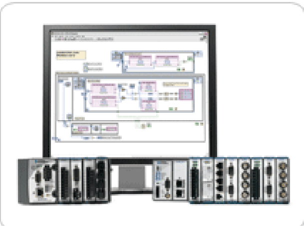
Document Type: Tutorial
NI Supported: Yes
Publish Date: Feb 24, 2011

Getting Started with CompactRIO - Logging Data to Disk

Overview



This article is part of a series on developing CompactRIO applications in LabVIEW.

Click here to view the list of other articles

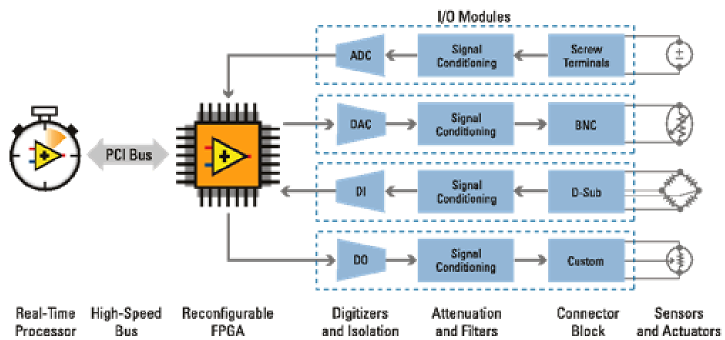
This tutorial provides an introduction to logging data to disk with CompactRIO in addition to an overview of the CompactRIO architecture and programming models available in LabVIEW.

Table of Contents

The RIO Architecture

The National Instruments CompactRIO programmable automation controller is an advanced embedded control and data acquisition system designed for applications that require high performance and reliability. With the system's open, embedded architecture, small size, extreme ruggedness, and flexibility, engineers and embedded developers can use COTS hardware to quickly build custom embedded systems. NI CompactRIO is powered by National Instruments LabVIEW FPGA and LabVIEW Real-Time technologies, giving engineers the ability to design, program, and customize the CompactRIO embedded system with easy-to-use graphical programming tools.

CompactRIO combines an embedded real-time processor, a high-performance FPGA, and hot-swappable I/O modules. Each I/O module is connected directly to the FPGA, providing low-level customization of timing and I/O signal processing. The FPGA is connected to the embedded real-time processor via a high-speed PCI bus. This represents a low-cost architecture with open access to low-level hardware resources. LabVIEW contains built-in data transfer mechanisms to pass data from the I/O modules to the FPGA and also from the FPGA to the embedded processor for real-time analysis, postprocessing, data logging, or communication to a networked host computer.



C Series I/O Modules

A variety of I/O types are available including voltage, current, thermocouple, RTD, accelerometer, and strain gauge inputs; up to ±60 V simultaneous-sampling analog I/O; 12, 24, and 48 V industrial digital I/O; 5 V/TTL digital I/O; counter/timers; pulse generation; and high voltage/current relays. Because the modules contain built-in signal conditioning for extended voltage ranges or industrial signal types, you can usually connect wires directly from the C Series modules to your sensors and actuators.

FPGA

The embedded FPGA is a high-performance, reconfigurable chip that engineers can program with LabVIEW FPGA tools. Traditionally, FPGA designers were forced to learn and use complex design languages such as VHDL to program FPGAs. Now, any engineer or scientist can use graphical LabVIEW tools to program and customize FPGAs. Using the FPGA hardware embedded in CompactRIO, you can implement custom timing, triggering, synchronization, control, and signal processing for your analog and digital I/O.

Real-Time Processor

The CompactRIO embedded system features an industrial 400 MHz Freescale MPC5200 processor that deterministically executes your LabVIEW Real-Time applications on the reliable Wind River VxWorks real-time operating system. LabVIEW has built-in functions for transferring data between the FPGA and the real-time processor within the CompactRIO embedded system. Choose from more than 600 built-in LabVIEW functions to build your multithreaded embedded system for real-time control, analysis, data logging, and communication. You can also integrate existing C/C++ code with LabVIEW Real-Time code to save on development time.

Size and Weight

Size, weight, and I/O channel density are critical design requirements in many embedded applications. A four-slot reconfigurable embedded system measures 179.6 by 88.1 by 88.1 mm (7.07 by 3.47 by 3.47 in.) and weighs just 1.58 kg (3.47 lb).

Application Examples

With the low cost and reliability of CompactRIO, as well as its suitability for high-volume embedded measurement and control applications, you can adapt it to solve a wide variety of industry and application challenges. Examples include:

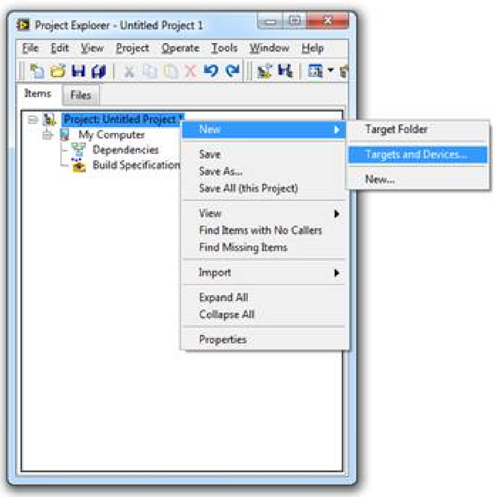
- In-vehicle data acquisition, data logging, and control
- Machine condition monitoring and protection
- Embedded system prototyping
- Remote and distributed monitoring
- Embedded data logging
- Custom multiaxis motion control
- Electrical power monitoring and power electronics control
- Servo-hydraulic and heavy machinery control
- Batch and discrete control
- Mobile/portable noise, vibration, and harshness (NVH) analysis

Starting a New CompactRIO Project in LabVIEW

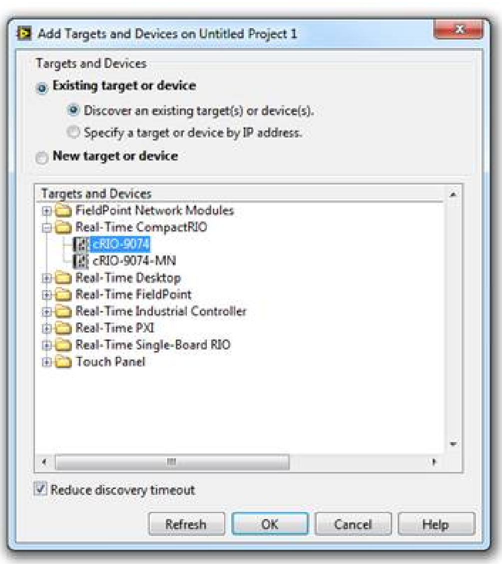
Begin by creating a new project in LabVIEW, where you will manage your code and hardware resources.

1. Create a new project in LabVIEW by selecting **File » New Project**

2. To add your CompactRIO system to the project, right-click on the Project item at the top of the tree and select **New » Targets and Devices…**



3. This dialog allows you to discover systems on your network or add offline systems. Expand the **Real-Time CompactRIO** folder, select your system, and click **OK**. *Note: If your system is not listed, LabVIEW could not detect it on the network. Ensure that your system is properly configured with a valid IP address in Measurement & Automation Explorer. If you system is on a remote subnet, you can also select to manually enter the IP address.*

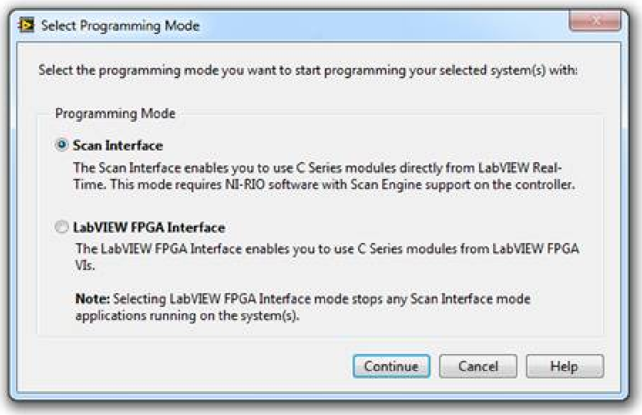


Select the Appropriate Programming Model

LabVIEW provides two programming models for CompactRIO systems. If you have LabVIEW Real-Time and LabVIEW FPGA on your development computer, you will be prompted to select which programming model you would like to use. You can change this setting later in the LabVIEW Project if needed.

Scan Interface (CompactRIO Scan Mode) – this option allows you to program the real-time processor of your CompactRIO system, but not the FPGA. In this mode, NI provides a pre-defined personality for the FPGA that periodically scans the I/O and places it in a memory map, making it available to LabVIEW Real-Time. CompactRIO Scan Mode is sufficient for applications that require single-point access to I/O at rates of a few hundred hertz. To learn more about scan mode, read the Using CompactRIO Scan Mode with NI LabVIEW white paper and view the benchmarks.

LabVIEW FPGA Interface – this option allows you to unlock the real power of CompactRIO by customizing the FPGA personality in addition to programming the real-time processor, achieving performance that would typically require custom hardware. Using LabVIEW FPGA, you can implement custom timing and triggering, off-load signal processing and analysis, create custom protocols, and access I/O at its maximum rate.



Select the appropriate programming model for your application.

LabVIEW will now attempt to detect the chassis and C Series I/O modules present in your system and automatically add them to the LabVIEW Project. *Note: If your system was not discovered and you choose to add it offline, you will need to add the chassis and C Series I/O manually. The LabVIEW Help online discusses this process for scan mode and FPGA mode.*

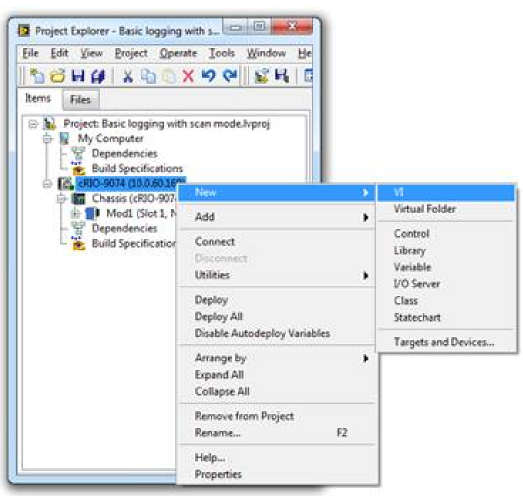
Once your system has been added to the LabVIEW Project, proceed to either the CompactRIO Scan Mode Tutorial or LabVIEW FPGA Tutorial below, depending on which programming model you selected.

CompactRIO Scan Mode Tutorial

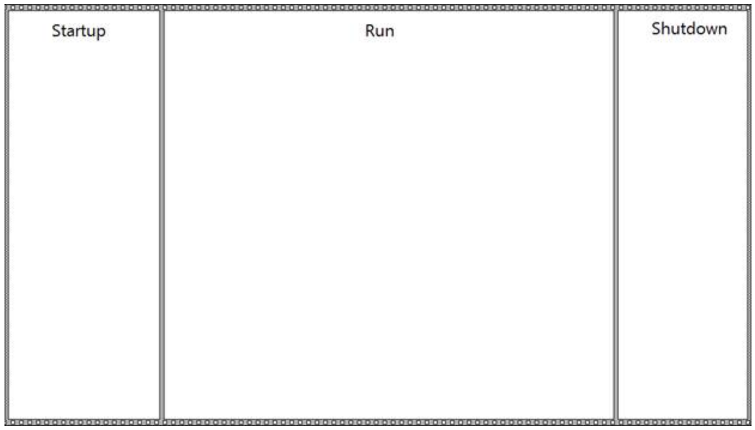
This section will walk you through creating a basic logging application on CompactRIO using scan mode. If you chose to use the LabVIEW FPGA Interface, see the LabVIEW FPGA Tutorial below. You should now have a new LabVIEW Project that contains your CompactRIO system, including the controller, chassis, and C Series I/O modules. In this tutorial we will be using an NI 9211 Thermocouple input module; however, the process can be followed for any analog input module. You can also download the solution from here.

1. Save the project by selecting **File»Save** and entering *Basic logging with scan mode***.** Click **OK**.

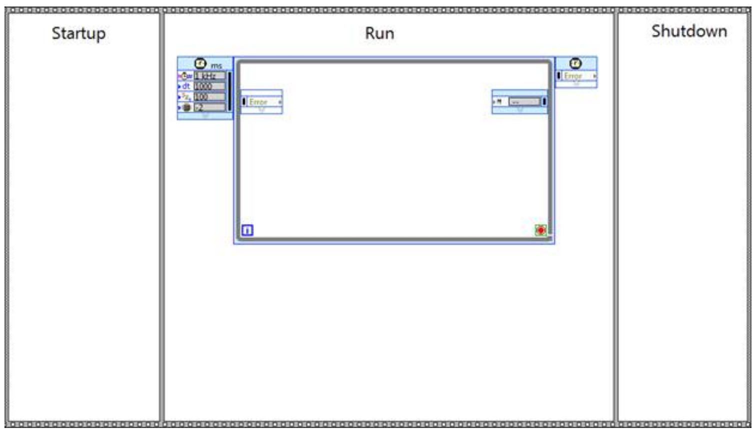
2. This project will only contain one VI, which is the LabVIEW Real-Time application that runs embedded on the CompactRIO controller. Create this the VI by right-clicking on the CompactRIO real-time controller in the project and selecting **New»VI.** Save the VI as *RT.vi*.



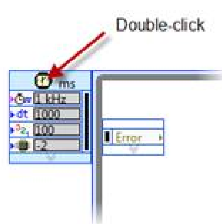
3. The basic operation of this application will include three routines: startup, run, and shutdown. A flat sequence structure is an easy way to enforce this order of operation. Place a **flat sequence structure** with three frames on your RT.vi block diagram as shown below.



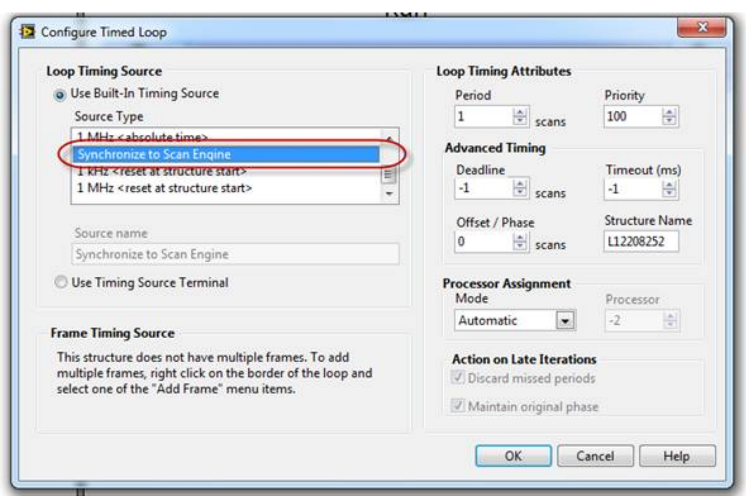
4. Now add a timed loop to the *Run* frame of the sequence structure. Timed loops provide the ability to synchronize code to various time basis, including the NI Scan Engine that reads and writes scan mode I/O.



5. To configure the timed loop, double-click on the clock icon on the left input node.

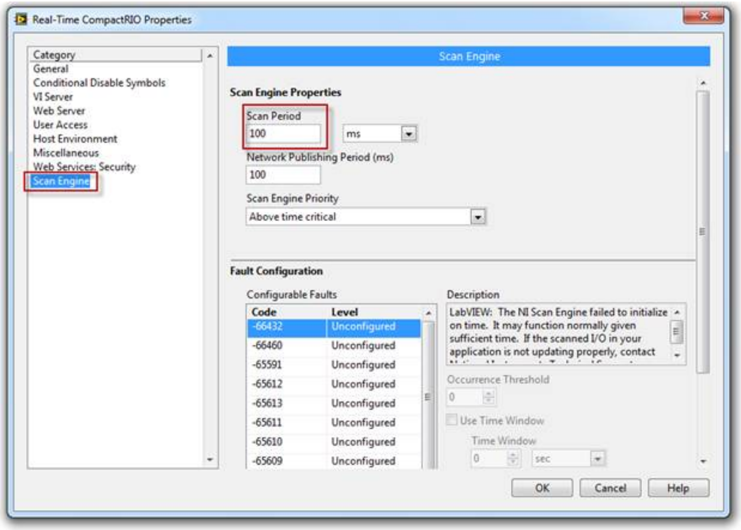


6. Select **Synchronize to Scan Engine** as the Loop Timing Source. Click **OK**. This will cause the code in the timed loop to execute once, immediately after each I/O scan, ensuring that any I/O values used in this timed loop are the most recent values.

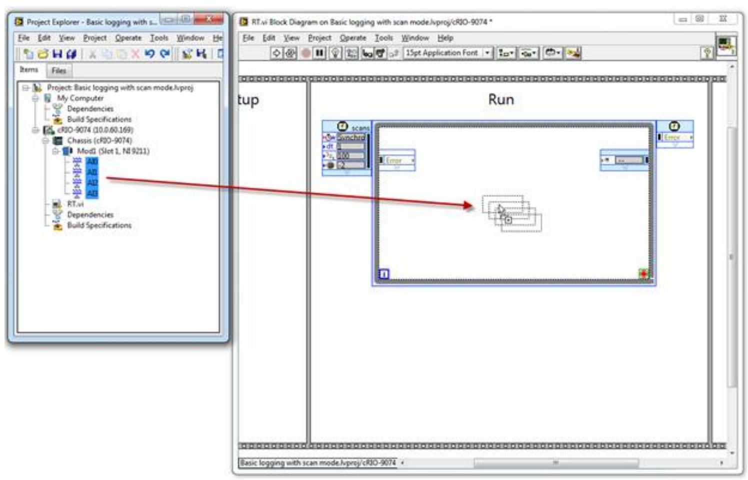


7. The previous step configured the timed loop to run synchronized to the scan engine. Now configure the rate of the scan engine itself by right-clicking on the CompactRIO real-time controller in the LabVIEW Project and selecting **Properties.**

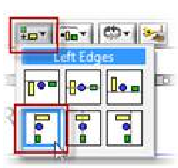
8. Select **Scan Engine** from the categories on the left and enter **100ms** as the *Scan Period*. This will cause all of the I/O in the CompactRIO system to be updated every 100ms (10Hz). The Network Publishing Period can also be set from this page, which controls how often the I/O values are published to the network for remote monitoring and debugging. Click **OK**.

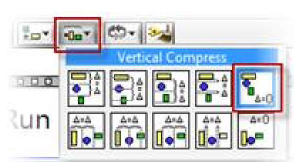


9. Now that you have configured the I/O scan rate, it is time to add the I/O reads to your application for logging. When using CompactRIO Scan Mode, you can simply drag and drop the I/O variables from the LabVIEW Project to the block diagram. Expand the CompactRIO real-time controller, chassis, and the I/O module you would like to log. Select all of the channels below the module by clicking on them and using the shift key, then drag and drop them into the timed loop on your RT.vi diagram as shown below.

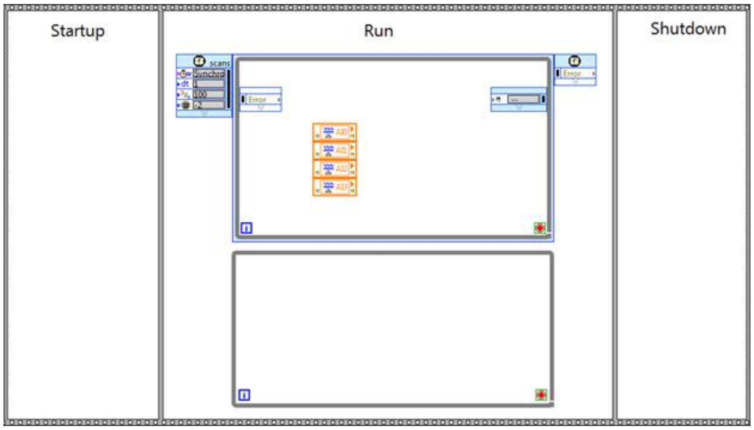


*Tip: Use the **Align Objects » Left Edges** and **Distribute Objects » Vertical Compress** items on the LabVIEW toolbar to organize the I/O variables on your diagram.*

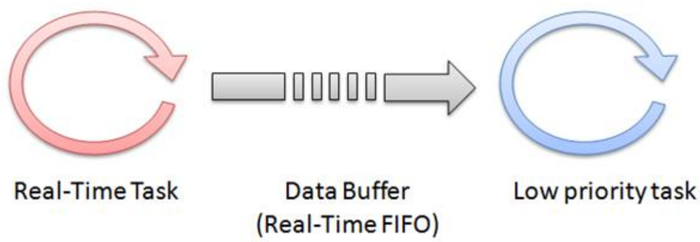




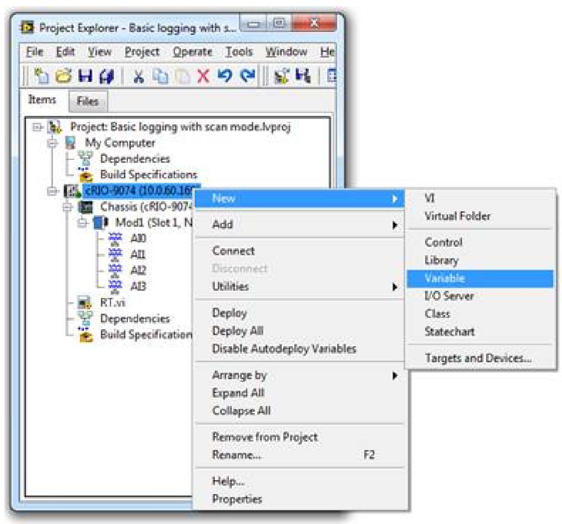
10. The next logical step would be to write the I/O variable values to disk using file I/O functions; however, because file I/O takes and undermined amount of time, it is necessary to separate the I/O acquisition task and the file I/O task. Neglecting this requirement could lead to losing data, since the file I/O may take longer than the I/O scan, causing a sample to be missed. Place a normal **while loop** in the *Run* frame under the timed loop, which will be used for the file I/O task.



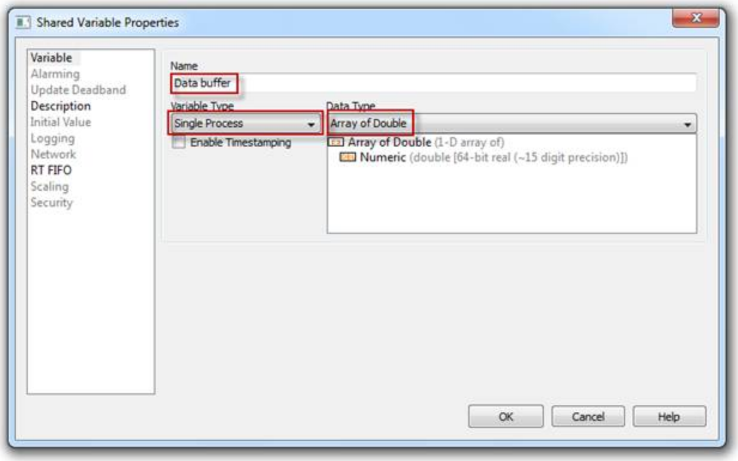
In order write the data to disk in the regular while loop, you need to transfer the I/O values from the timed loop using a real-time FIFO. This will provide a buffer between the two loops. The timed loop will run, synchronized the I/O scan, and write the new I/O values to the buffer each time. Then, the regular while loop will read the data out of the buffer and write it to disk. Separating the I/O task and disk access in this way allows your timed loop to run with "real-time" performance, meaning that it will always finish on time.



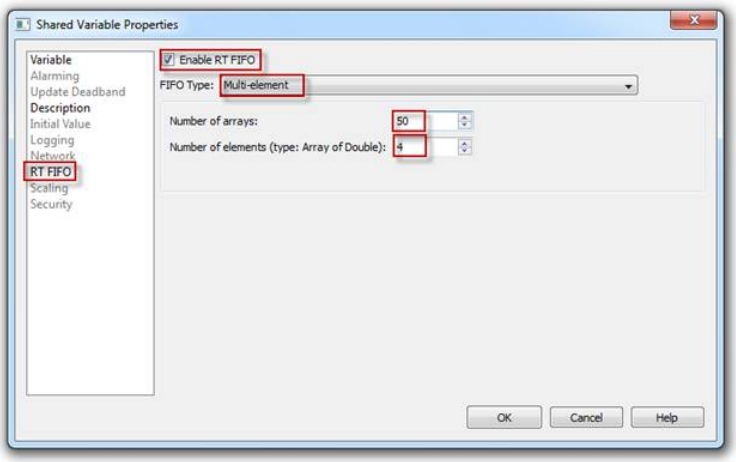
11. In the LabVIEW Project, right-click on the CompactRIO real-time controller and select **New»Variable.**



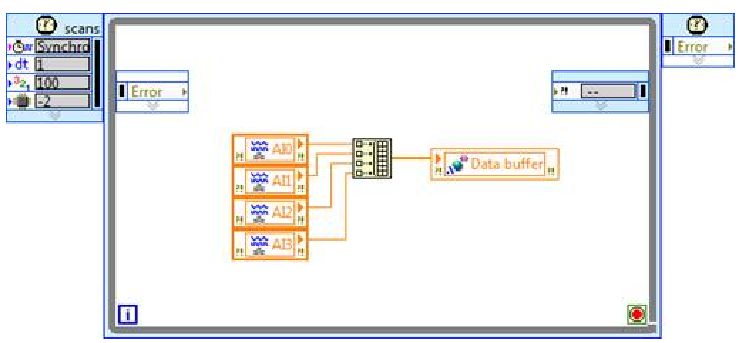
12. Name the variable *Data buffer,* select **Single Process** as the Variable Type, and **Array of Double** as the Data Type. This will create a locally scoped variable (no network publishing) that contains and array of double precision floating point numbers. Then select **RT FIFO** from the menu on the left.



13. With the RT FIFO category selected, select the **Enable RT FIFO** check box, select **Multi-element** for the FIFO Type, enter **50** for the Number of arrays, and enter **4** for the Number of elements (if you are logging a number of channels other than 4, enter that instead.) This configures the variable to operate as a real-time-safe FIFO, which can serve as the data buffer between our real-time and low priority tasks. The FIFO will hold fifty one dimensional arrays, each of which contain four double precision numbers. Click **OK.**

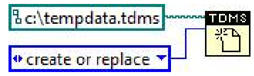


14. Drag and drop the **Data buffer** RT FIFO into your **timed loop** and use a **Build Array** function to build an array from the I/O variables and pass it into the RT FIFO.

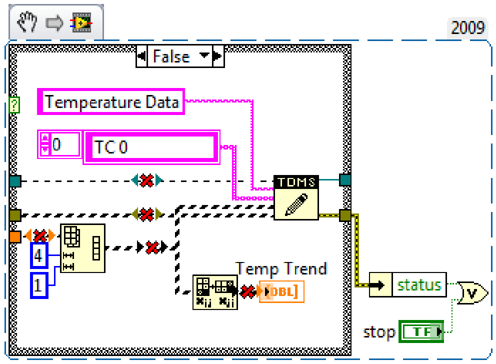


15. Drag and drop an additional copy of the **Data buffer** RT FIFO into the regular **while loop**, where you will read the data out and log it to disk. Also, right-click on the RT FIFO and select **Show Timeout** and wire a timeout of **100**. This will cause the RT FIFO to wait up to 100ms for new data to arrive in the buffer before it times out. If data is present in the buffer, the RT FIFO will return the oldest data in the buffer immediately.

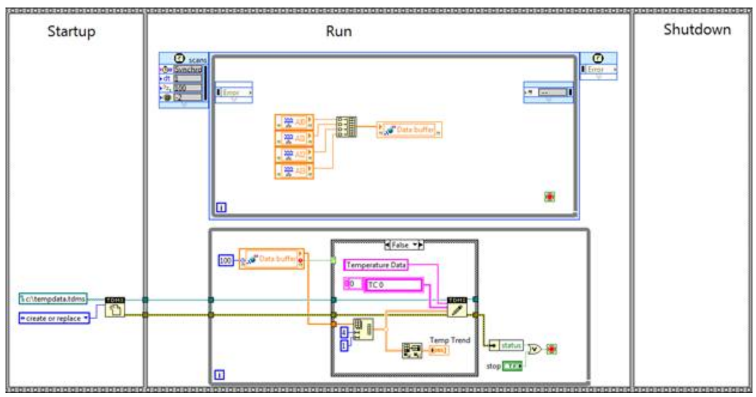
16. Now we will add the file I/O functions. Place a **TDMS Open** VI in the Startup frame of your sequence structure. Create constants for the **file path** and **operation** inputs. Type *c:\tempdata.tdms* in for the file path and select **create or replace** for the operation.



17. To perform the file I/O, a snippet of code is provided for you. You can download it here or drag and drop the VI Snippet below to your diagram from your Web browser. Place this code in the regular while loop and **wire** it as shown below.



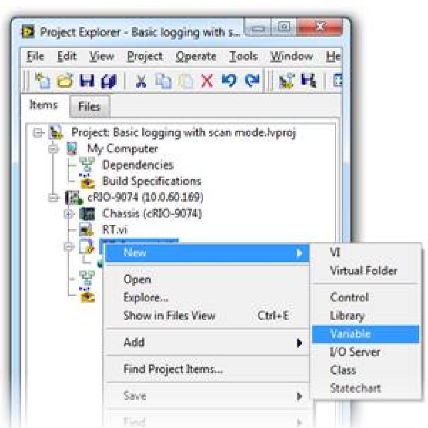
Drag this VI Snippet to your VI's block diagram



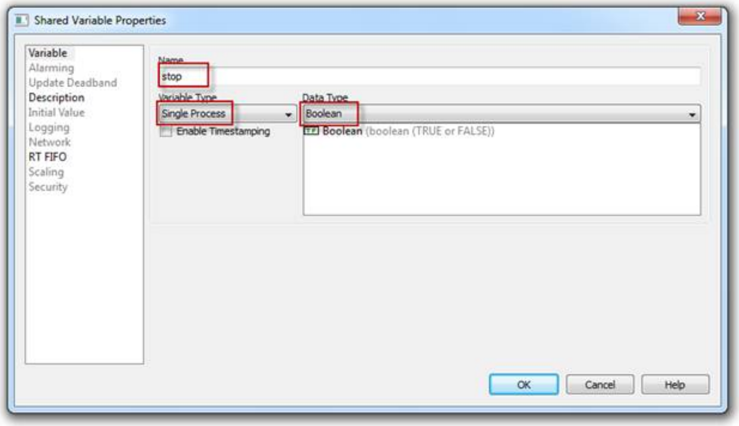
A case structure is used to execute this section of code only when the RT FIFO does not time out, indicating that new data has been returned from the buffer. Then, the data is formatted to be written to the TDMS file and displayed on the waveform chart. A stop button for the while loop is also provided in addition to checking for errors as a stop condition.

At this point, the application is almost complete. All that remains is to create a stop condition for the timed loop and to close the TDMS file in the *Shutdown* frame.

18. In the LabVIEW Project, create another shared variable by right-clicking on the library that contains the *Data buffer* RT FIFO and selecting **New»Variable.**



19. Name the variable *stop,* select **Single Process,** and select **Boolean.** Then select the RT FIFO category on the left.



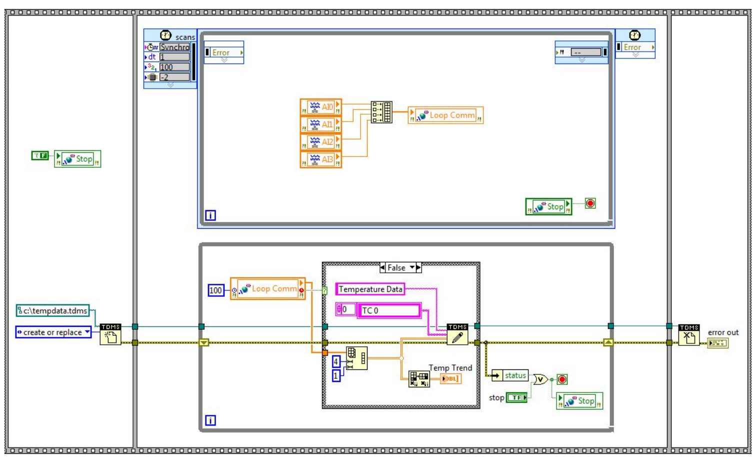
20. Select **Enable RT FIFO,** leave Single-element selected and click **OK**. This creates a variable that is safe to read from in tasks that require real-time performance.

21. Place a copy of the **stop** variable in the timed loop and wire it to the **stop condition**.

22. Place another copy of the **stop** variable in the regular while loop and write the result of the **OR** function to it as shown. This will cause the timed loop to stop when the regular while loop stops.

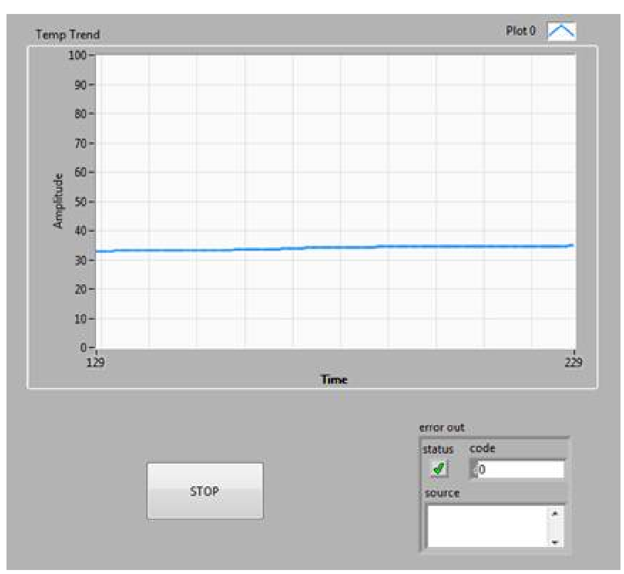
23. Place a **TDMS Close** function in the *Shutdown* frame and wire the file reference and error cluster through to it. Also, create and indicator from the TDMS Close **error out** terminal.

Your completed application should look like the image below:



24. Click **Run** on *RT.vi*, click **Save** for any unsaved items, and click **OK** on any dialogs or warnings about applying changes to the CompactRIO system. LabVIEW will now deploy your VI over Ethernet to run embedded on the CompactRIO system.

25. Once the VI deploys and begins running, view the front panel of your VI to see the current I/O values plotted on the waveform chart.



26. Click **STOP** once you are finished viewing and logging data.

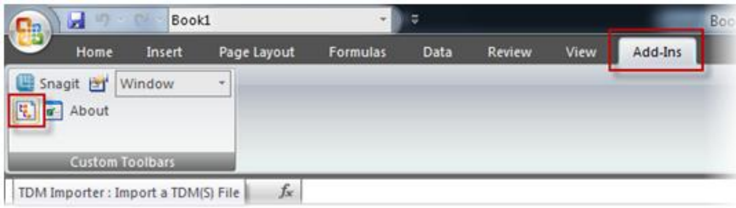
Congratulations! You have successfully created an embedded logging application with LabVIEW and CompactRIO. To continue learning, check out the additional resources on the CompactRIO Setup and Services page.

Note: Perform the following to retrieve and view the data logged on the CompactRIO system:

1. Using Windows Explorer or a Web browser, navigate to *ftp://<ip address>* where *<ip address>* is the IP address of your CompactRIO system

2. Download *tempdata.tdms* (or whatever you named the TDMS file)

3. If you have Microsoft Excel, you can view TDMS files by selecting the **Add-Ins** tab and clicking the **TDM Importer.** The first page of the workbook contains file information and the remaining sheets contain the channel data. *The TDM Importer installs with LabVIEW by default; however, if the TDM Importer is not present, you can install it from here.*



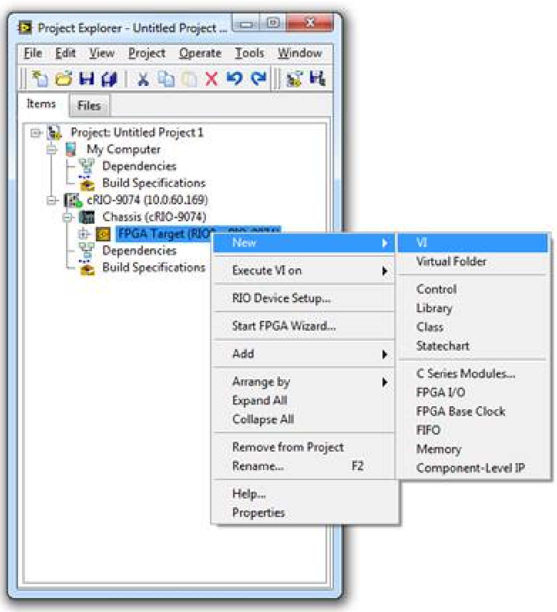
*4. If you do not have Excel, you can view the TDMS file using the **TDMS File Viewer** VI in LabVIEW.*

LabVIEW FPGA Tutorial

This section will walk you through creating a basic high-speed data logging application on CompactRIO using LabVIEW FPGA. You should now have a new LabVIEW Project that contains your CompactRIO system, including the controller, chassis, FPGA, and C Series I/O modules. In this tutorial we will be using an NI 9205 analog input module; however, the process can be followed for any analog input module. You can also download the solution from here.

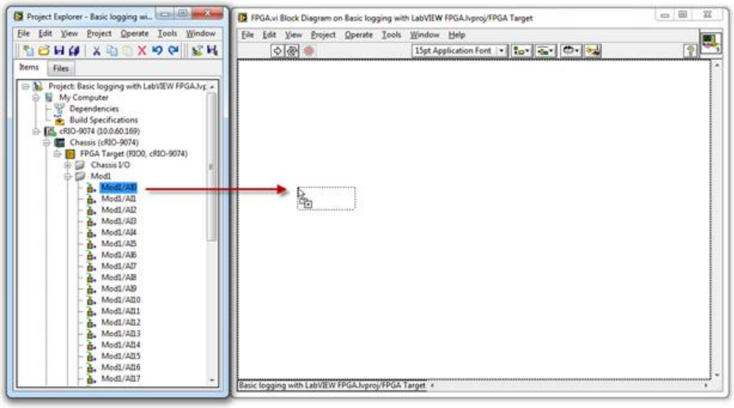
1. Save the project by selecting **File»Save** and entering *Basic logging with LabVIEW FPGA***.** Click OK.

2. In the LabVIEW Project, expand the CompactRIO real-time controller and chassis to find the FPGA item. Right-click on the FPGA item and select **New»VI.** This VI will perform the high-speed analog acquisition.

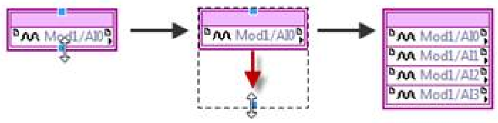


3. Save the VI as *FPGA.vi*

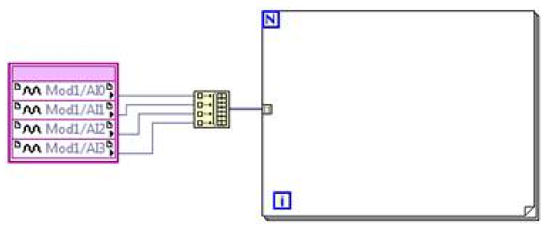
4. To add an I/O node that will acquire a sample from the analog input module, expand the folder named **Mod1** and drag **Mod1/AI0** to the FPGA VI diagram.



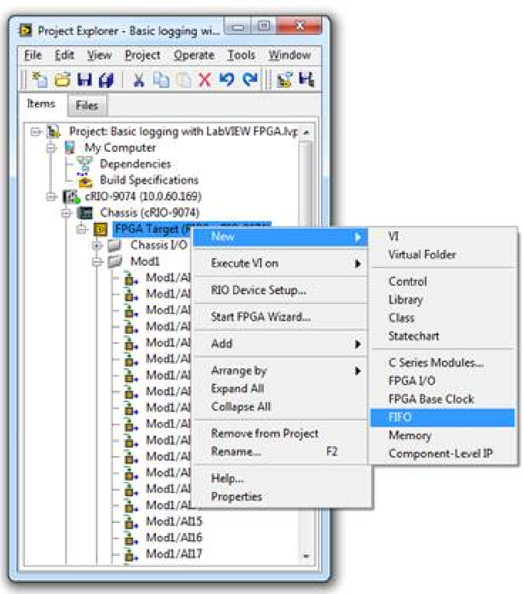
5. Expand the I/O node by clicking and dragging downward as shown, so that channels AI0 through AI3 are visible. This will cause the I/O node to sample all four channels each time it executes.



6. Use a **Build Array** function to combine the channels into an array. Also, place a **For Loop** on the diagram and wire the array into it. This For Loop will be used to interleave the samples into a FIFO that communicates to the real-time processor.



7. To create the FIFO that will pass data from the FPGA VI to the real-time VI, right-click on the FPGA in the LabVIEW Project and select **New»FIFO.**



8. Configure the FIFO as shown. This information was obtained by using the Context Help information on the wire returning data from the NI 9205 I/O node on the FPGA VI diagram.

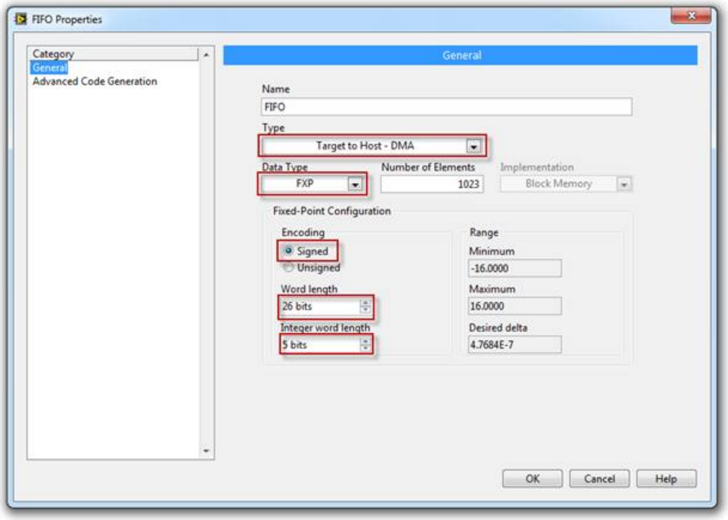
Type: **Target to Host – DMA**

Data Type: **FXP**

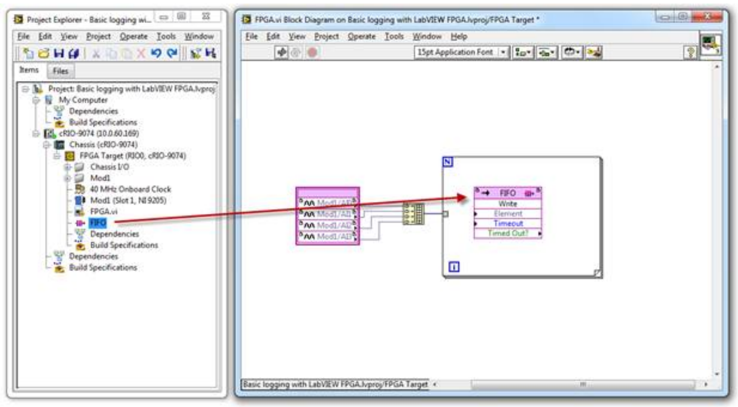
Encoding: **Signed**

Word length: **26 bits**

Integer word length: **5 bits**



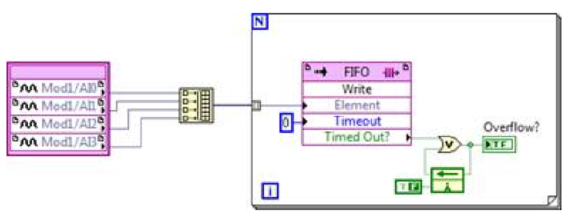
9. Drag and drop the FIFO into the For Loop on your FPGA diagram.



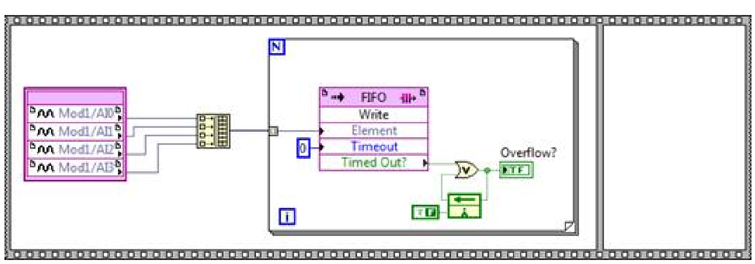
10. Wire the auto-indexed array terminal into the **Element** input of the FIFO. Create a constant for the **Timeout** input of the FIFO with a value of 0.

11. Wire the **Timed Out?** output of the FIFO into an **Or** function. Create an indicator for the result of the Or function and name it *Overflow?*

12. Place a **Feedback Node** on the diagram and wire the result of the Or function to its input (right side). Wire the output (left side) of the feedback node to the remaining input of the Or function. Create a false constant for the initialization input of the feedback node. The result, shown below, creates a latching mechanism that will cause *Overflow?* to become True and stay True, if the *Timed Out?* output of the FIFO is ever asserted. This output indicates that the FIFO was full and data was lost.

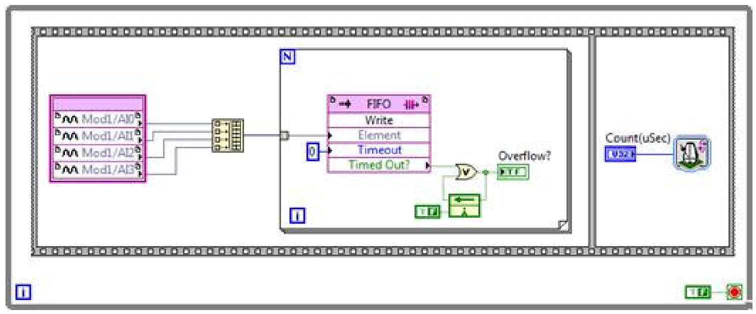


13. To enforce execution order and timing on the diagram, place a Flat Sequence Structure around the diagram and add one additional frame after as shown.

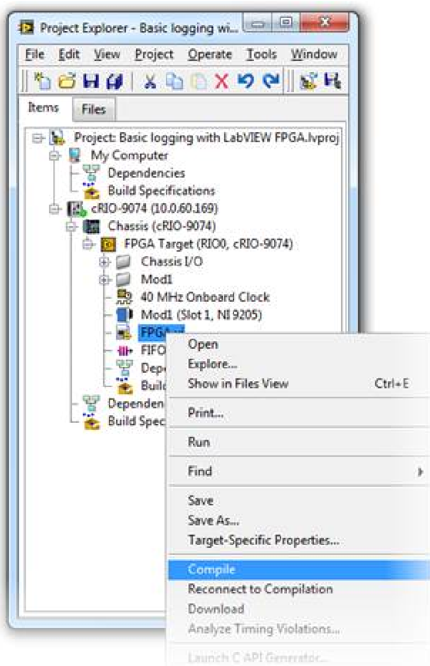


14. Place a **Loop Timer** in the right-hand frame. Select **uSec** from the Configure Loop Timer dialog and click **OK.** Create a **control** for the Loop Timer input.

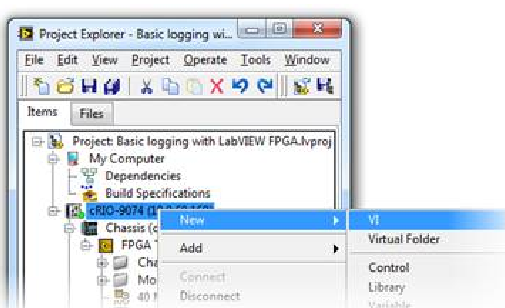
15. Place a **While Loop** around the Flat Sequence Structure and create a **False** constant for the loop stop condition.



16. Congratulations! You have completed the FPGA VI, which needs to be compiled into a bit stream that will be downloaded onto the FPGA. In order to begin the compilation process, save the VI and then right-click on the VI in the LabVIEW Project and select **Compile.**



17. Once LabVIEW finishes generating intermediate files, you can minimize the compilation process and begin writing the LabVIEW Real-Time VI. In the LabVIEW Project, right-click on the CompactRIO controller and select **New»VI.**

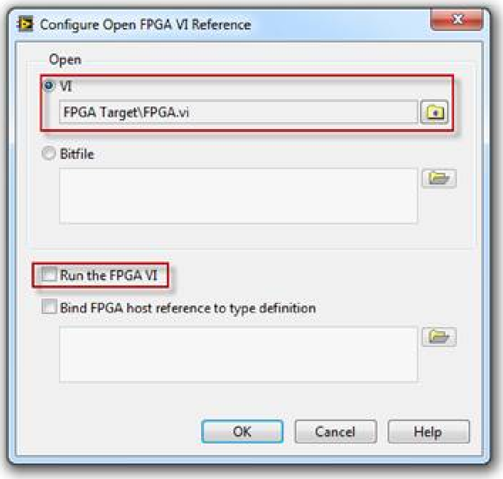


This VI will run on the real-time controller and continuously read data out of the FIFO set up between the FPGA and processor. The data will then be written to disk for offline analysis.

18. Save the VI as *RT.vi*

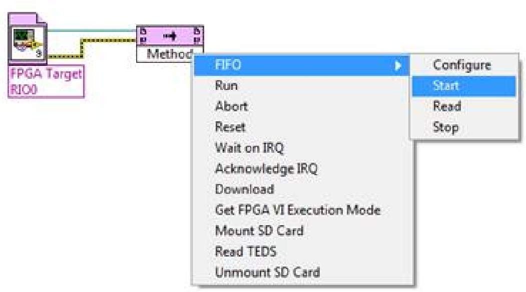
19. From the FPGA Interface palette, place an **Open FPGA VI Reference** on the *RT.vi* block diagram and double-click on it.

20. Select **VI»FPGA.vi** and uncheck **Run the FPGA VI.** This configuration will cause the function to download the FPGA VI, but not begin executing it. Click **OK.**



21. From the FPGA Interface palette, select **Invoke Method** and place it on the diagram. Wire the outputs of the Open FPGA VI Reference to the inputs of the Invoke Method.

22. Left-click on *Method* and select **FIFO»Start.** This function will initialize the DMA engine, which is powering the FIFO that between the processor and the FPGA.

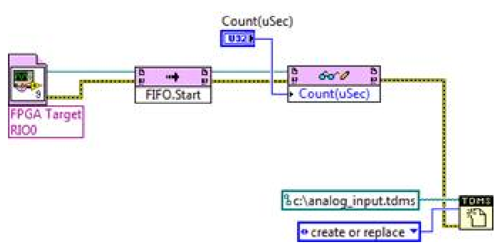


23. From the FPGA Interface palette, place a **Read/Write Control** function on the diagram. Connect the reference and error cluster wires. This function allows you to read and write to the controls and indicators on the FPGA VI.

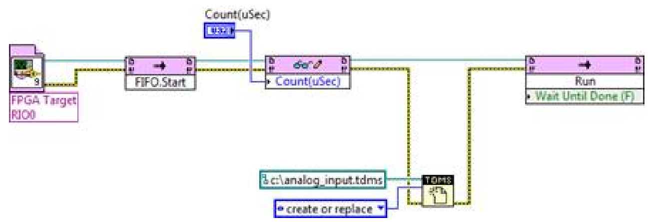
24. On the Read/Write Control, left-click *Unselected* and select **Count(uSec).** This corresponds to the Count(uSec) control on the FPGA VI front panel.

25. Create a control for the **Count(uSec)** input. This control will be used to set the analog input acquisition rate in microseconds. Enter a value of **500** for the Count(uSec) control on the front panel to set an acquisition rate of 2kHz.

26. Place a **TDMS Open** VI on the diagram. Create constants for the **file path** and **operation** inputs. Enter *analog_input.tdms* for the file name and select **create or replace** for the operation. Wire the error cluster from the Read/Write Control to TDMS Open. This function will create a new TDMS file where the data will be logged.



27. Place another **Invoke Method** function on the diagram and select the **Run** method. Wire it as shown. The Run method causes the FPGA VI to begin executing.

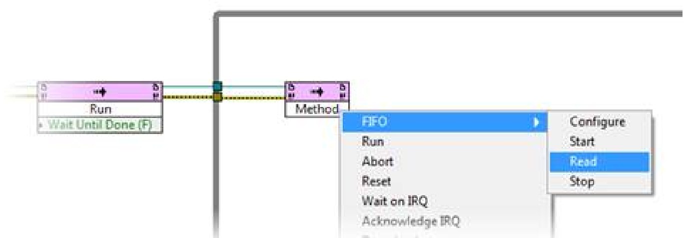


The code written so far downloads your FPGA VI, starts the DMA engine to prepare for high-speed data transfer through the FIFO, sets the acquisition loop rate on the FPGA, creates a new file for logging, and then starts the FPGA VI. Next, you will write the code to continuously read data from the FIFO and write it to disk.

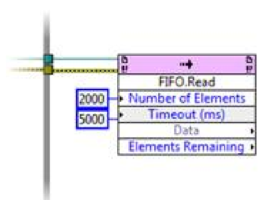
28. Place a **while loop** to the right of the existing code and place an **Invoke Method** function, from the FPGA Interface palette, in the while loop.

29. Wire the **FPGA VI Reference Out** and **error out** terminals of the Run method to the Invoke Method inputs.

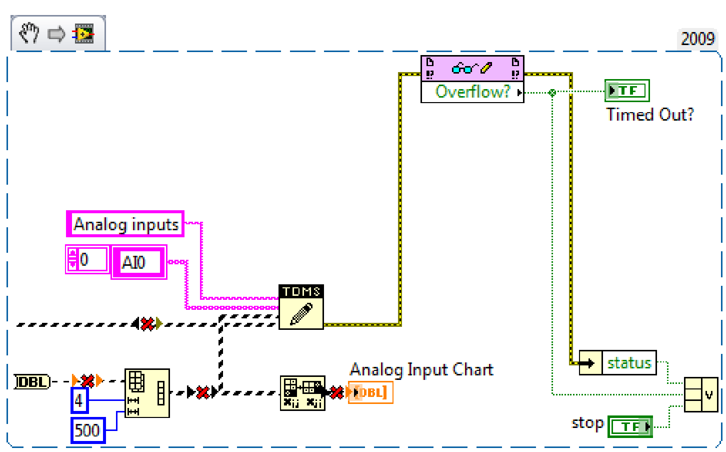
30. Left-click *Method* and select **FIFO»Read.**



31. Create constants for the **Number of Elements** and **Timeout (ms)** inputs of the FIFO.Read function. Enter **2000** for Number of Elements, which will cause the FIFO.Read function to wait until at least 2000 samples are available before executing.



32. A snippet of code is provided to perform the data logging and stop condition for the while loop. You can either drag and drop the image below to your block diagram or download the code here and copy and paste its contents to your diagram.



Drag and drop this VI Snippet to your block diagram

33. Connect the provided code snippet to your diagram as shown below.

a. Wire the **tdms file out** reference of the TDMS Open VI to the **tdms file** input of the TDMS Write VI

b. Wire the **FPGA VI Reference Out** output of the FIFO.Read function to the **FPGA VI Reference In** input of the Overwrite? function.

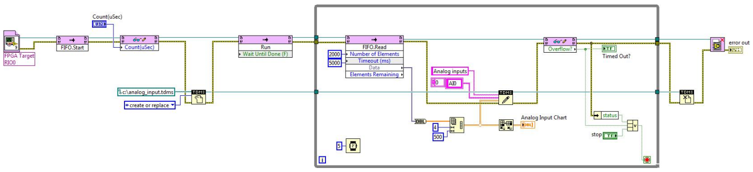
c. Wire the **error out** output of the FIFO.Read function to the **error in** input of the TDMS Write.

d. Wire the **Data** output of the FIFO.Read function to the **number** input of the To Double Precision Float function.

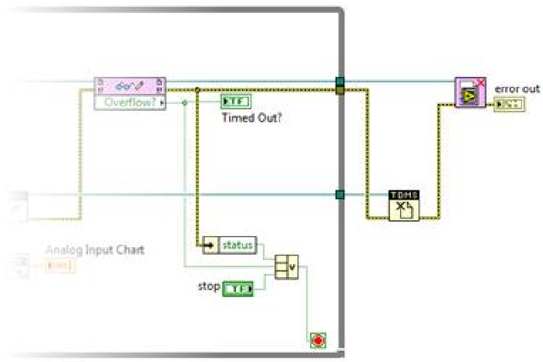
e. Wire the **result** output of the Compound Arithmetic OR function to the **stop condition** of the while loop.

34. Place a **Wait (ms)** function in the while loop and create a constant with a value of **5** for the **milliseconds to wait** input. This introduces some sleep time for the processor to perform other tasks each iteration.

The while loop will now continuously read data from the FIFO, format it, write it to the TDMS file, display it on a waveform chart, and then check to see if the overflow condition on the FPGA VI is true. The while loop will stop if the stop button is pressed, an error occurs, or if the overflow condition is met. To complete the application, you will add code to close the TDMS File and FPGA VI.

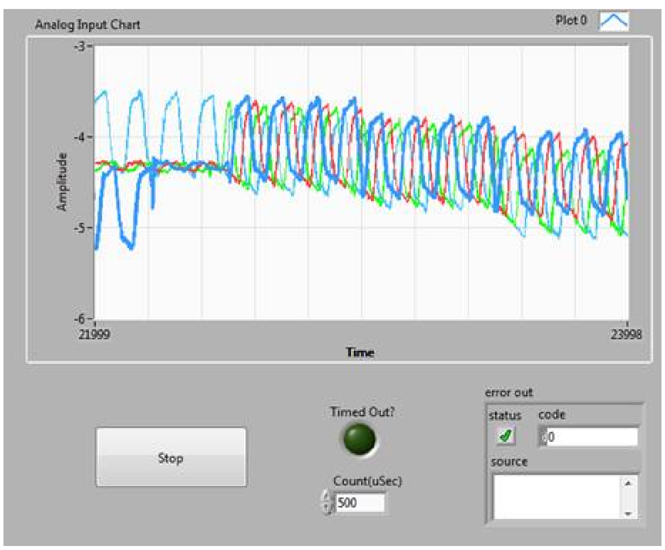


35. Place a **TDMS Close** VI and a **Close FPGA VI Reference** VI outside of the while loop and wire the file reference, FPGA reference, and error clusters as shown.



36. Click **Run** on *RT.vi*, click **Save** for any unsaved items, and click **OK** on any dialogs or warnings about applying changes to the CompactRIO system. LabVIEW will now deploy your VI over Ethernet to run embedded on the CompactRIO system.

37. Once the VI deploys and begins running, view the front panel of your VI to see the current I/O values plotted on the waveform chart.



38. Click **STOP** once you are finished viewing and logging data.

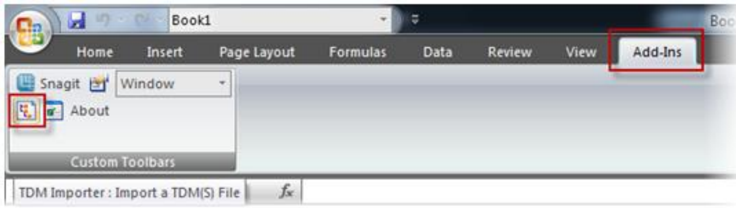
Congratulations! You have successfully created an embedded logging application with LabVIEW and CompactRIO. To continue learning, check out the additional resources on the CompactRIO Setup and Services page.

Note: Perform the following to retrieve and view the data logged on the CompactRIO system:

1. Using Windows Explorer or a Web browser, navigate to *ftp://<ip address>* where *<ip address>* is the IP address of your CompactRIO system

2. Download *tempdata.tdms* (or whatever you named the TDMS file)

3. If you have Microsoft Excel, you can view TDMS files by selecting the **Add-Ins** tab and clicking the **TDM Importer.** The first page of the workbook contains file information and the remaining sheets contain the channel data. *The TDM Importer installs with LabVIEW by default; however, if the TDM Importer is not present, you can install it from here.*



4. *If you do not have Excel, you can view the TDMS file using the **TDMS File Viewer** VI in LabVIEW.*

View Additional CompactRIO Development Resources

Congratulations on completing this tutorial! You are one step closer to becoming a skilled CompactRIO developer.

To continue learning, check out the additional resources on the CompactRIO Setup and Services page.

Discussion and Feedback

We do not regularly monitor Reader Comments posted on this page.

Please submit your feedback through the discussion forum thread so that we can improve this content.

Please direct support questions to NI Technical Support .

Downloads

fpgalogging.vi

criologgingsolution09.zip

scanmodelogging.vi