Embedded Programming

Preparing to Program

1. Ensure that your NI CompactRIO chassis is configured with the NI 9201 analog input module in Slot 1, the NI 9263 analog output module in Slot 2, and the NI 9401 digital I/O module in Slot 3.

2. Launch LabVIEW to get started.

3. To program the CompactRIO, use a LabVIEW project. You can create a new LabVIEW project by choosing “Create Project” from the LabVIEW Getting Started screen.
4. Select a blank project and save the project as “Learn RIO.”

5. LabVIEW projects are organized in a hierarchical fashion. “My Computer” refers to the machine that you are developing on, and any VIs placed under My Computer in the project run on the development machine. Because CompactRIO has its own processor, it is essentially another computer. To write code for that computer, or target, add the device to the project.

Right-click on the project title and choose New»Targets and Devices. This adds CompactRIO as a target in the project.
6. Select “Existing Target or Device.” LabVIEW then detects the connected device for you. Expand the folder called “Real-Time CompactRIO,” select your CompactRIO chassis, and click “OK.” Note that choosing “Existing Target or Device” indicates that you plan to program for a RIO target that is connected to your system. If you do not have access to hardware but still want to program for a RIO target, you can choose “New Target or Device” to develop code for hardware that you will connect to later.
7. A dialogue appears asking you if you are using the RIO Scan Interface or LabVIEW FPGA Interface. Choose LabVIEW FPGA Interface.

Use the RIO Scan Interface when you only need to write to inputs and outputs and do not need the functionality of the FPGA.

Use the LabVIEW FPGA Interface when you want to take advantage of the FPGA. This allows you to account for safety-critical functionality and achieve very high rates of analysis on the hardware, among other benefits that may be useful to your application.
Programming Architecture

Before programming, think about the best way to implement your ideas. With RIO technology, you can place functionality on the FPGA as well as on the real-time processor. You need to decide what goes on the FPGA versus the real-time processor. This choice depends on your application, but there are a few things to remember.

An FPGA can be thought of as a circuit. Anything you want a circuit to do should go on the FPGA like accessing I/O, I/O manipulation, signal processing, or emergency stop functionality. Tasks that a computer normally implements such as file management or logging should go on the real-time processor. It’s also much easier to rapidly create and test ideas on the real-time processor than on the FPGA because code running on the real-time processor does not have a compilation delay, and it can use floating-point numbers. Complex user interfaces should be kept on the host computer so that your real-time code can run as quickly as possible without needing to update a user interface.

A LabVIEW project is arranged in a hierarchy that makes understanding the LabVIEW RIO architecture intuitive. CompactRIO is a computer as is the machine you are developing on, so they are at the same level in the hierarchy.

1. Right-click on My Computer and select New»VI. This opens a new VI.
2. Save this VI as “My Computer.” This VI runs on the PC you are developing on, not on CompactRIO, because it falls under “My Computer” in the project hierarchy.

3. To create a new VI to run on the CompactRIO processor, right-click the processor and choose *New»VI.*
4. Save this VI as “RT” for real time. It runs on the real-time processor on CompactRIO because it falls under CompactRIO in the hierarchy.

5. To create a VI for programming the FPGA, right-click on FPGA Target and choose New»VI.
6. Save this VI as “FPGA.” It is deployed to the FPGA because it falls under the FPGA in the hierarchy.

You can drag and drop VIs throughout the project to change their execution targets.
Accessing I/O Through the FPGA

On RIO targets, all I/O goes through the FPGA. In this tutorial, FPGA Interface Mode is used; therefore, you must program the FPGA to access I/O. This tutorial guides you through reading and writing to some analog and digital inputs and outputs.

1. Open the “FPGA.vi” created in Step 5 of Programming Architecture.
2. To access I/O, right-click on the block diagram and navigate to the FPGA I/O palette.
3. Place an FPGA I/O Node onto the block diagram.
4. Click on the I/O node and choose Mod3»DIO0. LabVIEW detects all modules in the chassis, so you can choose from any I/O through the FPGA I/O Node.

Mod3 means that this module is in the third slot of the chassis, and DIO0 refers to a specific digital line of Mod3. Right now, the terminal is on the right side of the node, so this is set up to read the digital line.
5. To write to the digital line, right-click the I/O node and choose “Change to Write.”

6. Create a control from this I/O node by right-clicking the node, selecting Create»Control, and labeling it “Digital Write.”

7. Hover over the bottom edge of the I/O node and drag it down to reveal one more I/O element.

8. Select Mod3»DIO4.

9. Right-click on “Mod3/DIO4” and select “Change to Read.”

10. Create an indicator for this digital line by right-clicking and selecting Create»Indicator and call it “Digital Read.”

The NI 9401 digital module used in this tutorial is bidirectional. This means that the digital lines on this module can be set to input or output.
11. In the LabVIEW Project Explorer window, right-click on Mod3 in the project and select “Properties.” Here you can choose which lines to set to input and which to output.
12. Set lines 3-0 to output and 7-4 to input.
13. Right-click on the block diagram. From the Structures palette, choose Time Structures » Timed Loop and place it around the code.

In FPGA programming, Timed Loops are referred to as single-cycle Timed Loops because everything in the loop executes in one tick of the FPGA clock. By default, the FPGA clock is set to 40 MHz as seen in the LabVIEW project. A single-cycle Timed Loop is special because it guarantees timing. This can be useful if you need to keep track of time as in the case of a PWM.
Everything placed inside a single-cycle Timed Loop executes completely every 25 ns. If too many items are placed in this loop so that it cannot execute in one clock tick, the FPGA VI does not compile and you must rethink your code.

14. Drop another FPGA I/O Node onto the block diagram and choose Mod1»AI0.
15. Create an indicator for this analog channel called “Analog In.”
16. Drag down the I/O node by hovering over the bottom edge to reveal one more I/O element.
17. Choose AO0 from Mod2.
18. Create a control for this analog output channel called “Analog Out.”

Notice the light purple color of the wires going into and out of the analog channels. This wire color indicates a data type called fixed point, and this data type is commonly used in FPGA programming. FPGAs have fixed sizes, so only so much can fit on an FPGA. With fixed point, you can dictate how many bits to use to represent a number. If a particular number needs more bits and therefore more precision in an application, you can allow it to take up more space by right-clicking a control, an indicator, or a function and selecting Properties. With the Data Type tab, you can adjust these parameters.
In basic applications, you do not need to be concerned about this because LabVIEW has algorithms built in to handle this. But in more complex applications where FPGA space is a concern, you may need to manage your data representation. You can set word length, which is the number of bits dedicated to the entire number, and the Integer word length, which is the total number of bits dedicated to representing the part of that number in front of the decimal place.
19. Place a While Loop around the analog code by right-clicking and selecting **Structures>While Loop**. The analog-to-digital converters of analog modules cannot go fast enough to allow a single-cycle Timed Loop to execute in one clock tick, so a While Loop must be used in this case.

20. Create a constant for each loop condition terminal. Notice that the constants are “False” by default. This is desirable in FPGA programming because you want logic added to the FPGA to continuously execute. As long as the FPGA has power, the circuit you are programming runs. This is a major benefit of FPGAs.
21. To compile the code, right-click the FPGA VI in the project hierarchy (LabVIEW Project Explorer) and click “Create Build Specification.”
22. Right-click the build specification you just created and choose **Properties** to see some of the options you have such as naming and source files.

23. Select **“Build.”** (When prompted, select Local Compile Server for the purposes of this tutorial.)
Programming the Processor and Interacting With the FPGA

Once your FPGA code has compiled, you are ready to begin creating code to run on the CompactRIO processor. Remember, the real-time processor is where you want to implement file I/O and analysis or control algorithms with floating-point numbers.

1. Open the RT.vi created in Step 3 of Programming Architecture and switch to the block diagram by selecting Window » Show Block Diagram.

The FPGA Interface palette contains the VIs needed to interact with the functionality placed on the FPGA.

2. Place an Open FPGA VI Reference on the block diagram. With this VI, you can specify the FPGA functionality you have previously coded for the FPGA.
3. Right-click on the Open FPGA VI Reference and choose “Configure Open FPGA VI Reference.” Select the VI check box and choose FPGA.vi.

![Configure Open FPGA VI Reference](image)

4. Place down a **Read/Write Control** from the **FPGA Interface palette**.

5. Wire the reference wire and error wire from the Open FPGA VI Reference to the Read/Write Control. Once the reference is wired from the Open FPGA VI Reference to the Read/Write Control, the Read/Write Control is then associated with the particular functionality that has been placed on the FPGA.

![Read/Write Control](image)

6. Left-click on the **Read/Write Control** and choose the analog output channel defined earlier on the FPGA.
7. For this tutorial, you will write a basic program to prove that the architecture is working.

At this point in a typical application, you might do some analysis or write to a file with this data, among other things. For this tutorial, write a basic program to prove that the architecture is working.

8. Switch to the front panel and drop down a vertical toggle switch, round LED, knob, and numeric indicator.

10. On the block diagram, wire the toggle switch to the digital write, the LED to the digital read, the knob to analog output, and the numeric indicator to analog input.

11. Place the Read/Write Control and the associated controls and indicators in a Timed Loop found on the Structures palette.
12. Double-click on the Timed Loop to see some of the unique timing options it provides.

A Timed Loop can be tied to several different clocks to dictate timing. Additionally, a “priority” can be set with a Timed Loop. If multiple loops (Timed Loops or otherwise) are placed on a block diagram, you may want to prioritize some loops over others, meaning if LabVIEW has to make a choice as to which loop must execute on time, it prioritizes based on the settings specified in this dialogue. You can create a mix of Timed Loops and other loops on the block diagram. Timed Loops always have higher priority over other loops. Click “Cancel” when you have finished exploring.
13. Outside the loop, place down a Close FPGA VI Reference and wire the error wire and reference wire from the Read/Write Control to the Close FPGA VI reference.

14. Create a Stop button for the loop by right-clicking on the input to the Loop Condition and selecting Create » Control.

15. To verify that the code is working, wire AO0 to AI0 and DIO0 to DIO4 on your CompactRIO hardware.

16. Run the RT.vi.

17. Move the knob that writes voltage to the analog out module to see how the same value is read and displayed on the numeric indicator.

18. Toggle the switch to see how the LED on the front panel switches on and off.

Generally in embedded programming, you want your applications to be headless (meaning they run unattached from a development PC with no user interface). When the development PC is physically connected to a target, LabVIEW hosts a user interface on that development PC when you run code with the Run button. This helps you see what is happening in your code. This is beneficial for debugging, tuning, and in cases like the simple example shown in the tutorial. The only way to know this code is working is through this development PC’s hosted user interface, but it should be noted that this degrades your target’s performance. If you need a user interface in your application, you can set up networking so devices other than the development PC with LabVIEW can see target data and so you can manage target performance. Learn more about these techniques.