for all CompactRIO and NI Single-Board RIO hardware. You also can use this sample project with a real-time PXI controller and R Series device, but the System Reset node on the LabVIEW FPGA VI is not supported on R Series hardware. You can delete this node or put it inside a Disable Diagram structure.

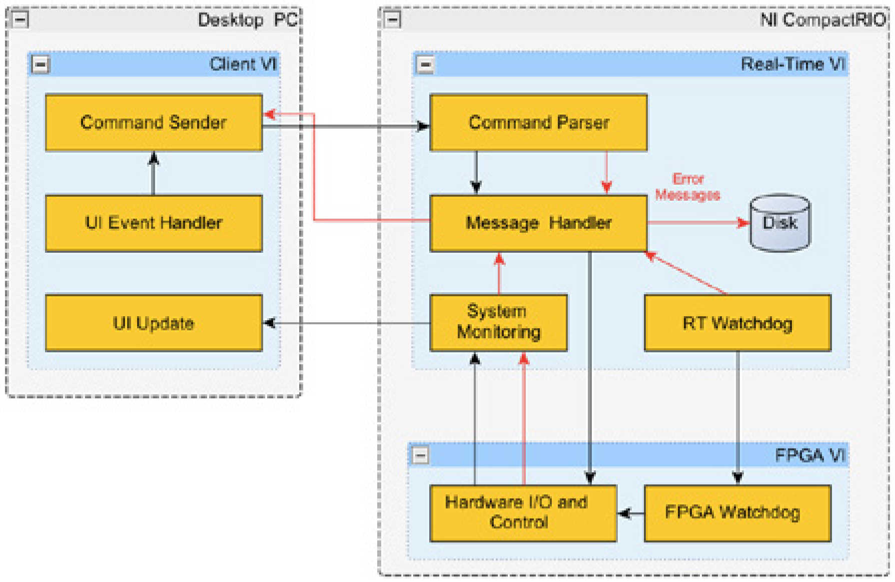


Figure 13.6. Data Communication Diagram for the LabVIEW FPGA Control on CompactRIO Sample Project

For an introduction to creating data communication diagrams for CompactRIO applications, see Chapter 1: Designing a CompactRIO Software Architecture.

Create a New Sample Project

To create your own application based on the LabVIEW FPGA Control on CompactRIO sample project, click the Create Project button in the Getting Started Window. This launches the Create Project dialog.

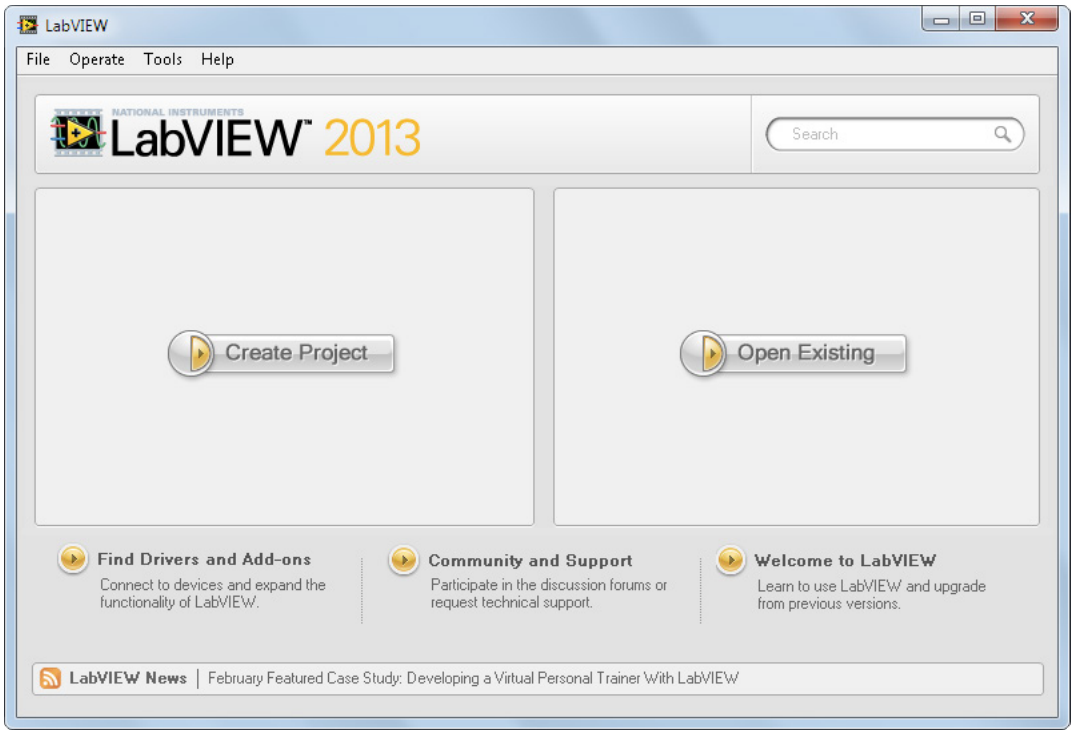
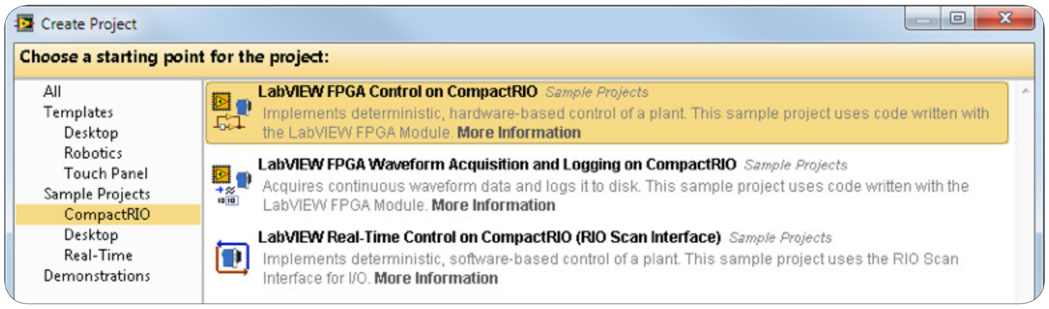

LabVIEW
File Operate Tools Help
NATIONAL INSTRUMENTS LabVIEW 2013
Search
Create Project
Open Existing
Find Drivers and Add-ons
Connect to devices and expand the functionality of LabVIEW.
Community and Support
Participate in the discussion forums or request technical support.
Welcome to LabVIEW
Learn to use LabVIEW and upgrade from previous versions.
LabVIEW News | February Featured Case Study: Developing a Virtual Personal Trainer With LabVIEW

Figure 13.7. Create a sample project by clicking the Create Project button.

If you choose the CompactRIO filter under the Sample Projects heading, you can see the LabVIEW FPGA Control on CompactRIO sample project. Follow the More Information link to launch the documentation for this sample project.



LabVIEW FPGA Control on CompactRIO *Sample Projects*
Implements deterministic, hardware-based control of a plant. This sample project uses code written with the LabVIEW FPGA Module. **More Information**

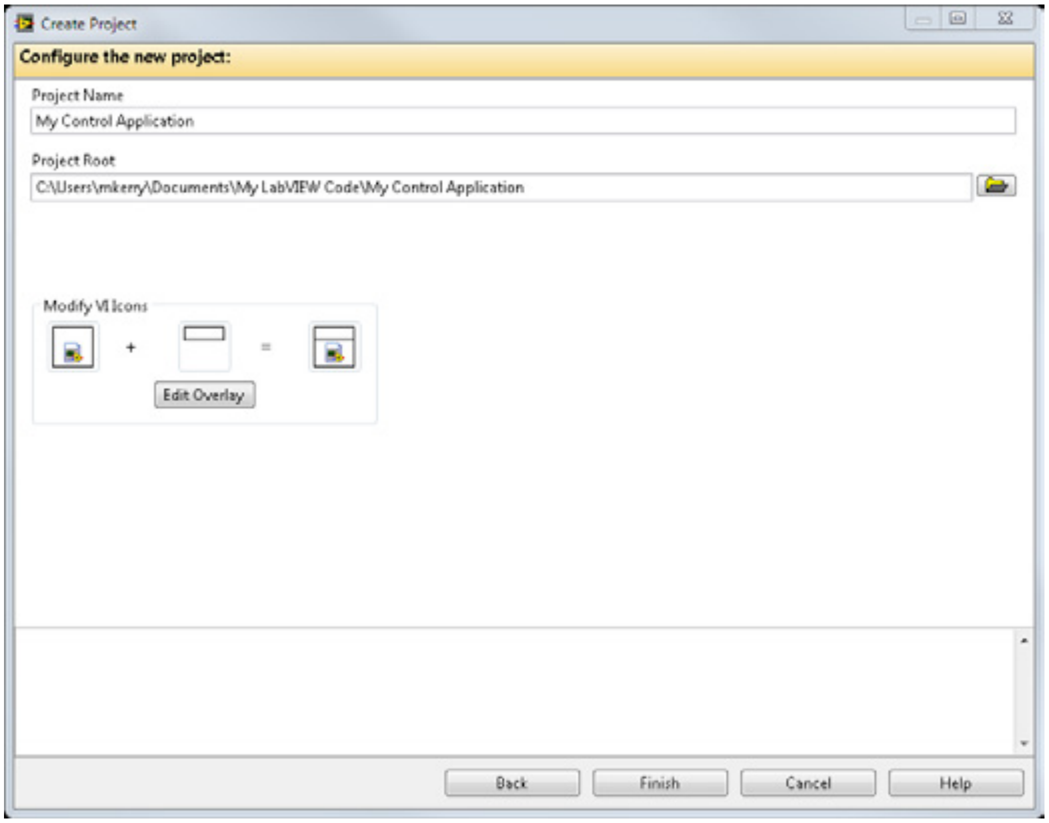
LabVIEW FPGA Waveform Acquisition and Logging on CompactRIO *Sample Projects*
Acquires continuous waveform data and logs it to disk. This sample project uses code written with the LabVIEW FPGA Module. **More Information**

LabVIEW Real-Time Control on CompactRIO (RIO Scan Interface) *Sample Projects*
Implements deterministic, software-based control of a plant. This sample project uses the RIO Scan Interface for I/O. **More Information**

Figure 13.8. Filter Sample Projects by CompactRIO

Before opening the documentation, start the process of creating your own copy of this project by clicking Next in the Create Project dialog.

You can see several options for customizing your own copy of the LabVIEW FPGA Control on CompactRIO sample project. In this case, change "Project Name" to "My Control Application." Specify a new folder titled My Control Application within your preferred directory. Leave all the other options as default. Click the Finish button to start copying the project. While the project is being made, examine the documentation.



Project Name
My Control Application

Project Root
C:\Users\mkerry\Documents\My LabVIEW Code\My Control Application

Figure 13.9. Configure your sample project.

Review Documentation

You can see in the overview of this documentation that the sample project is based on the Simple State Machine and Queued Message Handler project templates. You can also see a list of features, including the high-performance and hardware-based control of this application, because the control loop is running on the FPGA target.

If you scroll down, you can see the system requirements for this sample project, the required software and hardware, and an overview of all the parallel tasks running across the execution targets in this application.

Further down, a diagram illustrates what happens when you change a control setpoint on the user interface of this application. You can see that a message is sent over the network to the real-time target containing the new setpoints, and the real-time target sends those setpoints via read/write controls to the FPGA target. Explore how this works later when you open up the VIs.

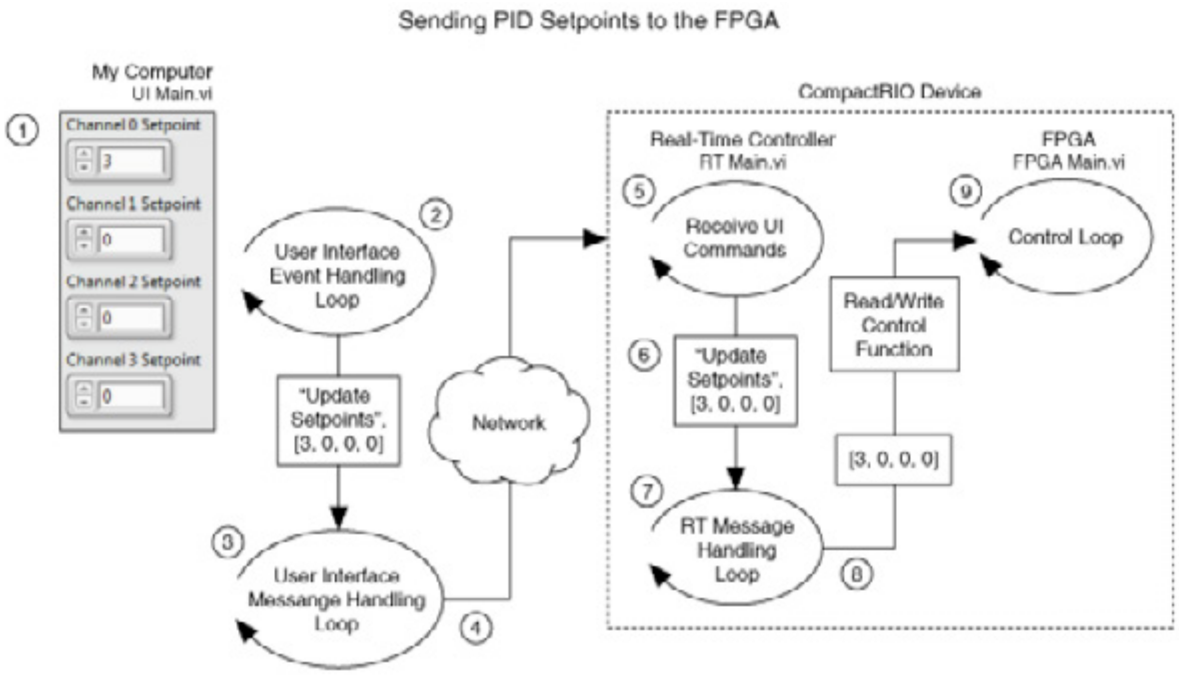


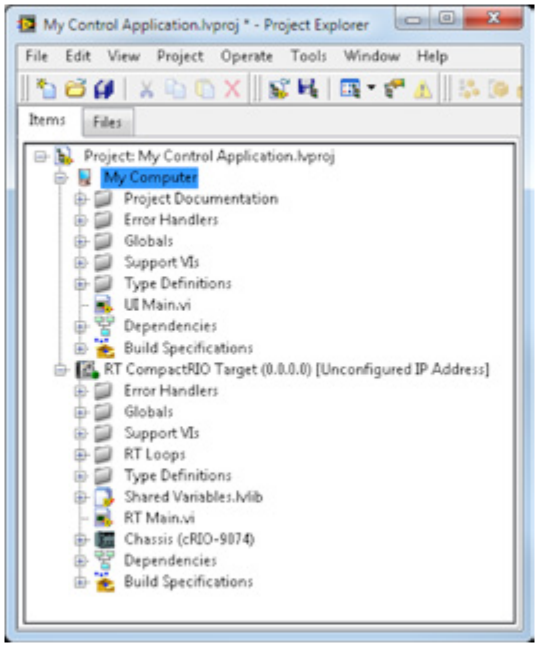
Figure 13.10. Review sample project documentation.

The documentation also contains instructions on how to modify the sample project to include your own hardware under "Modifying this Sample Project—Adapting the Sample Project to Your Hardware."

Now look at the completed project.

Review LabVIEW FPGA Control on CompactRIO Architecture

You can see you now have your own copy of the LabVIEW FPGA Control on CompactRIO application called My Control Application. The project features three execution targets: My Computer, the real-time CompactRIO target, and the FPGA target. For more documentation about the project, the documentation discussed previously is included in the project under the Project Documentation folder.



Project: My Control Application.lvproj
- My Computer
  - Project Documentation
  - Error Handlers
  - Globals
  - Support VIs
  - Type Definitions
  - UI Main.vi
  - Dependencies
  - Build Specifications
- RT CompactRIO Target (0.0.0.0) [Unconfigured IP Address]
  - Error Handlers
  - Globals
  - Support VIs
  - RT Loops
  - Type Definitions
  - Shared Variables.lvlib
  - RT Main.vi
  - Chassis (cRIO-9074)
  - Dependencies
  - Build Specifications

Figure 13.11. LabVIEW FPGA Control Sample Project

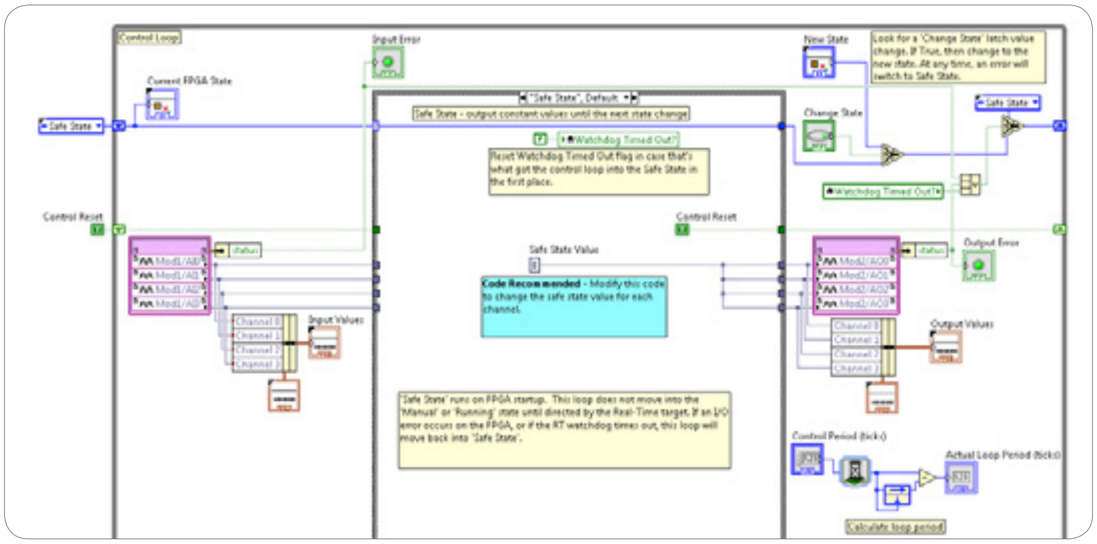
FPGA Main VI

The front panel of the FPGA Main VI features many controls and indicators. The Real-Time Main VI uses read/write controls to set and get the values of these controls and indicators. Now look at the block diagram.

Note: This FPGA VI was written to be loaded to flash memory using the NI RIO Device Setup utility and configured to reload the FPGA bitfile only upon cycling the power. After reviewing the LabVIEW FPGA code, see instructions for properly configuring the FPGA bitfile in the section "Downloading the FPGA Bitfile to Flash Memory."

Control Loop

Right away, you see multiple blue labels indicating places where you are required to add your own code for your specific application. You can see the control loop is a state machine that is initialized in the safe state. The only way for the control loop to move out of the safe state is by receiving a command from the Real-Time Main VI. While in the safe state, a safe state value is sent to all of the output controls for your control algorithm.



Reset Watchdog Timed Out flag in case that's what got the control loop into the Safe State in the first place.

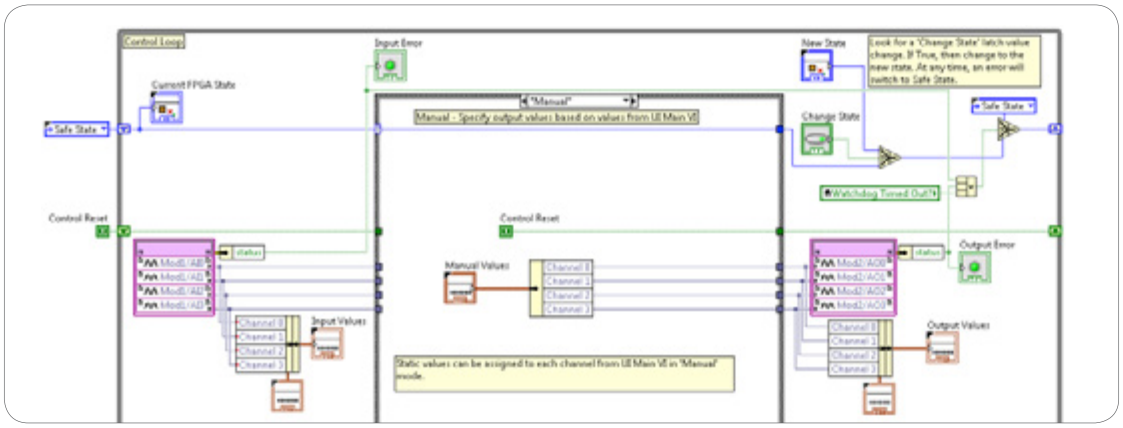
Code Recommended - Modify this code to change the safe state value for each channel.

'Safe State' runs on FPGA startup. This loop does not move into the 'Manual' or 'Running' state until directed by the Real-Time target. If an I/O error occurs on the FPGA, or if the RT watchdog times out, this loop will move back into 'Safe State'.

Look for a 'Change State' latch value change. If True, then change to the new state. At any time, an error will switch to Safe State.

Figure 13.12. The FPGA Main VI starts up in a safe state.

If the user moves the control algorithm into the Manual state on the user interface, then the Manual state runs. In this state, users can specify which manual values they wish to write to each of the output channels.

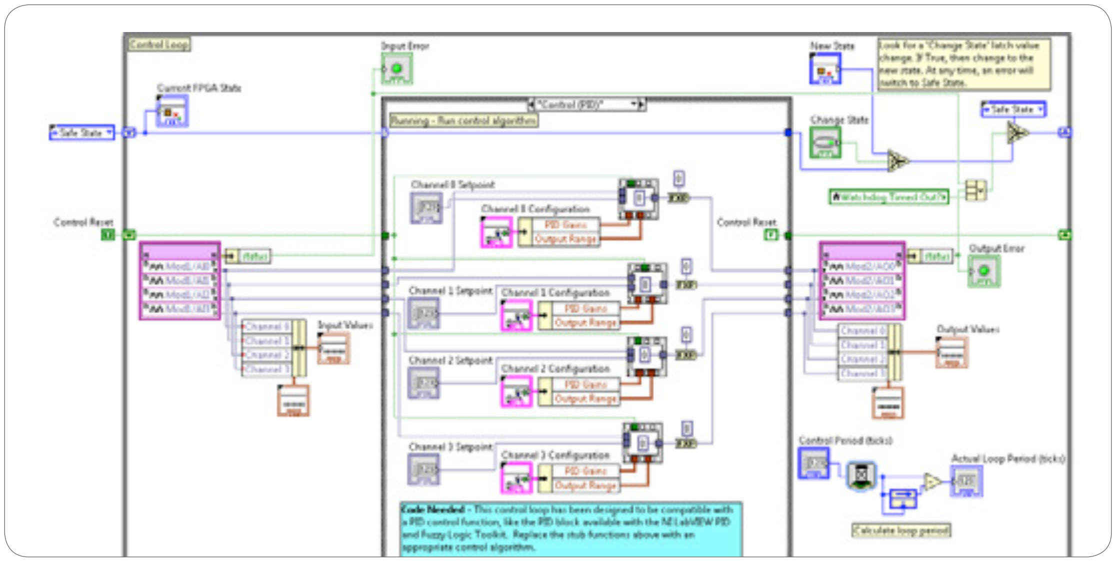


Static values can be assigned to each channel from UI Main VI in 'Manual' mode.

Look for a 'Change State' latch value change. If True, then change to the new state. At any time, an error will switch to Safe State.

Figure 13.13. Move into Manual state to manually control the hardware outputs.

If users run this application in the Control state (see Figure 13.14), then they are required to add a function, like the PID block that is available with the LabVIEW PID and Fuzzy Logic Toolkit, to actually perform the control algorithm because this code has no control function. The PID block installs with the LabVIEW Real-Time Module, and can be easily dropped into this VI. You can also create your own custom control algorithm.



Look for a "Change State" latch value change. If True, then change to the new state. At any time, an error will switch to Safe State.

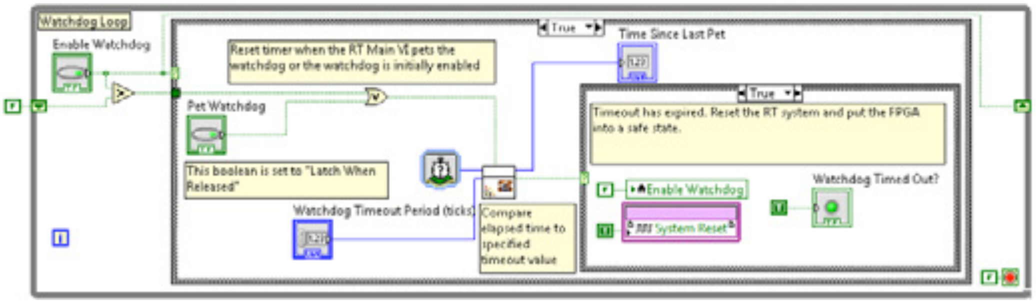
Code Needed - This control loop has been designed to be compatible with a PID control function, like the PID block available with the NI LabVIEW PID and Fuzzy Logic Toolkit. Replace the stub functions above with an appropriate control algorithm.

Figure 13.14. Move into the Control state to execute PID control on all four channels.

Watchdog Loop

In addition to the control loop, you can use the Watchdog Loop. This loop, once enabled by the RT Main VI, periodically "pets" the watchdog and resets the system if a timeout occurs. The timeout occurs when the real-time Watchdog Loop does not execute within a timeout period defined by the user. The Watchdog Loop runs on the FPGA VI instead of the real-time target because this code allows the FPGA to switch into a safe state and write known values to the outputs before resetting the system. For more information on watchdog timers, see the "Ensuring Reliability With Watchdog Timers" section of Chapter 3: Designing a LabVIEW Real-Time Application.

More information on configuring the watchdog timer is included in the real-time Watchdog Loop discussion later in the chapter.



Reset timer when the RT Main VI pets the watchdog or the watchdog is initially enabled

This boolean is set to "Latch When Released"

Compare elapsed time to specified timeout value

Timeout has expired. Reset the RT system and put the FPGA into a safe state.

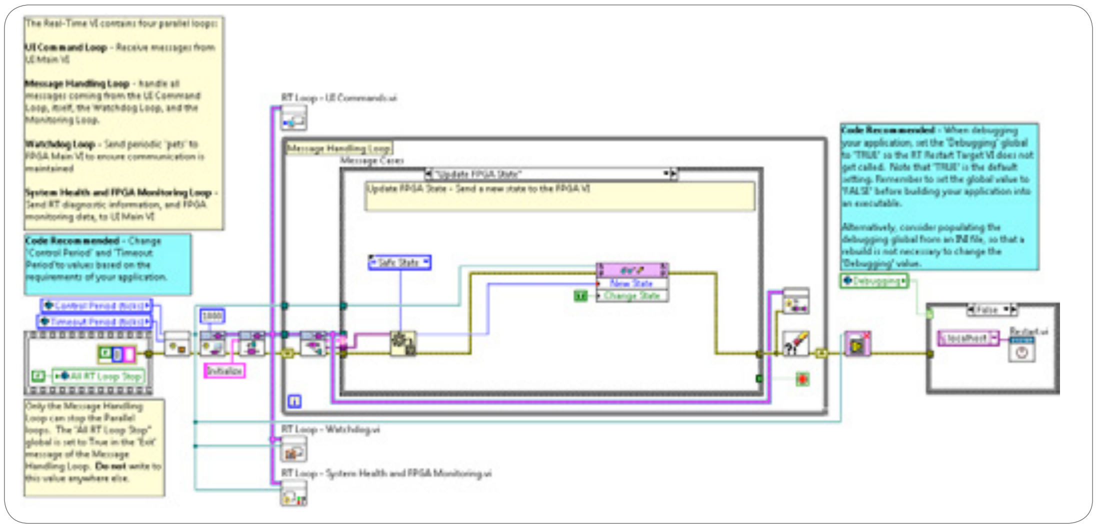
Figure 13.15. The watchdog timer is implemented in LabVIEW FPGA.

Note: If the FPGA bitfile is loaded to flash and set to reload only upon cycling the power, then when you call the System Reset node, the real-time controller reboots, but the FPGA bitfile continues to run. This ensures that the FPGA hardware outputs immediately go into a safe state and stay there until the real-time controller is back online. After reviewing the LabVIEW FPGA code, see instructions for properly configuring the FPGA bitfile in the section "Downloading the FPGA Bitfile to Flash Memory."

If you choose to not download the FPGA bitfile to flash memory, or if you choose to download it to flash but with the option to "autoload upon a device reboot," then you need to add some logic to this TRUE case to ensure that the hardware outputs go into a safe state prior to calling the System Reset node. Otherwise, a race condition could occur and you could end up resetting the FPGA prior to putting the hardware outputs into a safe state.

RT Main VI

Now look at the RT Main VI, which features several loops running in parallel. A comment on the diagram explains the functionality of each of the parallel loops. This VI opens with a broken run arrow since the FPGA VI is not compiled.


The Real-Time VI contains four parallel loops:

UI Command Loop - Receive messages from UI Main VI

Message Handling Loop - handle all messages coming from the UI Command Loop, itself, the Watchdog Loop, and the Monitoring Loop.

Watchdog Loop - Send periodic 'pets' to FPGA Main VI to ensure communication is maintained

System Health and FPGA Monitoring Loop - Send RT diagnostic information, and FPGA monitoring data, to UI Main VI

Code Recommended - Change 'Control Period' and 'Timeout Period' to values based on the requirements of your application.

Only the Message Handling Loop can stop the Parallel loops. The "All RT Loop Stop" global is set to True in the 'Exit' message of the Message Handling Loop. **Do not** write to this value anywhere else.

Update FPGA State - Send a new state to the FPGA VI

Code Recommended - When debugging your application, set the 'Debugging' global to "TRUE" so the RT Restart Target VI does not get called. Note that "TRUE" is the default setting. Remember to set the global value to "FALSE" before building your application into an executable.

Alternatively, consider populating the debugging global from an INI file, so that a rebuild is not necessary to change the 'Debugging' value.

Figure 13.16. The RT Main VI features four loops running in parallel.

UI Commands Loop

The top loop is the UI Commands VI. Open this VI.

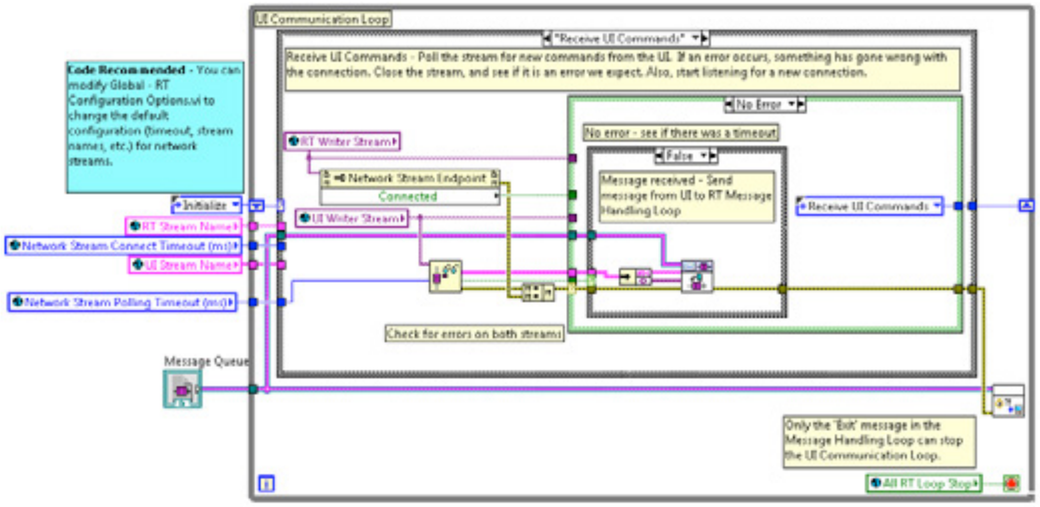
Figure 13.17 The UI Commands Loop

This VI is based on the Simple State Machine project template that is available with LabVIEW. You can see that this VI is using Network Streams to communicate with the user interface VI under My Computer. The main purpose of this loop is to receive UI commands and manage the connection of any Network Streams that are used within the application. This sample project uses two streams:

- UI Writer Stream—RT application receives commands from the UI
- RT Writer Stream—RT application sends error messages to the UI for display

Network Streams are used for this type of data transfer because when sending commands from a user interface to an embedded system, you need a reliable data communication mechanism. Network Streams were designed and optimized for lossless, high-throughput data communication. They feature enhanced connection management that automatically restores network connectivity if a disconnection occurs due to a network outage or other system failure. Streams use a buffered, lossless communication strategy that ensures data written to the stream is never lost, even in environments that have intermittent network connectivity.

The VI initializes by going into the Listen for UI Writer state and waits until it makes a connection with the UI, at which point it moves into the Receive UI Commands state.



Code Recommended - You can modify Global - RT Configuration Options.vi to change the default configuration (timeout, stream names, etc.) for network streams.

Receive UI Commands - Poll the stream for new commands from the UI. If an error occurs, something has gone wrong with the connection. Close the stream, and see if it is an error we expect. Also, start listening for a new connection.

No error - see if there was a timeout

Message received - Send message from UI to RT Message Handling Loop

Check for errors on both streams

Only the 'Exit' message in the Message Handling Loop can stop the UI Communication Loop.

Figure 13.18. The UI Commands loop receives UI commands and manages the network connection of Network Streams.

Upon receiving an error in both the Listen for UI Writer and Receive UI Commands cases, destroy the Network Stream and attempt to create a new one. This allows the client to disconnect and reconnect at any time while the RT application continues to run. When the client disconnects, the reference is no longer valid, so you need to create a new one.

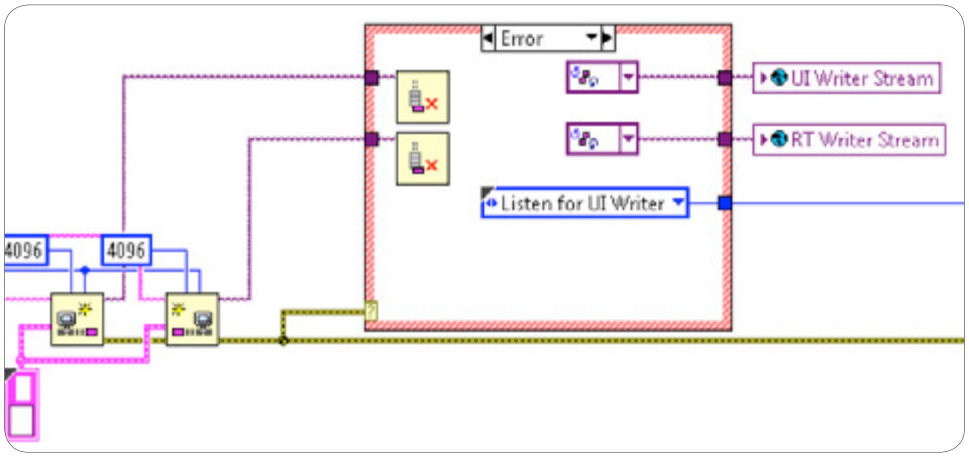


Figure 13.19. The UI Commands Loop is designed to allow the client to disconnect and reconnect at anytime.

You write data received from the Network Stream into the Enqueue Message VI shown in Figure 13.20. The Enqueue Message VI passes all messages or commands to the Message Handling Loop shown in Figure 13.21. This VI is borrowed from the Queued Message Handler Template available for the LabVIEW Core software option. The Message Queue API is basically a wrapper around Queue functions with usability tweaks for communicating messages between two or more loops. You can double-click the Enqueue Message VI to see how it works. This function forces the user to send messages as a cluster consisting of a string (the message or command) and a variant (any associated data). For example, the command might be "Change Setpoint" and the associated data might be "27 °C." This is a scalable and efficient method for communicating messages within a large application. You use the variant data type so that you can use the same API to send multiple data types.

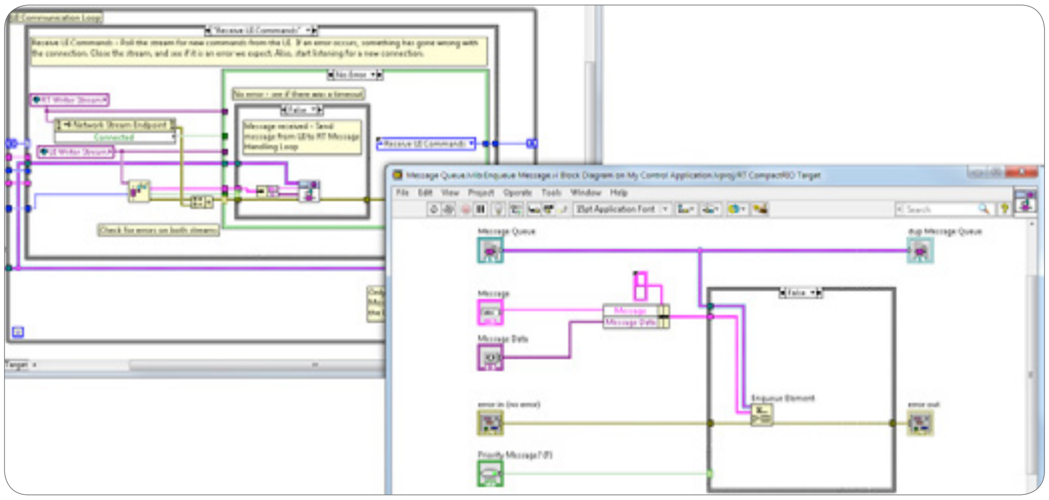


Figure 13.20. The Message Queue API is a wrapper around Queue functions.

The Message Handling Loop is part of the RT Main VI. The RT Main VI was built from the Queued Message Handler project template and thus uses a Queued Message Handler architecture for receiving and sending messages. The RT Main VI is set up to handle different messages like updating the state on the FPGA, handling errors, and sending data to the UI.

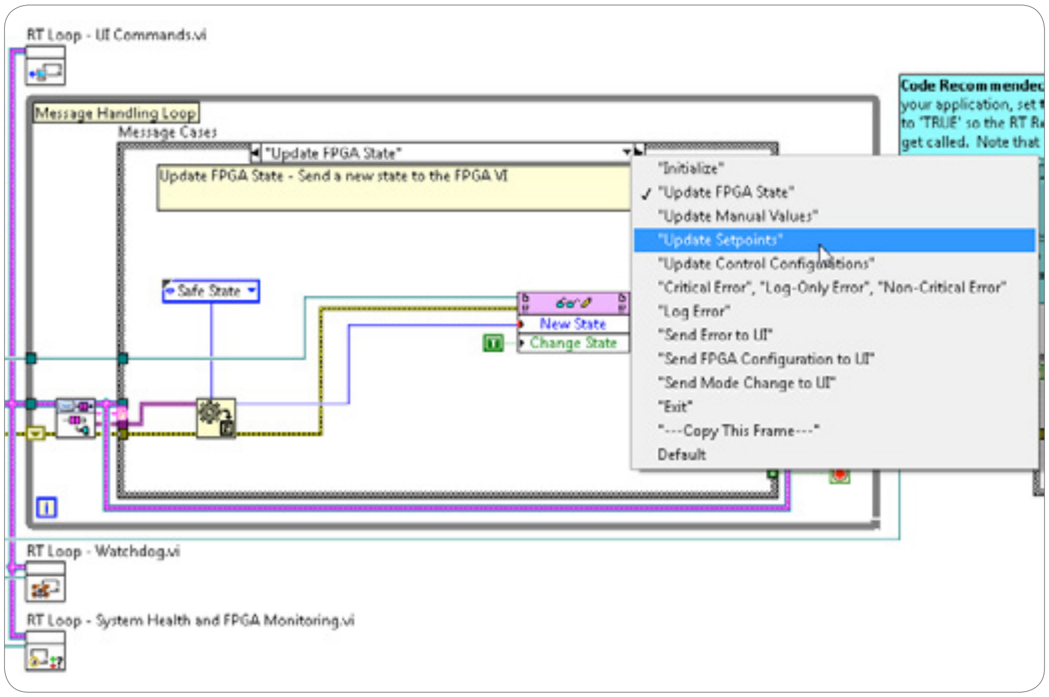
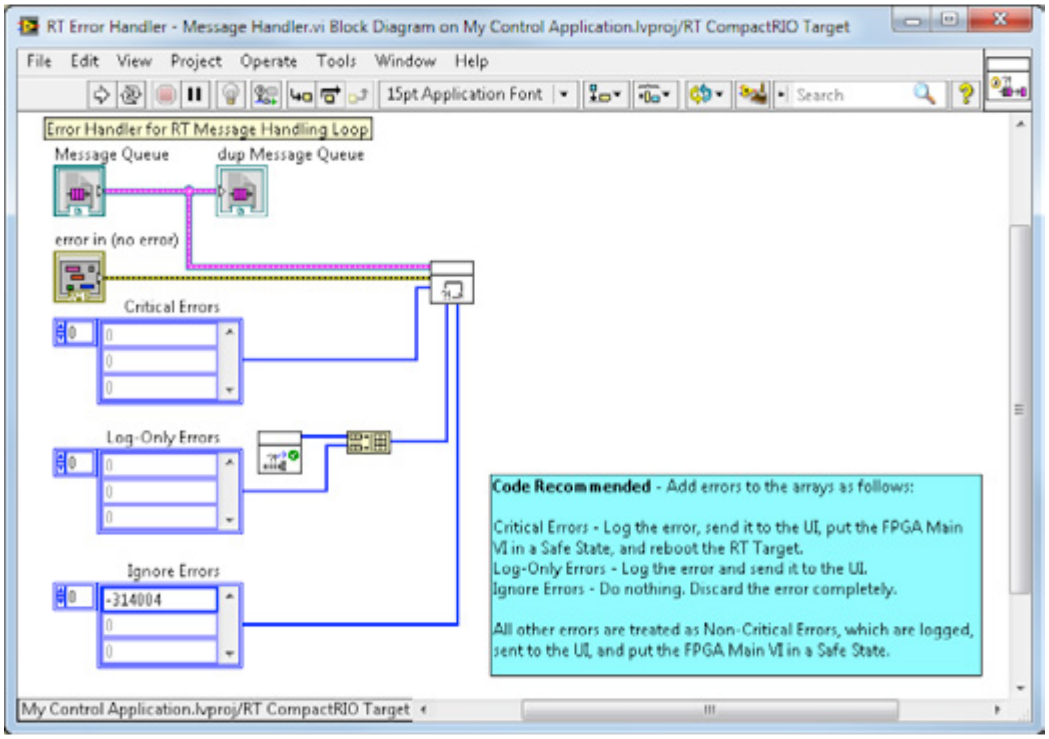


Figure 13.21. The Message Handling Loop receives messages such as Update Setpoints and Send Error to UI.

Note that every parallel loop on the RT Main VI has its own unique error handler VI, shown in Figure 13.22.


RT Error Handler - Message Handler.vi Block Diagram on My Control Application.lvproj/RT CompactRIO Target
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
Error Handler for RT Message Handling Loop
Message Queue
dup Message Queue
error in (no error)
Critical Errors
Log-Only Errors
Ignore Errors
-314004
My Control Application.lvproj/RT CompactRIO Target

Code Recommended - Add errors to the arrays as follows:

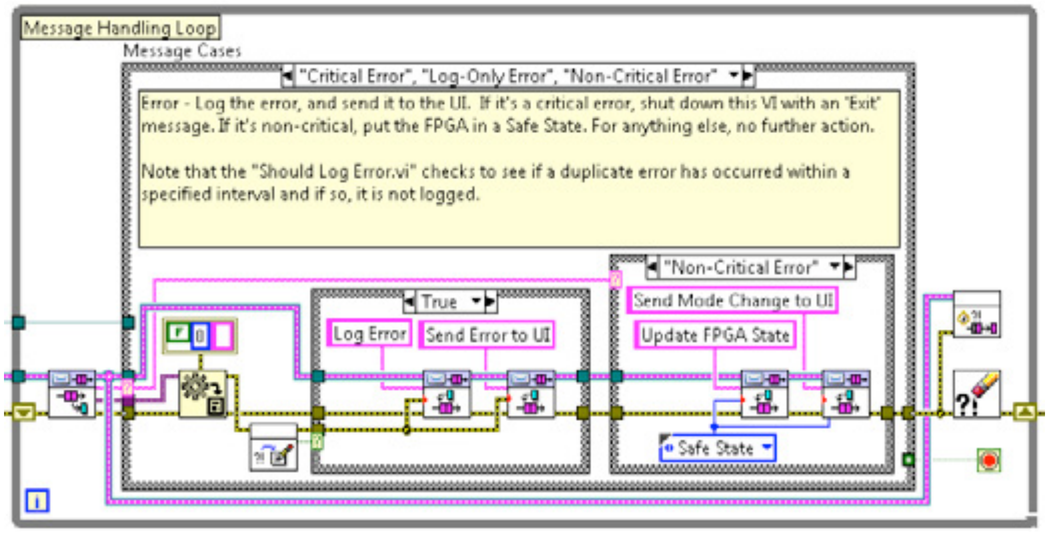
Critical Errors - Log the error, send it to the UI, put the FPGA Main VI in a Safe State, and reboot the RT Target.
Log-Only Errors - Log the error and send it to the UI.
Ignore Errors - Do nothing. Discard the error completely.

All other errors are treated as Non-Critical Errors, which are logged, sent to the UI, and put the FPGA Main VI in a Safe State.

Figure 13.22. Each loop has its own error handler subVI.

By using its own error handler, each loop can specify unique errors that the particular loop wants to handle in a certain way. Any error message that is generated within the real-time application can be classified by the user as critical, log-only, and ignore. Any error that isn't classified by the user is automatically classified as non-critical. All error messages are sent to the Message Handling Loop and are processed within the "Critical Error," "Log-Only Error," "Non-Critical Error" case, which is shown in Figure 13.23. Note that a feature of the Queued Message Handler is that in addition to receiving messages from the producer loop (UI Command Loop), the consumer loop can send messages to itself and make decisions on which state executes next.

In Figure 13.23, all error messages are logged to the local drive on the real-time controller and sent to the UI for display. In addition, upon a non-critical error, the FPGA goes into a safe state. Send a message to the UI notifying the user that the FPGA state has changed. Upon a critical error, exit the application.



Error - Log the error, and send it to the UI. If it's a critical error, shut down this VI with an "Exit" message. If it's non-critical, put the FPGA in a Safe State. For anything else, no further action.

Note that the "Should Log Error.vi" checks to see if a duplicate error has occurred within a specified interval and if so, it is not logged.

Figure 13.23. All error messages are processed in the Message Handling Loop.

Watchdog Loop

The next VI on the RT Main VI is the Watchdog Loop.

Figure 13.24. Watchdog Loop

The only purpose of this VI is to use a read/write control to periodically pet the watchdog in the Watchdog Loop on the FPGA that you already looked at.

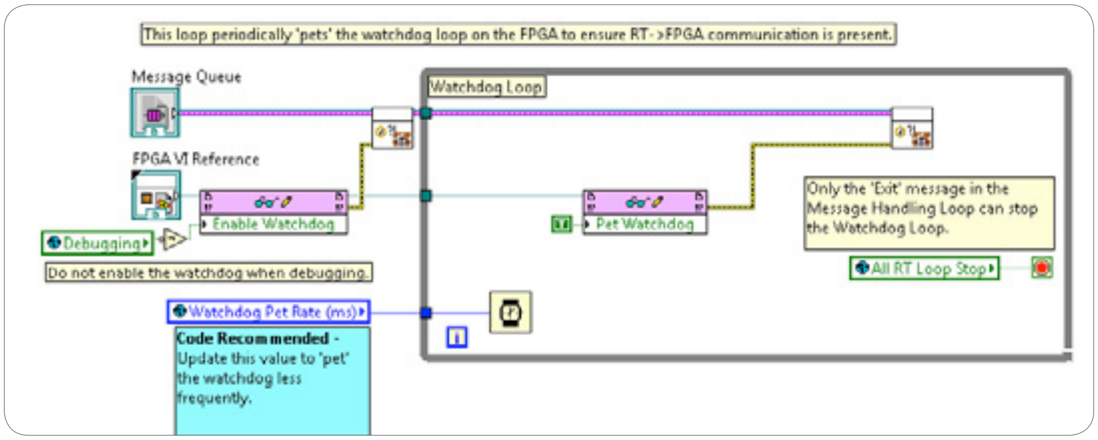
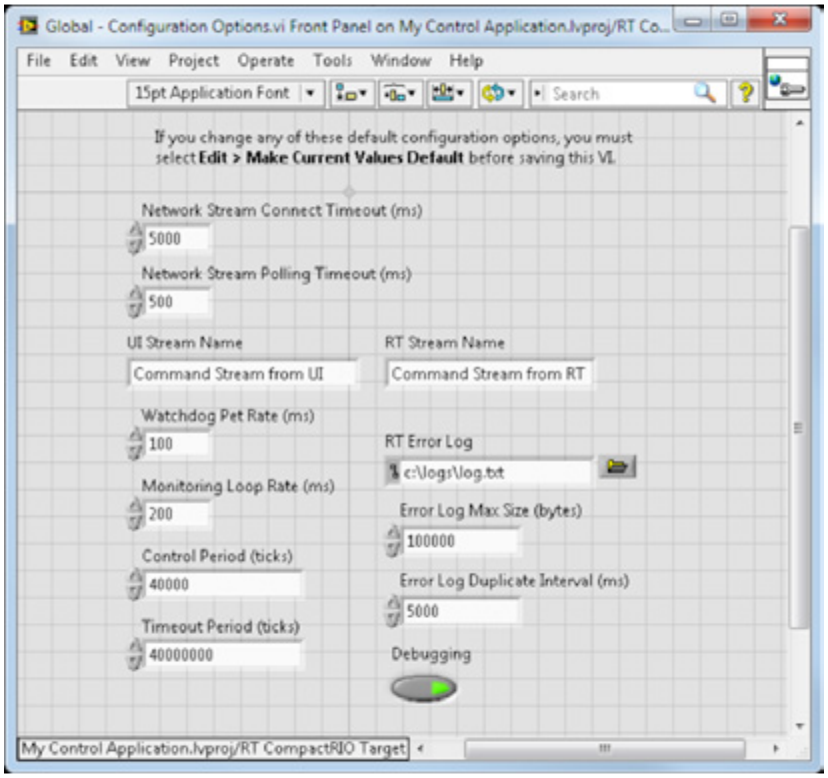

This loop periodically 'pets' the watchdog loop on the FPGA to ensure RT->FPGA communication is present.
Message Queue
FPGA VI Reference
Enable Watchdog
Debugging
Do not enable the watchdog when debugging.
Watchdog Pet Rate (ms)
Code Recommended - Update this value to 'pet' the watchdog less frequently.
Watchdog Loop
Pet Watchdog
Only the 'Exit' message in the Message Handling Loop can stop the Watchdog Loop.
All RT Loop Stop

Figure 13.25. The RT Watchdog Loop periodically pets the watchdog loop on the FPGA to ensure that RT to FPGA communication is present.

Several global variables are referenced throughout the RT Main VI. The RT Main VI includes one global variable that stores all configuration options titled Global-Configuration Options.vi. This global variable is located in the LabVIEW project under the **RT Target»Globals** folder and is shown in Figure 13.26. This is where users can specify whether they are in debugging mode, in addition to timeouts and loop rates.



Global - Configuration Options.vi Front Panel on My Control Application.lvproj/RT Co...

If you change any of these default configuration options, you must select **Edit > Make Current Values Default** before saving this VI.

Network Stream Connect Timeout (ms): 5000
Network Stream Polling Timeout (ms): 500
UI Stream Name: Command Stream from UI
RT Stream Name: Command Stream from RT
Watchdog Pet Rate (ms): 100
RT Error Log: c:\logs\log.txt
Monitoring Loop Rate (ms): 200
Error Log Max Size (bytes): 100000
Control Period (ticks): 40000
Error Log Duplicate Interval (ms): 5000
Timeout Period (ticks): 40000000
Debugging

My Control Application.lvproj/RT CompactRIO Target

Figure 13.26. Configuration options used within the RT Main VI are stored as a global variable.

The watchdog timeout by default is set to 40,000,000 ticks of the FPGA clock or 1 second. This means that if the Watchdog Loop fails to execute within 1 second on a given iteration, a watchdog timeout occurs and the FPGA goes into a safe state. Keep in mind that a real-time OS is extremely reliable and the only reason a loop might not finish on time is typically because of CPU starvation—your system runs out of memory and the software is hanging or is about to crash. You can ensure that the CPU bandwidth stays low and that the largest block of contiguous memory does not decrease by monitoring them in the System Health and FPGA Monitoring Loop.

You should disable the watchdog timer in debugging mode since debugging tools such as highlight execution cause the application to run much slower.

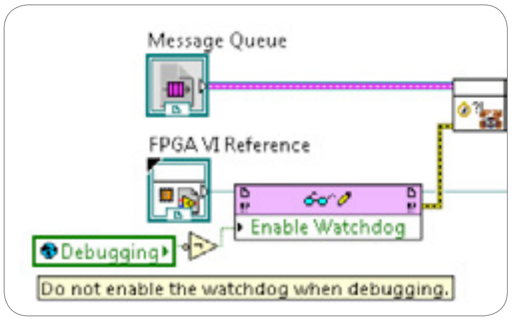


Figure 13.27. The Watchdog Loop checks to see if debugging mode is enabled before executing the watchdog.

System Health and FPGA Monitoring Loop

The last loop in RT Main VI is the System Health and FPGA Monitoring Loop.

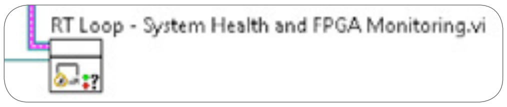
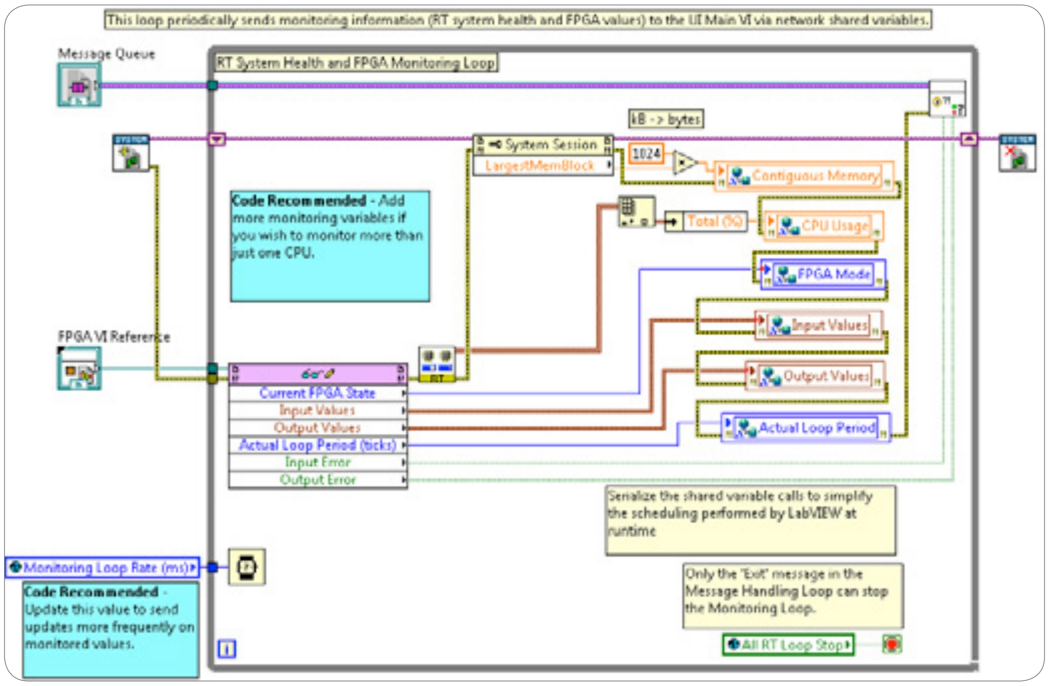


Figure 13.28. System Health and FPGA Monitoring Loop

This loop uses network-published shared variables to send monitoring information to the UI Main VI. You monitor input values (analog input) and output values (analog output) from the FPGA VI by using read/write controls and monitor some real-time system data, such as largest contiguous memory block and CPU usage percentage.


This loop periodically sends monitoring information (RT system health and FPGA values) to the UI Main VI via network shared variables.

Code Recommended - Add more monitoring variables if you wish to monitor more than just one CPU.

Serialize the shared variable calls to simplify the scheduling performed by LabVIEW at runtime

Only the "Exit" message in the Message Handling Loop can stop the Monitoring Loop.

Code Recommended - Update this value to send updates more frequently on monitored values.

Figure 13.29. The System Health and FPGA Monitoring Loop periodically sends information to the user interface.

You need to keep the CPU usage percentage below 80 percent. If the CPU usage reaches 100 percent, you probably have loops that are running slower than your specified rate. If a high-priority Timed Loop is active, the other loops may not be executing at all. If you find that the CPU usage is higher than 80 percent, you should review the LabVIEW Real-Time Help document Minimize CPU Usage for tips and tricks on how to reduce it.

You also should closely monitor the largest block of contiguous memory. The largest block of contiguous memory should be fairly constant because LabVIEW allocates and releases memory throughout the execution of the RT Main VI. If you see that the contiguous memory falls below a certain limit, go into a safe state and restart the system. When you restart the real-time controller, the memory is defragmented. Often an embedded software application crashes not because it ran out of memory but because the available memory is very fragmented. This is caused by dynamic memory allocations. Some applications cannot be restarted while they are deployed out in the field, and for these applications, the developer should spend extra time removing a majority if not all of the dynamic memory allocations. Some common sources of dynamic memory allocation include Queues without a fixed-sized buffer, variable-sized arrays (and waveforms), variable-sized strings, and variants.

Finally, note that any input or output errors from the LabVIEW FPGA VI are also handled in this loop. If an error occurs on either the analog input node or the analog output node in the LabVIEW FPGA VI, a custom error message is sent to the UI for display.

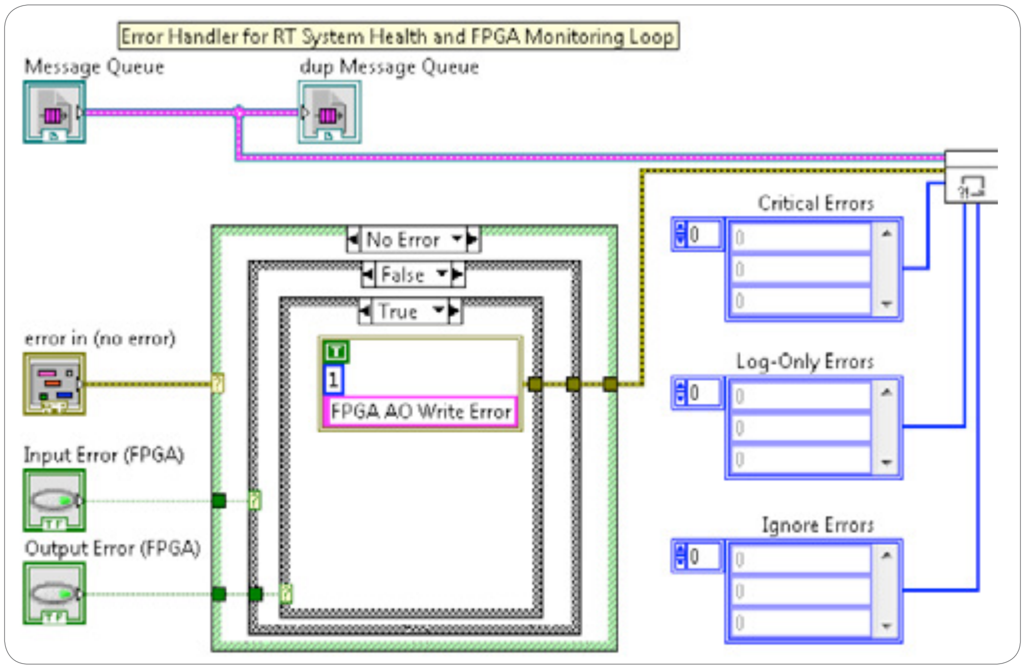


Figure 13.30. A custom error message is sent to the UI if an error occurs on the LabVIEW FPGA I/O nodes.

Stopping the Loops in the RT Main VI

The only way to stop the RT Main VI from running is to execute the Exit case within the RT Message Handling Loop. This case executes only if the user presses the Exit button on the UI Main VI, or if a critical error occurs within the RT Main VI. If either of these conditions occurs, the Exit case executes, which writes TRUE to the All RT Loop Stop global variable.

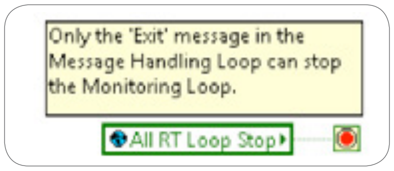

Only the 'Exit' message in the Message Handling Loop can stop the Monitoring Loop.
All RT Loop Stop

Figure 13.31. The RT Main VI stops if the user presses the Exit button on UI Main VI or if a critical error occurs within the RT Main VI.

UI Main VI

Open the UI Main VI. This is the user interface that your user interacts with when operating the LabVIEW FPGA Control on CompactRIO application. In the upper-left corner of the user interface, you see the Connection Settings. When this application first runs, these are the only settings that the user can change. All other controls are disabled.

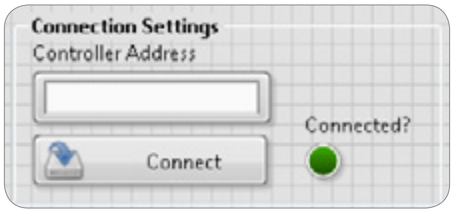


Figure 13.32. Connect to a CompactRIO device within the Connection Settings portion of the user interface.

The user can now change the IP address of the controller and choose to connect to that IP address. Once connected, the rest of the user interface enables.

If you want to run the FPGA control algorithm in the Control state, you can click the Run Control button.

Once the control is running, you can set setpoint values, which are sent to the target automatically. Note that the setpoints automatically coerce to the range of the input module being used for the PID control task.

You typically want to tweak the Control Configuration settings while the control loop is running. To send the updated Control Configuration settings to the target, you can click the Update Control Configurations button.

On the Manual tab, you can set the values of all the FPGA outputs directly. When you click the Run Manual button, the FPGA Main VI switches into the Manual state.

The Data Monitoring tab features the raw channel values of the inputs and the outputs on the FPGA.

Finally, the System Monitoring tab shows monitoring information like CPU usage, largest continuous memory, and the loop period of the VI running on the FPGA. If any errors occur while running this application, they appear in the System Status string on the front panel.

Now look at the block diagram. The UI Main VI has three parallel loops.

Event Handling Loop

The top loop is the Event Handling Loop, which sends messages to the UI Message Loop when any value change occurs on the front panel. By giving the Event Handling task its own loop separate from the UI Message Loop, you can ensure that the user interface is responsive.

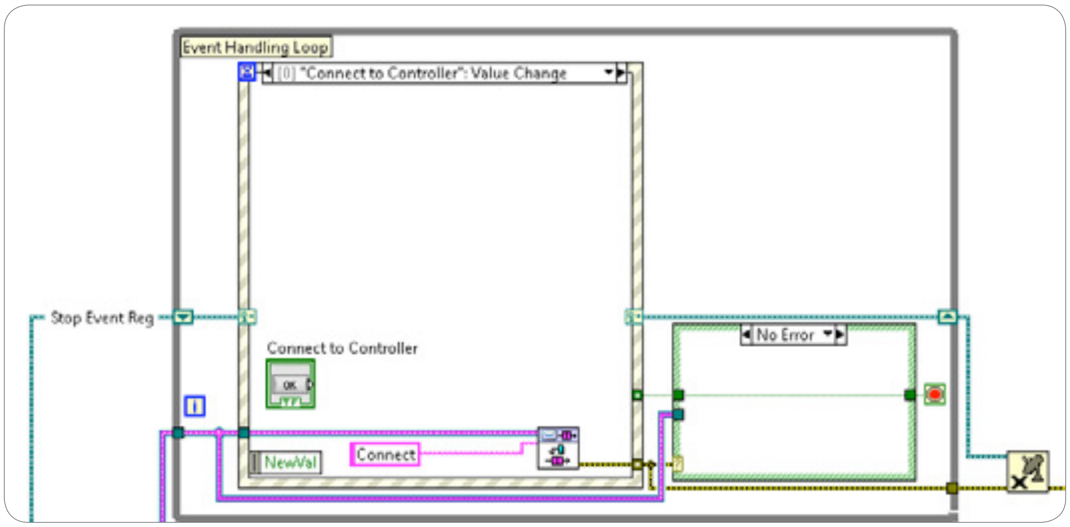


Figure 13.33. Event Handling Loop

Note that this loop is registered for the user event Stop. You use the user event so the Message Handling Loop is in charge of shutting everything down. If an error (as opposed to a UI event) is what causes the UI Main VI to exit, you need a way for the Message Handling Loop to tell the Event Handling Loop to stop. The user event is the NI-recommended way.

When designing applications involving a UI, ensure that the loop handling front panel events is the last one to shut down. In this sample project, when you press the Exit button, you immediately send an Exit command down to the UI Message Loop, as shown in Figure 13.34. Once the Event Handling Loop has received an acknowledgement that the UI Message Loop has stopped (or is executing the Exit case), stop the Event Handling Loop. You send this acknowledgment by sending a user event called Stop from the UI Message Loop to the Event Handling Loop.

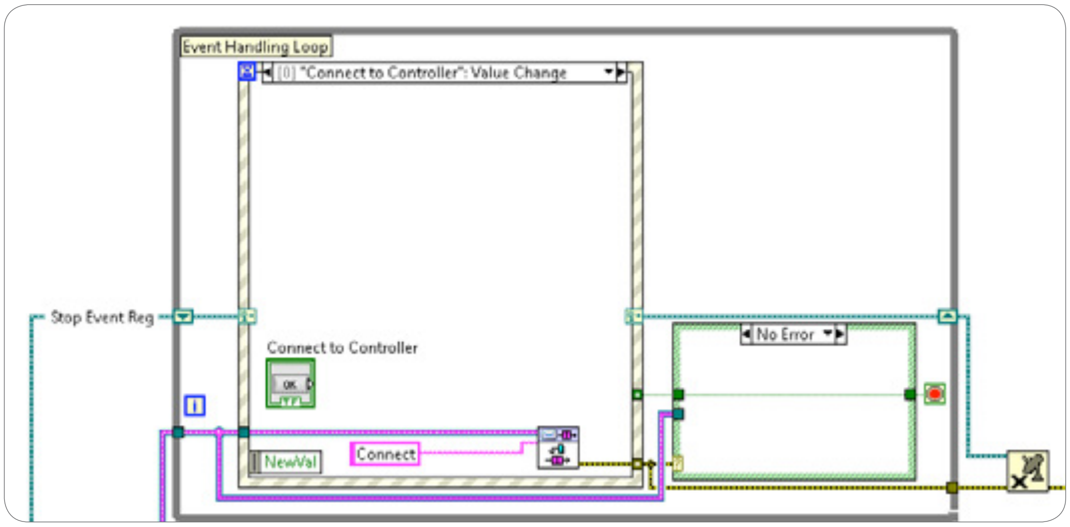
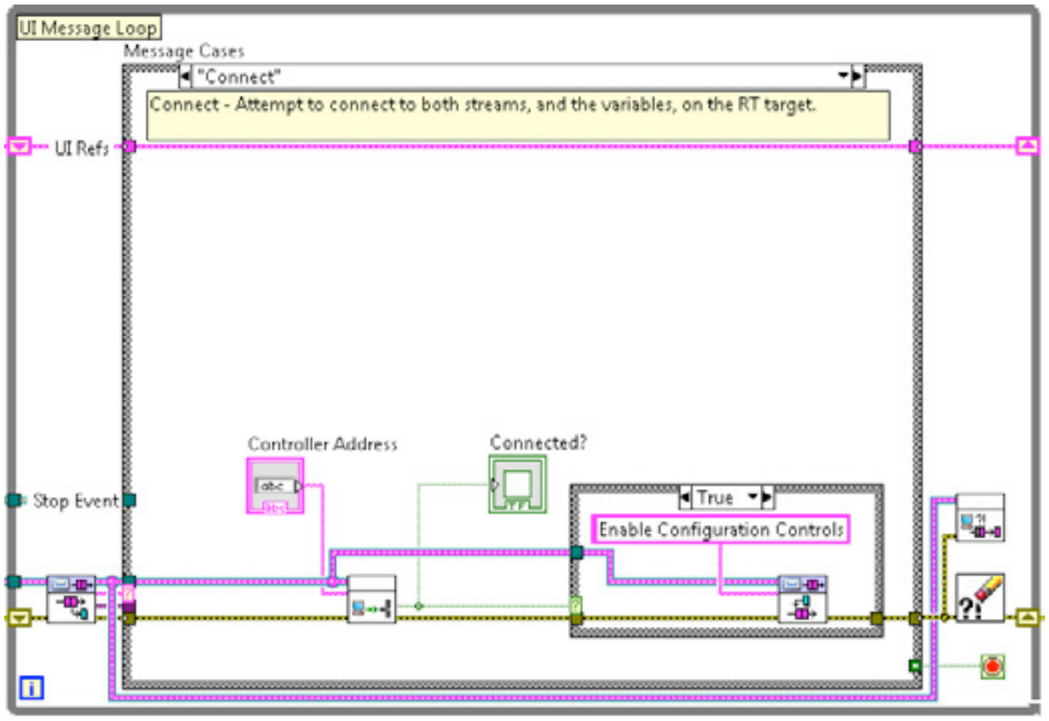


Figure 13.34. The Event Handling Loop sends messages to the UI Message Loop that are generated from front panel events.

UI Message Loop

The UI Message Loop, which is based on the Queued Message Handler project template, handles messages from the user interface and any error messages received from the real-time target via Network Streams.



UI Message Loop
Message Cases
"Connect"
Connect - Attempt to connect to both streams, and the variables, on the RT target.
UI Refs
Controller Address
Connected?
Stop Event
True
Enable Configuration Controls

Figure 13.35. The UI Message Loop sends messages to the CompactRIO device using Network Streams and handles the configuration of front panel controls.

Specifically, this loop implements the following tasks:

- "Initialize UI Refs," "Disable Controls," "Enable Configuration Controls"
  - Enables and disables front panel controls based on the connection status.
- "Connect"
  - Creates a Network Stream endpoint for each Network Stream.
  - Initializes network-published shared variables.
  - Updates the Connected? indicator if both the Network Streams and network-published shared variables are successfully created.
- "Close Connections"
  - Closes network connections upon an error in the UI Message Loop and/or the UI Monitoring Loop.
- "Disconnect UI"
  - Disconnects the UI after closing network connections (if something went wrong). User needs to attempt to reconnect and address any networking issues.
- "Get Settings from FPGA"

    - Once the connection is established within the "Connect" case, a command is sent to the RT Main VI to retrieve all current FPGA Main VI parameter values and send them back to the UI. This is important since the FPGA Main VI control loop continues to run even when the client disconnects and reconnects.

- "Initial UI Values"
    - Initializes the UI with the current values of the FPGA Main VI's read/write controls upon successful connection to the real-time target.
- "Run Control," "Run Manual," "Run Safe State," "Stop Control," "Stop Manual"
    - Sends any of the commands listed above to the RT Main VI, which then sends the commands to the FPGA Main VI.
- "Update Control Configuration"
    - Sends the control configuration on the user interface to the FPGA Main Control Loop when the user hits the "Update Control Configuration" button.
- "Update Manual Values," "Update Setpoints"
    - When the user modifies the value of any setpoint control or manual value control on the front panel, the new value is sent to the FPGA control loop.
- "Update Display Mode"
    - Enables/disables controls depending on the state of the FPGA (this case is called when the FPGA state changes).
- "Critical Error," "Log-Only Error," "Non-Critical Error," "RT Error"
    - Identifies any incoming errors as "RT" or "UI" and displays the error message to the System Status indicator on the front panel.
    - Exits upon a critical error.
- "Update System Status"
    - Adds a new string to the beginning of the System Status string on the UI.
- "Exit"
    - Fires the user event to stop the Event Handling Loop and toggles the local variable to stop the Monitoring Loop. This message alone causes all three loops to stop.

Monitoring Loop

The Monitoring Loop runs periodically when connected and uses network-published shared variables to display the monitored values from the real-time target. You use network-published shared variables instead of Network Streams because you are concerned only with the current value of the data, and you are OK with missing a few data points here and there.

Note that you use the Dynamic Shared Variable API instead of the static shared variable nodes. This is because you can dynamically change the IP address of the real-time target from the front panel. To ensure that you connect to the Shared Variable Library that is hosted on the real-time target corresponding to the IP address on the front panel, use the IP address on the front panel when opening a connection to each variable.

When you use static shared variable nodes, the IP address is statically configured in the LabVIEW project. If you change the IP address on the front panel, the UI Main VI continues to connect to the IP address defined in the LabVIEW project.

The shared variable connection is created in the UI-Initiate Connection.vi found in the Connect state of the UI Message Loop.

For more information on the LabVIEW FPGA Control on CompactRIO sample project, view the documentation in the created project or follow the More Information link in the Create Project window.

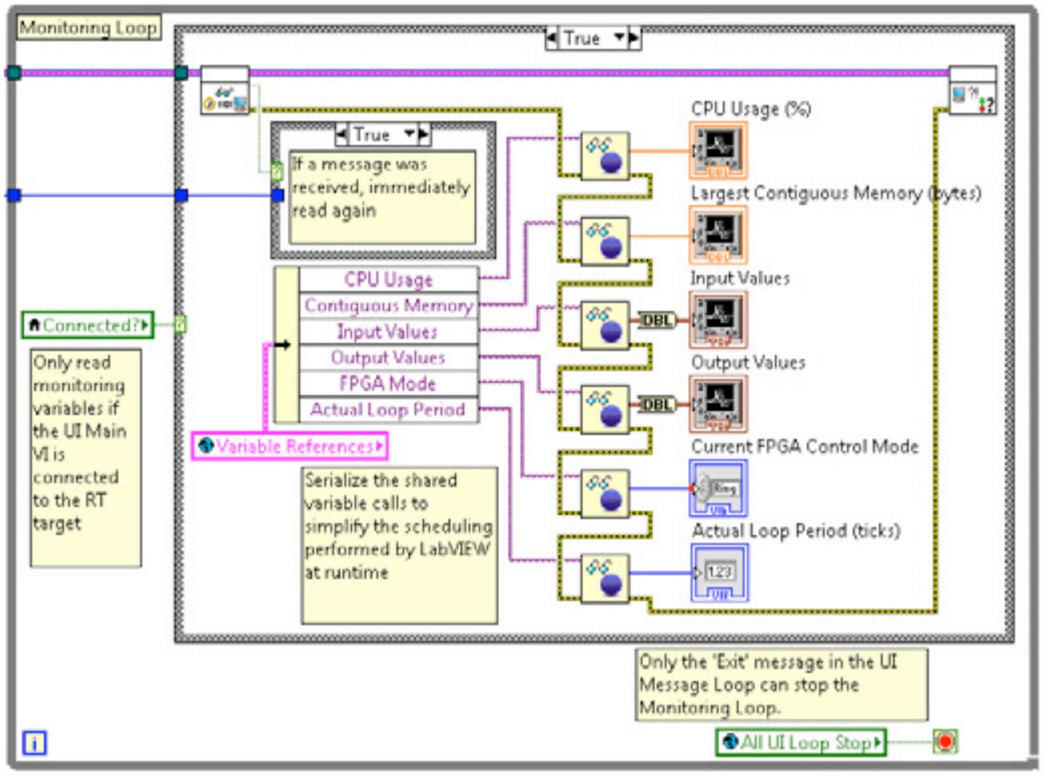

Monitoring Loop
True
True
If a message was received, immediately read again
CPU Usage
Contiguous Memory
Input Values
Output Values
FPGA Mode
Actual Loop Period
Connected?
Only read monitoring variables if the UI Main VI is connected to the RT target
Variable References
Serialize the shared variable calls to simplify the scheduling performed by LabVIEW at runtime
CPU Usage (%)
Largest Contiguous Memory (bytes)
Input Values
Output Values
Current FPGA Control Mode
Actual Loop Period (ticks)
DBL
DBL
Only the 'Exit' message in the UI Message Loop can stop the Monitoring Loop.
All UI Loop Stop

Figure 13.36. The Monitoring Loop receives current value data from the CompactRIO device and displays it to the UI.

Downloading the FPGA Bitfile to Flash Memory

This FPGA VI was written to be loaded to flash memory using the NI RIO Device Setup utility and configured to reload the FPGA bitfile only upon cycling the power. The benefit of loading the bitfile to flash memory is that you ensure that the bitfile is always loaded and running, even if you need to restart the real-time controller. This is especially important for this specific application since the hardware outputs must stay in a safe state upon rebooting the real-time controller. For more information, see the "Deploying Stand-Alone FPGA Applications" section of the white paper Managing FPGA Deployments.

Configure your FPGA bitfile to download directly to flash memory using the following instructions:

1. Right-click the LabVIEW FPGA Build Specification in the LabVIEW project and select Properties.

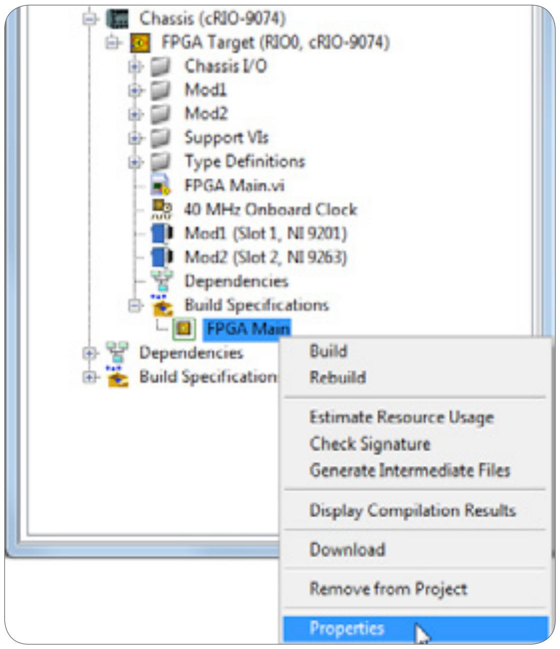
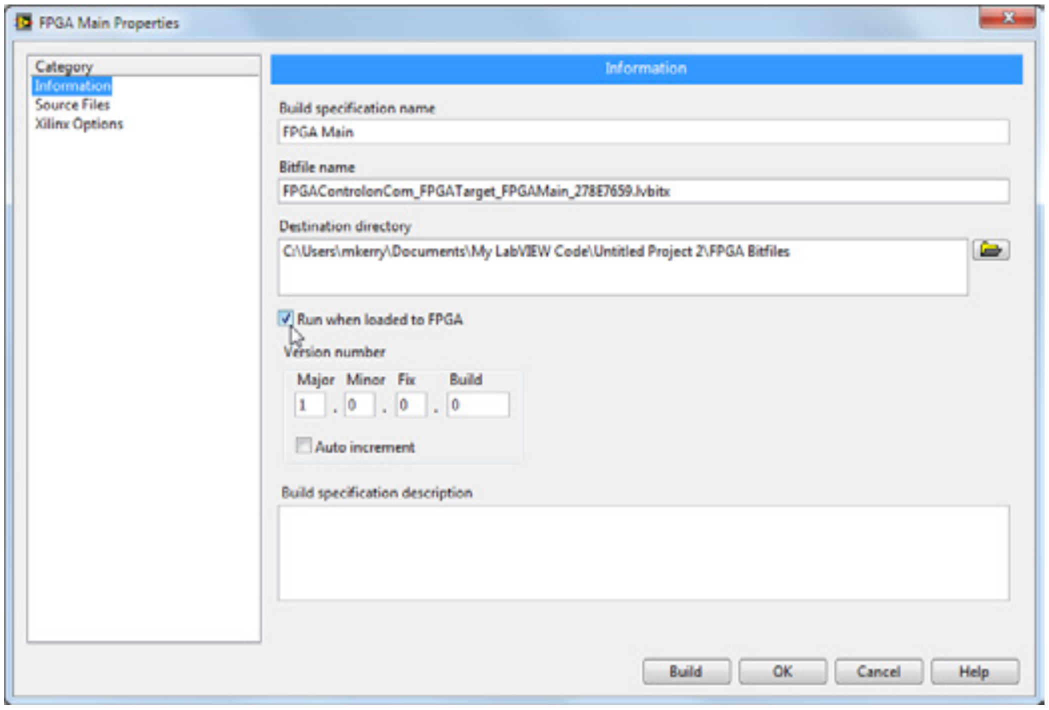


Figure 13.37. To download a LabVIEW FPGA bitfile to flash, right-click the LabVIEW FPGA Build Specification and go to Properties.

2. On the Information page, ensure that “Run when loaded to FPGA” is checked. If it’s not checked, check it and rebuild the bitfile.


FPGA Main Properties

Category: Information, Source Files, Xilinx Options

Information

Build specification name: FPGA Main

Bitfile name: FPGAControlonCom_FPGATarget_FPGAMain_278E7659.lvbitx

Destination directory: C:\Users\mkerry\Documents\My LabVIEW Code\Untitled Project 2\FPGA Bitfiles

[✓] Run when loaded to FPGA

Version number

Major	Minor	Fix	Build
1	0	0	0

[ ] Auto increment

Build specification description

Figure 13.38. Ensure that “Run when loaded to FPGA” is checked.

3. To load your FPGA bitfile to flash memory, launch the RIO Device Setup utility by right-clicking your FPGA target.

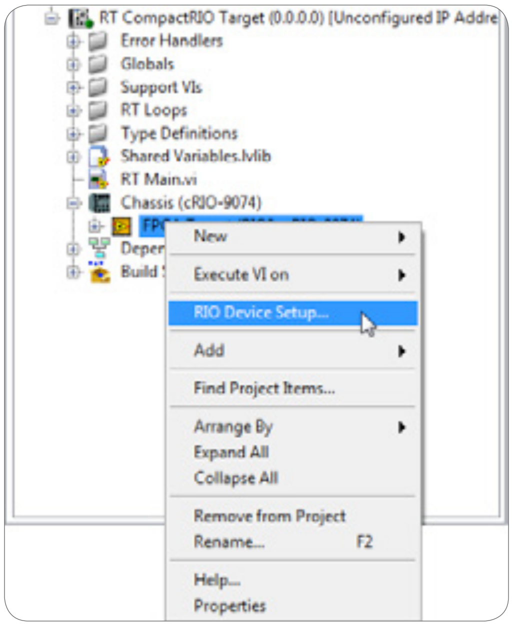
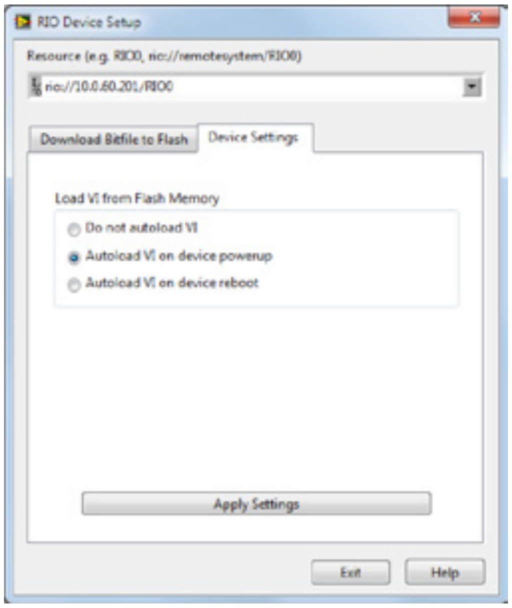


Figure 13.39. Launch the RIO Device Setup Utility from the LabVIEW project.

4. On the Download Bitfile to Flash tab, select the bitfile you want to use and click the Download Bitfile button.

5. On the Device Settings tab, select "Autoload VI on device powerup" and press the Apply Settings button.



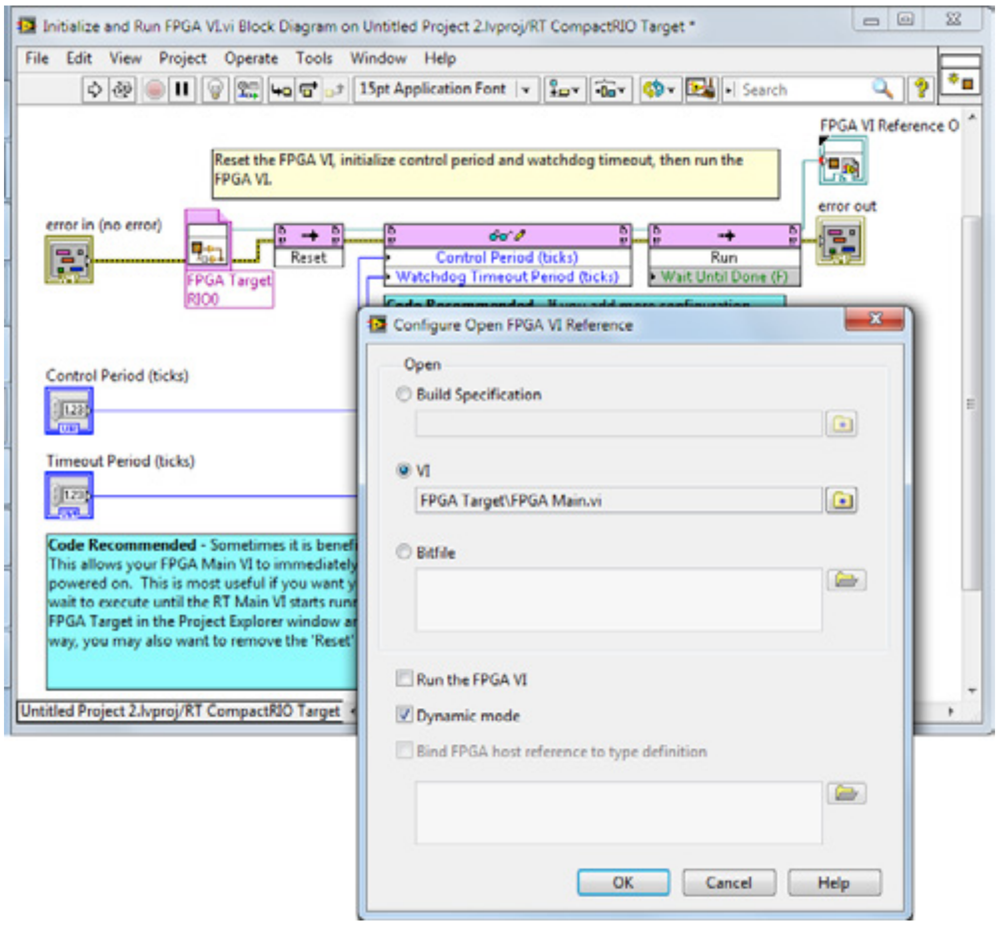
Resource (e.g. RIO0, rio://remotesystem/RIO0)
rio://10.0.60.201/RIO0

Load VI from Flash Memory
- ○ Do not autoload VI
- ● Autoload VI on device powerup
- ○ Autoload VI on device reboot

Figure 13.40. Configure your FPGA bitfile to load to your device every time you cycle the power.

6. Now your FPGA bitfile loads to your device every time you cycle the power. If you perform a system reset, the FPGA bitfile stays loaded, and the real-time OS restarts.

7. Finally, open the Initialize and Run FPGA subVI within the RT Main VI and ensure that the Open FPGA VI Reference is configured to not run the FPGA VI when called. Since the FPGA bitfile already is loaded, calling this function resets the FPGA bitfile, which is not necessary.


Initialize and Run FPGA VI.vi Block Diagram on Untitled Project 2.lvproj/RT CompactRIO Target *
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
Reset the FPGA VI, initialize control period and watchdog timeout, then run the FPGA VI.
error in (no error)
FPGA Target RIO0
Reset
Control Period (ticks)
Watchdog Timeout Period (ticks)
Run
Wait Until Done (F)
FPGA VI Reference O
error out
Control Period (ticks)
Timeout Period (ticks)
Untitled Project 2.lvproj/RT CompactRIO Target
Configure Open FPGA VI Reference
Open
Build Specification
VI
FPGA Target\FPGA Main.vi
Bitfile
Run the FPGA VI
Dynamic mode
Bind FPGA host reference to type definition
OK
Cancel
Help

Code Recommended - Sometimes it is benef
This allows your FPGA Main VI to immediately
powered on. This is most useful if you want y
wait to execute until the RT Main VI starts run
FPGA Target in the Project Explorer window a
way, you may also want to remove the 'Reset'

Open: ( ) Build Specification [✓ (•) VI] ( ) Bitfile

☐ Run the FPGA VI
☑ Dynamic mode
☐ Bind FPGA host reference to type definition

Figure 13.41. Uncheck the Run the FPGA VI option in the Configure Open FPGA VI Reference dialog.