2011 NI TECHNICAL SYMPOSIUM

Hands-On: CompactRIO Part I

Programming with LabVIEW Real-Time
LabVIEW
• Powerful Graphical Programming language
• >20 years old
• Intuitive Development Environment
  • Control
  • Analysis
  • Logging
• Complex industrial measurement and control applications
• Used to program the entire system
  • Controller
  • HMI
  • FPGA
Three main components to any cRIO system

• RT Processor – Deterministic
  • Microsecond Decision Making
  • Options include USB, Serial, Dual-Ethernet, Triggering, etc.
  • Volatile and non volatile-memory

• FPGA/Backplane – Powerful, Rugged
  • True parallelism
  • Nano-Second Decision Making
  • Reliability
  • Types
    • **Virtex-6
    • Virtex-5
    • Spartan

• I/O Modules – Versatility
  • >50 C series modules for cRIO
  • Plug and Play
  • Almost any sensor or actuator
  • Open Platform for third party or custom built modules
  • Examples of NI module types
    • Thermocouple
    • Analog Input (Example: ±10 V Simultaneous Sampled 24-bit)
    • Industrial DIO up to 1 A current drive
    • Strain
    • RTD
    • Analog Output
      • 24-bit IEPE Accelerometer
      • Power Measurements
      • CAN (controller area network)
      • SD Card Logging
• cRIO is FLEXIBLE!
• FPGA enables cRIO to adapt to many different types of input and control.
• Easily adapted to wide range of applications.
• CompactRIO has been a very successful platform for National Instruments and has been used in a myriad of applications including:
  • Advanced Hydraulic Control
  • Process Optimization
  • Environmental Monitoring
  • Industrial Control
  • Structural Monitoring
  • Unmanned Systems
  • FDA approved medical devices
Typical programming model in cRIO:
- I/O <-> FPGA <-> RT
- I/O: Acquire or Output I/O
- FPGA: Send data to FPGA, perform processing, output data
- RT: Send data to Real Time for more processing, UI, network transfer, file I/O

Getting Started: Scan Interface is QUICK and EASY!

We have added a scan engine to LabVIEW and CompactRIO to make it easier to program I/O while retaining the flexibility of FPGAs.
It is a set of special IP that NI has written for you, which allows you to synchronize acquisition and output, then sends it to the scan engine on LabVIEW Real-Time to scale and provide to the user program.

This feature allows users to quickly program I/O and also take advantage of the flexibility of the FPGA. For example, the RIO Scan interface can be configured to use digital channels on a module to decode quadrature encoder signals or to serve as counter.

Users can also configure a module to be programmed directly on the FPGA, allowing them to use the FPGA for faster control and custom signal processing. This gives users the best of both worlds in programming on cRIO, a quick and easy way to get I/O, and a way to go deep and program the FPGA directly.
Exercise 1
CompactRIO Scan Mode Basics
Exercise 1 – CompactRIO Scan Mode Basics

1. Select **Empty Project** from the LabVIEW Getting Started window.

2. Save the project as **CompactRIO Scan Mode.lvproj** in \C:\NITS\cRIO 1 RT\Hands On work\.

3. Right-click on the project item and select **New » Targets and Devices...**

4. Expand **Real-Time CompactRIO**, select **cRIOHandsOn** and click **OK**.

5. Select **Scan Interface** and click **OK**. LabVIEW will now detect your I/O modules and add them to the project.

6. Configure the NI Scan Engine from the cRIOHandsOn properties page. Right-click cRIOHandsOn and select **Properties**...
7. Select **Scan Engine** from the Category field. Configure the **Scan Period** for **100ms** and click **OK**. This setting tells the NI Scan Engine to update I/O every 100 milliseconds.

8. Right-click **cRIOHandsOn** and select **Add » File...**

9. Add **Temperature threshold.vi** located at:

   C:\NITS\cRIO 1 RT\LabVIEW Real-Time and CompactRIO Hands On\Exercises\CompactRIO Scan Mode basics\

10. Open **Temperature threshold.vi** and press `<Ctrl+E>` to view the block diagram. The UI has been completed for you.
11. Expand **Mod1 (Slot 1, NI 9211)** in the LabVIEW Project. AI0 through AI3 are I/O variables that allow you to read scaled calibrated I/O directly in LabVIEW Real-Time with no FPGA programming or compiling.

![LabVIEW Project Explorer](image)

12. Rename **AI0** to **Temp** and drag-and-drop it to the block diagram. Wire the Temp I/O variable to the **Temperature** chart and to the **Greater?** function. This code monitors and displays the temperature, while determining if it has exceeded a threshold.

![Block Diagram](image)

13. Expand **Mod2 (Slot 2, NI 9474)** in the project. DO0 through DO7 are I/O variables that allow you to write to the NI 9474 digital lines.

14. To control the intensity of the heat lamp, we will use a 24V pulse width modulation signal (PWM). CompactRIO Scan Mode supports PWM, counter, and quadrature encoder input functionality for any eight-channel digital NI C Series Module. Right-click **Mod2 (Slot 2, NI 9474)** and select **Properties**...
15. From the Category list on the left, select **Specialty Digital Configuration**. Select **Pulse-Width Modulation** from the Specialty Mode field and set the Frequency (Period) to **1kHz** for channel PWM0 and click **OK**. Notice that the I/O variables for the NI 9474 repopulate as PWM channels.

![Specialty Digital Configuration](image1)

16. Rename **PWM0** to **Lamp intensity** and drag-and-drop it to the block diagram. Wire the output of the **Select** function to the Lamp intensity I/O variable.

![Block Diagram](image2)

17. Expand **Mod3 (Slot 3, NI 9423)** in the project. DO0 through DO7 are I/O variables that allow you to read the NI 9423 digital lines. Rename **DI0** to **Enable Lamp** and drag-and-drop it to the block diagram. Wire the Enable Lamp I/O variable to the input of the **Select** function and to the **Lamp Enable** indicator.

![Block Diagram](image3)
18. To synchronize your code to the NI Scan Engine, use a timed loop. Place all of the code in a **timed loop** by right-clicking the block diagram and selecting **Programming » Structures » Timed Structures » Timed Loop**. Wire the **stop** control to the **stop terminal**.

19. Double-click the timed loop input node to open the **Configure Timed Loop** dialog.

20. Select **Synchronize to Scan Engine** in the Loop Timing Source field and click **OK**. This synchronizes your code to the NI Scan Engine, causing it to run once, each time the I/O variable values are updated.
21. Place an instance of the **Lamp intensity** I/O variable outside the timed loop and wire a value of 0 to the input. This turns off the lamp when the loop stops.

![Lamp intensity I/O variable](image)

22. Save and **Run** Temperature threshold.vi. Click **OK** and **Apply** to any deployment dialog windows.

23. Use the far right red hardware button on the demo box to enable control of the lamp. Then use the **Lamp Power** slide to increase the lamp brightness.

**NI Distributed System Manager**

1. In the LabVIEW Project, right-click on **cRIOHandsOn** and select **Utilities » View in System Manager**. The NI Distributed System Manager allows you to remotely monitor CompactRIO I/O values, force I/O values, and monitor system performance.

![NI Distributed System Manager](image)

2. Expand the controller and **Mod1** to remotely monitor the I/O values. Choose **Temp** to see a graph of the current temperature trend.
3. Select the CompactRIO controller (10.0.0.50) in the left window and select the **CPU/Memory** tab to monitor CPU load and memory usage.

4. CompactRIO Scan Mode adds I/O forcing, allowing you to override physical input values and program output values. In Temperature threshold.vi set the **Temp Threshold** to **99** degrees.

5. In the system manager, select the **Scan Engine** tab and click **Enable Forcing** if it is not enabled.

6. Select **Temp** under Mod1. Enter **100** in the New Value field and click **Force**. Notice that the system manager now reports a value of 100 for Temp. All instance of the Temp I/O variable also read a value of 100, including Temperature threshold.vi. The **Over threshold?** indicator now
shows that the temperature exceeds 99 degrees. I/O forcing can be used for debugging and troubleshooting. Try forcing other values in your application.

7. Press Unforce to return the I/O Variable to hardware control.

End of Exercise 1
FPGA may be the power house, but RT is great too. Check out the cool features shown here!!

Key Points
• Micro-Second decisions
• Priorities and Scheduling provide determinism
• Synchronize to scan engine for tight I/O
• Access to most LabVIEW functions
• Control Design and Simulation Module
• RT FIFOs to prevent jitter and allow communication between loops
• Network Communication (Network Streams)
Exercise 2
PID Control
Exercise 2 – Closed Loop PID Control with Scan Mode

You will now use the basic concepts of CompactRIO Scan Mode and new LabVIEW function blocks to implement a closed loop PID control system. The temperature chamber in the demo box can be heated with the lamp and cooled with the fan. You will use the NI 9474 to control each of these with a PWM signal. The PID function block will be used to perform the control algorithm and provide remote monitoring and tuning of the system.

1. Close open VIs and projects. Open PID Control.lvproj from:
   
   C:\NITS\cRIO 1 RT\LabVIEW Real-Time and CompactRIO Hands On\Exercises\PID Control\  

2. Open PID Control.vi from the project window  

3. Expand Mod1, Mod2, and Mod3 in the project and drag Temp (AI0), Lamp (PWM0), Fan (PWM1), and Enable Control (DI0) to the block diagram  

4. Wire the Enable Control I/O variable to Control Enabled and to the Select function. Place the Lamp I/O variable inside the True case of the case structure and the Fan I/O variable inside the False case  

5. Place instances of the Lamp and Fan I/O variables outside the timed loop with a shutdown value of 0  

6. Right-click on the block diagram to open functions palette, and select Real-Time » Function Blocks » Control » PID (or use quick drop by pressing <Ctrl+Space> and typing PID [NI_FunctionBlockMergeVIs.lvlib]) and place the PID function block on the block diagram.  

7. Notice that the PID function block is added to the LabVIEW Project under PID Control.vi. Rename the function block to Temp chamber PID by right-clicking the function block project
item and selecting **Rename**. Each function block is reentrant with its own memory space and has a unique name.

8. Expand **Temp chamber PID** to view its parameter variables. These parameters are single process shared variables with RT FIFO enabled. This gives you jitter free access to all parameter values throughout your application. Drag the **setpoint** and **process variable** variables to the normal while loop on the block diagram.

9. Wire **setpoint** and **process variable** to the **bundle** function. This while loop executes at a lower priority than the timed loop and is responsible for updating the chart on the front panel. Removing user interface (UI) components from your control loops maintains determinism.

10. Function block parameter variables can be configured for various use cases. Double-click the **Temp chamber PID** function block on the block diagram to open the Function Block Properties dialog.

11. For **setpoint** and **process variable** change Variable Scope to **Network-published**

12. For **proportional gain (Kc)**, **integral time (Ti, min)**, and **derivative time (Td, min)** uncheck **Visible?**, change Variable Scope to **Network-published**, and change Data Source to **Variable**. Change the default **proportional gain (Kc)** value to **10** and click **OK** (see below).
13. Wire the **Setpoint** control and **Temp** I/O variable to the **setpoint** and **process variable** inputs of the Temp chamber PID function block, respectively. Wire the **output** of the function block to the **t** input of the **Select** function. Your completed block diagram should appear as below.

14. Synchronize the PID control code to the NI Scan Engine by double-clicking the timed loop input node and selecting **Synchronize to Scan Engine** as the Loop Timing Source.

15. Save and **Run** the VI. Click **OK** and **Apply** to deployment dialogs.

16. Press the **hardware button** on the demo box to enable PID control. View the front panel of PID Control.vi and adjust the **Setpoint** control. The PID algorithm will attempt to reach and hold the setpoint by powering the lamp and the fan.
1. In the LabVIEW Project, right-click on cRIOHandsOn and select Utilities » View in System Manager.

2. Expand your CompactRIO controller then PID Control.vi » Temp chamber PID. Listed are the PID parameter variables configured as network published.

3. Right-click on process variable and select Probe. Use the probe to monitor the process variable as you tune the PID gain settings.

4. Select proportional gain (Kc) and enter a value of 100. Click Set. Since the proportional gain (Kc) input of the Temp chamber PID function block has the Data source field configured as Variable, it can be updated here in the system manager, or from writing to the variable in a VI.

5. Modify the proportional gain (Kc) and integral time (Ti, min) variables in the list to find the optimal PID gains for the system.

End of Exercise 2
Scan Engine is great for certain use cases. Here we have a graph that displays RT CPU usage as a function of the number of channels being read by the scan engine.

Here we see the usage of a high channel count, low update rate application. Even at 80 channels, only around 25% CPU usage! Perfect use case for Scan Interface!

Here we see the usage of a two slightly higher update rates. CPU usage starts to jump as we gain more channels, and this is not ideal. These are cases that may be more suited to using FPGA.
A good portion of our cRIO and C-series product lines are compatible with the Scan Engine, including many controllers and chassis, and most standard DAQ (AI,AO,DIO) modules.

However, some of our specialty modules (motor drives, CAN, Serial) are not supported, and some of the older controllers/chassis are unsupported as well.
Optional Exercise – Using LabVIEW FPGA with CompactRIO Scan Mode

In this exercise you use the NI 9211 and 9274 in scan mode to read temperature from the thermocouple and control the lamp. Using LabVIEW FPGA, you will read an acceleration input of a balanced and unbalanced fan, calculate the mean square of the acceleration and use a DMA FIFO to stream the result back to your real-time VI. The VIs have been completed and compiled for you. You only need to configure the project.

1. Open **Using LabVIEW FPGA with scan mode.lvproj** from

   `C:\NITS\cRIO 1 RT\LabVIEW Real-Time and CompactRIO Hands On\Exercises\LabVIEW FPGA with scan mode`

2. Add the FPGA target to your project by right-clicking on the **Chassis** item and selecting **New » FPGA Target**

3. Select **Deploy Now**. This puts the CompactRIO chassis in LabVIEW FPGA Mode, so that it is expecting a custom FPGA VI to be downloaded and run, instead of using the scan mode FPGA code.

4. Drag-and-drop **Mod5 (Slot 5, NI 9234)** onto the FPGA target in the project. This gives you the ability to program the NI 9234 directly in LabVIEW FPGA, while the NI 9211 (Mod1) and NI 9274 (Mod2) remain in scan mode allowing you access to I/O variables without programming the FPGA.
5. An FPGA VI has already been created and compiled for you. Add FPGA VI.vi to the project by right-clicking on the FPGA Target item and selecting Add » File... and browsing to:
C:\NITS\cRIO 1 RT\LabVIEW Real-Time and CompactRIO Hands On\Exercises\LabVIEW FPGA with scan mode\FPGA VI.vi

6. Open FPGA VI.vi and view the block diagram. The VI is broken because the FIFO does not exist in the project yet.

7. Right-click on the FPGA Target item in the project and select New » FIFO. Configure the FIFO with the following options as show below and click OK:
   - General Category
     i. Name: FIFO
     ii. Type: Target to Host – DMA
   - Data Type Category
     i. Data Type: FXP
     ii. Encoding: Unsigned
     iii. Word length: 47 bits
     iv. Integer word length: 7 bits
8. As shown below, this FPGA VI reads the accelerometer analog input of the NI 9234 and calculates the mean square of the acceleration. The resulting values are then streamed to the real-time VI with a DMA FIFO. The FIFO is also monitored for overflow.

When this FPGA VI is compiled, the scan mode FPGA code for the NI 9211 and 9274 is combined with the FPGA VI and one bit file is created. This bit file supports both the custom FPGA VI created and the scan mode features for any modules under the Chassis item at compile time.
9. From the project, open the LabVIEW Real-Time VI called RT VI.vi. This VI opens a reference to the FPGA VI and deploys the bit file. The timed loop is synchronized to the scan engine and reads from the I/O variable associated with the NI 9211 and writes to the I/O variable associated with the NI 9274. The regular while loop reads the DMA FIFO from the FPGA VI.

10. Run RT VI.vi and vary the speed of the fan with the fan control knob on the Sound and Vibration Signal Simulator. Notice that the mean square acceleration increases with the vibration of the fan. Use the black switch above the fans on the Sound and Vibration Signal Simulator to switch between a balanced and unbalanced fan to see further changes in vibration. Adjust the Lamp power with the slider on the front panel to change the temperature read by the thermocouple.

Switching back to scan mode...

1. The bit file created when using LabVIEW FPGA with scan mode fixes each slot in the chassis to be accessed with either scan mode or LabVIEW FPGA, depending on the configuration in the project at compile time. Modules under the Chassis are in scan mode and modules under the FPGA are in LabVIEW FPGA Mode.

2. To put the chassis and all modules back in to scan mode, right-click on the Chassis project item and select Properties.... Select Scan Interface and click OK.

3. Drag any modules you want to access in scan mode under the Chassis item in the project. Right-click on the Chassis project item and select Deploy. Now you can access all modules under the Chassis item with I/O variables without deploying an FPGA VI. You can now remove the FPGA target from the project.

End of Exercise