Contents

CHAPTER 7
Adding I/O to the CompactRIO System

Adding I/O to CompactRIO

NI products based on the LabVIEW reconfigurable I/O (RIO) architecture are increasingly adopted for system-level applications that require high-channel counts, intensive processing, and distributed I/O. Adding RIO expansion I/O to the NI RIO product offering enables a 1:N system topology with one controller and many FPGA I/O nodes for flexible, high-channel-count systems that can perform both distributed control and centralized processing.

NI offers three types of expansion chassis systems for expanding your I/O. Each expansion chassis features an FPGA and is based on the same C Series module architecture. The main differences between expansion I/O options stem from the choice of bus, with each bus being best suited for a subset of expansion I/O applications. The NI I/O expansion chassis systems include the following:

- MXI-Express RIO
- Ethernet RIO
- EtherCAT RIO

Figure 7.1. Expand your CompactRIO I/O with NI expansion chassis.

Each expansion chassis has unique capabilities. Choose an expansion chassis depending on your application requirements. Table 7.1 provides a quick comparison of the three different types of expansion chassis.

	MXI-Express RIO	Ethernet RIO	EtherCAT RIO
	NI 9157, 9159	NI 9148	NI 9144
Slots	14	8	8
FPGA	Virtex-5 (LX85 or LX110)	Spartan 2M	Spartan 2M
Network Topology	Daisy chain	Same as Ethernet	Daisy chain
Distance	7 m between nodes	100 m before repeater	100 m before repeater
Multichassis Synchronization	FPGA-based DIO	FPGA-based DIO	Implicit in bus operation
Communication Jitter	<10 µs	No Spec	<1 µs
Bus Throughput	250 MB/s	100 Mbit/s	100 Mbit/s
API Support	FPGA Host Interface	FPGA Host Interface/Scan Engine	FPGA Host Interface/Scan Engine
Host	Windows/Real-Time	Windows/Real-Time	Real-Time Only

Table 7.1. Comparison of RIO Expansion Chassis

MXI-Express RIO

Featuring a high-performance Xilinx Virtex-5 FPGA and a PCI Express x1 cabled interface, an NI 9157/9159 MXI-Express RIO 14-slot chassis for C Series I/O expands the RIO platform for large applications requiring high-channel counts and a variety of signal conditioning and custom processing and control algorithms. You can use these MXI-Express RIO expansion chassis with high-performance NI cRIO-9081/9082 CompactRIO systems in addition to industrial controller and PXI systems. When using a MXI-Express RIO expansion chassis, you interface to the expansion chassis I/O within your main controller VI using LabVIEW FPGA Host Interface functions.

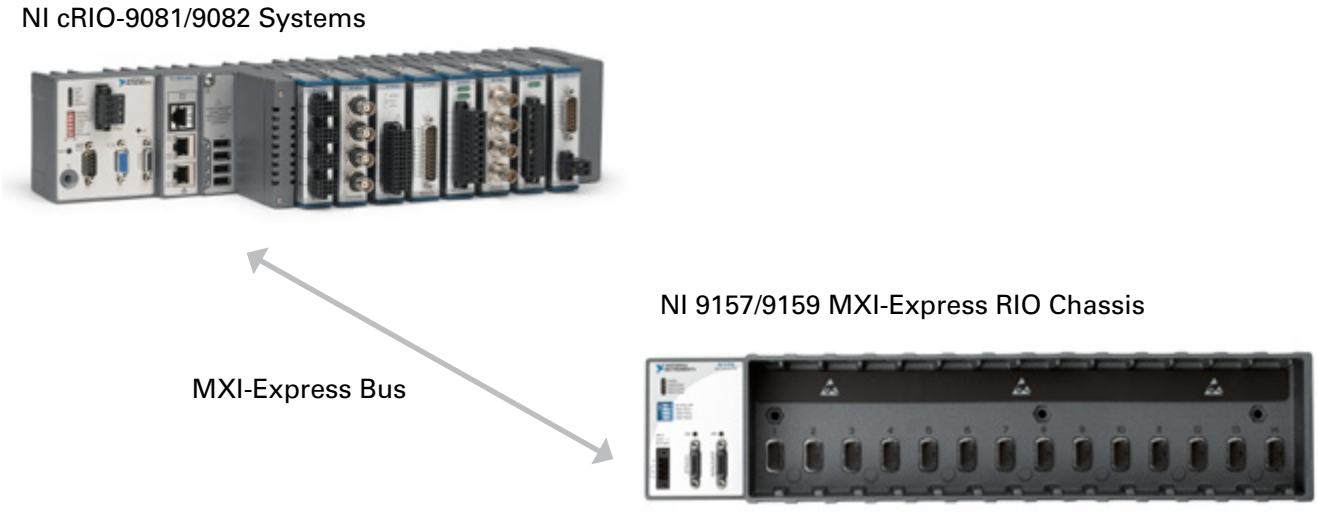


Figure 7.2. A MXI-Express RIO Chassis Paired With an NI cRIO-9081/9082 High-Performance CompactRIO System

Ethernet RIO

When expanding your I/O using the standard Ethernet protocol, you can use the NI 9148 Ethernet RIO expansion chassis. Programmers can take advantage of the existing network infrastructure such as switches and routers. Although full duplex switch networks eliminate packet collisions, switches introduce jitter, and you should use general Ethernet only in applications that do not require deterministic communication. If you need synchronization between the local I/O and expansion I/O, see the EtherCAT RIO section for more information.

Figure 7.3. The NI 9148 Ethernet RIO Expansion Chassis

When using the Ethernet RIO expansion chassis, the main controller is responsible for running the real-time control loop using I/O from its own chassis in addition to the I/O from one or more Ethernet RIO chassis. The expansion chassis provides expansion or distributed I/O for the main controller.

The Ethernet RIO expansion chassis works with both LabVIEW FPGA and the Scan Engine. If you use LabVIEW FPGA with the expansion chassis, you can embed decision-making capabilities to quickly react to the environment without host interaction. The FPGA can also offload processing from the main controller by conducting inline analysis, custom triggering, and signal manipulation.

When using LabVIEW FPGA, you should create a regular While Loop or a Timed Loop with a lower priority to handle the communication since Ethernet is nondeterministic (see Figure 7.4). This allows your control task to run deterministically and reliably because it is not affected by the possibly high-jitter I/O device. When using LabVIEW FPGA, you interface to the I/O within your real-time VI using the FPGA Host Interface functions.

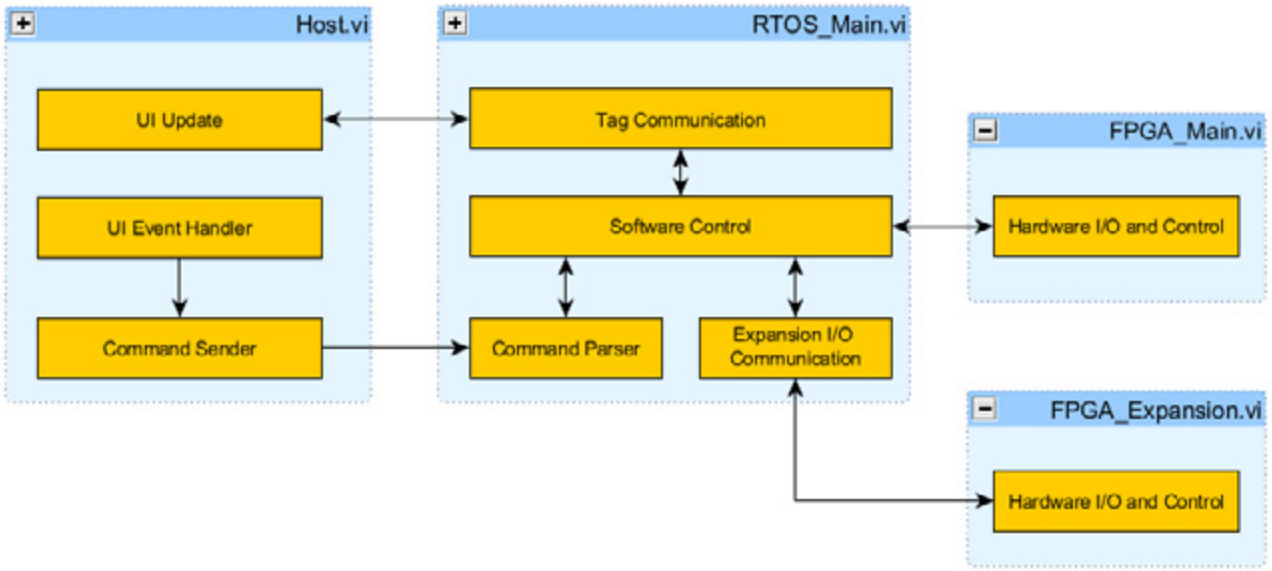


Figure 7.4. Add a new process to handle the I/O expansion task when using LabVIEW FPGA Interface Mode.

The Ethernet RIO expansion chassis also works with the Scan Engine. You have the option to choose Scan Mode or FPGA Interface Mode when adding the Ethernet RIO chassis to your LabVIEW project. When using Scan Mode, your design diagram might look like Figure 7.5, where you have all of your system I/O accessible from the Scan Engine. When using Scan Mode, you interface to the I/O within your real-time VI using the Scan Engine I/O variables.

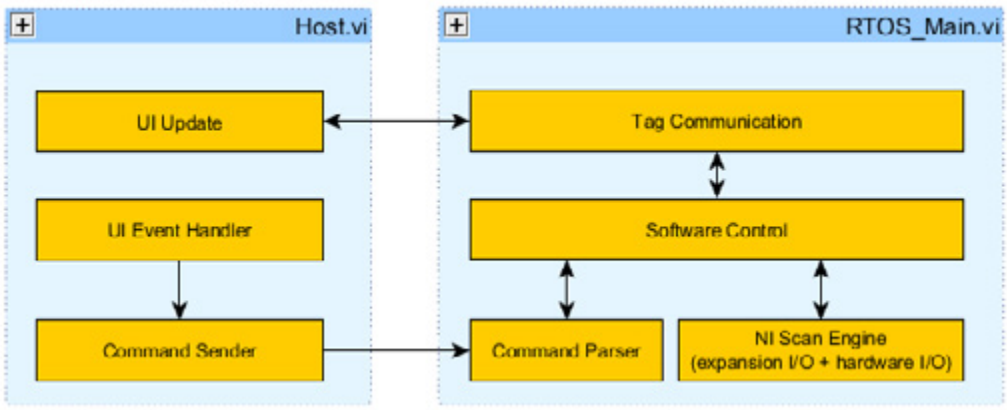


Figure 7.5. You can use the NI Scan Engine to handle I/O with the NI 9148 Ethernet RIO chassis.

To get started with the NI 9148 Ethernet RIO expansion chassis, see the NI Developer Zone tutorial Getting Started With the NI 9148 Ethernet RIO Expansion Chassis.

EtherCAT RIO

In certain applications, the main I/O and expansion I/O systems require tight synchronization—all of the inputs and outputs must be updated at the same time. Using a deterministic bus allows the main controller to know not only when the expansion I/O is updated but also exactly how long the data takes to arrive. You can easily distribute CompactRIO systems with deterministic Ethernet technology using the NI 9144 expansion chassis.

The NI 9144 is a rugged expansion chassis for expanding CompactRIO systems using a deterministic Ethernet protocol called EtherCAT. With a master-slave architecture, you can use any CompactRIO controller with two Ethernet ports as the master device and the NI 9144 as the slave device. The NI 9144 also has two ports that permit daisy chaining from the controller to expand time-critical applications.

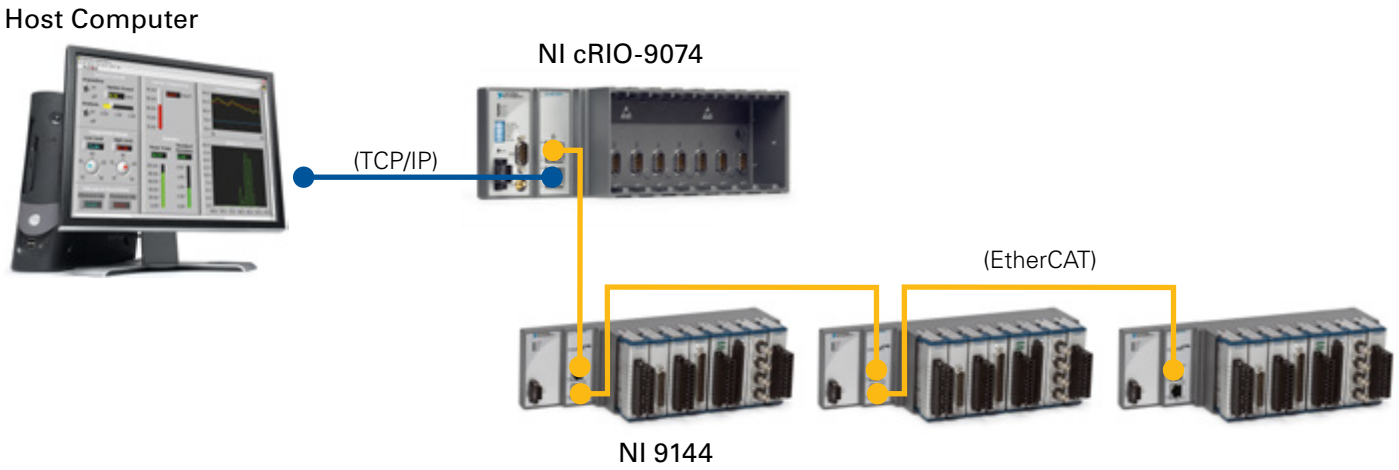


Figure 7.6. The CompactRIO hardware architecture expands time-critical systems using the NI 9144 deterministic Ethernet chassis.

Tutorial: Accessing I/O Using the EtherCAT RIO Chassis

The following step-by-step tutorial provides instructions for configuring your EtherCAT system and communicating I/O with both the Scan Engine and LabVIEW FPGA. This example uses an NI cRIO-9074 integrated system, but any CompactRIO system with two Ethernet ports can work with EtherCAT.

Step 1: Configure the Deterministic Expansion Chassis

1. To enable EtherCAT support on the CompactRIO controller, you must install the NI-Industrial Communications for EtherCAT driver on the host computer. This driver is on the CD included with the NI 9144 chassis and on ni.com for free download.

2. To configure the deterministic Ethernet system, connect Port 1 of the cRIO-9074 to the host computer and connect Port 2 to the NI 9144 using standard Ethernet cables. To add more NI 9144 chassis, use the Ethernet cable to connect the OUT port of the previous NI 9144 to the IN port of the next NI 9144.

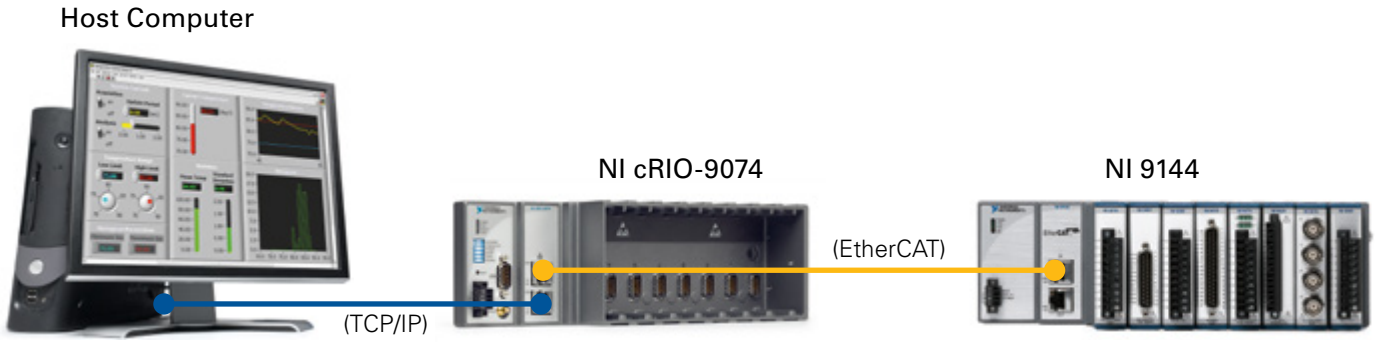


Figure 7.7. Hardware Setup for Ethernet RIO Tutorial

Configure the CompactRIO Real-Time Controller to Be the Deterministic Bus Master

3. Launch MAX and connect to the CompactRIO controller.

4. In MAX, expand the controller under **Remote Systems** in the **Configuration** pane. Right-click **Software** and select **Add/Remove Software**.

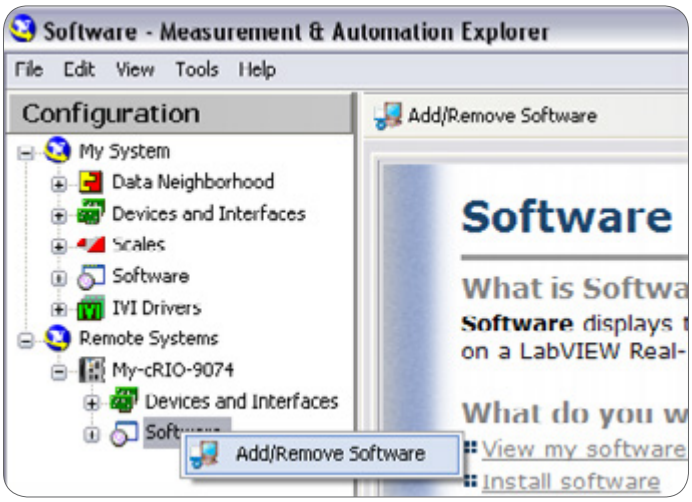


Figure 7.8. Install the appropriate software in MAX.

5. If the controller already has LabVIEW Real-Time and NI-RIO installed on it, select **Custom software installation** and click **Next**. Click **Yes** if a warning dialog box appears. Click the box next to **NI-Industrial Communications for EtherCAT**. The required dependencies are automatically checked. Click **Next** to continue installing software on the controller.

6. Once the software installation is finished, select the controller under Remote Systems in the Configuration pane. Navigate to the **Network Settings** tab in the bottom right portion of the window. In the Network Adapters window, select the secondary Ethernet Adapter (the one that does not say primary). Next to Adapter Mode, select **EtherCAT** in the pull-down box and hit the **Save** button at the top of the window.

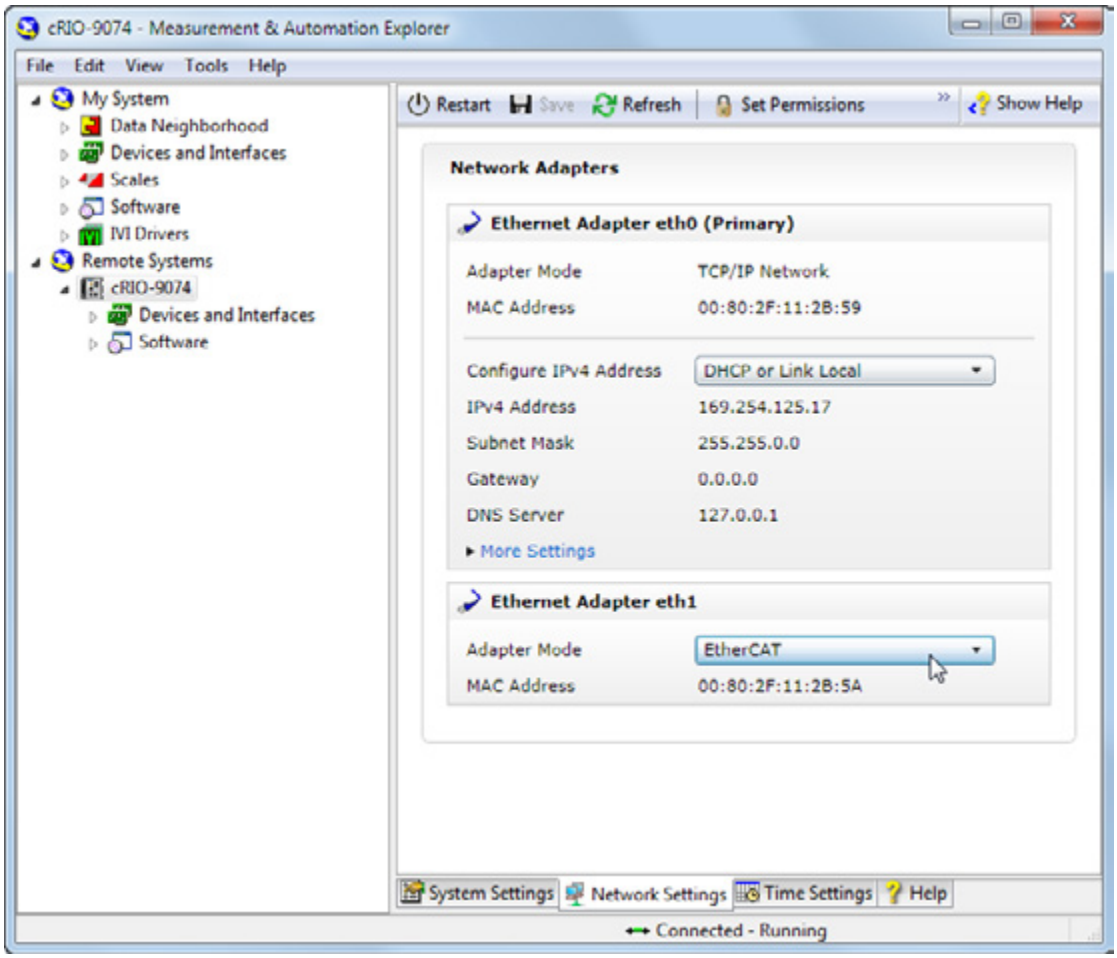


Figure 7.9. Select EtherCAT as the mode for the second Ethernet port of the CompactRIO controller.

Step 2: Add Deterministic I/O (Option 1—Scan Engine)

1. In the LabVIEW Project Explorer window, right-click on the CompactRIO controller and select **New»Targets and Devices**.

2. In the Add Targets and Devices dialog window, expand the category **EtherCAT Master Device** to autodiscover the EtherCAT port on the master controller. Select your EtherCAT device and click **OK**. The LabVIEW project now lists the master controller, the NI 9144 chassis, their I/O modules, and the physical I/O on each module.

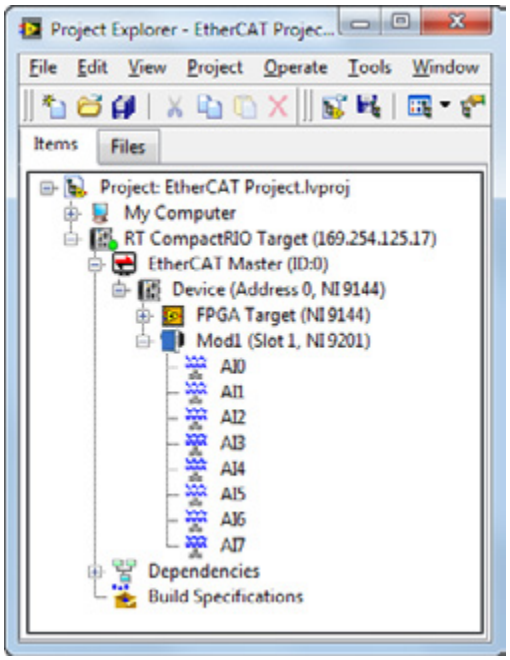


Figure 7.10. The LabVIEW project lists the master controller, NI 9144 chassis, and I/O modules.

3. By default, the I/O channels show up in the project as Scan Engine I/O variables. The Scan Engine automatically manages I/O synchronization so that all modules are read and updated at the same time once each **scan cycle.**

Note: For high-channel-count systems, you can streamline the creation of multiple I/O aliases by exporting and importing .csv spreadsheet files using the Multiple Variable Editor.

Step 2: Add Deterministic I/O (Option 2—LabVIEW FPGA)

1. In the LabVIEW Project Explorer window, right-click on the CompactRIO controller and select **New»Targets and Devices**.

2. In the Add Targets and Devices dialog window, expand the category **EtherCAT Master Device** to autodiscover the EtherCAT port on the master controller. Select your EtherCAT device and click **OK**. The LabVIEW project now lists the master controller, the NI 9144 chassis, their I/O modules, and the physical I/O on each module. By default the I/O channels show up in the project as Scan Engine I/O variables.

3. Right-click the EtherCAT device in the LabVIEW project and select **New»FPGA Target**. You can now create a LabVIEW FPGA VI to run on the EtherCAT target. By default the chassis I/O is added to the FPGA target but not the I/O modules. To program the C Series module I/O using LabVIEW FPGA, drag the C Series modules from the EtherCAT device target onto the FPGA target in the LabVIEW Project Explorer window.

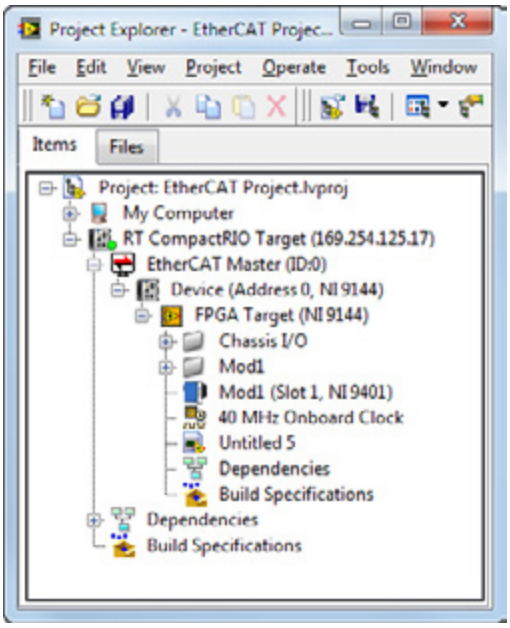


Figure 7.11. Drag C Series I/O modules under the FPGA target to access them with LabVIEW FPGA.

4. Create an FPGA VI under the EtherCAT device target and program using the EtherCAT I/O. The Figure 7.12 example uses the FPGA on an EtherCAT chassis to output a PWM signal.

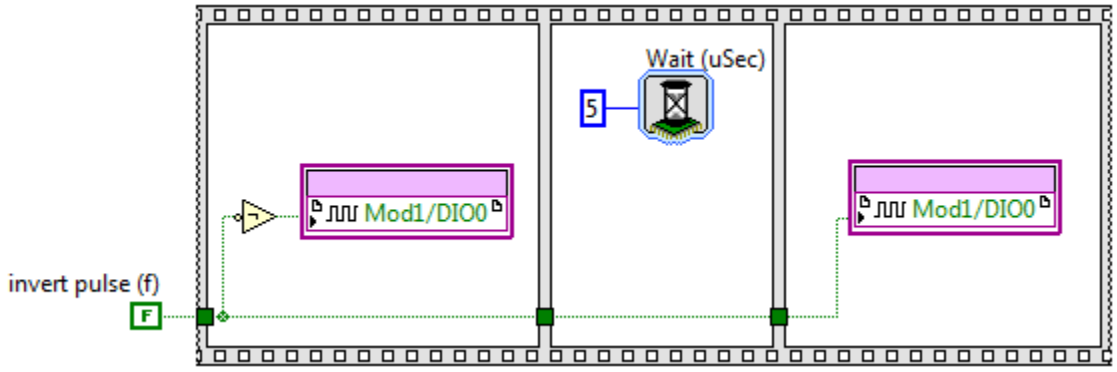


Figure 7.12. Program the EtherCAT FPGA using LabVIEW FPGA.

Note: There is a limit to the number of user-defined I/O variables that you can create in FPGA Interface Mode. The NI 9144 can hold a total of 512 bytes of input data and 512 bytes of output data for both I/O variables in Scan Mode and user-defined I/O variables in FPGA Interface Mode. For example, if you are using four 32-channel modules in Scan Mode and each channel takes up 32 bits of data, then Scan Mode I/O variables are using 256 bytes of input data. With the remaining 256 bytes of input data, you can create up to 64 input user-defined I/O variables (also of 32-bit length) in FPGA Interface Mode.

5. The next step is creating an interface that allows the Real-Time VI to communicate to the FPGA VI on the EtherCAT expansion chassis. For example, you need the user to be able to control the pulse width of the PWM signal, along with the invert pulse input. To transfer data to or from an FPGA on an EtherCAT chassis to a Real-Time VI, you use a new mechanism called user-defined I/O variables. To create a user-defined I/O variable, right-click the EtherCAT device target and select **New»User-Defined Variable**.

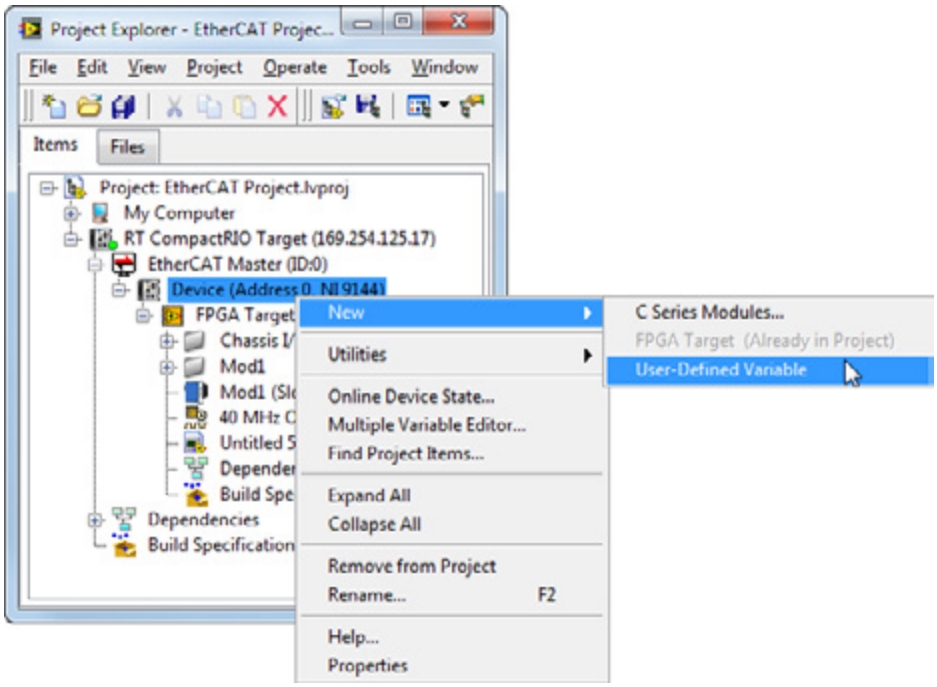


Figure 7.13. Create user-defined I/O variables to communicate between a Real-Time VI and the EtherCAT FPGA VI.

6. In the Properties dialog box, specify the variable name, data type, and direction (FPGA to host or host to FPGA). Drag and drop the user-defined I/O variables onto your FPGA VI.

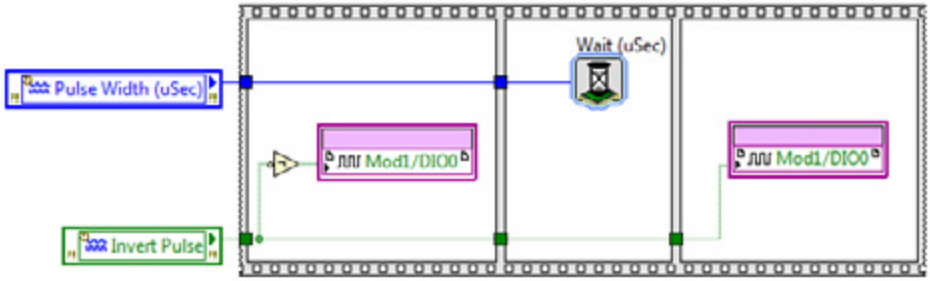


Figure 7.14. Drag and drop user-defined I/O variables onto the FPGA VI.

Note: These user-defined I/O variables transfer single-point data to and from the master controller at each scan cycle, so they are best used for passing processed data from the NI 9144 FPGA.

7. Drag and drop the same variables onto your Real-Time VI to communicate between the two targets.

CHAPTER 8
Communicating With Third-Party Devices

This section **examines different methods for communicating with third-party devices from a CompactRIO controller.** Because CompactRIO has built-in serial and Ethernet ports, it can easily work with Modbus TCP, Modbus Serial, and EtherNet/IP protocols. You can find information on using the Ethernet port to communicate to an HMI in Chapter 4: Best Practices for Network Communication.

When interfacing to a third-party device in LabVIEW Real-Time using RS232, USB, or Ethernet ports, you should consider adding a separate process for handling the communication since these protocols are nondeterministic. This allows the control task to run deterministically and reliably because it is not affected by the possibly high-jitter I/O device.

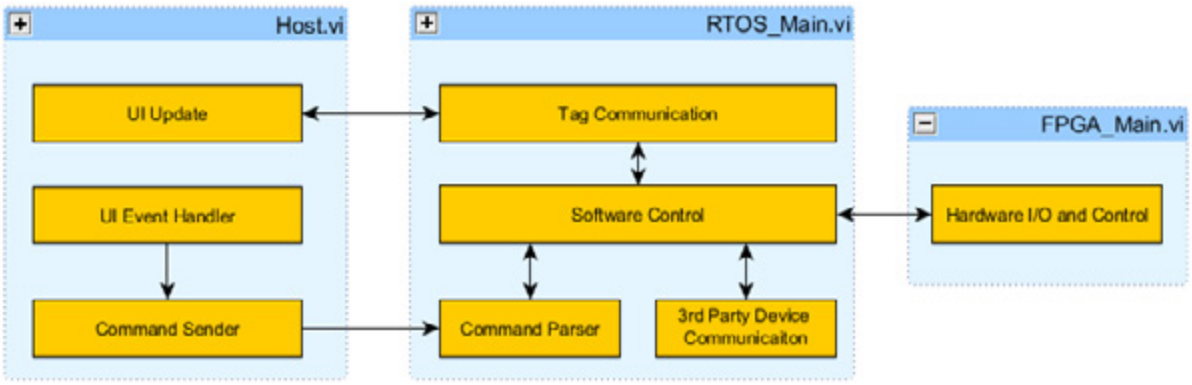


Figure 8.1. Add a separate process for handling any nondeterministic communication to a third-party device over serial or Ethernet.

Serial Communication From CompactRIO

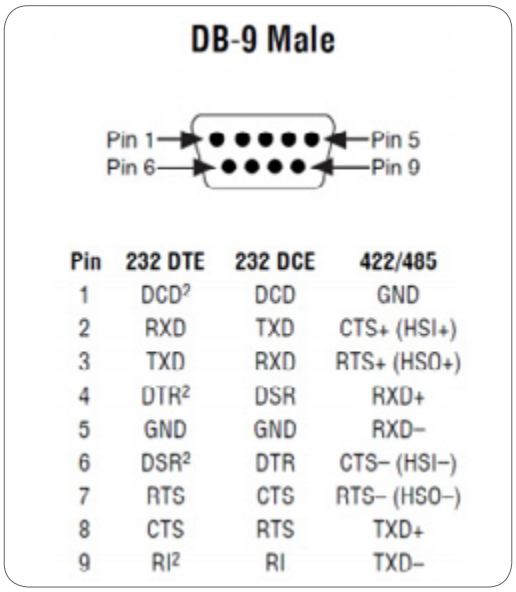
A standard for more than 40 years, RS232 (also known as EIA232 or TIA232) ports can be found on all CompactRIO systems and a wide variety of industrial controllers, programmable logic controllers (PLCs), and devices. RS232 remains a de facto standard for low-cost communications because it requires little complexity, low processing and overhead, and modest bandwidth. Though it is fading from the PC industry in favor of newer buses such as USB, IEEE 1394, and PCI Express, RS232 continues to be popular in the industrial sector because of its low complexity, low cost to implement, and wide install base of existing RS232 ports.

The RS232 specification covers the implementation of hardware, bit-level timing, and byte delivery, but it does not define fixed messaging specifications. While this allows for easy implementation of basic byte reading and writing, more complex functionality varies between devices. Several protocols use RS232-based byte messaging such as Modbus RTU/ASCII, DNP3, and IEC-60870-5.

This section covers how to program byte-level communications for the serial port built into CompactRIO real-time controllers using the NI-VISA API.

RS232 Technical Introduction

RS232 is a single-drop communications standard, meaning only two devices can be connected to each other. The standard defines many signal lines used in varying degrees by different applications. At a minimum, all serial devices use the Transmit (TXD), Receive (RXD), and ground lines. The other lines include flow control lines, Request to Send, and Clear to Send, which are sometimes used to improve synchronization between devices and prevent data loss when a receiver cannot keep up with its sender. The remaining lines are typically used with modem-to-host applications and are not commonly implemented in industrial applications with the exception of radio modems.



Pin	232 DTE	232 DCE	422/485
1	DCD[2]	DCD	GND
2	RXD	TXD	CTS+ (HSI+)
3	TXD	RXD	RTS+ (HSO+)
4	DTR[2]	DSR	RXD+
5	GND	GND	RXD–
6	DSR[2]	DTR	CTS– (HSI–)
7	RTS	CTS	RTS– (HSO–)
8	CTS	RTS	TXD+
9	RI[2]	RI	TXD–

Figure 8.2. D-SUB 9-Pin Connector Pinout

As a single-ended bus, RS232 is ideal for shorter-distance communications (under 10 m). Shielded cables reduce noise over longer runs. For longer distances and higher speeds, RS485 is preferred because it uses differential signals to reduce noise. Unlike buses such as USB and IEEE 1394, RS232 is not designed to power end devices.

CompactRIO controllers have an RS232 port that features the standard D-SUB 9-pin male connector (also known as a DE-9 or DB-9). You can use this port to monitor diagnostic messages from a CompactRIO controller when the console out switch is engaged or to communicate with low-cost RS232 devices in your application.

RS232 Wiring and Cabling

RS232 ports feature two standard pinouts. You can classify RS232 devices as data terminal equipment (DTE), which is like a master or host, and data communications equipment (DCE), which is similar to slaves. DCE ports have inverted wiring so that DCE input pins connect to DTE output pins and vice versa. The serial port on CompactRIO controllers is a DTE port, and the serial port on a device such as a GPS, bar code scanner, modem, or printer is a DCE port.

When connecting a CompactRIO DTE RS232 port to a typical end device with a DCE RS232 port, you use a regular straight-through cable. When connecting two DTE or DCE devices together, such as a CompactRIO controller to a PC, you need to use a null modem cable, which swaps the transmit and receive signals in the cabling.

Device 1	Device 2	Cable Type
DTE	DTE	Null modem cable
DTE	DCE	Straight-through cable
DCE	DTE	Straight-through cable
DCE	DCE	Null modem cable

Table 8.1. Selecting a Cable to Run Between Different RS232 Ports

Loopback Testing

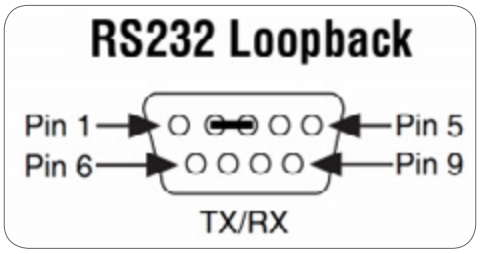


Figure 8.3. RS232 Loopback Wiring

A common technique for troubleshooting and verifying serial port functionality is loopback testing. At a basic level, all you need to do is connect the TXD to the RXD pin. Use a standard RS232 straight-through or loopback cable and a small bent paper clip to short pins 2 and 3. With the loopback cable installed, bytes sent from the read function can be read by the read function. You can verify that software, serial port settings, and drivers are working correctly with this technique.

Serial Communications From LabVIEW

The native CompactRIO RS232 serial port is attached directly to the CompactRIO real-time processor, so you should place code that accesses the serial port in the LabVIEW Real-Time VI. To send and receive bytes to the serial port, use NI-VISA functions.

NI-VISA is a driver that provides a single interface for communicating with byte-level interfaces such as RS232, RS485, GPIB, and so on. Code written with the NI-VISA functions is usable on any machine with a serial port and NI-VISA installed. This means you can write and test a Serial VI on a Windows machine with LabVIEW and then reuse the same code in LabVIEW Real-Time on CompactRIO. You can then directly use thousands of prewritten and verified instrument drivers located at ni.com/idnet.

To get started with the serial port, locate the Virtual Instrument Software Architecture (VISA) functions in the palettes at **Data Communication»Protocols»Serial**.

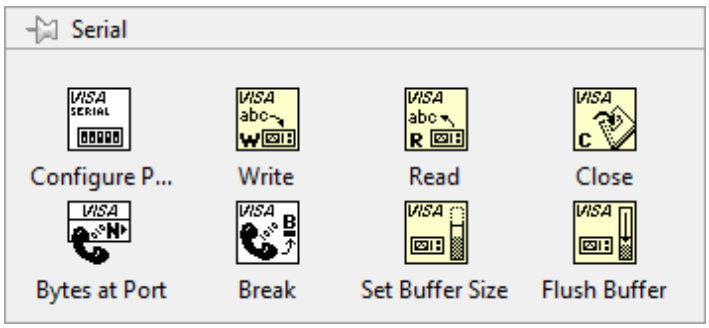


Figure 8.4. VISA Functions

For most simple instrument applications, you need only two VISA functions, VISA Write and VISA Read. Refer to the Basic Serial Write and Read VI in the labview\examples\instr\smplserl.llb for an example of how to use VISA functions.

Most devices require you to send information in the form of a command or query before you can read information back from the device. Therefore, the VISA Write function is usually followed by a VISA Read function. Because serial communication requires you to configure extra parameters such as baud rate, you must start the serial port communication with the VISA Configure Serial Port VI. The VISA Configure Serial Port VI initializes the port identified by the VISA resource name.

It is important to note that the VISA resource name refers to resources on the machine for which the VI is targeted.

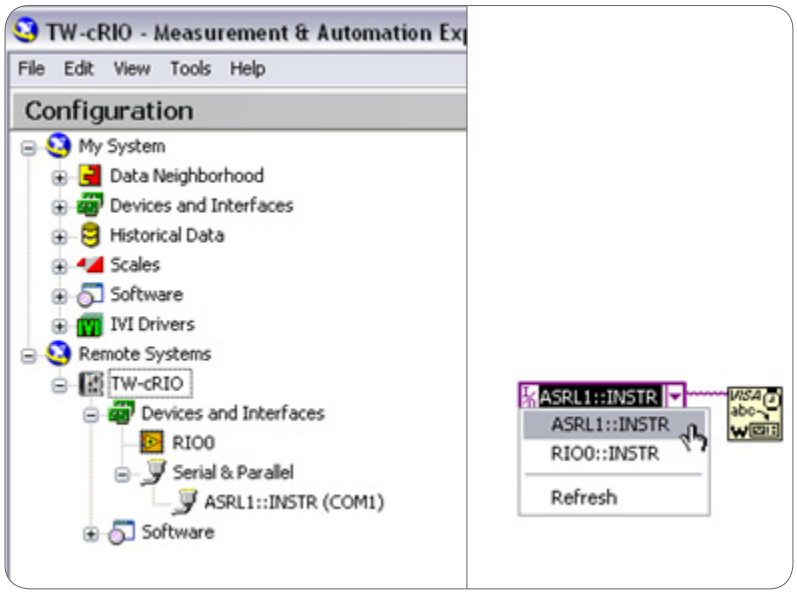


Figure 8.5. You can directly browse to the VISA resource name from the VISA resource control in LabVIEW or from MAX.

Check the bottom left corner of the VI to determine which target the VI is linked to. For CompactRIO, use COM1 to use the built-in port. COM 1 on a CompactRIO controller is ASRL1::INSTR. If you are connected to the CompactRIO controller, you can directly browse to the VISA resource name from the VISA resource control in LabVIEW or from MAX.

Timeout sets the timeout value for the serial communication. Baud rate, data bits, parity, and flow control specify those particular serial port parameters. The error in and error out clusters maintain the error conditions for this VI.

Though the ports work at the byte level, the interfaces to the read and write functions are strings. In memory, a string is simply a collection of bytes with a length attached to it, which makes working with many bytes easier to program. The string functions in LabVIEW provide tools to break up, manipulate, and combine data received from a serial port. You can also convert string data to an array of bytes for using LabVIEW array functions to work with low-level serial data.

Because serial operations deal with varying-length strings, these tasks are nondeterministic. Additionally, serial interfaces by nature are asynchronous and do not have any mechanisms to guarantee timely delivery of data. To maintain the determinism of your control loops, when programming serial applications keep any communications code in a separate loop from the control code.

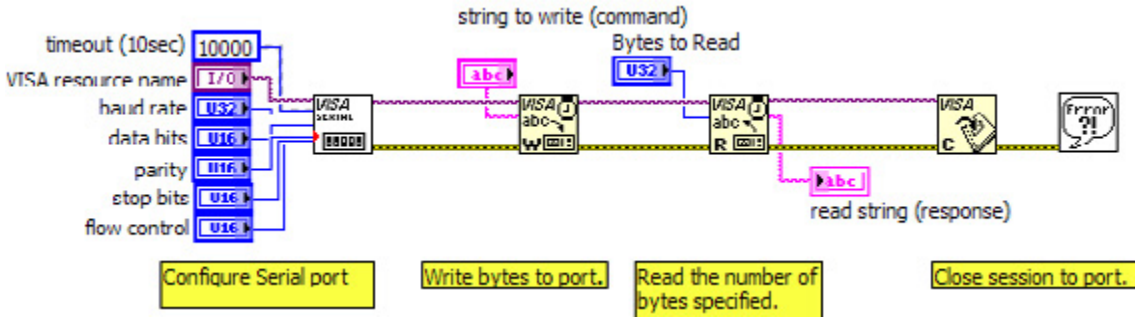


Figure 8.6. Simple Serial Communication Example

The VISA Read function waits until the number of bytes requested arrives at the serial port or until it times out. For applications where this is known, this behavior is acceptable, but some applications have different lengths of data arriving. In this case, you should read the number of bytes available to be read, and read only those bytes. You can accomplish this with the Bytes at Serial port property accessible from the Serial palette.

You can access serial port parameters, variables, and states using property nodes. If you open the VISA Configure Serial Port VI and examine the code, you see that this VI simply makes a call to many property nodes to configure the port. You can use these nodes to access advanced serial port features, including the auxiliary data lines, bytes waiting to be read/written, and memory buffer sizes. Figure 8.7 shows an example of using a property node to check the number of bytes available at the port before reading.

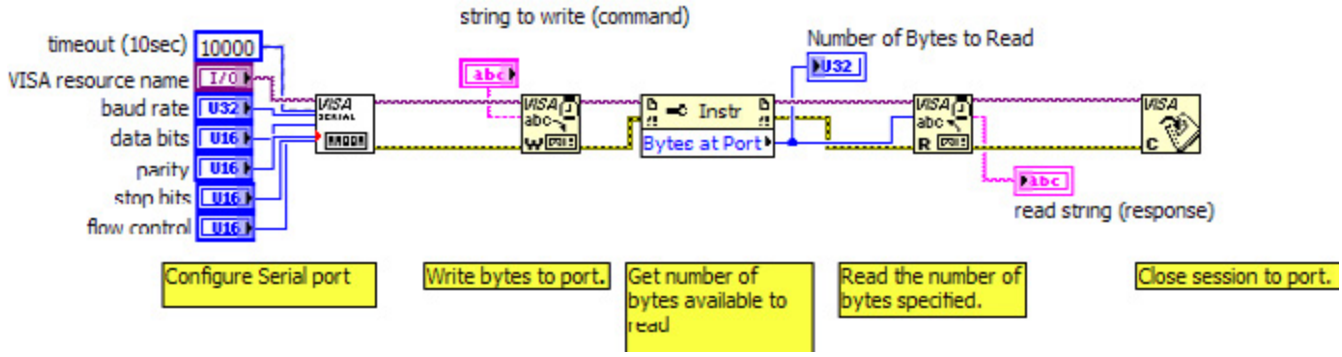


Figure 8.7. Using a VISA Property Node to Read the Number of Bytes

In general, working with devices can range from simply reading periodically broadcasted bytes such as GPS data to working with complex command structures. For some devices and instruments, you can find preprogrammed instrument drivers on ni.com/idnet, which can simplify development. Because these drivers use NI-VISA functions, they work with the onboard CompactRIO serial port.

Instrument Driver Network

To help speed development, NI has worked with major measurement and control vendors to provide a library of ready-to-run instrument drivers for more than 10,000 devices. An instrument driver is a set of software routines that control a programmable instrument. Each routine corresponds to a programmatic operation such as configuring, reading from, writing to, and triggering the instrument. Instrument drivers simplify instrument control and reduce program development time by eliminating the need to learn the programming protocol for each device.

Where to Find Instrument Drivers and How to Download Them

You can find and download instrument drivers in two ways. If you are using LabVIEW 8.0 or later, the easiest way is to use the NI Instrument Driver Finder. If you have an older version of LabVIEW, then you can use the Instrument Driver Network (ni.com/idnet).

Use the NI Instrument Driver Finder to find, download, and install LabVIEW Plug and Play drivers for an instrument. Select **Tools»Instrumentation»Find Instrument Drivers** to launch the Instrument Driver Finder. This tool searches IDNet to find the specified instrument driver.

You can use an instrument driver for a particular instrument as is. However, LabVIEW Plug and Play instrument drivers are distributed with their block diagram source code, so you can customize them for a specific application. You can create instrument control applications and systems by programmatically linking instrument driver VIs on the block diagram.

RS232 and RS422/RS485 Four-Port NI C Series Modules for CompactRIO

If your application requires higher performance or additional RS232 ports and/or RS485/RS422 ports, you can use the NI 9870 and NI 9871 serial interfaces for CompactRIO to add more serial ports. These modules are accessed directly from the FPGA using LabVIEW FPGA or from the Scan Engine.

Using NI 987x Serial Modules With Scan Mode

To enable support for these modules using the Scan Engine, add NI-Serial 9870 and 9871 Scan Engine Support to your CompactRIO target in Measurement & Automation Explorer (MAX). Once the **NI-Serial 9870 and 9871 Scan Engine Support** and your CompactRIO target are correctly configured, the ports of an NI 987x appear in MAX when the **Serial & Parallel** branch is expanded. ASRL1 is your built-in serial port, and ASRL 2 through 5 are the ports of an NI 987x.

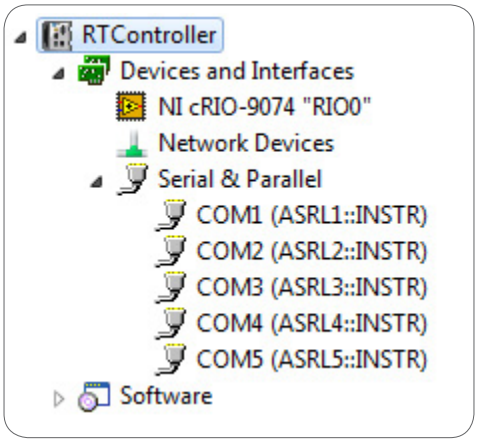


Figure 8.8. Access your NI 987x serial ports in MAX by installing Scan Engine support.

In LabVIEW, place a VISA Configure Serial Port VI or a Visa Property Node to configure the settings of your individual ports.

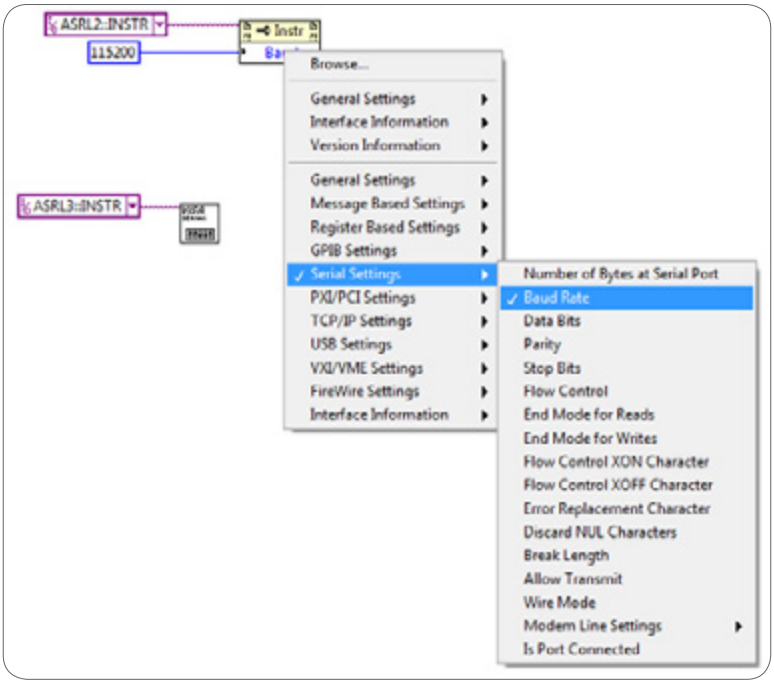


Figure 8.9. Program an NI 987x with the VISA API when using Scan Mode.

Once you have set the serial port to match that of the connected device, use the VISA API as you would when using any other VISA device.

Note: This maximum nonstandard baud rate for the NI 9871 module (3.6864 Mbit/s) can be achieved only while using the FPGA interface. The maximum baud rate while using either module with the RIO Scan Interface is 115.2 kbit/s.

Using NI 987x Serial Modules With LabVIEW FPGA

You also can access NI 987x serial modules with LabVIEW FPGA. See the Figure 8.10 example titled NI 987x Serial Loopback.lvproj in the NI Example Finder. This example demonstrates communication to the NI 987x serial port and streaming serial data to a real-time VI.

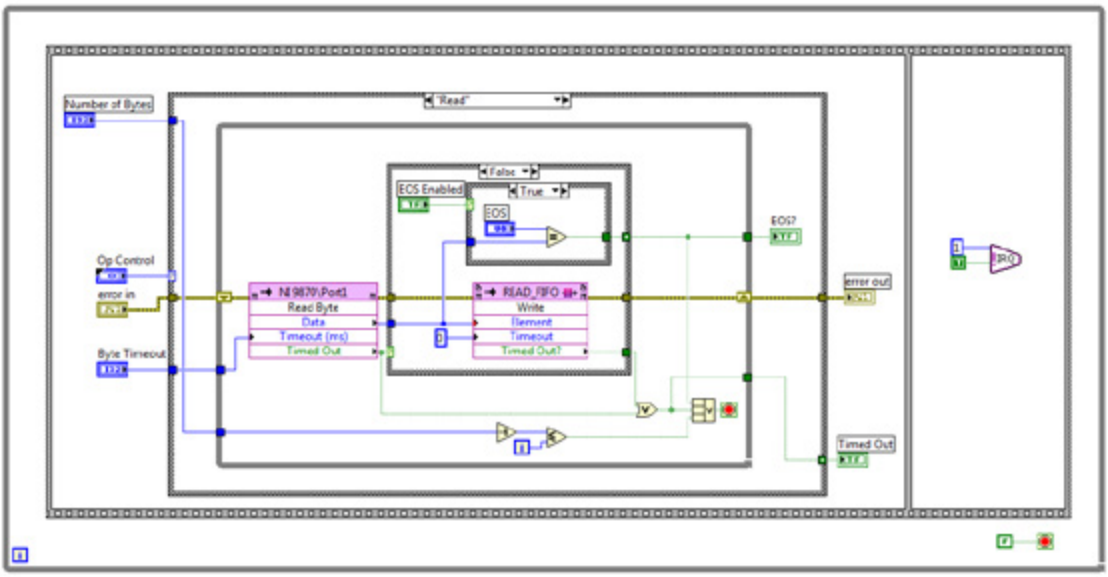


Figure 8.10. You can find the NI 987x serial loopback example for LabVIEW FPGA in the NI Example Finder.

Communicating With PLCs and Other Industrial Networked Devices

Often applications call for integrating NI programmable automation controllers (PACs) such as CompactRIO into an existing industrial system or with other industrial devices. These systems can consist of traditional PLCs, PACs, or a variety of specialty devices such as motor controllers, sensors, and HMIs. These devices feature a wide range of communication options from simple RS232 to specialized high-speed industrial networks.

You can choose from three common methods to connect the CompactRIO controller to an industrial system or PLC. You can directly wire the CompactRIO controller to the PLC using standard analog or digital I/O. This method is simple but scales for only small systems. For larger systems, you can use an industrial communications protocol to communicate between the CompactRIO controller and the PLC. For large Supervisory Control and Data Acquisition (SCADA) applications, OPC is the most commonly used protocol. OPC scales to high channels and can be used with either Windows-based PCs or real-time OSs depending on the OPC standard.

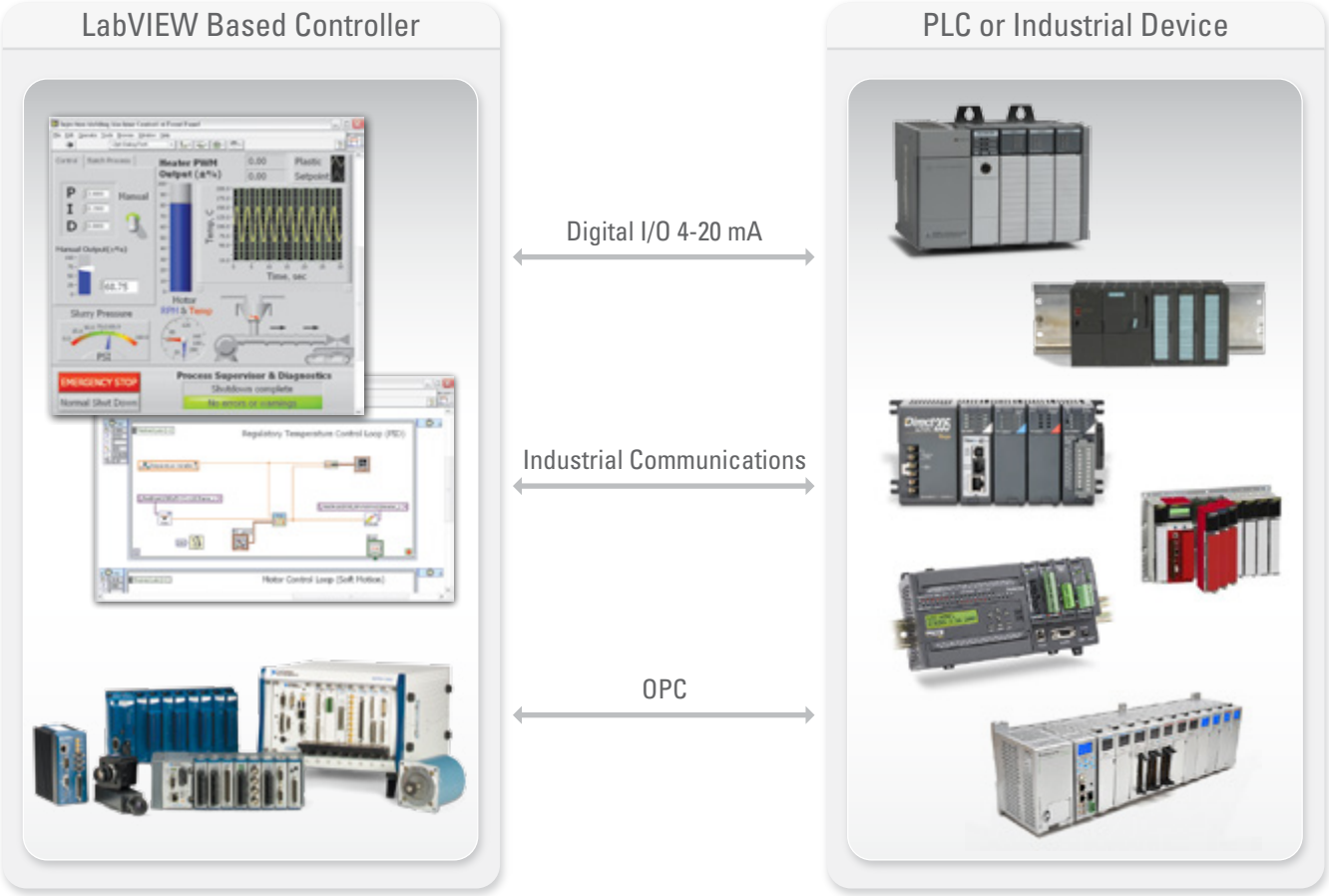


Figure 8.11. Three Methods for Connecting a CompactRIO Controller to an Industrial Device

Industrial Communications Protocols

Beyond using standard I/O, the most common way to connect a CompactRIO controller to other industrial devices is using industrial communications protocols. Most protocols use standard physical layers such as RS232, RS485, CAN, or Ethernet. As discussed earlier, buses like RS232 and Ethernet provide the physical layer for simple communications but lack any predefined methods of higher level communication, so they offer just enough for a user to send and receive individual bytes and messages. When working with large amounts of industrial data, handling those bytes and converting them into relevant control data quickly become tedious. Industrial communications protocols provide a framework for how data is communicated. They are fundamentally similar to the instrument drivers discussed earlier although they typically run a more sophisticated stack.

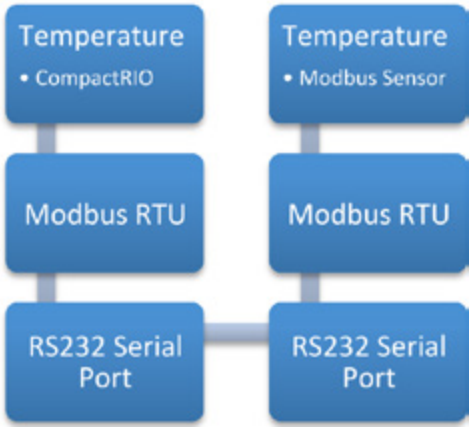


Figure 8.12. Typical Industrial Protocol Implementation

Industrial protocols take a low-level bus like RS232 or TCP/IP and add predefined techniques for communication on top. This is similar to how HTTP, the protocol for web content delivery, sits on top of TCP/IP. It is possible to write your own HTTP client like Internet Explorer or Firefox, but it takes a large amount of effort and may not accurately connect to all servers without hours of testing and verification. Similarly, devices and controllers that support industrial networking standards are much easier to integrate at the program level and to use for abstracting the low-level communication details from end users, so engineers can focus on the application.

Because CompactRIO has built-in serial and Ethernet ports, it can easily work with many protocols including Modbus TCP, Modbus Serial, and EtherNet/IP. You also can use plug-in modules for communication with most common industrial protocols such as PROFIBUS and CANopen.

When you cannot use native communication, another option is gateways. Gateways are protocol translators that can convert between different industrial protocols. For example, you can connect to a CC-Link network by using a gateway. The gateway can talk Modbus TCP on the PAC end and translate that to CC-Link on the other end.

Modbus Communications

Modbus is the most common industrial protocol in use today. Developed in 1979, it supports both serial and Ethernet physical layers. Modbus is an application layer messaging protocol for client/server communication between devices connected on a bus or network. You can implement it using asynchronous serial transmission to communicate with touch screens, PLCs, and gateways to support other types of industrial buses. Modbus works with a CompactRIO controller and its onboard serial or Ethernet modules. Note that the onboard serial port of CompactRIO hardware uses RS232 though some Modbus devices may use the RS485 electrical layer. In this case, you can use a common RS485-to-RS232 adapter.

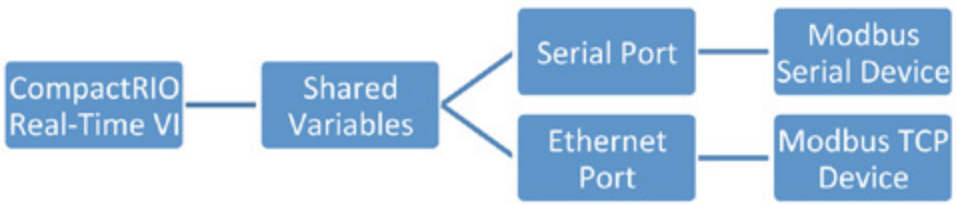


Figure 8.13. Software Architecture for Communicating With Modbus Devices

The Modbus serial protocol is based on a master/slave architecture. An address, ranging from 1 to 247, is assigned to each slave device. Only one master is connected to the bus at any given time. Slave devices do not transmit information unless a request is made by the master device, and slave devices cannot communicate to other slave devices.

Information is passed between master and slave devices by reading and writing to registers located on the slave device. The Modbus specification distinguishes the use of four register tables, each capable of 65,536 items and differentiated by register type and read-write access. Register address tables do not overlap in this implementation of LabVIEW.

Tables	Object Type	Type of Access	Comments
Discrete Inputs	Single-Bit	Read Only	Only the master device can read. Only the slave device can change its register values.
Coils	Single-Bit	Read-Write	Both master and slave devices can read and write to these registers.
Input Registers	16-Bit Word	Read Only	Only the master device can read. Only the slave device can change its register values.
Holding Registers	16-Bit Word	Read-Write	Both master and slave devices can read and write to these registers.

Table 8.2. Information is passed between master and slave devices by reading and writing to registers located on the slave device.

Before starting with Modbus programming, you must determine several important parameters of your Modbus device by consulting its documentation:

1. Master or slave? If your Modbus device is a slave, configure your CompactRIO controller as a master. This is the most common configuration. Likewise, connecting to a Modbus master (such as another PLC) requires configuring the CompactRIO controller as a slave.
2. Serial or Ethernet? Modbus Ethernet devices are sometimes called "Modbus TCP" devices.
3. For Modbus serial
   - RS232 or RS485? Many Modbus devices are RS232, but some use the higher power RS485 bus for longer distances. RS232 devices can plug directly into a CompactRIO system, but an RS485 device needs an RS232-to-RS485 converter to function properly.
   - RTU or ASCII mode? RTU/ASCII refers to the way data is represented on the serial bus. RTU data is in a raw binary form, whereas ASCII data is human readable on the raw bus. This parameter is required by the Modbus VIs.
4. What are the register addresses? Every Modbus device has a mapping of its I/O to registers, coils, and discrete inputs for reading and writing that is represented by a numerical address. Per Modbus convention, a register address is always one less than the register name. This is similar to the first element of an array being element 0. The Modbus LabVIEW Library requires register addresses, not register names.

Modbus Tutorial

The NI Developer Zone document How to Turn an RT Target Into a Modbus Slave Using I/O Servers walks you through the steps of turning your CompactRIO controller into a Modbus slave. It also shows you how to read from this server with LabVIEW for Windows if you do not have a third-party Modbus server. Before starting the tutorial, install the Modbus I/O Server onto your CompactRIO target through MAX.

Additionally, with the LabVIEW Real-Time and LabVIEW Datalogging and Supervisory Control (DSC) modules, you can configure Modbus I/O servers on the development computer to communicate with Modbus devices. If you do not have the LabVIEW Real-Time or LabVIEW DSC modules, you can use the free LabVIEW Modbus Library to create Modbus master or slave applications with any Ethernet or serial ports on your development computer. This

library provides a set of lower-level VIs for greater flexibility, performance, and customization. Download the LabVIEW Modbus Library to begin programming your Modbus applications today.

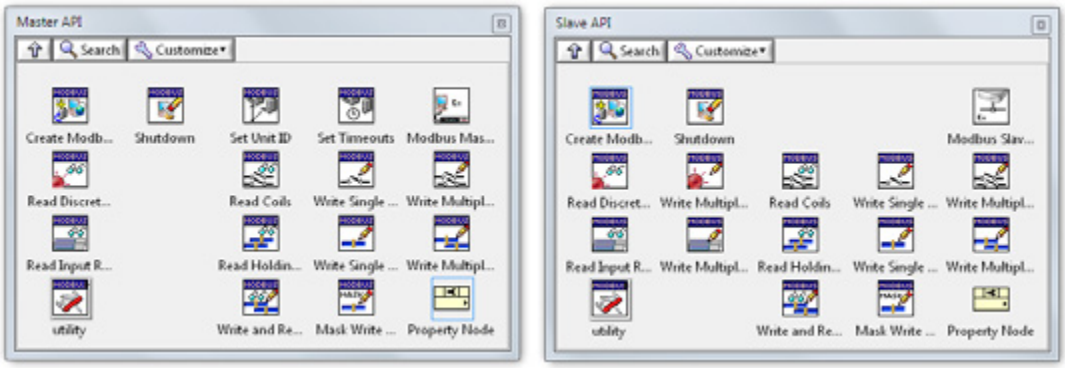


Figure 8.14. The LabVIEW Modbus Library provides low-level functions for greater flexibility and performance.

Modbus Serial- and TCP-based communications are also available as a third-party add-on on the LabVIEW Tools Network called ModBusVIEW over TCP.

EtherNet/IP

EtherNet/IP is a protocol built on the Common Industrial Protocol (CIP) for communications with EtherNet/IP devices. This protocol, developed and commonly found on Rockwell Automation (Allen-Bradley) PLCs, uses standard Ethernet cabling for communications with industrial I/O. EtherNet/IP is an application layer protocol that uses TCP for general messages and User Datagram Protocol (UDP) for I/O messaging and control.

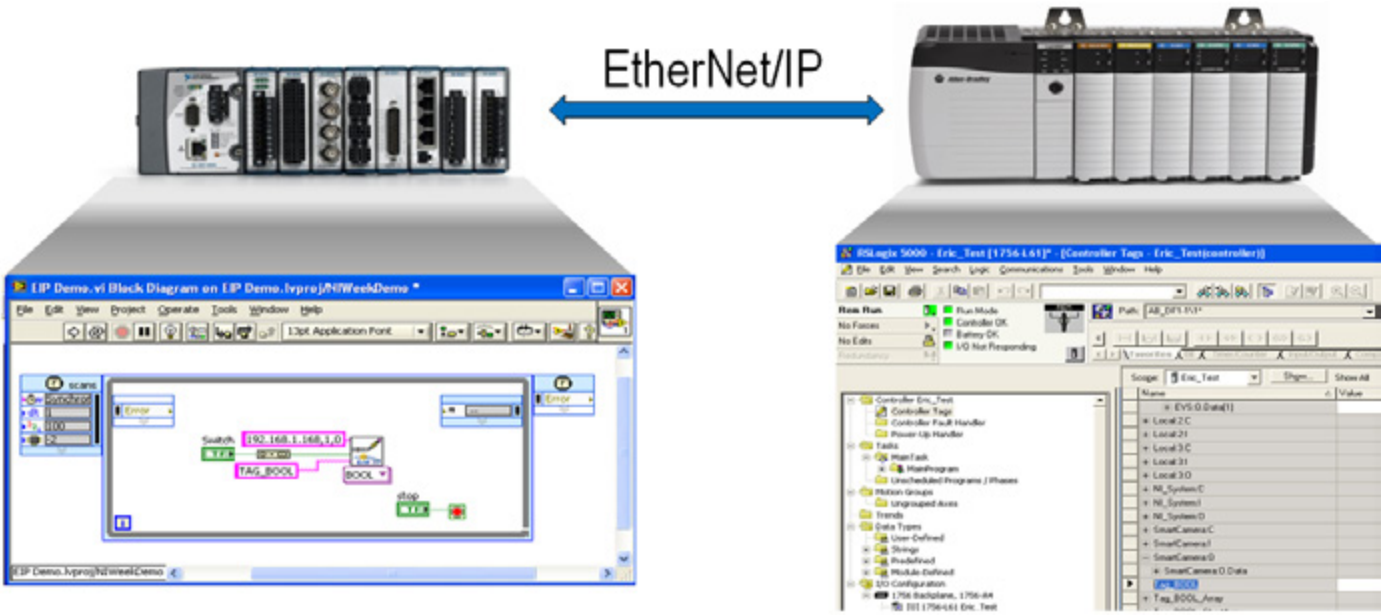


Figure 8.15. With EtherNet/IP, you can directly read and write to tags on a Rockwell PLC from CompactRIO.

The LabVIEW Driver for EtherNet/IP provides both explicit messaging API and implicit I/O data communication with a wide range of PLCs and other EtherNet/IP devices. With this driver, the CompactRIO controller supports direct communication to Rockwell Automation PLCs, allowing for easy integration into a new or an existing system.

Energy Protocol: DNP3

DNP3 (Distributed Network Protocol) is a protocol specification for vendors of power grid SCADA components. This protocol is commonly used in electrical and water utilities to communicate between a master station and other devices such as remote terminal units (RTUs) and other intelligent electronic devices (IEDs).

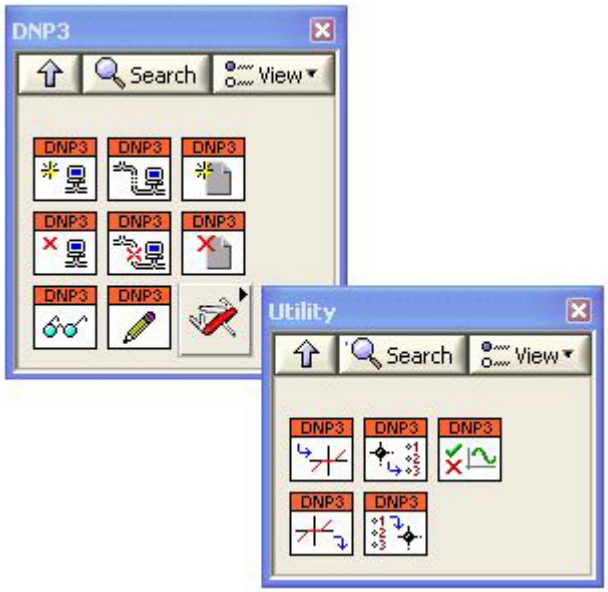


Figure 8.16. DNP3 functions provide capabilities to create outstations in LabVIEW.

You can use the LabVIEW Driver for DNP3 Outstation Support to program a real-time target, such as a CompactRIO system, as an outstation device with advanced functionality like power quality monitoring, phasor measurements, and other smart grid-related analysis. The DNP3 software driver supports Ethernet and serial communication, file transfer, and time synchronization between the master and outstation. Multiple communication channels per outstation and multiple sessions (logical devices) per channel also work with this driver.

OPC

OPC, or Open Platform Communications, is a common protocol used in many applications with high channel counts and low update rates, which are common in process industries such as oil and gas and pharmaceutical manufacturing. OPC is designed for connecting HMI and SCADA systems to controllers—it is not generally used for communication between controllers. A common use for OPC is to integrate a CompactRIO controller into a third-party system for HMI and SCADA applications.

The OPC specification is a large collection of standards that span many applications. This section covers OPC Data Access (DA), the core specification that defines how to universally move data around using common Windows technologies and OPC Unified Architecture (UA), a cross-platform standard optimized for security and with extended functionality.

OPC Data Access (DA)

OPC DA is a standard interface defined by the OPC Foundation that allows communication between numerous data sources, including devices on a factory floor, laboratory equipment, test system fixtures, and databases. The LabVIEW Real-Time and LabVIEW DSC modules provide OPC Client I/O servers that can communicate with any server implementing the OPC Foundation OPC server interface through the OPC DA standard. When LabVIEW acts as an OPC server, it uses the Shared Variable Engine to create OPC tags out of network-published shared variables that an OPC DA client can connect to. This allows LabVIEW VIs to easily communicate with other OPC DA client software. The network-published shared variable in LabVIEW provides a gateway to OPC.

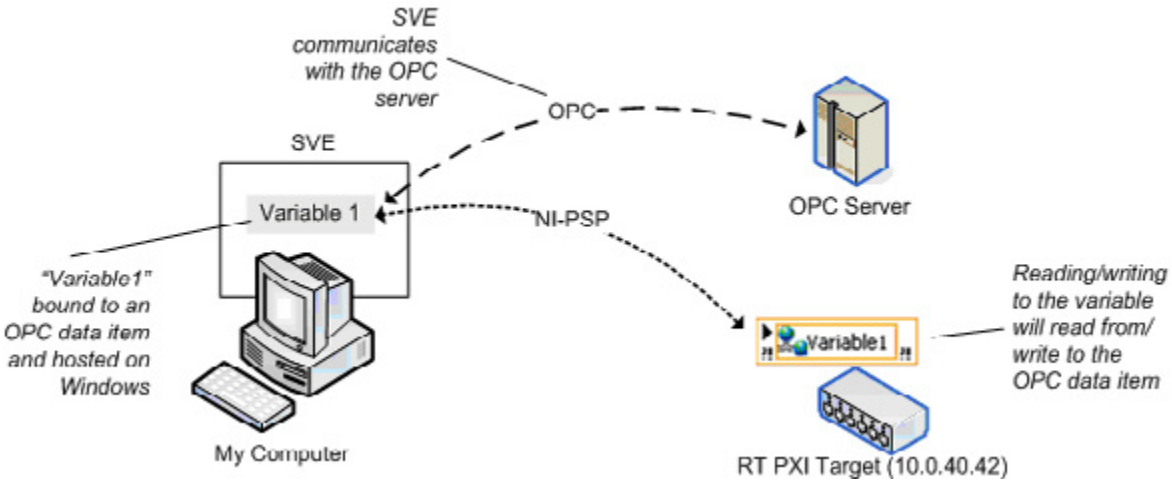


Figure 8.17. Communicate OPC data items with network-published shared variables.

OPC DA is a Windows-only technology and the CompactRIO controller cannot directly communicate using the OPC DA protocol. Instead, a Windows PC can communicate to the CompactRIO controller using the NI-PSP protocol and can translate and republish the information via OPC. When running on a Windows machine, the Shared Variable Engine functions as an OPC server and performs this translation and republishing automatically. This means other OPC DA clients, such as third-party SCADA packages or process control software, can access and retrieve data from the CompactRIO device.

Publishing Data From a CompactRIO System Over OPC DA

The following example describes how to publish network shared variables as OPC DA items using the Shared Variable Engine as an OPC DA server. The example assumes you are already publishing data on the network via network-published shared variables hosted by the CompactRIO system. If the network-published shared variables are hosted on a Windows PC, they are published as OPC DA items by default. You can verify that the variable is working correctly by launching an OPC DA client such as the NI Distributed System Manager as described in step 6.

1. The CompactRIO controller publishes data onto the network using network-published shared variables hosted on the CompactRIO controller. In this example, the network shared variable is named SV_PID_SetPoint.

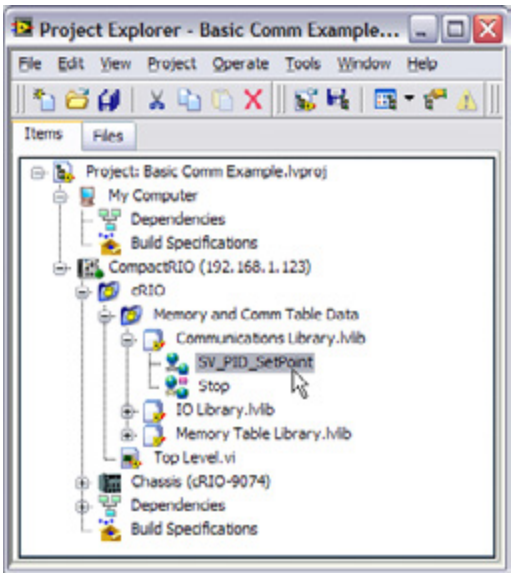


Figure 8.18. Network-Published Shared Variables Hosted on the CompactRIO System

2. To publish the variables as OPC DA data items, bind each variable hosted on the CompactRIO controller to a variable hosted on the Windows PC. To create variables hosted on a Windows PC, right-click My Computer in the LabVIEW Project and select **New»Variable**.

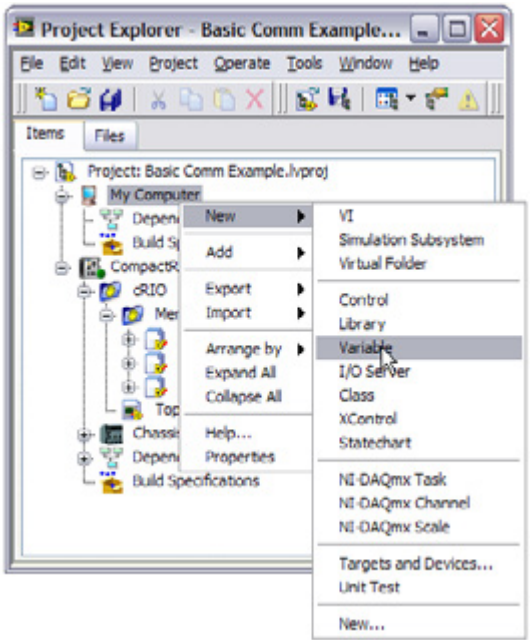


Figure 8.19. Network-published shared variables hosted on Windows are automatically published via OPC.

3. In the Shared Variable Properties window, give the variable a useful name such as SV_PID_SetPoint_OPC. Check the Enable Aliasing box and click the Browse button to browse to the SV_PID_SetPoint variable on the CompactRIO controller.

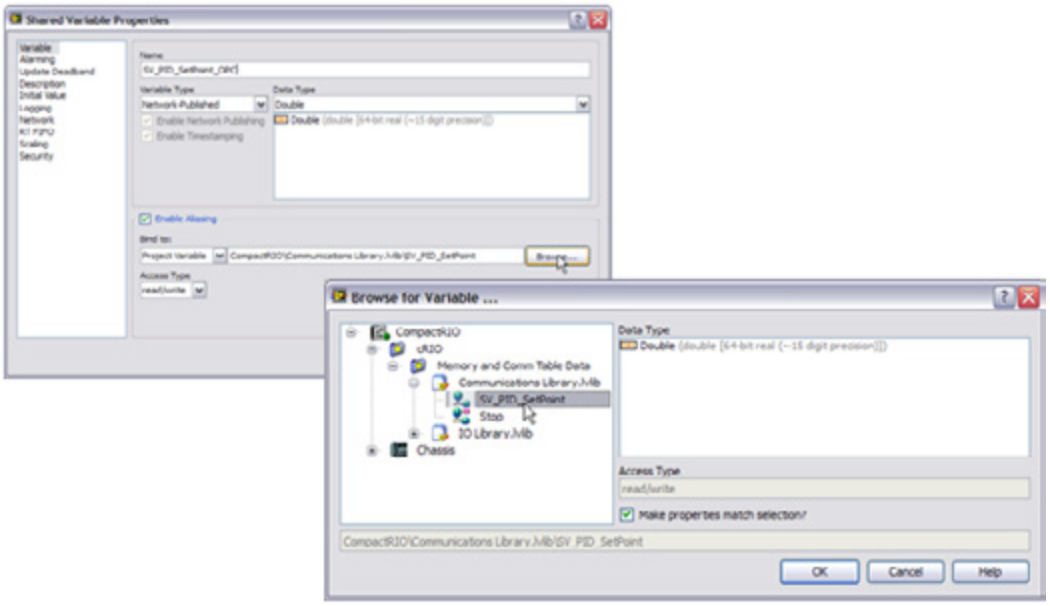


Figure 8.20. An aliased network-published shared variable provides a gateway between CompactRIO and OPC.

4. Save the new library you create with a useful name, such as OPC Library.lvlib.

5. Deploy the variables to the Shared Variable Engine by right-clicking the library and selecting Deploy.

6. The shared variable is now published as an OPC DA item. Other OPC DA clients can now connect to the Windows PC and use the data for display, HMI, and so on. You can verify that the variable is working correctly by launching an OPC DA client such as the NI Distributed System Manager (**Start»Programs»National Instruments»NI Distributed System Manager**).

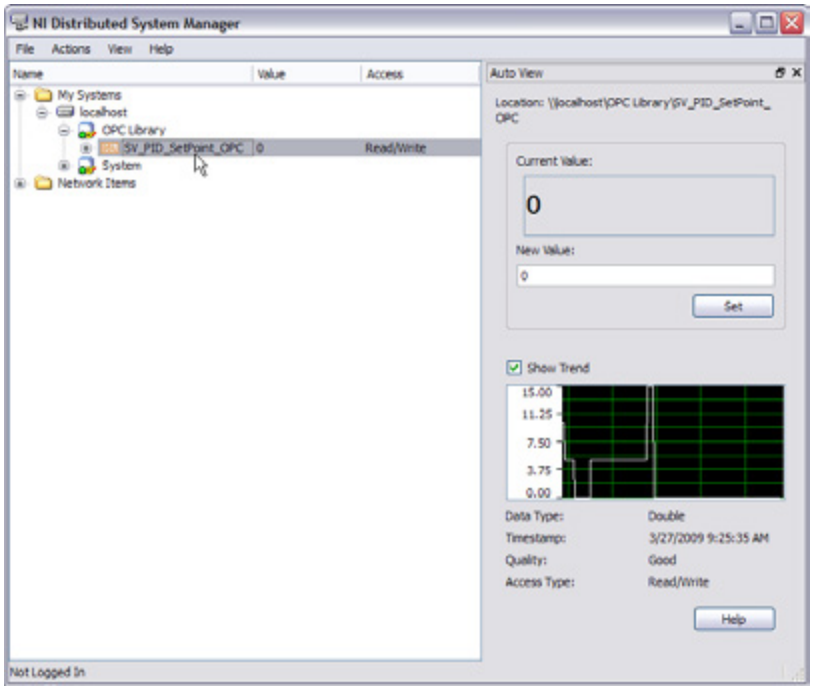


Figure 8.21. The OPC clients, such as the NI Distributed System Manager, can read and write to the OPC item.

7. If you have an OPC DA client, such as the LabVIEW DSC Module, you can verify that the variable is working from that client as well. NI shared variable OPC DA items are published under the National Instruments.Variable Engine.1 server name.

8. Once variables are deployed in the Shared Variable Engine, they stay deployed with subsequent reboots of the OPC DA server machine.

9. Find more information on the Shared Variable Engine and OPC DA in the LabVIEW Help file Connecting to OPC Systems Using LabVIEW (Windows Only).

OPC Unified Architecture (UA)

OPC UA is a new communication technology standard that was first released by the OPC Foundation in 2006 as an improvement on its predecessor, Classic OPC. OPC UA includes the functionality of Classic OPC to offer current data access, alarms, and events. Furthermore, OPC UA is based on a cross-platform, business-optimized service-oriented architecture (SOA), which expands on the security and functionality found in Classic OPC, instead of Microsoft-based COM/DCOM technology.

The LabVIEW Datalogging and Supervisory Control (DSC) Module and the LabVIEW Real-Time Module include the OPC UA VIs to exchange data between OPC UA servers and clients and create certificates that protect data. You need LabVIEW DSC to use the OPC UA VIs on Windows targets, and you need LabVIEW Real-Time to use the OPC UA VIs on LabVIEW Real-Time targets.

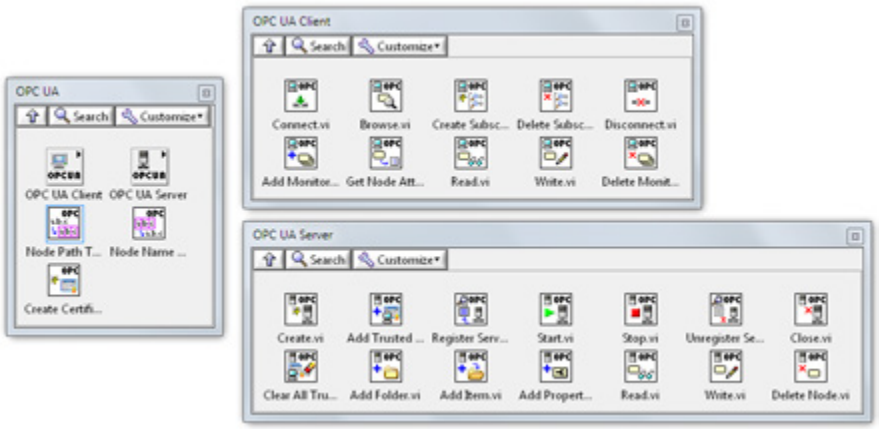


Figure 8.22. The OPC UA VIs support both nonsecure connections and secure connections between an OPC UA server and an OPC UA client.

To learn more about the OPC UA standard, read the white paper Why OPC UA Matters. For example code on how to use the OPC UA API, review the project **C:\Program Files\National Instruments\LabVIEW XXXX\examples\comm\OPCUA\OPC UA Demo.lvproj**.

OPC Servers

In addition to OPC DA I/O servers and the OPC UA API, National Instruments provides an OPC Server solution that contains a list of drivers for many of the industry's PLCs. For a list of supported PLCs, refer to the NI Developer Zone article Supported Device & Driver Plug-in List for NI-OPC Servers.

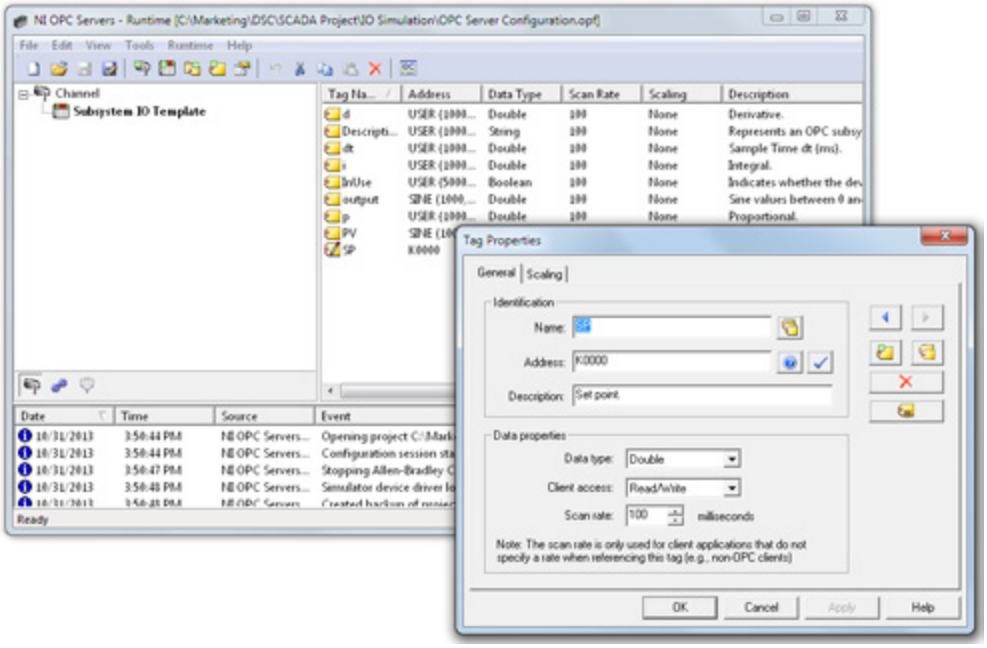


Figure 8.23. OPC servers allow connectivity to hundreds of third-party PLCs.

To learn more about OPC connectivity, visit ni.com/opc.

Additional Resources

For information on the other industrial communications protocols NI supports, visit ni.com/industrialcommunications.

CHAPTER 9
Designing a Touch Panel HMI

Building User Interfaces and HMIs Using LabVIEW

This chapter examines one software design architecture for building an operator interface with a scalable navigation engine for cycling through different human machine interface (HMI) pages.

You can use this architecture to build an HMI based on LabVIEW for any HMI hardware targets including the NI TPC-2212 touch panel computer running the Windows Embedded Standard 7 (WES7) OS or the NI TPC-2206 running the Windows XP Embedded (XPe) OS. LabVIEW is a full programming language that provides one solution for a variety of development tasks ranging from HMI/SCADA systems to reliable and deterministic control applications. The LabVIEW Touch Panel Module offers a graphical programming interface that you can use to develop an HMI in a Windows development environment and then deploy it to an NI touch panel computer (TPC) or any HMIs running Windows Embedded Standard 7 or Windows XP Embedded.

Basic HMI Architecture Background

An HMI can be as simple or as complex as the functionality you require. The software architecture defines the functionality of an HMI as well as its ability to expand and adapt to future technologies. A basic HMI has three main routines:

1. Initialization and shutdown (housekeeping) routines
2. I/O scan loop
3. Navigation (user interface) loop

Before executing the I/O scan loop and the navigation loop, the HMI needs to perform an initialization routine. This initialization routine sets all controls, indicators, internal variables, and variables communicating with the hardware (controller) to default states. You can add more logic to prepare the HMI for operations such as logging files. Before stopping the system, the shutdown task closes any references and performs additional tasks such as logging error files. Initialization and shutdown tasks are not diagrammed since they are not processes (they execute only once).

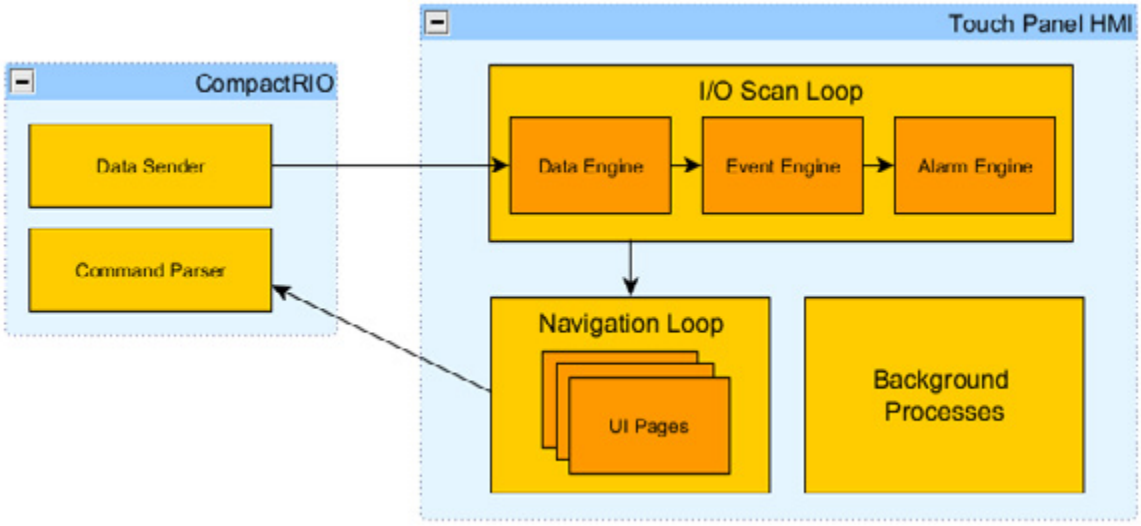


Figure 9.1. Basic Software Architecture of a Touch Panel HMI Communicating With a CompactRIO Controller

I/O Scan Loop

The I/O scan loop typically consists of three components: the Data Engine, the Event Engine, and the Alarm Displays Engine.

Data Engine

The Data Engine exchanges tag values with the controller via an Ethernet-based communication protocol or Modbus. It receives this data across the network and makes it available throughout the HMI application.

Event Engine

The I/O scan loop might also handle alarms and events. The Event Engine compares a subset of tag values to a set of predefined conditions (value equal to X, value in range or out of range, and so on) and logs an event when a tag value matches one of its event conditions.

Alarm Engine

Some events are simply logged while other events require operator intervention and are configured as alarms. The alarm event data is sent to the Alarm Displays Engine, which manages how the alarm is presented to the operator. When the tag value leaves the alarm state, the Event Engine sends an alarm canceling the event to the Alarm Displays Engine.

Navigation Loop

The navigation loop handles the management and organization of different UIs in your application. You can implement a navigation loop as a simple state machine built with a While Loop and a Case structure. Each case encloses an HMI page VI that, when called, is displayed on the HMI screen. Figure 9.2 is an example of a navigation loop.

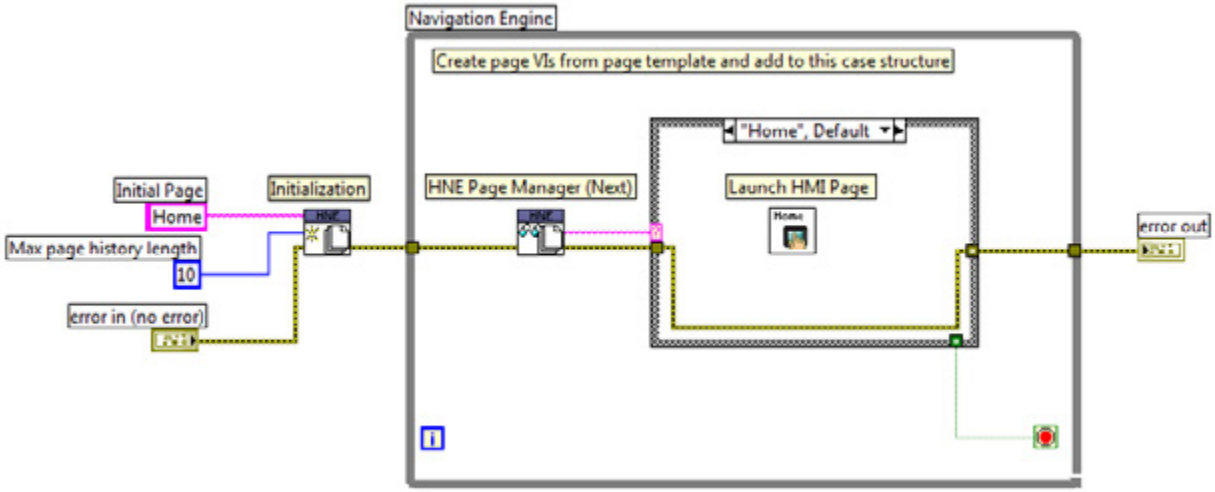


Figure 9.2. Example Navigation Loop Block Diagram

This example uses the HMI Navigation Engine (HNE) Reference Library, which was created for HMI page management and navigation. The HNE is based on VIs included with the LabVIEW Touch Panel Module, but it features an additional history cache so the user can define a button to jump backward toward the previously viewed screen. The HNE also includes basic templates and examples for getting started. Refer to the NI Developer Zone document HMI Navigation Engine (HNE) Reference Library to download the HNE library.

The HNE installs a page manager API palette named HNE to the User Libraries palette in LabVIEW.

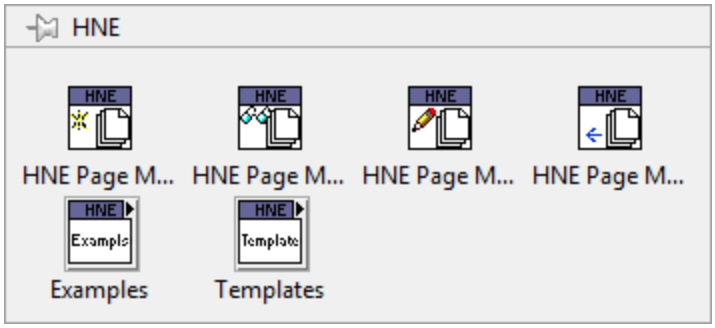


Figure 9.3. HNE (HMI Navigation Engine) Palette

- **HNE Page Manager (Init)**—This VI initializes the HNE by setting the navigation history depth and setting the name of the first page to be displayed.
- **HNE Page Manager (Next)**—This VI returns the name of the next page and passes it to the Case structure in the HNE.
- **HNE Page Manager (Set)**—This VI sets the name of the next page to be displayed. It is used within HMI pages to support navigation buttons.
- **HNE Page Manager (Back)**—This VI returns the name of the previous page from the page history. It is used within HMI pages to support the operation of a "back" navigation button.
- **Examples subpalette**—This is a subpalette that contains the example VI for the HNE API.
- **Templates subpalette**—This is a subpalette that contains two template VIs for the HNE API. The first is a template VI for the navigation loop called HMI_NavigationEngine VI and the second is a template for an HMI page called HMI_Page VI.

The navigation engine uses VIs from the Touch Panel Navigation palette.

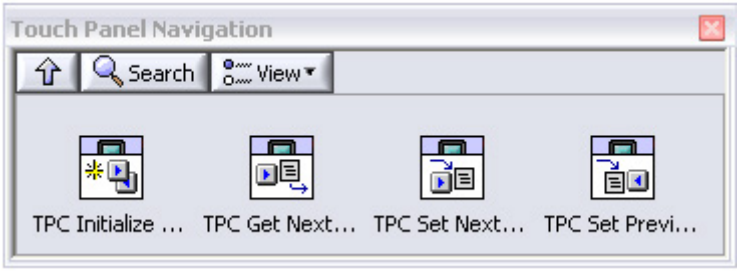


Figure 9.4. The Navigation Palette

- TPC Initialize Navigation—This VI initializes the navigation engine by setting the navigation history depth and the name of the first page to be displayed.
- TPC Get Next Page—This VI returns the name of the next page and passes it to the Case structure in the HMI navigation engine.
- TPC Set Next Page—This VI sets the name of the next page to be displayed. Use it within HMI pages to support navigation buttons.
- TPC Set Previous Navigation Page—This VI returns the name of the previous page from the page history. Use it within HMI pages to support the operation of a "back" navigation button.

The page state and history are stored in a functional global variable that each of these VIs accesses.

UI Pages

As stated above, the navigation loop contains all of the HMI pages for an application. Each HMI page is a LabVIEW VI created to monitor and configure a specific process or subprocess in the machine. The most common elements on a page front panel are navigation buttons, action buttons, numeric indicators, graphs, images, and Boolean controls and indicators. Figure 9.5 shows an example page containing a typical set of front panel elements.

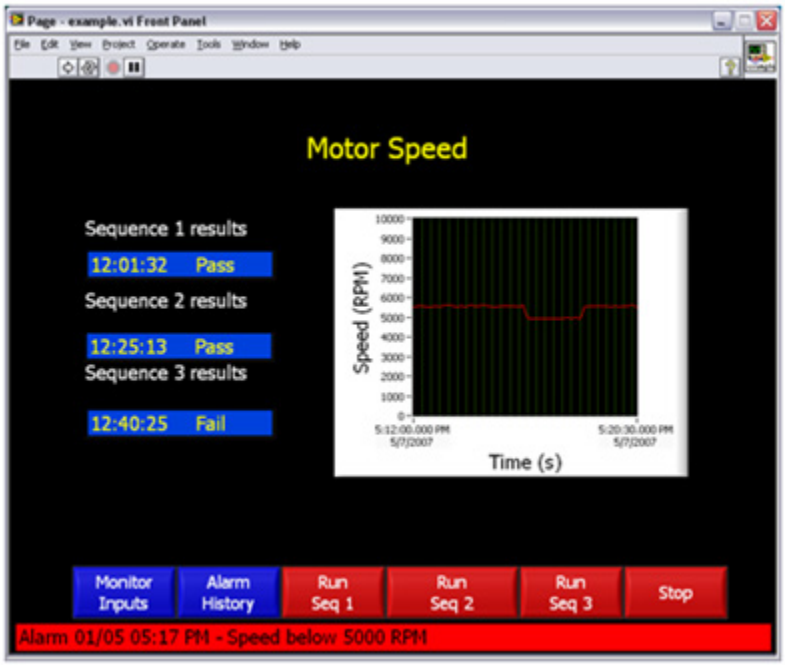


Figure 9.5. Example of a Typical UI Page in LabVIEW

The page block diagram uses the event-based producer consumer design pattern to implement a responsive, event-driven UI. The example in Figure 9.6 uses the Asynchronous Message Communication (AMC) reference library for interprocess communication. You can download the AMC reference library, which uses queues for interprocess communication, from the NI Developer Zone document Asynchronous Message Communication (AMC) Reference Library. The AMC library also has an API based on UDP that you can use to send messages across the network. You can implement the Figure 9.6 example using queue functions for communicating between the Event Handler process and the Message process.

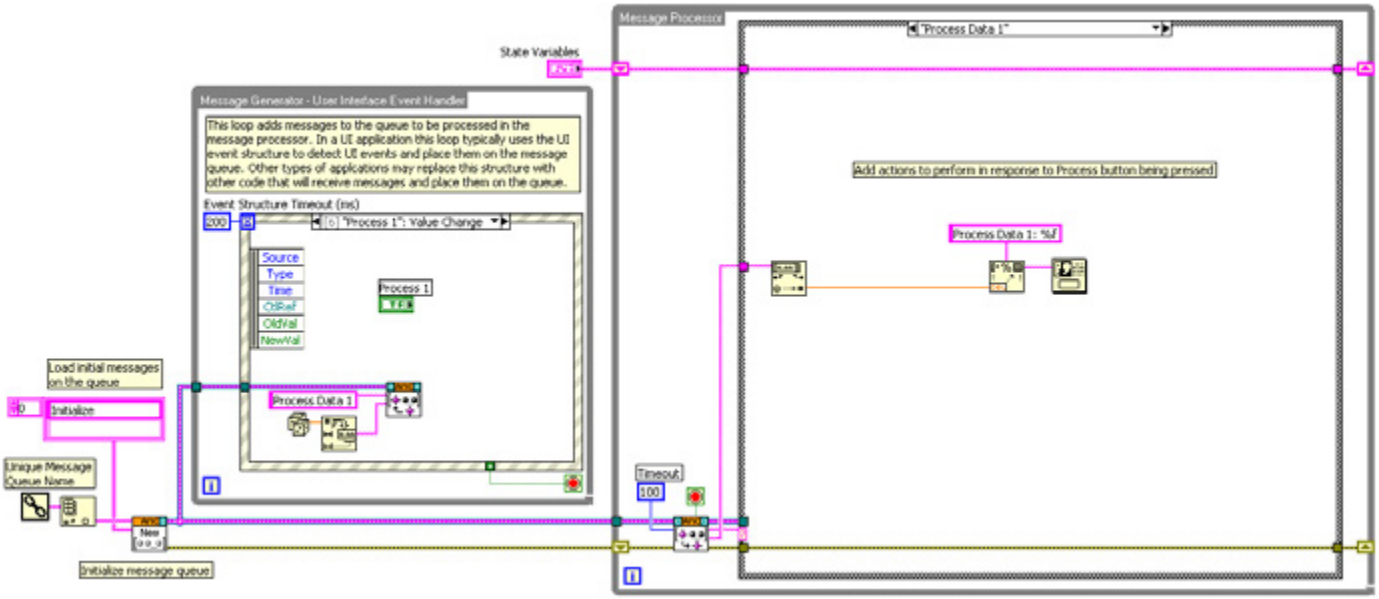


Figure 9.6. Example Page Block Diagram Using AMC Design Pattern

For more information on creating UI pages, see the NI Developer Zone document Creating HMI Pages for the LabVIEW Touch Panel Module.

CHAPTER 10
Adding Vision and Motion

Machine Vision/Inspection

Machine vision is a combination of image acquisition from one or more industrial cameras and the processing of the images acquired. These images are usually processed using a library of image processing functions that range from simply detecting the edge of an object to reading various types of text or complex codes.

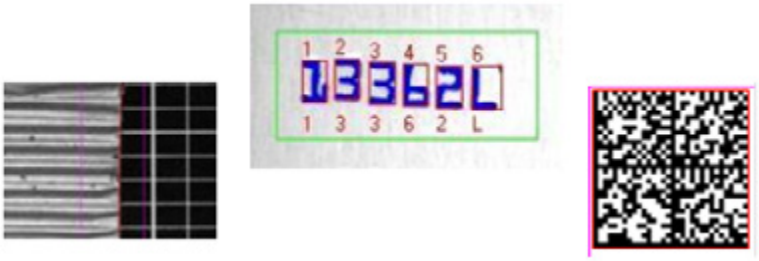


Figure 10.1. Edge detection, optical character recognition, and 2D code reading are common machine vision tasks.

Often more than one of these measurements is made on one vision system from one or more images. You can use this for many applications including verifying that the contents of a container match the text on the front of the bottle or ensuring that a code has printed in the right place on a sticker.

The information from these processed images is fed into the control system for data logging, defect detection, motion guidance, process control, and so on.

For information on the algorithms in NI vision tools, see the Vision Concepts Manual.

Machine Vision System Architecture

Typical machine vision systems consist of an industrial camera that connects to a real-time vision system, usually via a standardized camera bus such as IEEE 1394, Gigabit Ethernet, USB, or Camera Link. The real-time system processes the images and has I/O for communicating to the control system.

A few companies have combined the camera with the vision system, creating something called a smart camera. Smart cameras are industrial cameras that have onboard image processing and typically include some basic I/O.

NI offers both types of embedded machine vision systems. The NI Compact Vision System (Figure 10.2) is a real-time embedded vision system that features direct connectivity to GigE Vision cameras as well as FPGA-based I/O channels for synchronization and triggering. This system features Ethernet connectivity to your industrial network, allowing communication back to CompactRIO hardware. Vision integration is also directly available on certain CompactRIO targets through GigE Vision cameras and USB3 Vision cameras connected through USB 2.0 ports.

NI Smart Cameras (Figure 10.2) are industrial image sensors combined with programmable processors to create rugged, all-in-one solutions for machine vision applications. They have multiple resolutions from VGA (640x480 pixels)

to 5MP with color and monochrome options. These cameras feature dual Gigabit Ethernet ports, digital inputs and outputs, and a built-in lighting controller.

Figure 10.2. NI Compact Vision System and NI Smart Camera

NI also offers plug-in image acquisition devices called frame grabbers that provide connectivity between industrial cameras and PCI, PCI Express, PXI, and PXI Express slots. These devices are commonly used for scientific and automated test applications, but you also can use them to prototype a vision application on a PC before you purchase any industrial vision systems or smart cameras.

All NI image acquisition hardware uses the same driver software called NI Vision Acquisition Software. With this software, you can design and prototype your application on the hardware platform of your choice and then deploy it to the industrial platform that best suits your application requirements with minimal code changes.

Lighting and Optics

This guide does not cover lighting and optics in depth, but they are crucial to the success of your application. The following resources examine the majority of the basic and some more advanced machine vision lighting concepts:

A Practical Guide to Machine Vision Lighting, Part I

A Practical Guide to Machine Vision Lighting, Part II

A Practical Guide to Machine Vision Lighting, Part III

The lens used in a machine vision application changes the field of view. The field of view is the area under inspection that is imaged by the camera. You must ensure that the field of view (FOV) of your system includes the object you want to inspect. To calculate the horizontal and vertical FOV of your imaging system, use Equation 10.1 and the specifications for the image sensor of your camera.

$$FOV = \frac{Pixel\ Pitch \times Active\ Pixels \times Working\ Distance}{Focal\ Length}$$

Equation 10.1. Field of View Calculation

Where

- FOV is the field of view in either the horizontal or vertical direction
- Pixel pitch measures the distance between the centers of adjacent pixels in either the horizontal or vertical direction
- Active pixels is the number of pixels in either the horizontal or vertical direction

- Working distance is the distance from the front element (external glass) of the lens to the object under inspection
- Focal length measures how strongly a lens converges (focuses) or diverges (diffuses) light

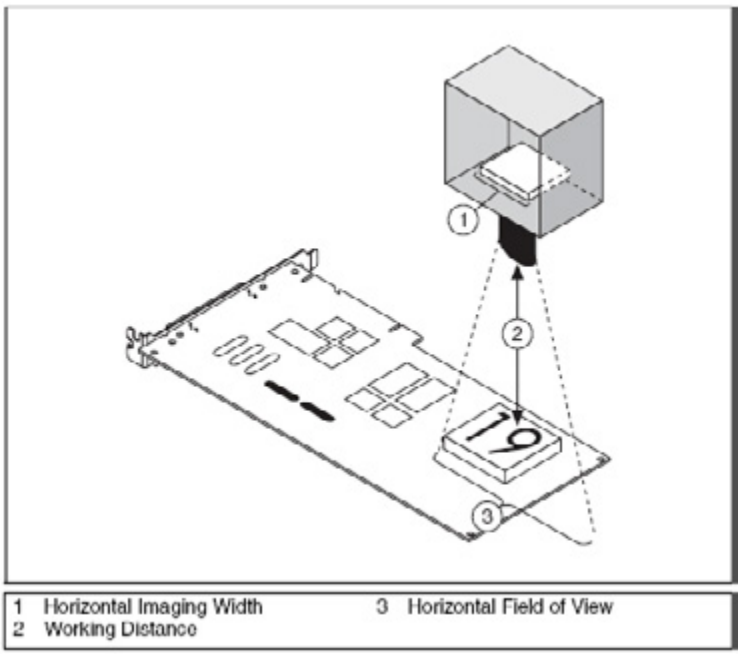


Figure 10.3. The lens selection determines the FOV.

For example, if the working distance of your imaging setup is 100 mm, and the focal length of the lens is 8 mm, then the FOV in the horizontal direction of an NI Smart Camera using the VGA sensor in full Scan Mode is

$$FOV_{horizontal} = \frac{0.0074\ \text{mm} \times 640 \times 100\ \text{mm}}{8\ \text{mm}} = 59.2\ \text{mm}$$

Similarly, the FOV in the vertical direction is

$$FOV_{vertical} = \frac{0.0074\ \text{mm} \times 480 \times 100\ \text{mm}}{8\ \text{mm}} = 44.4\ \text{mm}$$

Based on the result, you can see that you may need to adjust the various parameters in the FOV equation until you achieve the right combination of components that match your inspection needs. This may include increasing your working distance, choosing a lens with a shorter focal length, or changing to a high-resolution camera.

Software Options

Once you have chosen the hardware platform for your machine vision project, you need to select the software platform you want to use. NI offers two application development environments (ADEs) for machine vision. Both NI Compact Vision Systems and NI Smart Cameras are LabVIEW Real-Time targets, so you can develop your machine vision application using the LabVIEW Real-Time Module and the NI Vision Development Module.

The Vision Development Module is a library of machine vision functions that range from basic filtering to pattern matching and optical character recognition. This library also includes the NI Vision Assistant and the Vision Assistant Express VI. The Vision Assistant is a rapid prototyping tool for machine vision applications. With this tool, you can use

click-and-drag, configurable menus to set up most of your application. With the Vision Assistant Express VI, you can use this same prototyping tool directly within LabVIEW Real-Time.

Another software platform option is NI Vision Builder for Automated Inspection (AI). Vision Builder AI is a configurable machine vision ADE based on a state diagram model, so looping and decision making are extremely simple. Vision Builder AI features many of the high-level tools found in the Vision Development Module. Both hardware targets work with Vision Builder AI as well, giving you the flexibility to choose the software you are most comfortable with and the hardware that best suits your application.

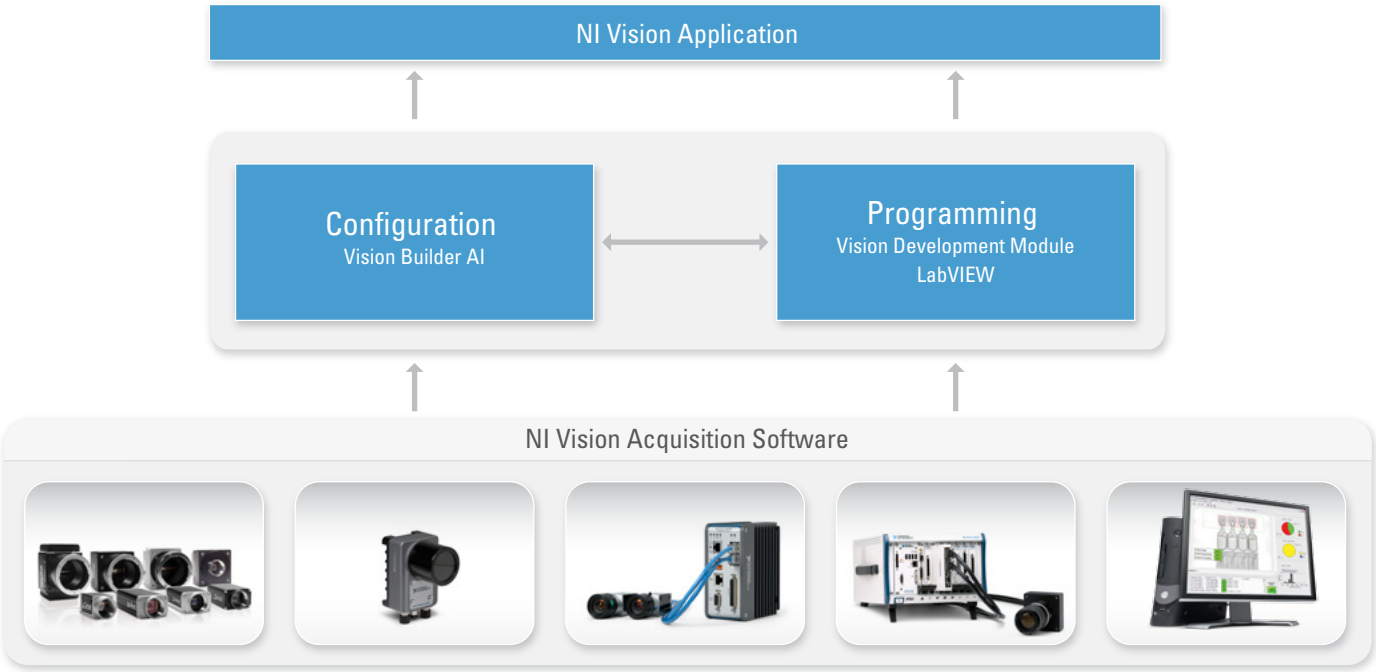


Figure 10.4. NI offers both configuration software and a full programming environment for machine vision application development.

Machine Vision/Control System Interface

Most smart camera or embedded vision system applications provide real-time inline processing and give outputs that you can use as another input into the control system. The control system usually has control over when this image acquisition and processing starts via sending a trigger to the vision system. The trigger can also come from hardware sensors such as proximity sensors or quadrature encoders.

The image is processed into a set of usable results such as the position of a box on a conveyor or the value and quality of a 2D code printed on an automotive part. These results are reported back to the control system and/or sent across the industrial network for logging. You can choose from several methods to report these results, from a simple digital I/O to shared variables or direct TCP/IP communication, as discussed previously in this document.

Machine Vision Using LabVIEW Real-Time

The following example demonstrates the development of a machine vision application for an NI Smart Camera using LabVIEW Real-Time and the Vision Development Module.

Step 1. Add an NI Smart Camera to the LabVIEW Project

You can add the NI Smart Camera to the same LabVIEW project as the CompactRIO system. If you wish to prototype without the smart camera connected, you can also simulate a camera.

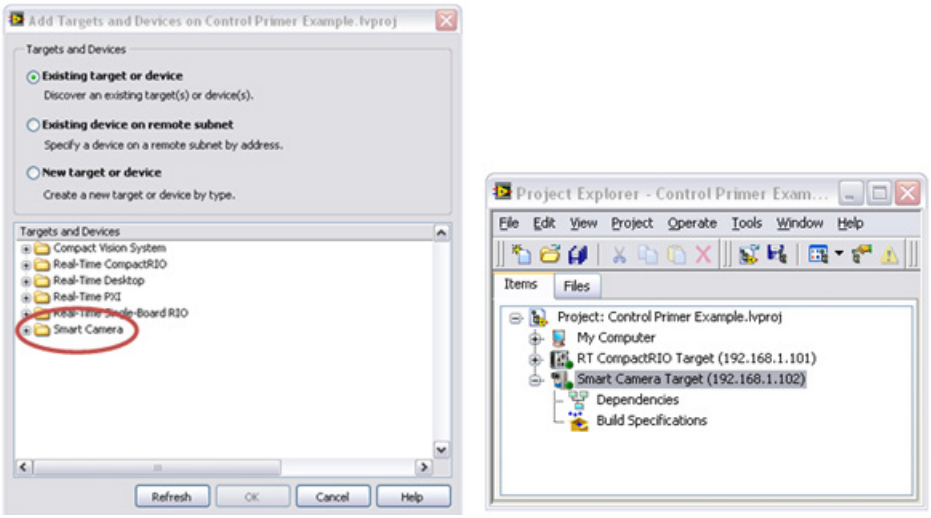


Figure 10.5. You can add NI Smart Camera systems to the same LabVIEW project as CompactRIO systems.

Step 2. Use LabVIEW to Program the NI Smart Camera

Creating an application for the smart camera is almost identical to creating an application for a CompactRIO real-time controller. The main difference is using the NI Vision Acquisition Software driver to acquire your images and algorithms in the Vision Development Module to process them.

You can create a new VI and target it to the smart camera just as you have created VIs for CompactRIO.

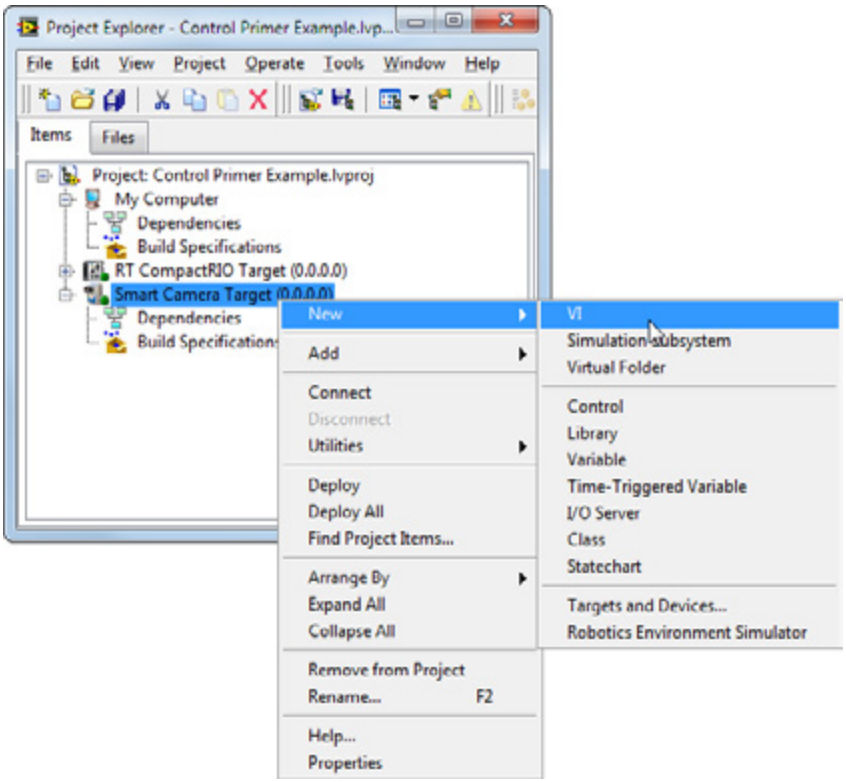


Figure 10.6. Adding a VI on Your NI Smart Camera to Your Project in LabVIEW Real-Time

Upon deployment, this VI resides in smart camera storage and runs on the smart camera during run time.

To simplify the process of acquiring and processing images, NI includes Express VIs in the Vision palette. Use these Express VIs in this example to acquire images from the smart camera (or vision system) as well as process the images. To access these Express VIs, right-click on the block diagram and choose **Vision»Vision Express.**

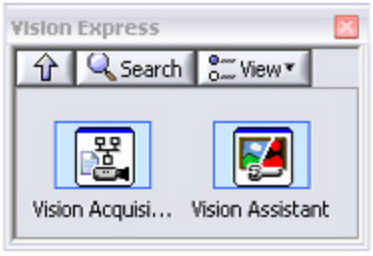


Figure 10.7. The Vision Express Palette

The first step is to set up an acquisition from your camera. If your camera is not available or not fixtured correctly yet, you also can set up a simulated acquisition by opening images stored on your hard drive.

Start by dropping the Vision Acquisition Express VI onto the block diagram. This menu-driven interface is designed so that you can quickly acquire your first image. If you have any recognized image acquisition hardware connected to your computer, it shows up as an option. If not, you have the option on the first menu to open images from disk.

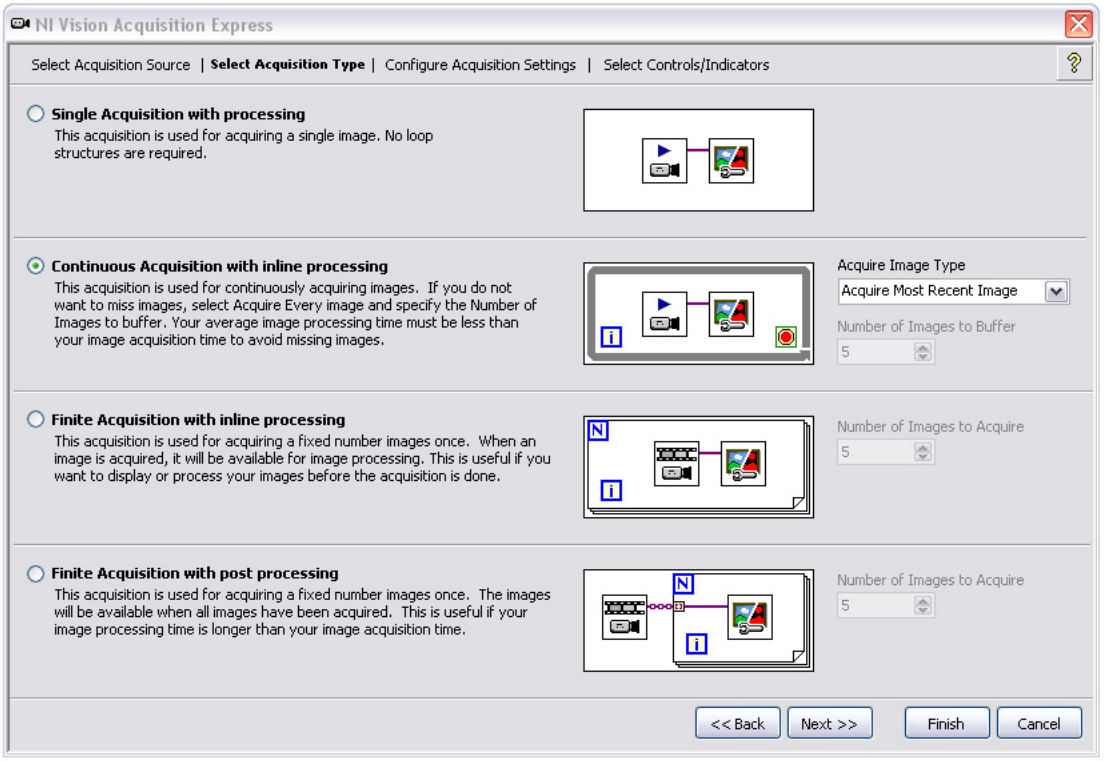


Figure 10.8. The Vision Acquisition Express VI guides you through creating a vision application in LabVIEW.

Next, choose which type of acquisition you are implementing. For this example, select Continuous Acquisition with inline processing so you can sit in a loop and process images. Next, test your input source and verify that it looks right. If it does, then click the finish button at the bottom.

Once this Express VI generates the LabVIEW code behind the scenes, the block diagram appears again. Now drop the Vision Assistant Express VI just to the right of the Vision Acquisition Express VI.

With the Vision Assistant, you can prototype your vision processing quickly. You can deploy this same tool to real-time systems, although traditional LabVIEW VIs are usually implemented to provide greater efficiency in real-time systems. For an overview of the tools in this assistant, view the NI Vision Assistant Tutorial.

Step 3. Communicate With the CompactRIO System

Once you have set up the machine vision you plan to conduct with the Vision Assistant Express VI, the last thing to do is communicate the data with the CompactRIO system. You can use network-published shared variables to pass data between the two systems. Network communication between LabVIEW systems is covered in depth in Chapter 4: Best Practices for Network Communication. In this example, you are examining a battery clamp, and you want to return the condition of the holes (if they are drilled correctly) and the current gap shown in the clamp.

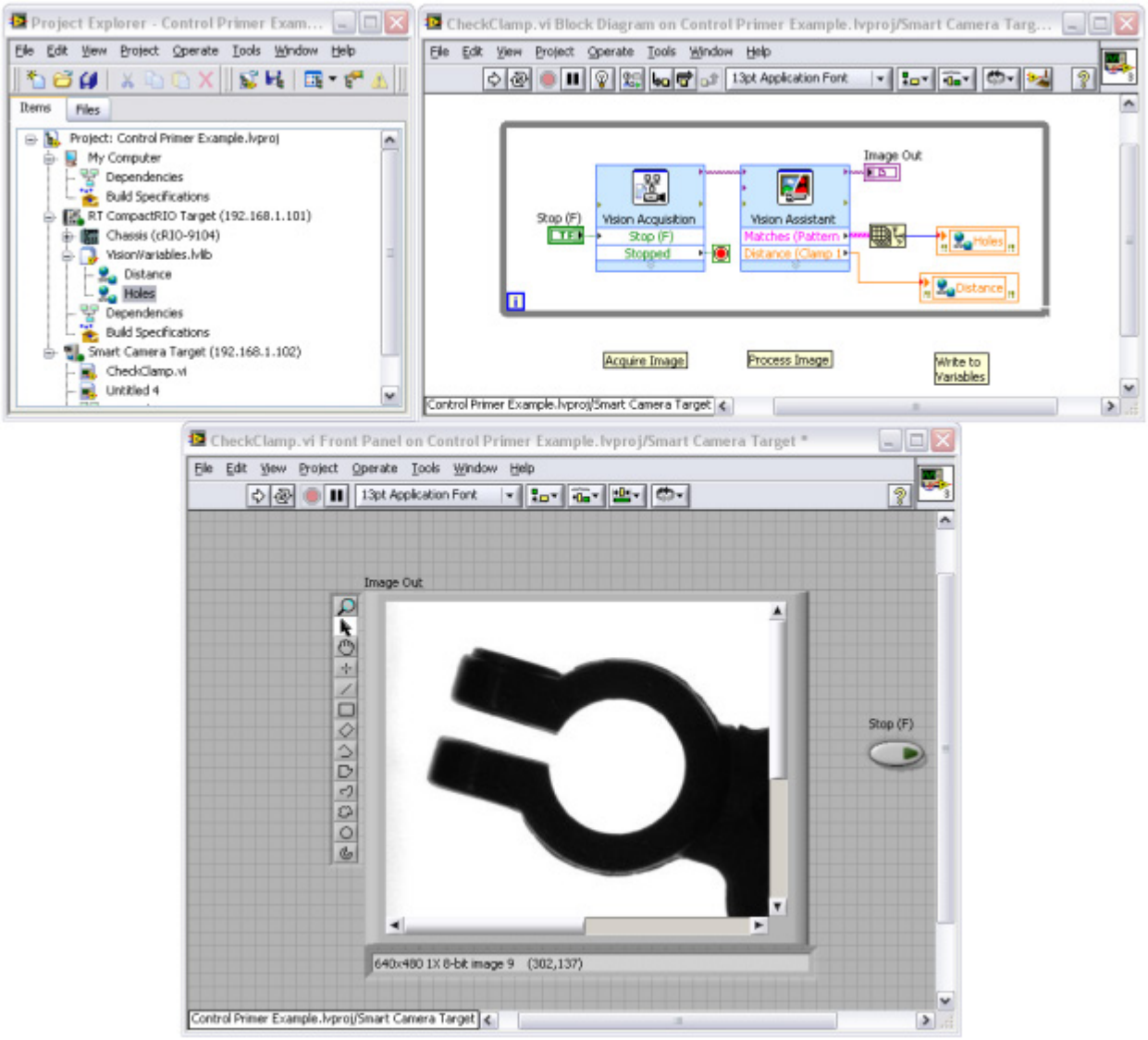


Figure 10.9. A Complete Inspection in LabVIEW

As you can see in Figure 10.9, the results of the inspection are passed as current values to the CompactRIO system via shared variables hosted on the CompactRIO system. You can also pass the data as commands to the CompactRIO system.

Machine Vision Using Vision Builder AI

As mentioned previously, Vision Builder AI is a configurable environment for machine vision. You implement all of the image acquisition, image processing, and data handling through configurable menus.

Step 1. Configure an NI Smart Camera With Vision Builder AI

When you open the environment, you see a splash screen where you can choose your execution target. If you do not have a smart camera connected, you can simulate a smart camera. Choose this option for this example.

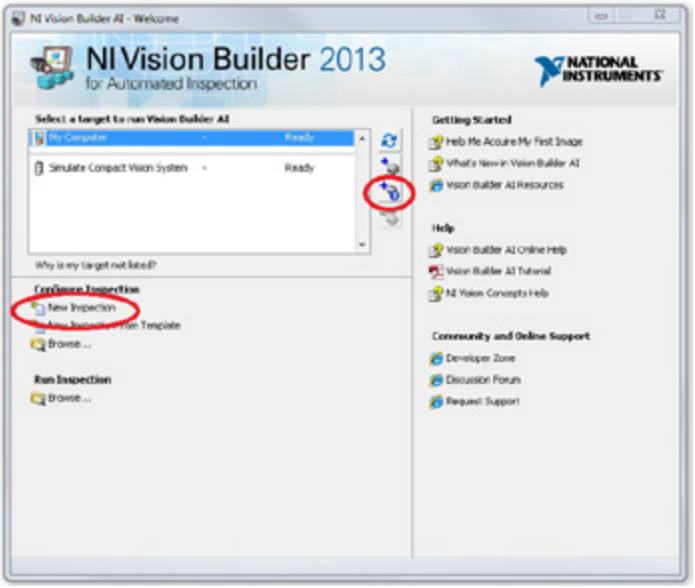


Figure 10.10. Choose your execution target from the Vision Builder AI splash screen.

With this emulator, you can set up lighting and trigger options, arrange I/O to communicate to the CompactRIO hardware, and complete many of the other actions required to configure the smart camera without having one available for your development system. Once you have selected your execution target, click on New Inspection under Configure Inspection. This takes you into the development environment, which features four main windows. Use the largest window, the Image Display Window, to see the image you have acquired as well as any overlays you have placed on the image. Use the window to the upper right, the State Diagram Window, to show the states of your inspection (with your current state highlighted). The bottom right window, the Inspection Step palette, displays all the inspection steps for your application. Lastly, the thin bar at the bottom is the Step Display Window, where you see all of the steps that are in the current state.

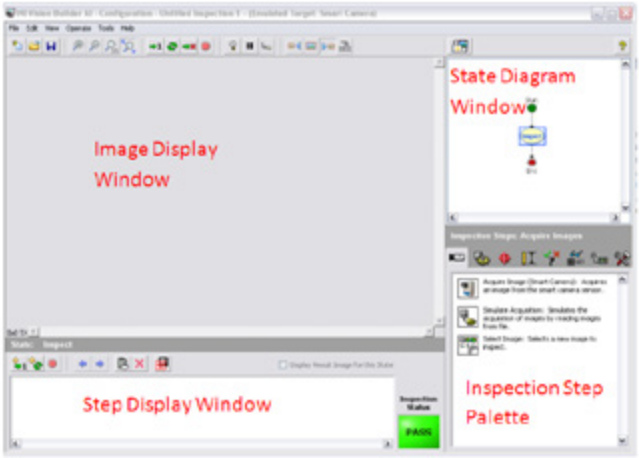


Figure 10.11. The Vision Builder AI Development Environment

This example implements the same inspection as the LabVIEW example that inspected battery clamps and returned the number of holes and the gap of the clamp back to CompactRIO via the two shared variables hosted on the CompactRIO hardware.

Step 2. Configure the Inspection

The first step of the inspection, just like in LabVIEW, is to acquire the image. Implement this with the Acquire Image (Smart Camera) step. Select this step and configure the emulation by clicking on the Configure Simulation Settings button. For example, set the image to grab the images from the following path:

C:\Program Files\National Instruments\Vision Builder AI 2013\DemoImg\Battery\BAT0000.PNG

This library of images should install with every copy of Vision Builder AI (with differing version numbers). After selecting the file above, also make sure to check the box for Cycle through folder images.

You also see that with this window, you can configure exposure times, gains, and other settings for the camera. These settings do not make any difference in the emulator, but in the real world, they go hand-in-hand with lighting and optics choices.

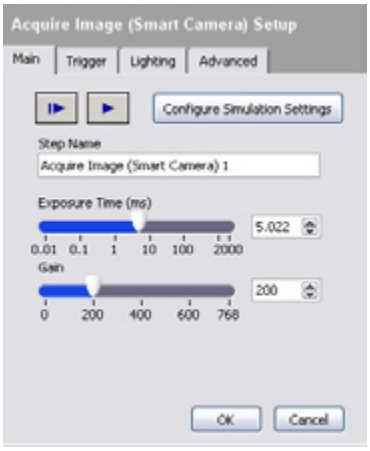


Figure 10.12. The Image Acquisition Setup Step

From here, set up pattern matching, edge detection, object detection, code reading, or any other algorithms you need. For example, implement a pattern match to set the overall rotation of the object, a detect objects to see if both of the holes were in the clamp, and a caliper tool to detect the distance between the clamp prongs. This generates a few pass/fail results that you can use to set the overall inspection status, which you can display on the overlay to the user.

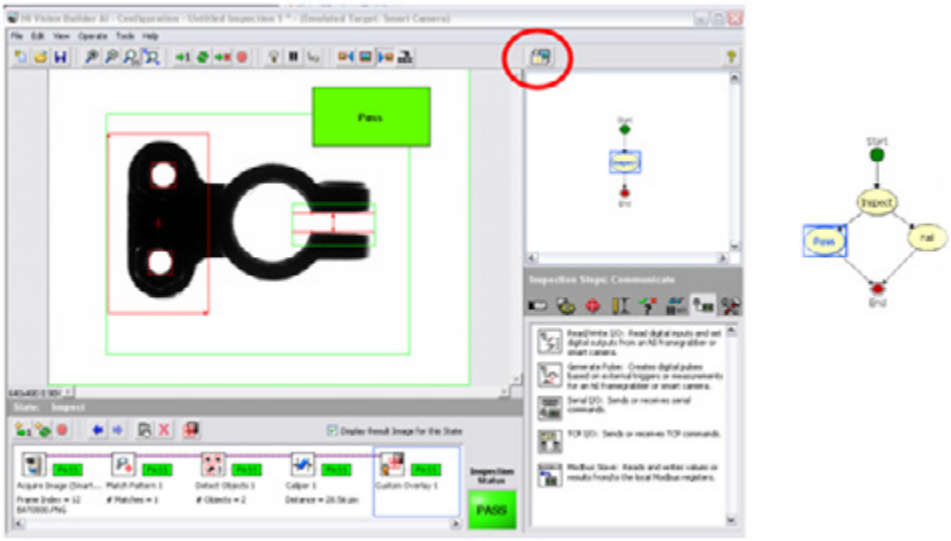


Figure 10.13. A Completed Inspection in Vision Builder AI

Now create two new states by clicking the toggle main window view button (circled in red). Create a pass state and a fail state. In both states, report the values back to CompactRIO but, in the fail state, also reject the part using some digital I/O.

Step 3. Communicate With the CompactRIO System

Vision Builder AI provides access to many of the I/O types discussed previously, including network-published shared variables, RS232, Modbus, Modbus TCP, and raw TCP/IP. In this example, use the variable manager to access the shared variables hosted on CompactRIO. Access the variable manager by navigating to **Tools» Variable Manager**.

Here you can discover the variables on the CompactRIO system by navigating to the Network Variables tab. From there, select and add the variables and bind them to variables within the inspection.

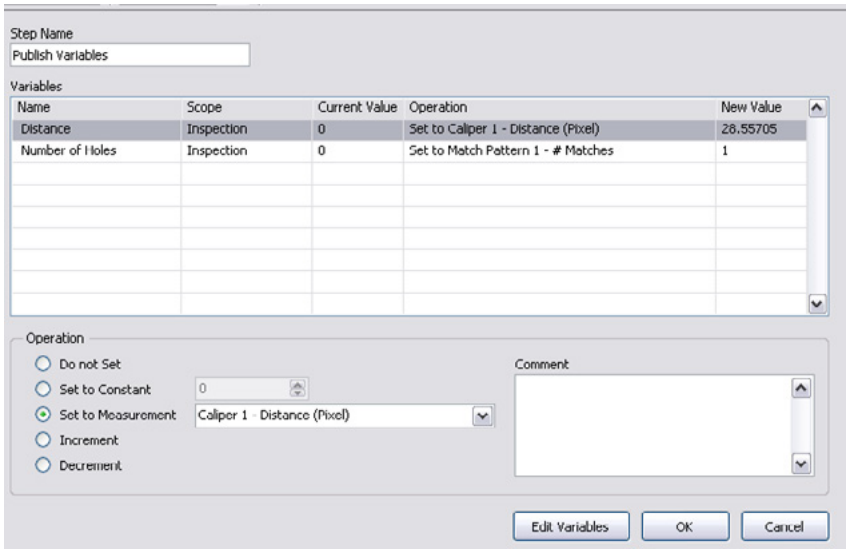


Figure 10.14. Setting Up Variable Communication in Vision Builder AI

Now these results are sent back to the CompactRIO system, and the inspection waits on the next camera trigger.

As you can see, both methods (programmable and configurable) offer the user a way to acquire images, process them, and then use the pertinent information to report results back to a CompactRIO control system or to directly control I/O from a real-time vision system.

Motion Control

The term "motion control" is applied to applications ranging from simple on-off or open-loop control of rotary equipment like fans and pumps to high-precision multiaxis position or velocity control synchronized to measurement and control I/O. You can implement simple motion control through industrial control I/O: a standard digital output to a motor starter or a PWM signal to a motor drive. Complex motion control applications, on the other hand, require specialized software and hardware: supervisory control, cascaded control loops, motor commutation schemes, advanced tuning algorithms, specialized sensors, actuators, feedback devices, and high-speed/high-precision I/O.

You can use CompactRIO for both simple and complex motion applications. For simple applications like standard pump or fan control, generic analog or digital I/O modules and user-defined logic are sufficient. For more complex applications that require precise position or velocity control, multiaxis coordination, or synchronization with I/O, NI offers the LabVIEW NI SoftMotion Module.

NI SoftMotion turns a CompactRIO controller into a configurable custom motion controller by providing modular motion control components that run on the CompactRIO real-time controller and user-configurable FPGA. Components include APIs for motion programming in LabVIEW or in C, a motion engine that handles supervisory control and trajectory generation, and motion control IP blocks for control loops and motion I/O. The modular components of NI SoftMotion are combined differently depending on the system architecture and motion I/O selection. Because NI SoftMotion is implemented in the LabVIEW Real-Time and LabVIEW FPGA modules with user-accessible components, it provides direct integration with all the other capabilities of CompactRIO at both the LabVIEW Real-Time and LabVIEW FPGA levels.

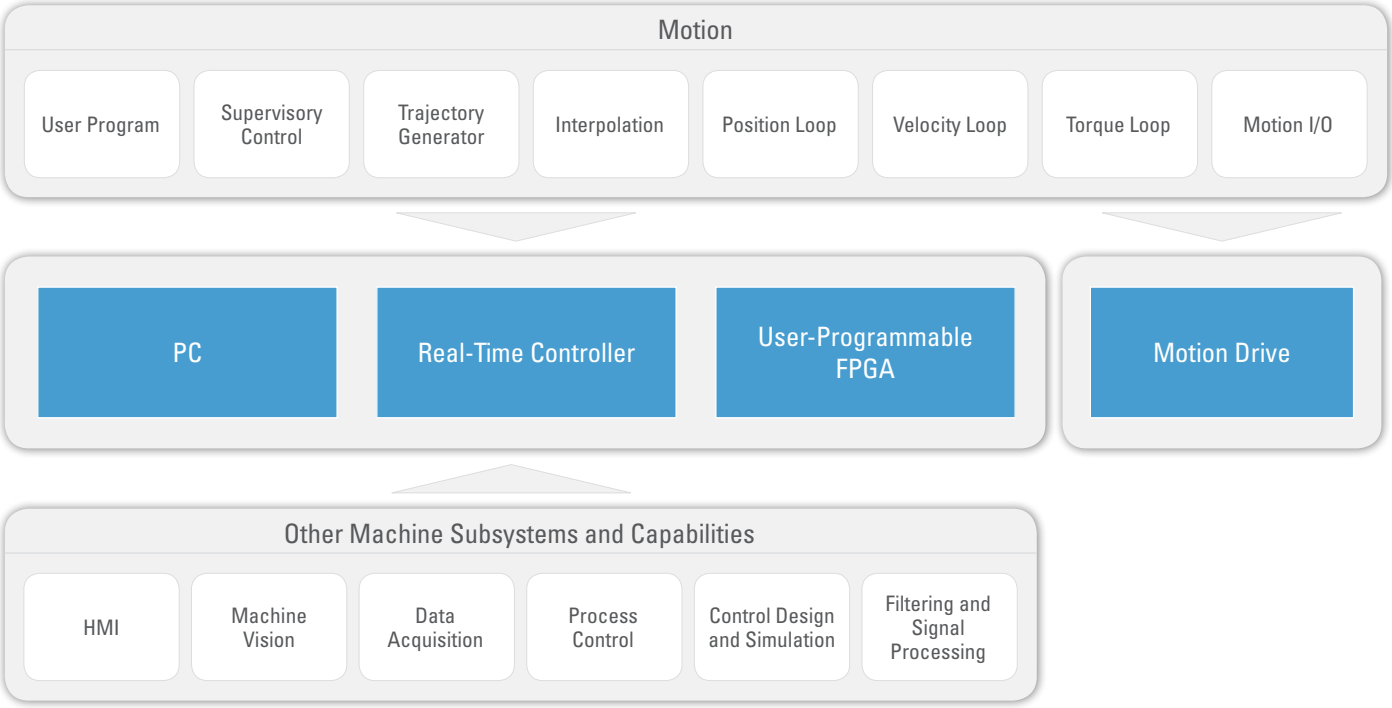


Figure 10.15. The LabVIEW NI SoftMotion Module turns a CompactRIO controller into a configurable custom motion controller.

You can choose from three versions of NI SoftMotion: Lite, Standard, and Premium.

- **LabVIEW NI SoftMotion Module Lite**—Provides a subset of functionality offered by the LabVIEW NI SoftMotion Module for free. With this version, you can perform simple single-axis point-to-point motion using NI C Series drive interfaces, create motion applications for NI CompactRIO hardware, and connect to AKD drives through EtherCAT.
- **LabVIEW NI SoftMotion Module Standard**—Offers full support for coordinated motion. You can use it to create coordinate spaces, easily achieve multiaxis synchronization, and perform electronic gearing and camming. The NI SoftMotion for SolidWorks interface is also included. With this tool, you can apply your custom motion application to a 3D CAD model and visualize, simulate, and validate your application before deploying to hardware.
- **LabVIEW NI SoftMotion Module Premium**—Includes everything that comes with LabVIEW NI SoftMotion Module Standard and adds support for creating custom motion applications that target hardware options other than the C Series drive interfaces and the AKD servo drive. Create custom axes using standard I/O devices such as C Series I/O modules for CompactRIO or third-party EtherCAT drives.

LabVIEW NI SoftMotion Features	Lite	Standard	Premium
Point-to-point motion	X	X	X
Motion I/O	X	X	X
Coordinated motion		X	X
Gearing/camming		X	X
Customization			X

Table 10.1. LabVIEW NI SoftMotion Module Features by Version

Visit the LabVIEW NI SoftMotion Module page on ni.com for more information and for the option to download an evaluation copy. This evaluation copy contains all the features of LabVIEW NI SoftMotion Premium for the evaluation period and then defaults to LabVIEW NI SoftMotion Lite until you provide an activation code for Standard or Premium.

Components of a Motion System

A typical motion control system used for position or velocity control consists of the following components:

- **Motion controller**—The processing element that runs software algorithms and closed control loops to generate the motion profile commands based on the move constraints from the user-defined application software and I/O feedback.
- **Communication interface**—Converts the command signals from a motion controller to actual digital or analog values that the drive can interpret.
- **Drive**—Receives the digital or analog data from a motion controller after conversion in the communication interface and converts it to real current/voltage that it applies to the motor to perform the commanded motion. In many cases, it also has a processing element that closes high-speed control loops. Some advanced drives can offload the position and velocity control loops from the motion controller and run them on board the drive.
- **Motor**—Converts the electrical energy from the drive/amplifier into mechanical energy. The motor torque constant, kt, and the motor efficiency define the ratio between motor current and mechanical torque output.
- **Mechanical transmission**—Consists of the components connected to the motor that direct the rotary motion of the motor to do work. This typically involves mechanical devices such as gearboxes or pulleys and lead screws that convert the rotary motion at the motor shaft to a linear movement at the payload with a certain transmission gear ratio. Common examples are belts, conveyors, and stages.
- **Feedback sensors**—Help close the control loops to provide instantaneous position and velocity information to the drive/amplifier and the motion controller.
- **Motion I/O**—Relays information such as limit switches, drive status, and other synchronization information back to the motion controller through the communication interface.

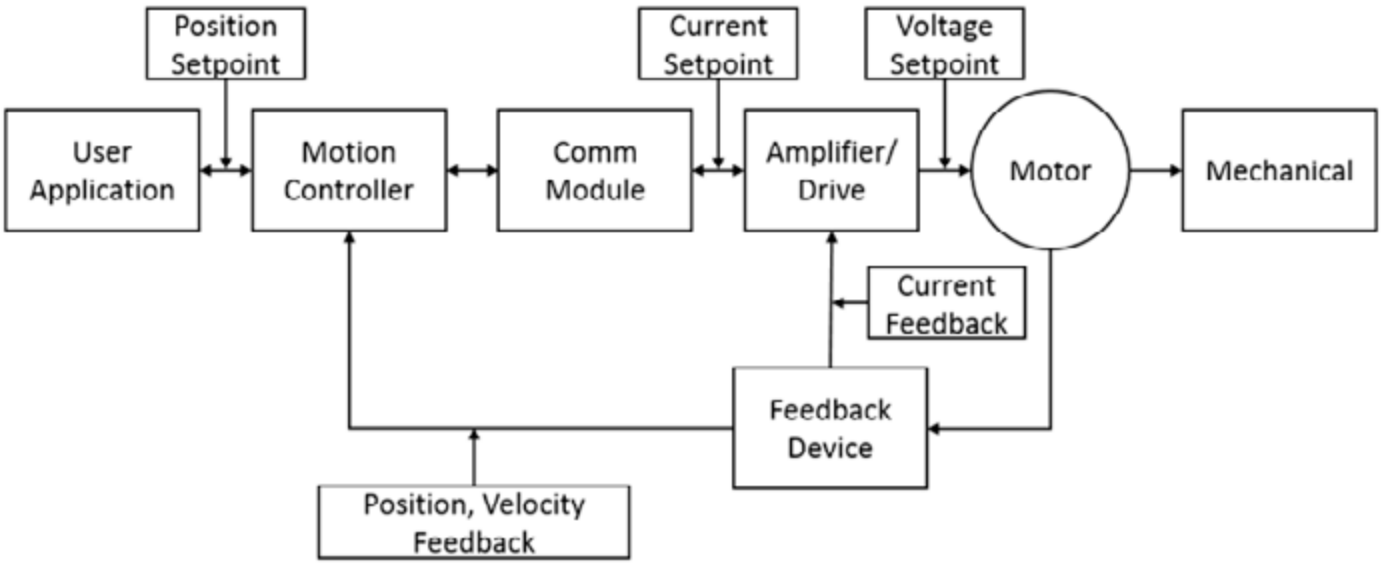


Figure 10.16. Simplified Motion System Diagram

The motion controller component of the system is divided into four components:

- Supervisory control—This top control loop executes command sequencing and passes commands to the trajectory generation loops. This loop performs the following:
    - System initialization, which includes homing to a zero position
    - Event handling, which includes triggering outputs based on position or sensor feedback and updating profiles based on user-defined events
    - Fault detection, which includes stopping moves on a limit switch encounter, safe system reaction to emergency stop or drive faults, and other watchdog actions
- Trajectory generator—This loop receives commands from the supervisory control loop and generates path planning based on the profile specified by the user. It provides new location setpoints to the control loop in a deterministic fashion. As a rule of thumb, this loop should execute with a 5 ms or faster loop rate.
- Control loop(s)—These are fast control loops that execute every 50 µs. They use position and velocity sensor feedback and the setpoint from the trajectory generator to create the commands to the drive. Because these loops run faster than the trajectory generator, they also generate intermediate setpoints based on time with a routine called spline interpolation. For stepper systems, one of the control loops is replaced with a step generation component.
- Feedback devices—These are sensors such as encoders and limit switches that provide instantaneous position and velocity information to the drive/amplifier and the motion controller.

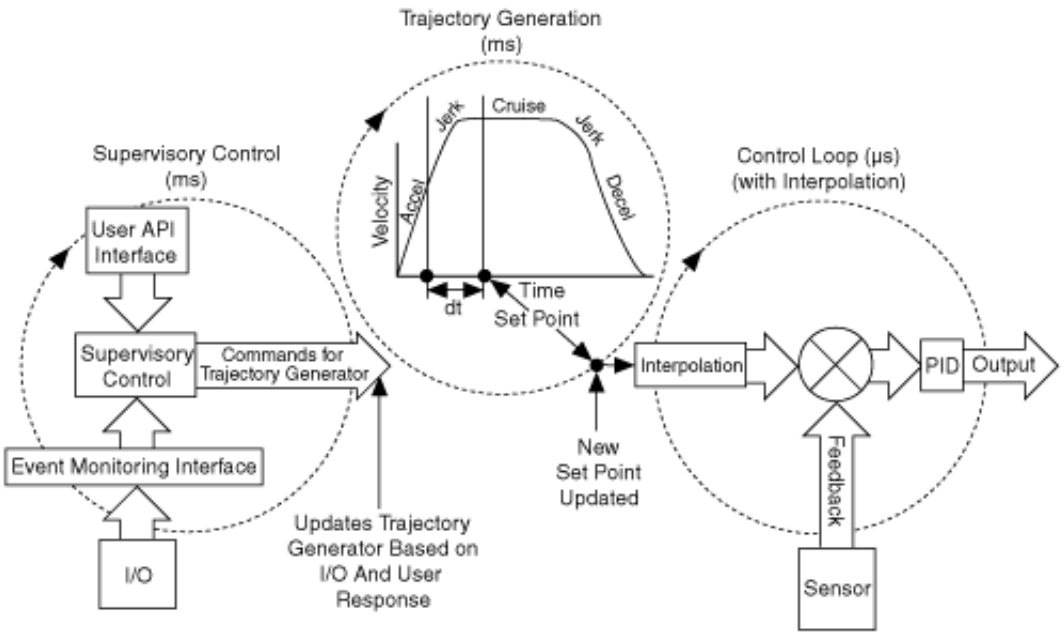


Figure 10.17. Functional Architecture of NI Motion Controllers

The LabVIEW NI SoftMotion Module provides these motion controller components implemented in the LabVIEW Real-Time and LabVIEW FPGA modules as well as an API in LabVIEW and C for creating motion control applications on top of these core components.

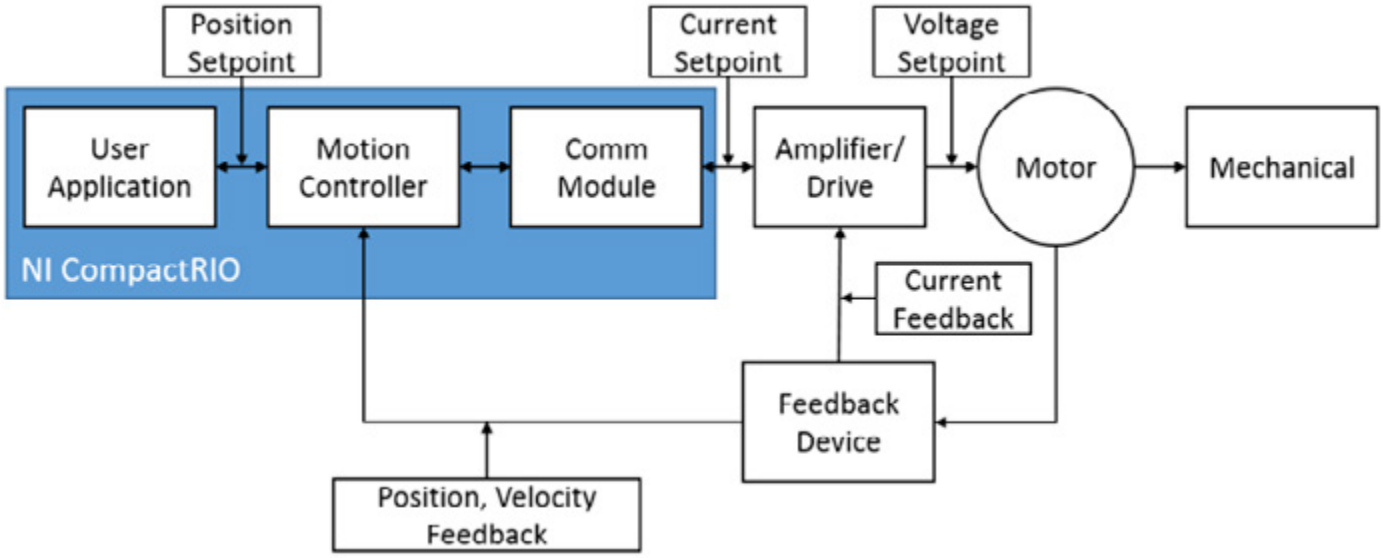


Figure 10.18. Simplified Motion System Diagram Showing Which Components Are Implemented in the LabVIEW RIO Architecture (in this case, with a CompactRIO controller)

NI SoftMotion Architecture

The LabVIEW NI SoftMotion Module contains components that run in LabVIEW for Windows, LabVIEW Real-Time, and LabVIEW FPGA. It also contains interfaces to a variety of motion specific devices and generic I/O.

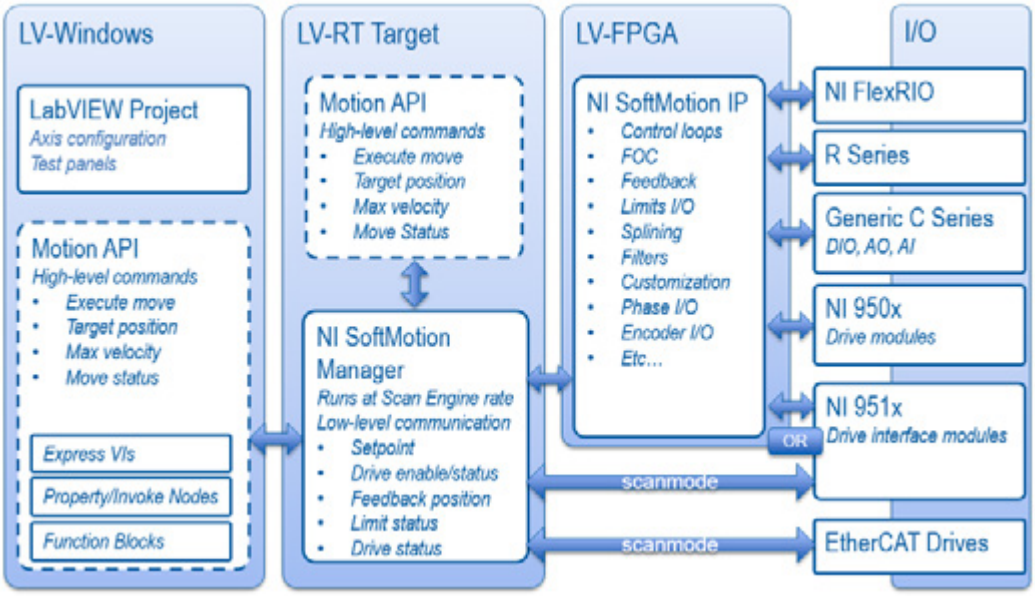


Figure 10.19. LabVIEW NI SoftMotion Components and Motion I/O Hardware Options

NI SoftMotion provides an architectural framework and common API that are hardware agnostic; it can be used to control a variety of different motion hardware configurations. NI SoftMotion is also implemented with open IP in the LabVIEW Real-Time and LabVIEW FPGA modules, so you can modify specific components as necessary but still use custom components with the other parts of the NI SoftMotion architecture. Because NI SoftMotion is designed to be hardware agnostic and open, it provides

- Easy coordination of different axis types
- High-level programming consistency
- Tight integration with other components running in LabVIEW Real-Time or LabVIEW FPGA
- Customization for unique application requirements

Examples of application-specific customization include FPGA triggering and timing based on other C Series I/O in the system, changing the trajectory generation method, or modifying the control-loop algorithms to implement a sensorless startup routine.

This section of the guide examines LabVIEW NI SoftMotion Module components and the motion-specific hardware with which they interface.

NI SoftMotion Project Items and APIs

PC-based NI SoftMotion components include the LabVIEW project-based configuration and high-level motion programming. All NI SoftMotion configuration and hardware setup is managed through the LabVIEW project. When NI SoftMotion is installed, you can access a list of NI SoftMotion items by right-clicking on a target in the system.

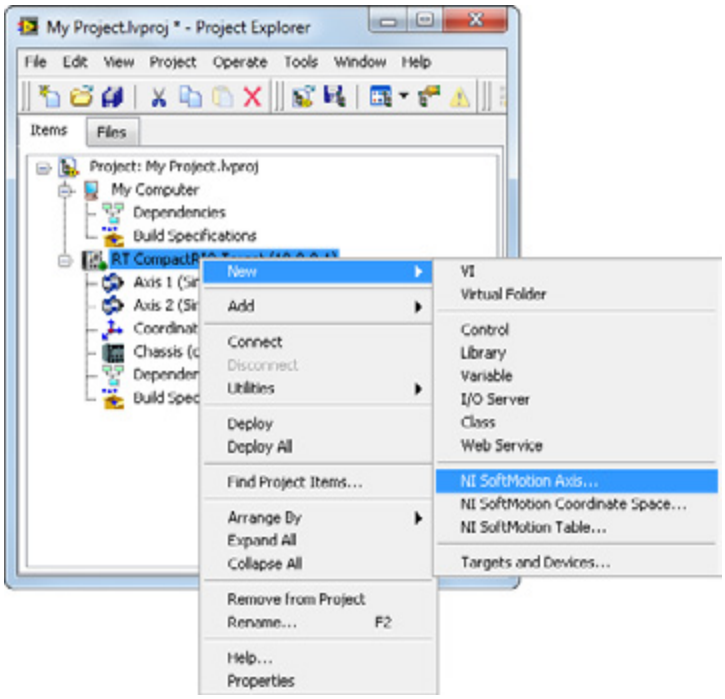


Figure 10.20. NI SoftMotion Resource Items Available From the LabVIEW Project

An NI SoftMotion axis is a resource that contains the settings and configuration for a specific motion drive, motor, or feedback device. You use axis resources on the block diagram to specify which resources interact with the NI SoftMotion API. An axis is bound to a specific hardware type to give it the necessary characteristics to configure and operate with that physical hardware. Axes are bound from the Axis Manager, which you access by right-clicking on an axis in the LabVIEW project.

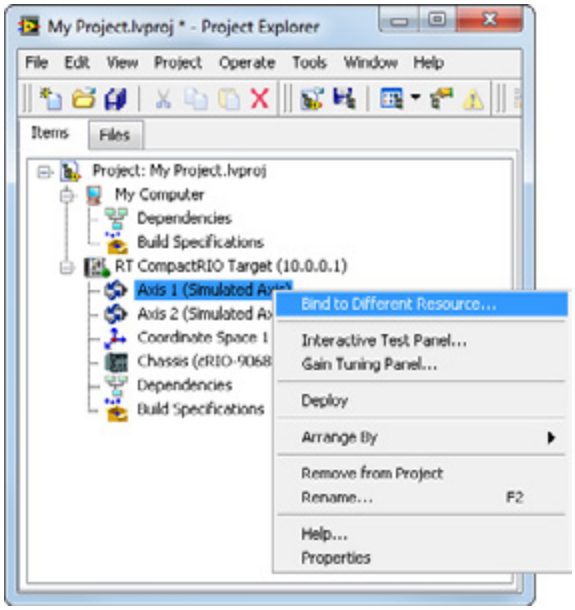


Figure 10.21. Access the Axis Manager by right-clicking on an axis in the LabVIEW project.

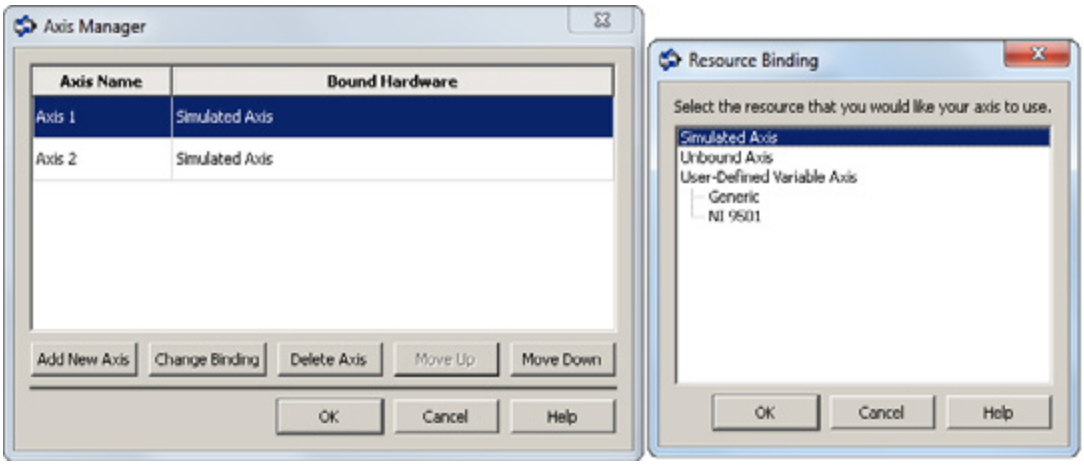


Figure 10.22. Use the NI SoftMotion Axis Manager and Resource Binding Dialog to associate an axis with a specific type of hardware.

NI SoftMotion contains the following axis types:

- **Simulated axis**—Configure axes and create and execute motion applications even if you do not have hardware installed. This is useful for simple prototyping and setup, or for advanced applications as a trajectory master for electronic gearing and camming operations. However, because you are not connected to actual hardware, you cannot test control-loop settings or any I/O. The simulated axis, which is available only as a servo axis, features the following characteristics.

I/O Type	Value
Encoder 0 position	Trajectory generator setpoint
Encoder 0 velocity	Trajectory generator commanded velocity
Encoder 1 position and velocity	0
Analog input	Sine wave
Steps generated	0
Limits and Home input status	High or On
Position error, capture, watchdog, compare, fault status	FALSE
Digital input	Pulse

Table 10.2. A simulated axis returns simulated data for testing NI SoftMotion applications without requiring a physical drive and motor in the system.

- **SolidWorks axis**—The SolidWorks axis type provides a connection to a motor defined in the SolidWorks Motion Analysis add-on. With this axis type, you can synchronize Dassault Systemes SolidWorks assemblies for motion system simulation. Using NI SoftMotion with SolidWorks to simulate your system with actual motion profiles, you can design motion profiles; detect collisions; simulate the mechanical dynamics of your machine including mass and friction effects; estimate machine cycle time performance; validate component selections for motors, drives, and mechanical transmissions; and evaluate engineering trade-offs between the mechanical, electrical, control, and embedded system aspects of the design. For more details, see the Virtual Prototyping Training Series.

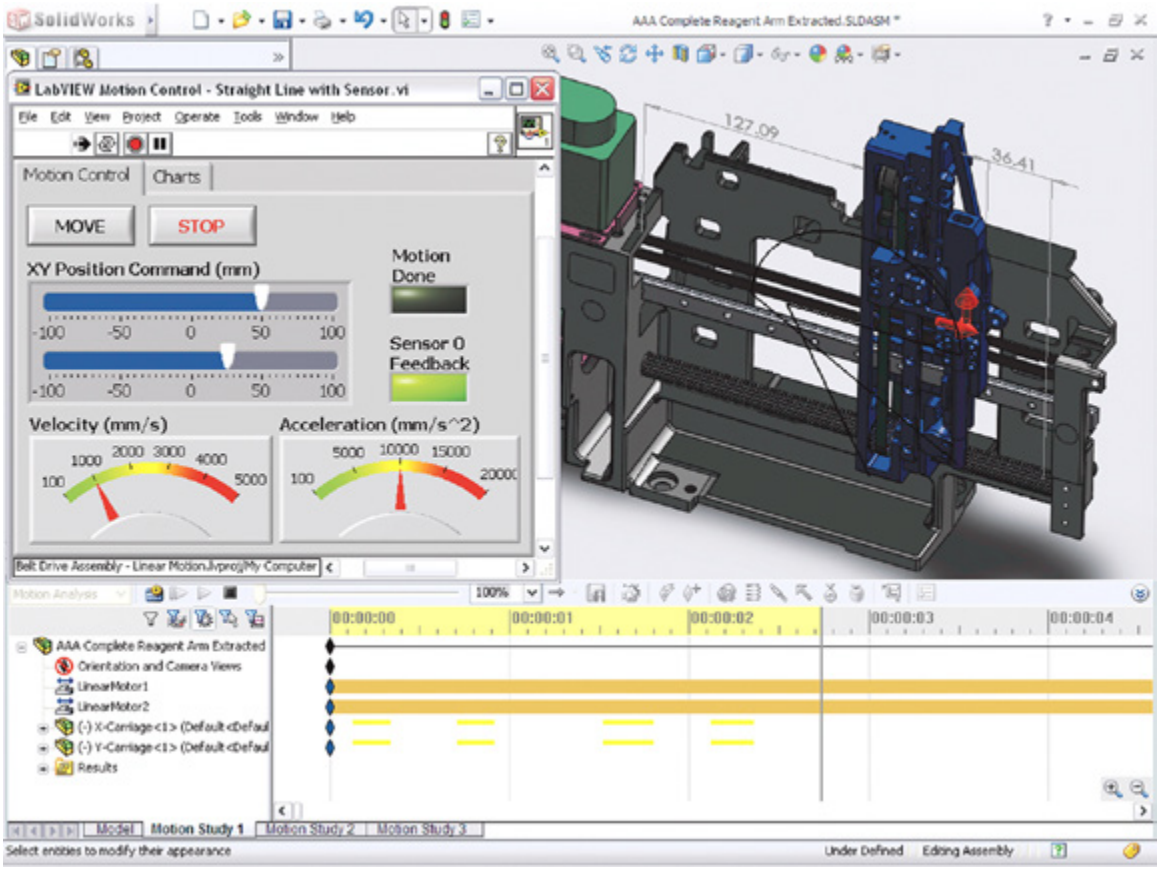


Figure 10.23. Interface NI SoftMotion with a SolidWorks assembly through the LabVIEW project to control the model with LabVIEW code.

- **NI 951x axis**—Use NI 951x C Series modules directly from LabVIEW Real-Time with the NI Scan Engine, or use these modules in FPGA mode with the FPGA I/O nodes for the modules.
- **EtherCAT drive**—Configure and use an EtherCAT AKD servo drive completely through NI SoftMotion. You do not need to use the I/O variables under the EtherCAT device when using the EtherCAT AKD drive with NI SoftMotion.
- **Unbound axes**—Create custom motion applications in situations where the system requires features or functionality not available in other motion controllers. This may be a non-supported communication interface, specialized I/O, or customized control algorithms for precise control. If you use unbound axes with the Axis Interface nodes, your main application VI can use the same NI SoftMotion APIs for all axis types, which means that you can have unbound axes and axes bound to other resources in the same application. Unbound axes are used with most NI 950x C Series drive modules and when implementing motion control with generic C Series I/O modules.
- **User-defined variable axes**—Use user-defined variable (UDV) axes to implement an interface for communication between the NI SoftMotion Engine and the LabVIEW FPGA Module. UDV axes allow your main application VI to use the same NI SoftMotion API for all axis types, which means that you can have UDV axes and axes bound to other resources in the same application. The UDV communication module handles communication between the NI SoftMotion Engine and the LabVIEW FPGA VI using user-defined variables that you add to the project. NI SoftMotion supports the following UDV axis types:

- **Generic user-defined variable axis**—Works with UDV axes not associated with a currently supported C Series module. When you use a generic UDV axis, the axis configuration dialog box contains all available configuration options.
- **NI 9501 user-defined variable axis**—Works with UDV axes associated with an NI 9501 C Series stepper drive module. When you use an NI 9501 UDV axis, the axis configuration dialog box contains configuration options customized for the NI 9501 properties.

As the above list attests, the axis is an abstraction that you can use to bind various types of hardware to it. The benefit to this is the ability to combine dissimilar axis types in the same VI using a consistent high-level programming API. You can prototype a program using a simulated axis and then deploy it to a real-time system, change the axis binding to the real hardware, and run the code on the real-time system without changing anything about the way the high-level code is written.

Right-click on each axis to bind that axis to a specific physical interface or drive in the system, and to configure the settings and properties for that axis.

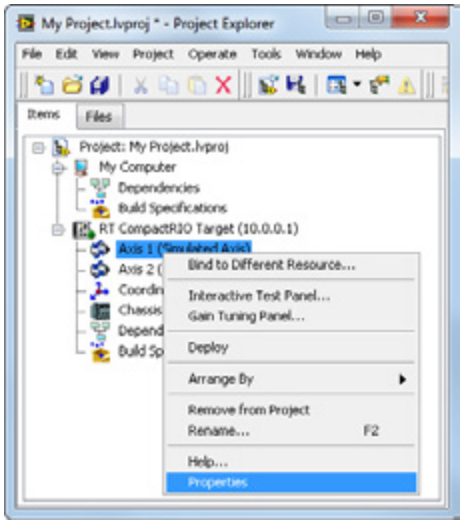


Figure 10.24. Right-click on an axis in the LabVIEW project to bind it to a physical resource, change its properties, or test the settings with the interactive test panel.

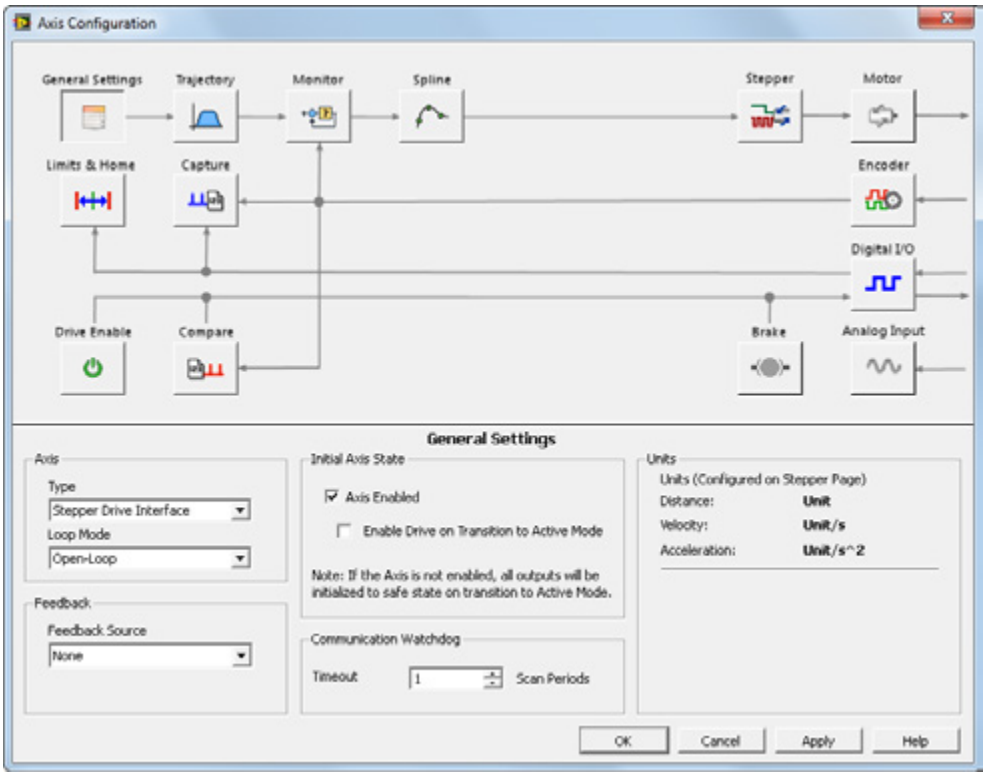


Figure 10.25. The items and settings on the Axis Configuration panel change depending on the axis type.

You can use the Interactive Test Panel to test and debug your motion system. With the Interactive Test Panel, you can perform a simple straight line move and monitor move and I/O status information, change move constraints, obtain information about errors and faults in the system, and view the position or velocity plots of the move. If you have a feedback device connected to your system, you can also obtain feedback position and position error information.

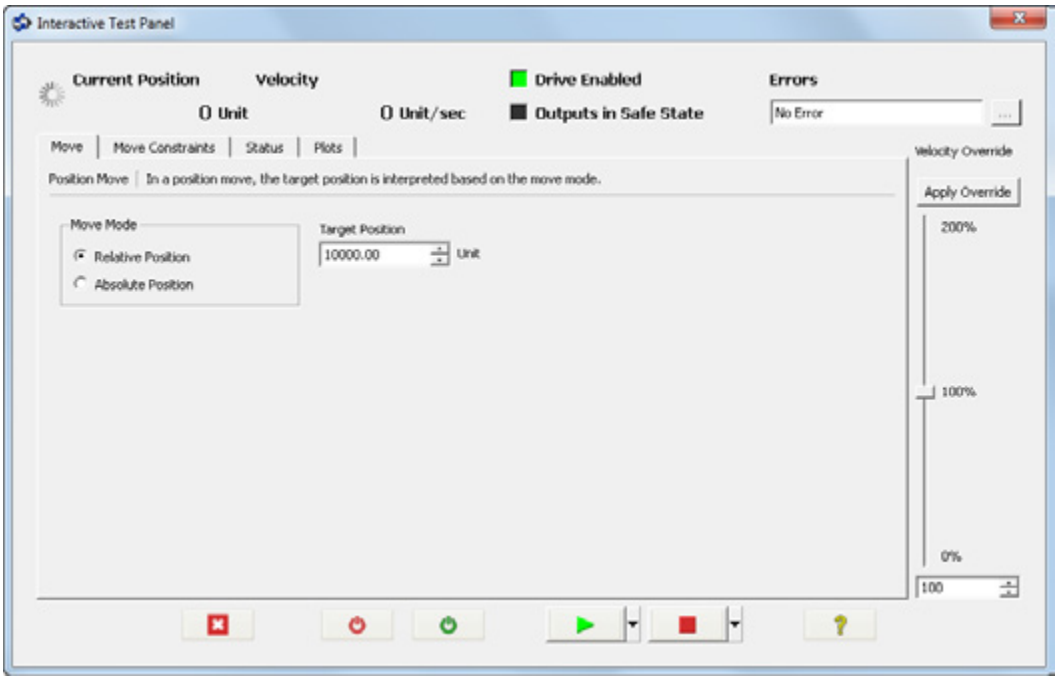


Figure 10.26. Perform test moves, check limit switch settings, enable/disable drives, and see any errors from the Interactive Test Panel.

To open the interactive window, right-click the axis in the LabVIEW Project Explorer window and select Interactive Test Panel. Set the desired position, move mode, and move constraints using the tabs. Click the Start button on the bottom of the dialog box to start the move with the configured options. Use the Status and Plots tabs to monitor the move while it is in progress.

Once you configure an axis, you use it on the LabVIEW block diagram as a reference to tell the NI SoftMotion API which axis to use when executing functions. You can drag and drop axes from the project to the block diagram or place them from the NI SoftMotion palette.

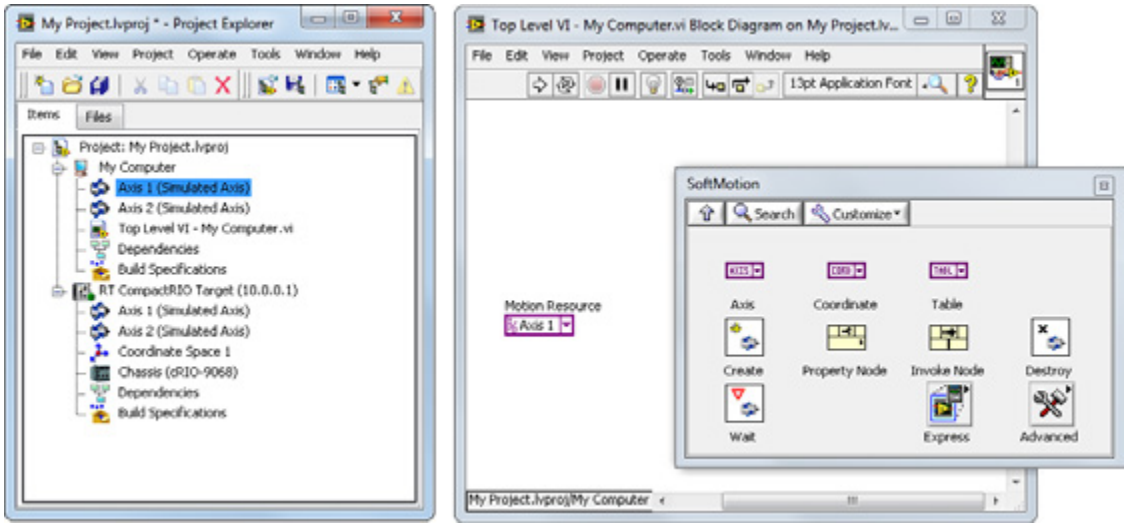


Figure 10.27. You can drag an Axis resource from the LabVIEW project to the block diagram or drop it from the palette and associate it with an item in the LabVIEW project.

When you install LabVIEW NI SoftMotion, you get a palette of motion VIs that includes two APIs: the Property/Invoke node API and the Express VI API.

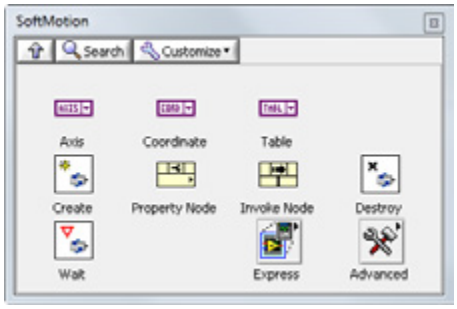


Figure 10.28. The NI SoftMotion Palette

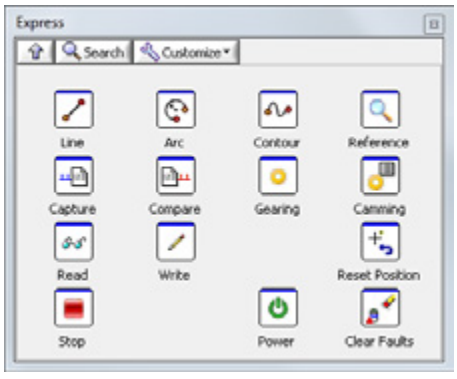


Figure 10.29. The NI SoftMotion Express VI Palette

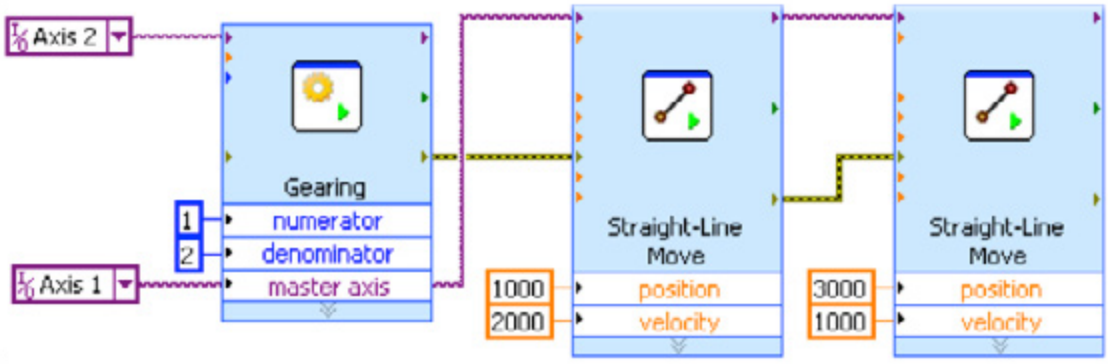


Figure 10.30. Axis 2 is geared to Axis 1 with a 1:2 gear ratio for the two straight line moves shown.

The Express VI API contains Express configuration dialogs with visualizations that help you quickly put together an application. Double-clicking on an Express VI opens up the properties dialog where you can configure the Express VI node, see the visualizations, and set parameter values.

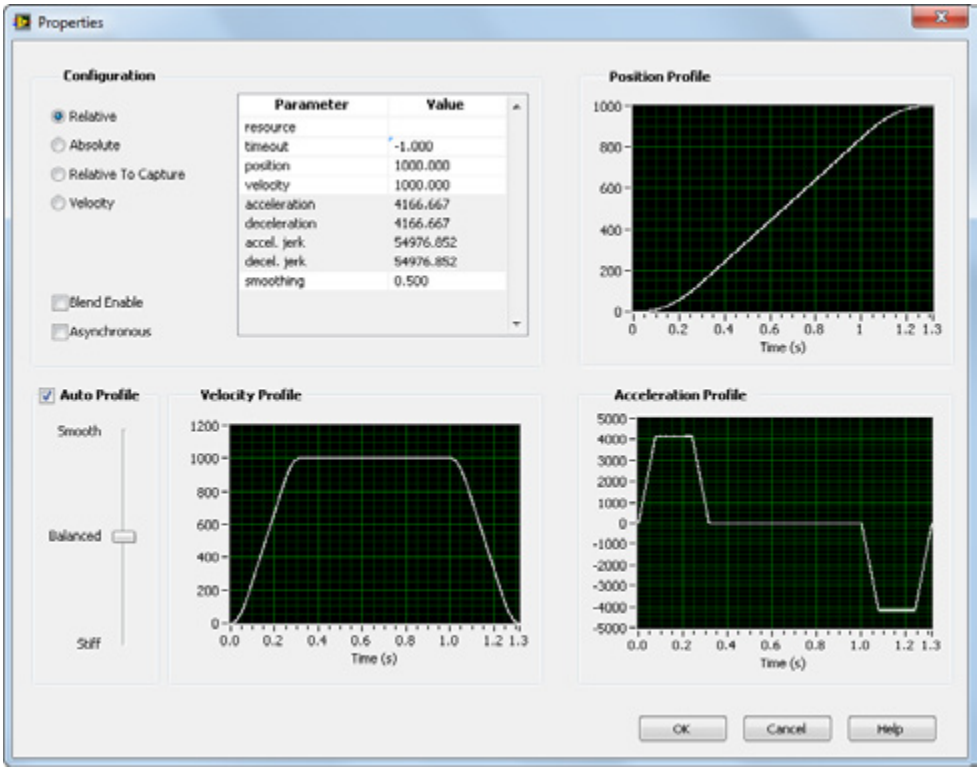


Figure 10.31. Use the Straight Line Move Express VI Properties Page to configure settings, change values, and see the resulting profiles.

Palette Object	Palette Symbol	Description
Line		Performs a straight-line move using an axis or coordinate resource. A straight-line move connects two points using one or more axes. The behavior of the move changes based on the Straight-Line Move Mode.
Arc		Performs a circular, spherical, or helical arc move. An arc move produces motion in a circular shape using a radius you specify. The type of arc to perform changes based on the Arc Move Mode.
Contour		Performs a contour move using an axis or coordinate resource. A contour move is a move expressed as a series of positions that the software uses to extrapolate a smooth curve. These positions are stored in a table. Each point in the move is interpreted as an absolute position using the starting point of the move as a temporary "zero" position. The type of contour move changes based on the Contour Mode.
Reference		Performs a reference move, such as locating a home or limit position, on an axis resource. Reference moves are used to initialize the motion system and establish a repeatable reference position. The behavior of the move changes based on the Reference Move Mode.
Capture		Records encoder position based on an external input, such as the state of a sensor. You can use the captured position to execute a move relative to a captured position, or simply record the encoder position when the capture event occurs.
Compare		Synchronizes the motor with external activities and specified encoder positions. When the specified position is reached, a user-configurable pulse is executed. The behavior of the position compare operation changes based on the Compare Mode.
Gearing		Configures the specified axis for gearing operations. Gearing synchronizes the movement of a slave axis to the movement of a master device, which can be an encoder or the trajectory of another axis. The movement of the slave axes may be at a higher or lower gear ratio than the master. For example, every turn of the master axis may cause a slave axis to turn twice. The type of gearing operation to perform changes based on the Gearing Mode.
Camming		Configures the specified axis for camming operations. These ratios are handled automatically by LabVIEW NI SoftMotion, allowing precise switching of the gear ratios. Camming is used in applications where the slave axis follows a nonlinear profile from a master device. The type of camming operation changes based on the Camming Mode.
Read		Reads status and data information from axes, coordinates, feedback, and other resources. Use the read methods to obtain information from different resources.
Write		Writes data information to axes, coordinates, or feedback resources. Use the write methods to write information to different resources.
Reset Position		Resets the position on the specified axis or coordinate.
Stop		Stops the current motion on an axis or coordinate. The behavior of the move changes based on the Stop Mode.
Power		Enables and disables axes and/or drives on the specified axes or coordinate resources.
Clear Faults		Clears LabVIEW NI SoftMotion faults.

Table 10.3. Motion Express VI Overview

You can configure NI SoftMotion Express VIs for either synchronous or asynchronous operation by right-clicking and selecting **Timing Model** from the menu.

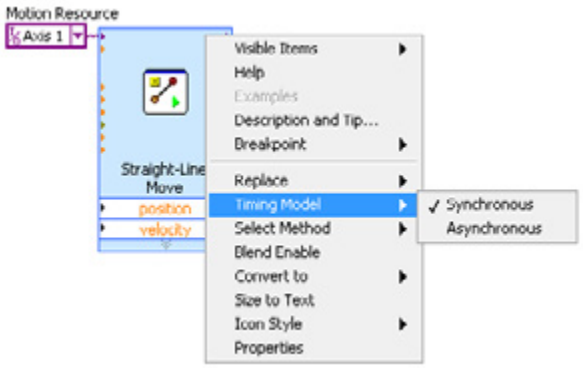


Figure 10.32. Select the timing model for each Express VI by right-clicking and selecting Timing Model.

When configured for synchronous operation, Express VIs wait until the node finishes executing before returning, and you can use standard LabVIEW programming methods (dataflow).

When configured for asynchronous operation, the Express VIs return immediately, and you must control execution by using the status parameters provided: execute, error out, done, aborted, busy, and active.

These two programming methods are very different, as illustrated from the two versions of a simple straight line move example shown in figures 10.33 and 10.34.

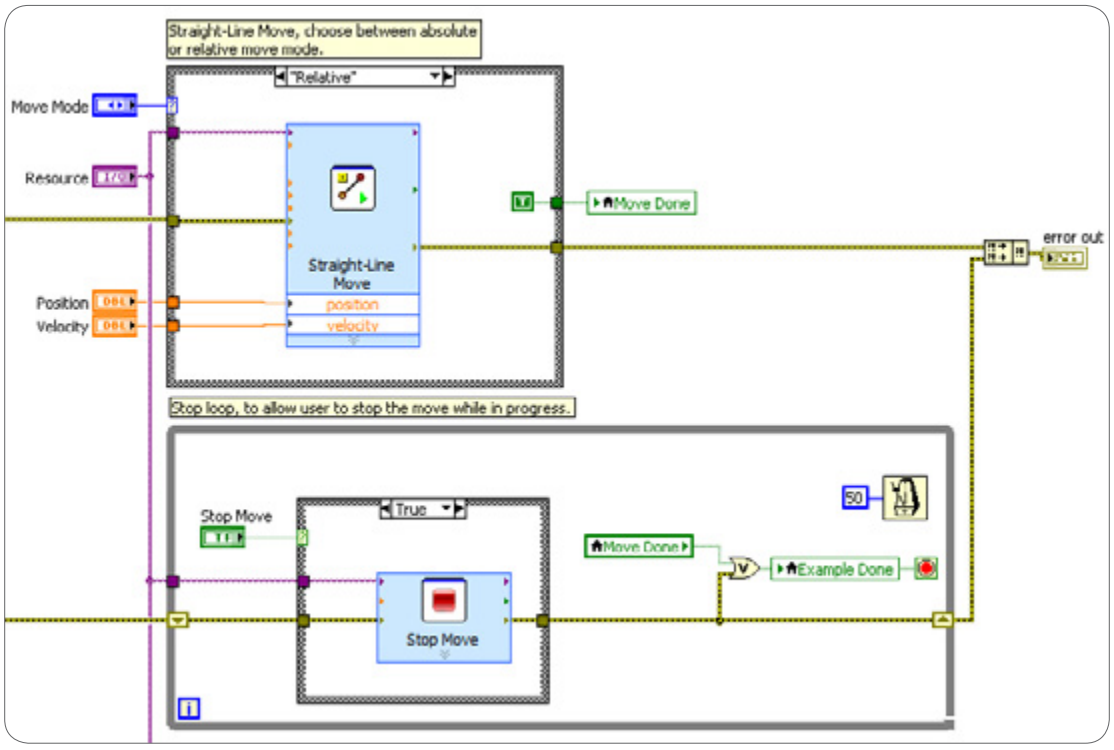


Figure 10.33. Straight Line Move Example Using Synchronous Express VIs

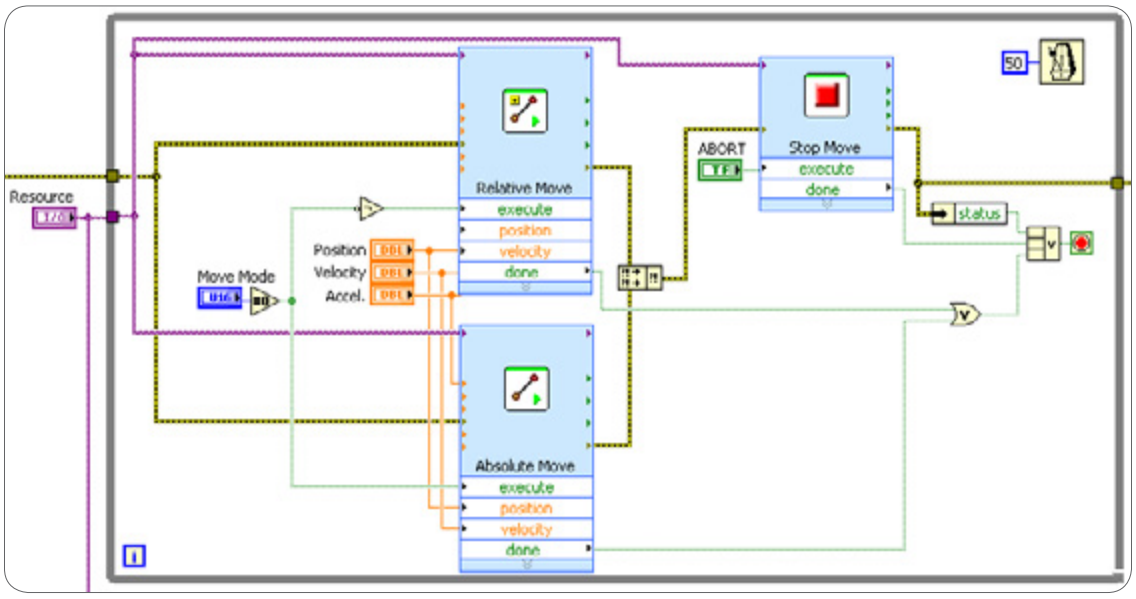


Figure 10.34. Straight Line Move Example Using Asynchronous Express VIs

You can convert Express VIs to subVIs that uses the Property/Invoke node API by right-clicking on the Express VI and selecting **Convert To»SubVI**.

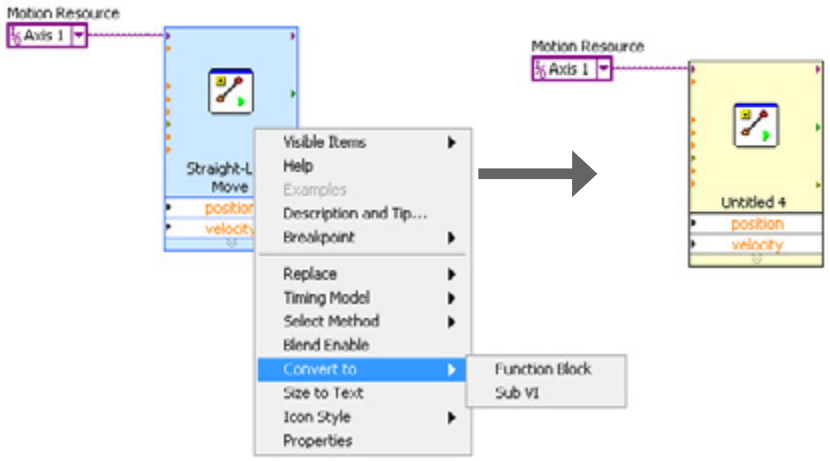


Figure 10.35. Right-click on an NI SoftMotion Express VI to convert it to a subVI.

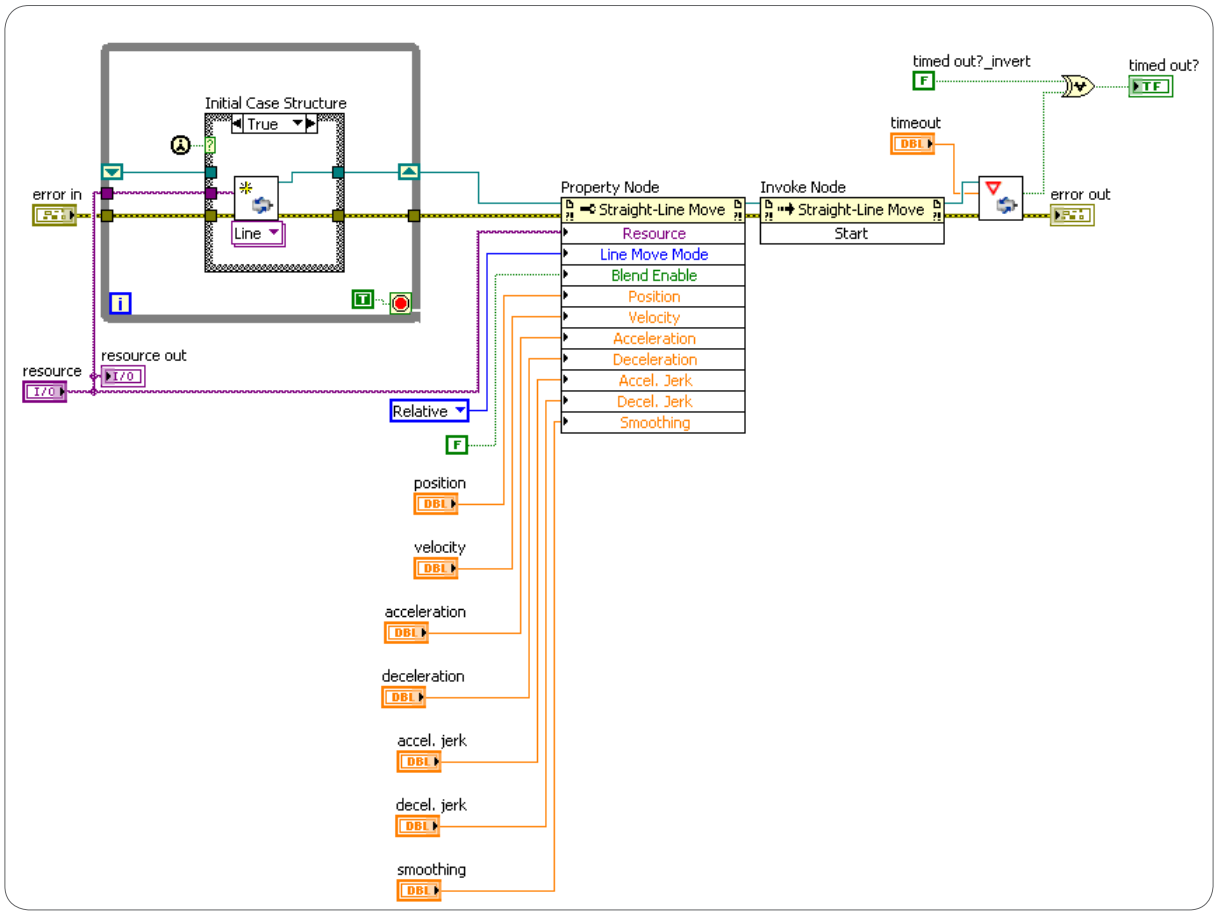


Figure 10.36. Resulting Block Diagram After Converting the Straight Line Move Express VI to a SubVI

See examples of using both the Express VI and Property/Invoke node APIs in the NI Example Finder.

The Property/Invoke node API provides functionality similar to the Express VI API but with a more granular level of control over how NI SoftMotion processes and executes commands. In addition to the axis resource, the Property/Invoke node API uses the concept of a motion resource, which creates a reference to specific NI SoftMotion interface objects.

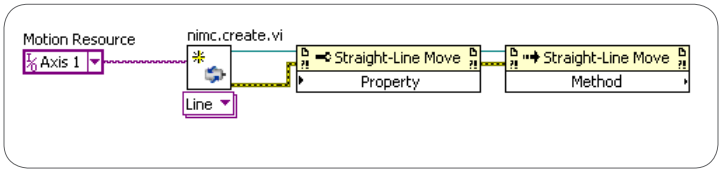


Figure 10.37. The NI SoftMotion Property/Invoke node API uses motion resources as well as axis resources.

With this property node, you can configure all of the same properties and settings available from the Express VI configuration dialog. But where the Express VI execution is determined by dataflow in the synchronous case and by the done terminal in the asynchronous case, the Property/Invoke node API execution is controlled through the Invoke node.

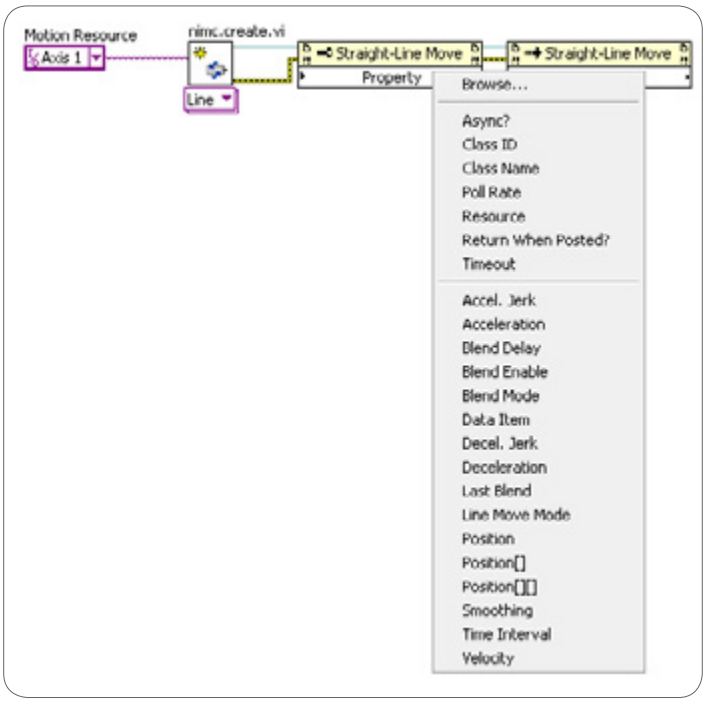


Figure 10.38. Available Properties of a Straight Line Move

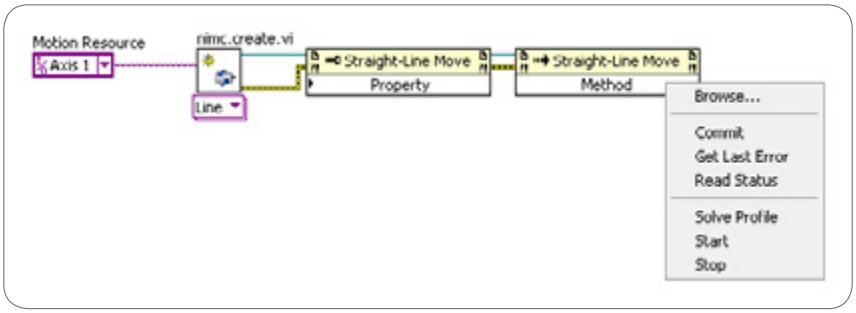


Figure 10.39. Available Methods of a Straight Line Move

You can also use the Property/Invoke node API to programmatically configure settings for all axes in the LabVIEW project.

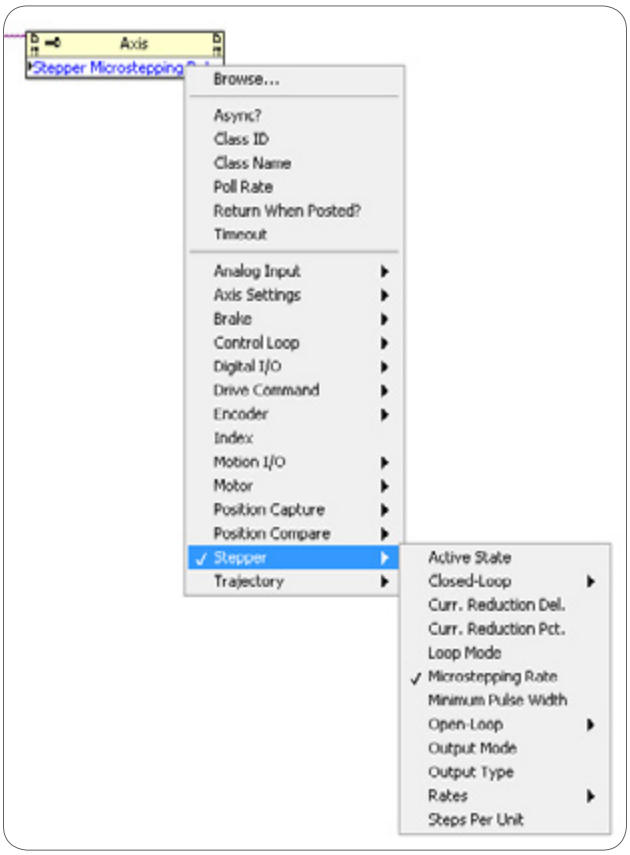


Figure 10.40. All the same settings that you can configure through the axis in the LabVIEW project can also be configured programmatically through the Property/Invoke node API.

Which API to use is largely determined by your application architecture. Use the Property/Invoke node API when you need tight synchronization with the rest of your application or when you use a state machine architecture. In all other cases, use the NI SoftMotion Express VI API, which is lean and efficient.

You can program VIs written using these motion APIs and deploy them to the Windows machine, or you can run them in LabVIEW Real-Time on a CompactRIO controller. If you deploy and run the VIs on a Windows system while using resources deployed to a CompactRIO controller, then use Web Service APIs to act on those resources remotely. Typically, it is best to deploy the code to LabVIEW Real-Time and then implement a communications protocol back to a high-level API to execute certain move sequences and interact with the application.

Real-Time-Based NI SoftMotion Components: The NI SoftMotion Engine

As mentioned, you can run the high-level motion code on the host PC in LabVIEW for Windows or in LabVIEW Real-Time on the CompactRIO controller . All the deterministic components of the motion system also run on the real-time system, including trajectory generation and supervisory control. The NI SoftMotion Manager runs at the scan rate of the CompactRIO controller and communicates between the high-level API code and the various motion hardware and I/O set up through the axis in the system.

FPGA-Based NI SoftMotion Components

The goal for NI SoftMotion low-level implementation on the FPGA is to keep the high-level API programming but add FPGA flexibility and performance. The following section provides an overview of the motor control IP included in NI SoftMotion.

When you open a VI targeted to the CompactRIO FPGA, you can see an NI SoftMotion palette that includes the following motor control IP:

- PID/PI
- PWM
- FOC
- I^2T
- Motion I/O
- Feedback
- Filters
- Stepper

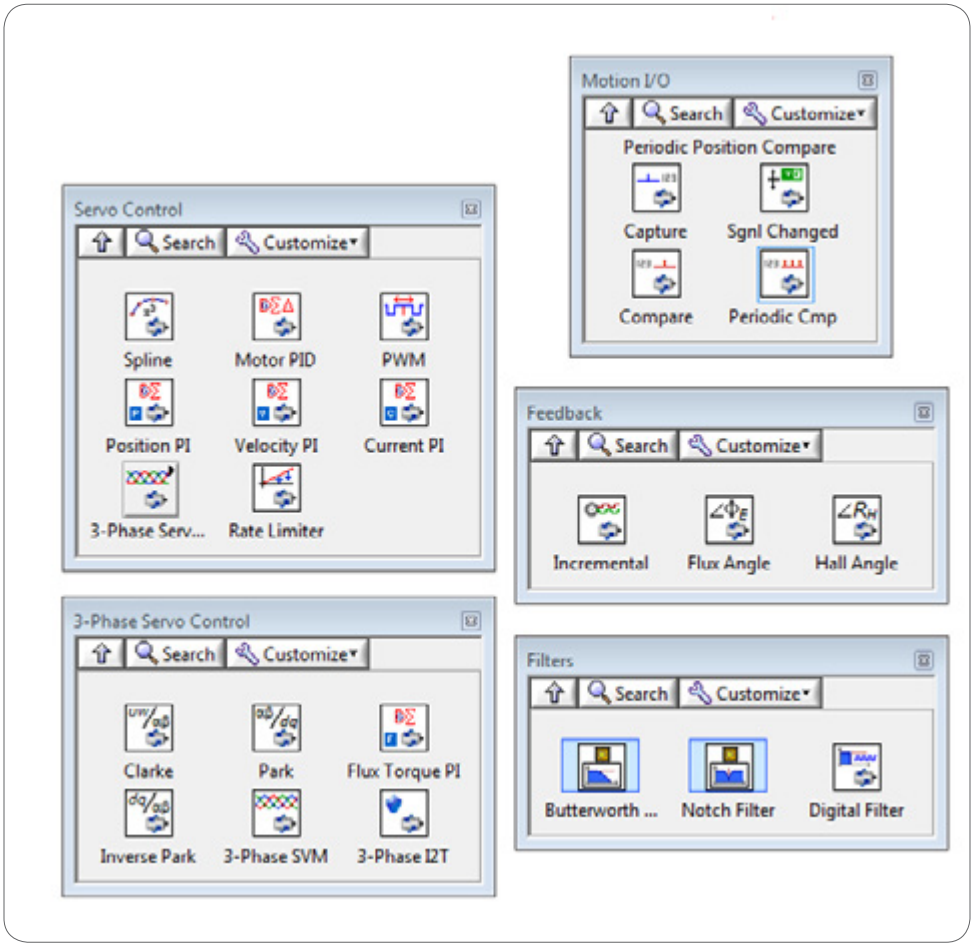


Figure 10.41. NI SoftMotion Motor Control IP Palettes in LabVIEW FPGA

Use this IP to build all the necessary components to control a motion axis. There are two basic elements to controlling a torque amplifier: control and feedback. The control loop provides a current setpoint and the feedback loop provides an encoder position.

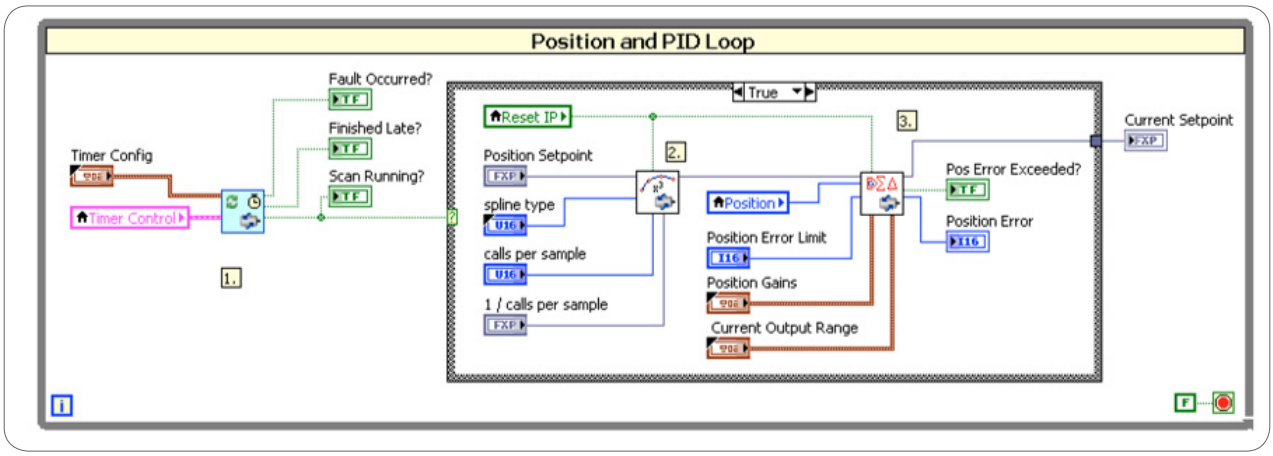


Figure 10.42. Spline Interpolation and PID Loop Executed in LabVIEW FPGA With NI SoftMotion IP

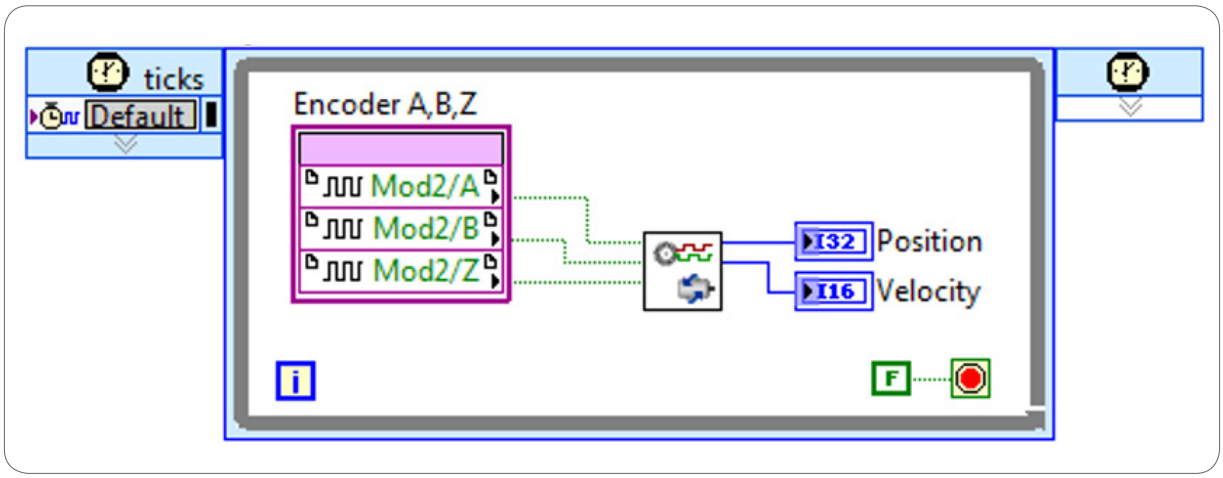


Figure 10.43. Incremental (Quadrature) Encoder Decoder

To control the NI 9505 brushed DC motor drive module, you also need current control and PWM control of H-bridges.

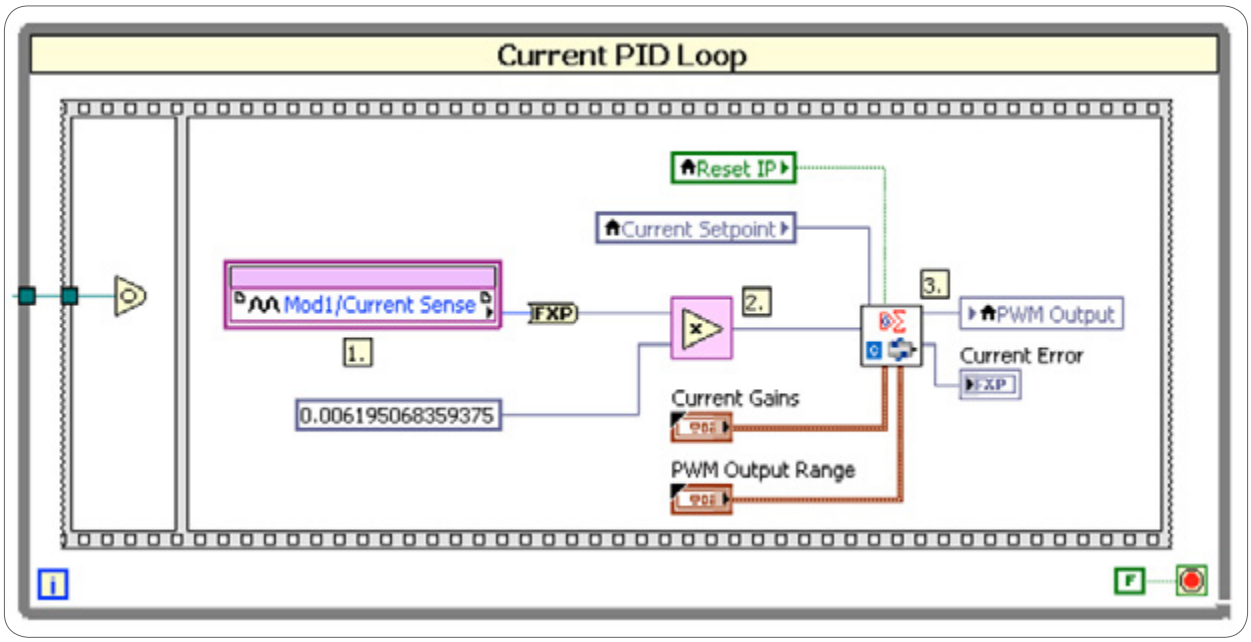


Figure 10.44. Current Control IP

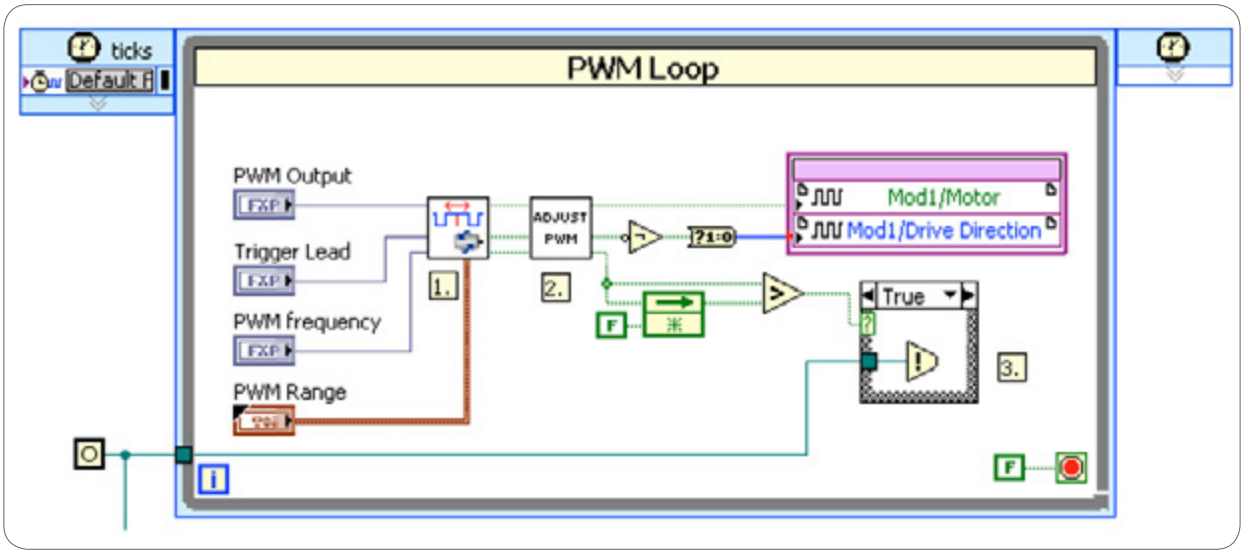


Figure 10.45. PWM Control of H-Bridges

To control a brushless servo drive, you need to use two new elements: field oriented control and Hall effect sensors. This additional complexity is necessary because a brushless motor is electrically commutated by the drive instead of mechanically commutated due to the inherent construction of the motor.

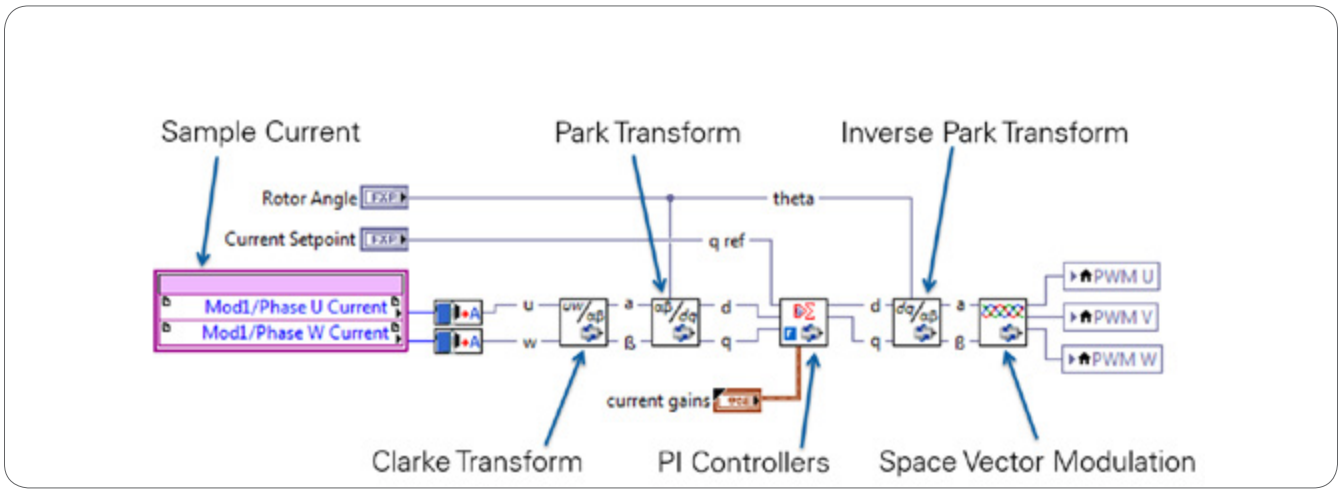


Figure 10.46. Field Oriented Control (FOC) Algorithm Implemented in LabVIEW FPGA

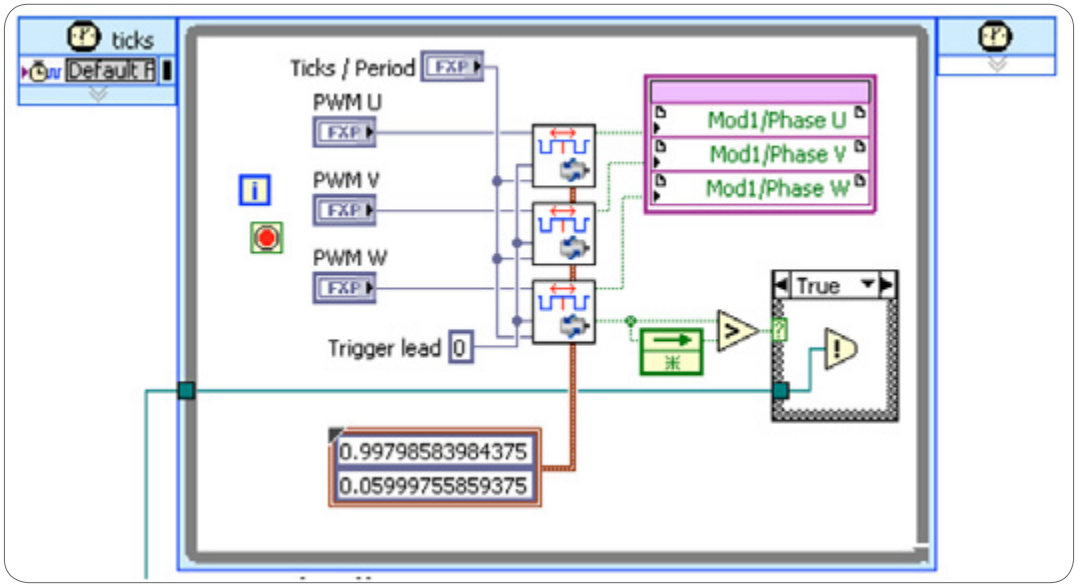


Figure 10.47. Three PWM Generators, One for Each Motor Phase

Use Hall effect sensor inputs to determine the angle of the motor shaft, and use the angle upon startup to eliminate a jump in rotor position.

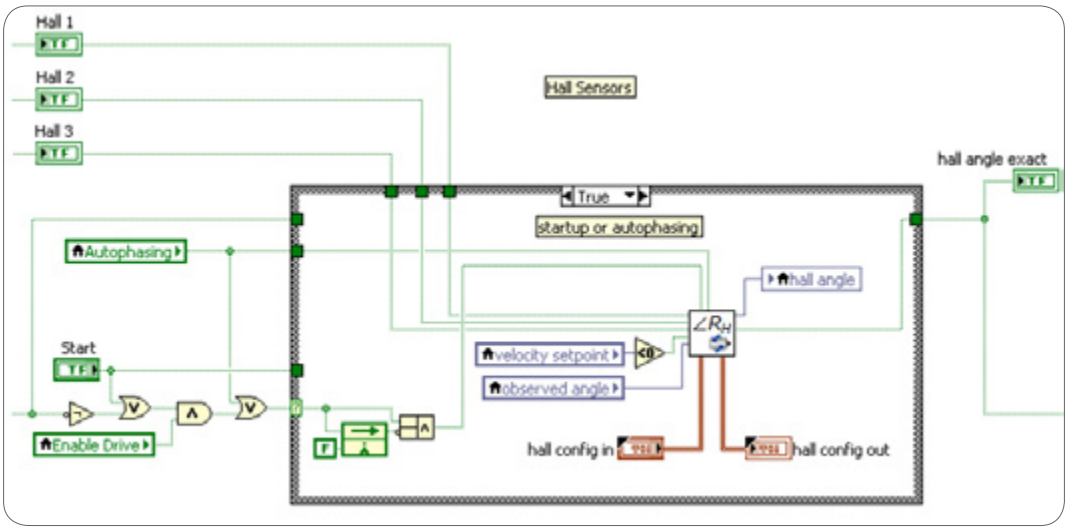


Figure 10.48. Hall Sensors for Initialization

Motion I/O is also implemented through the LabVIEW FPGA Module.

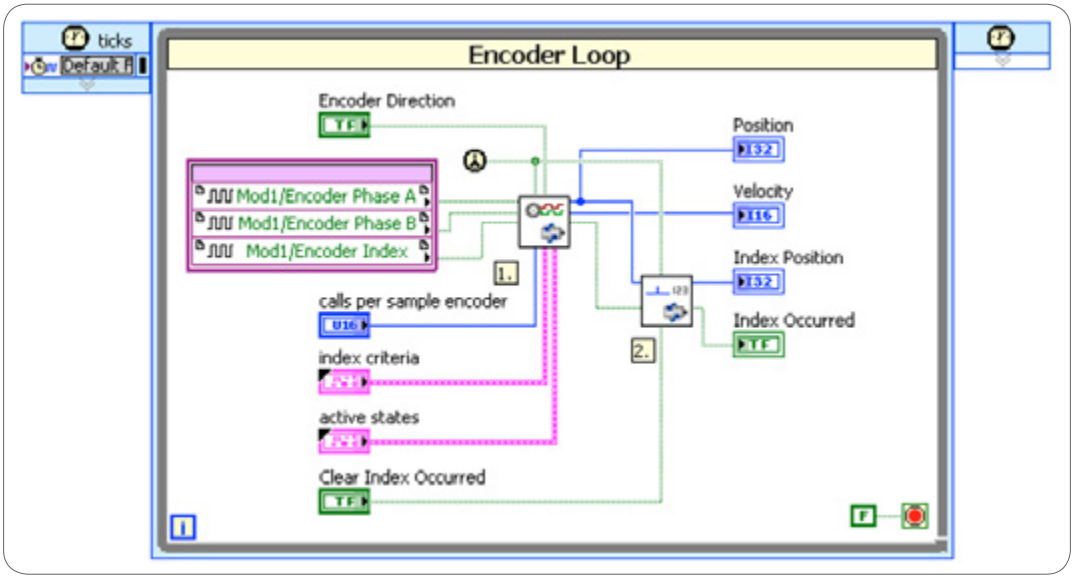


Figure 10.49. Encoder Loop With Index Read Functionality

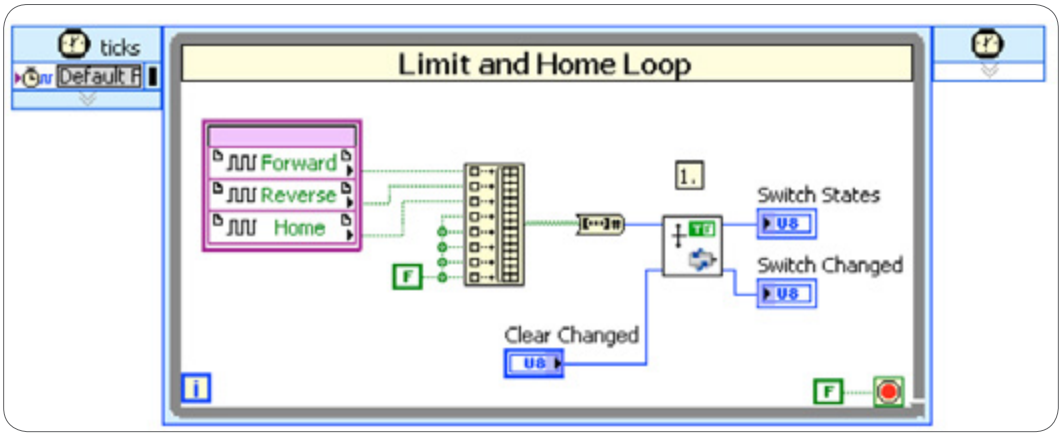


Figure 10.50. Limit and Home Loop

Consider an example application for which you need to trigger gripper action based on multiple conditions:

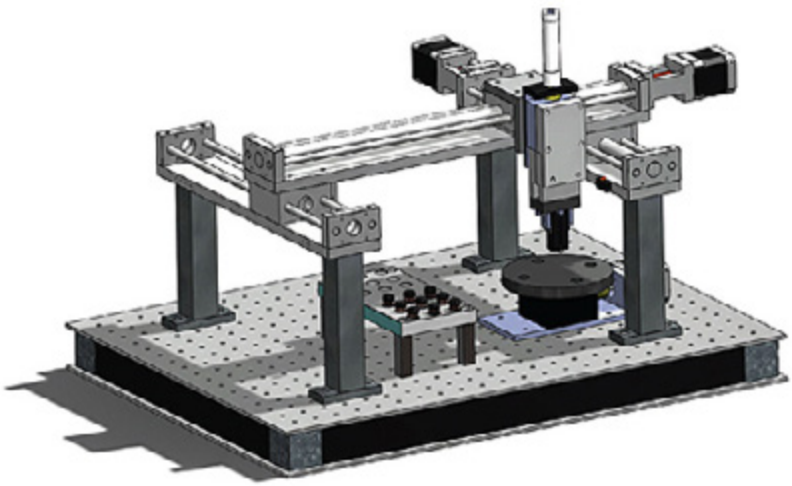


Figure 10.51. Example Gantry System

1. Position of the gantry (X, Y, and Z)
2. Rotation angle of the stage
3. Force exerted by the gripper

It's possible to get the position of the gantry (X, Y, Z, and Θ) at 25 ns intervals from the single-cycle Timed Loop housing the quadrature decoder code.

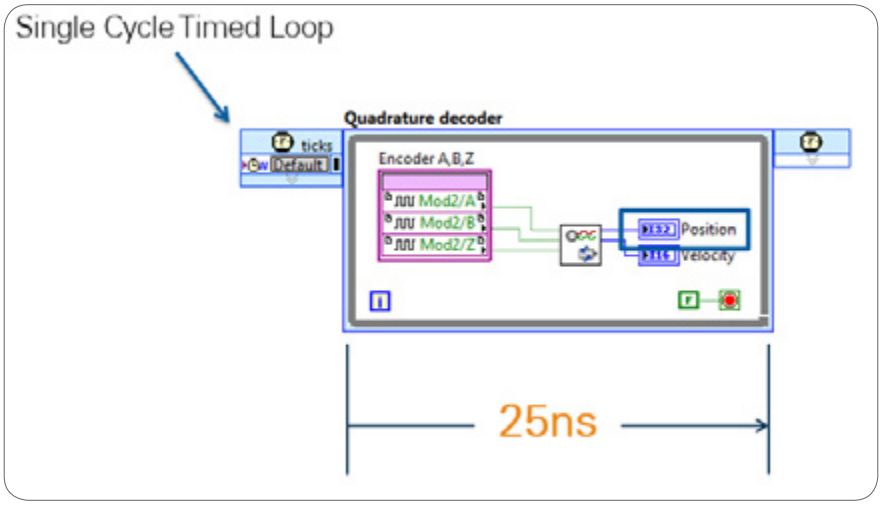


Figure 10.52. You can read position from the quadrature decoder loop.

You can determine the force of the gripper from the sampled quad current in the FOC algorithm every 1.4 µs.

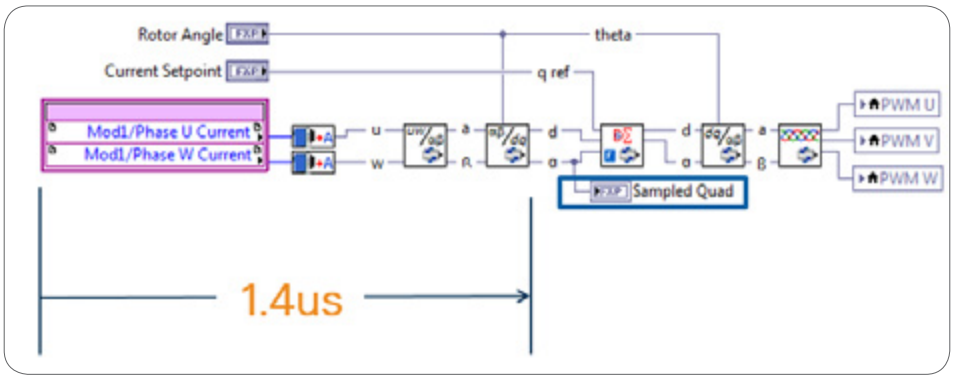


Figure 10.53. FOC Algorithm

You can then define this information in a user-defined trigger running in a single-cycle Timed Loop on the FPGA and executing as fast as every 25 ns.

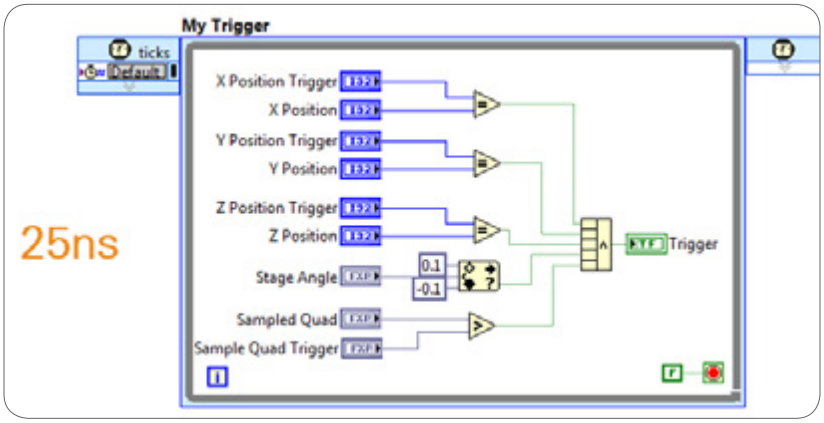


Figure 10.54. User-Defined Trigger

NI SoftMotion has IP for triggering, including position compare, position capture, and periodic position compare functionality. These functions are full featured with enable/disable, pulse width, periodic trigger, digital filter, and trigger window capabilities. They are single-cycle Timed Loop compatible and open so that you can modify them if needed.

Hardware Interfaces to NI SoftMotion

You can choose from a variety of hardware interfaces to NI SoftMotion. The LabVIEW NI SoftMotion Module is written so that you can write custom hardware interfaces for specific hardware such as third-party drives or C Series modules. The most common hardware interfaces are

- NI SoftMotion and EtherCAT AKD drives
- NI SoftMotion and C Series drive interface modules (NI 951x)
- NI SoftMotion and C Series drive modules (NI 950x)

NI SoftMotion and EtherCAT AKD Drives

NI SoftMotion contains an interface to the EtherCAT AKD servo drive. Use this interface when drives are connected to an NI real-time controller over EtherCAT.

Figure 10.55. Any CompactRIO controller with two Ethernet ports can act as an EtherCAT master to control multiple daisy-chained AKD drives from the second Ethernet port.

All NI real-time targets with two Ethernet ports and supported by the NI-Industrial Communications for EtherCAT driver are configurable as NI EtherCAT masters. This includes PXI controllers, NI industrial controllers, the NI 9792 programmable wireless sensor network gateway, and CompactRIO controllers.

You can add coordinated high-performance multiaxis servo control to any system based on these NI real-time controllers. Which controller to select depends on the number of axes in the system (amount of data being processed each scan cycle on the EtherCAT bus) and on your desired control-loop rate. The higher the axis count, the more data that must be processed within a scan cycle. The faster the control-loop rate, the less time the controller has to complete that processing. These amount of data on the bus and control-loop rate determine the practical limit on the number of slave devices allowed in a particular system.

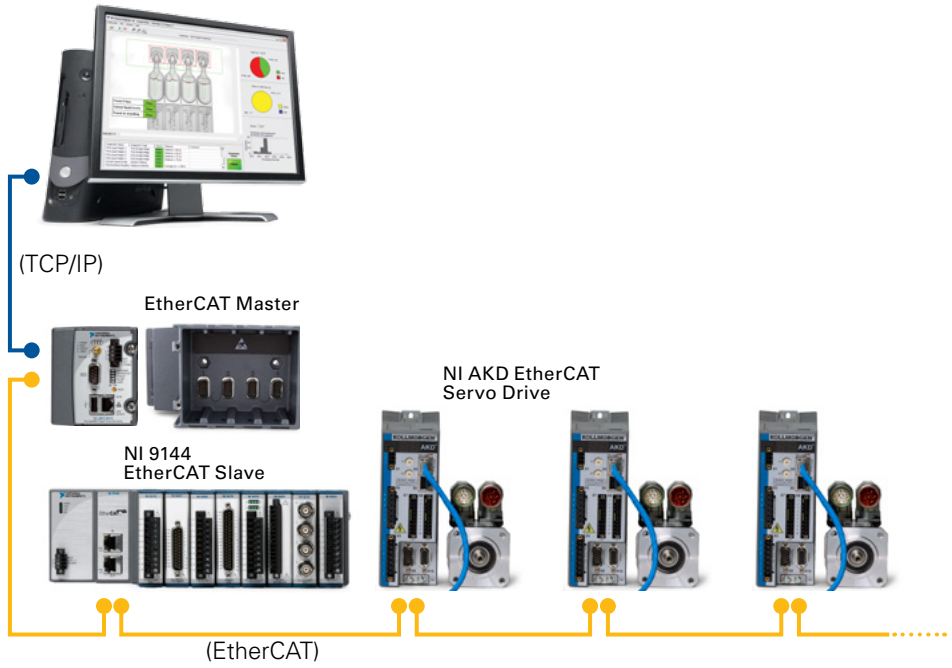


Figure 10.56. CompactRIO acts as the EtherCAT master. You can add multiple NI 9144 and EtherCAT AKD slaves to extend this system up to the practical limit imposed by the processing power of the EtherCAT master.

When added to the system, EtherCAT drives show up under the EtherCAT master in the system. Then you can bind them to an axis.

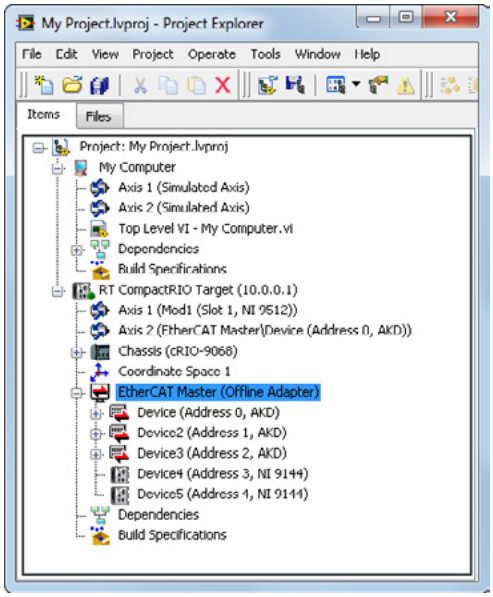


Figure 10.57. EtherCAT devices show up under the EtherCAT master adapter (second Ethernet port of the CompactRIO system). You can bind EtherCAT AKD drives to an NI SoftMotion axis as shown.

Once bound to an axis, NI SoftMotion takes care of all drive communication over EtherCAT, and you can use that axis with the high-level motion APIs. This is because NI SoftMotion includes a specific drive interface and LabVIEW project integration for the AKD drive. This drive interface abstracts all the EtherCAT programming and variables; just use the axis bound to the AKD with the high-level NI SoftMotion API.

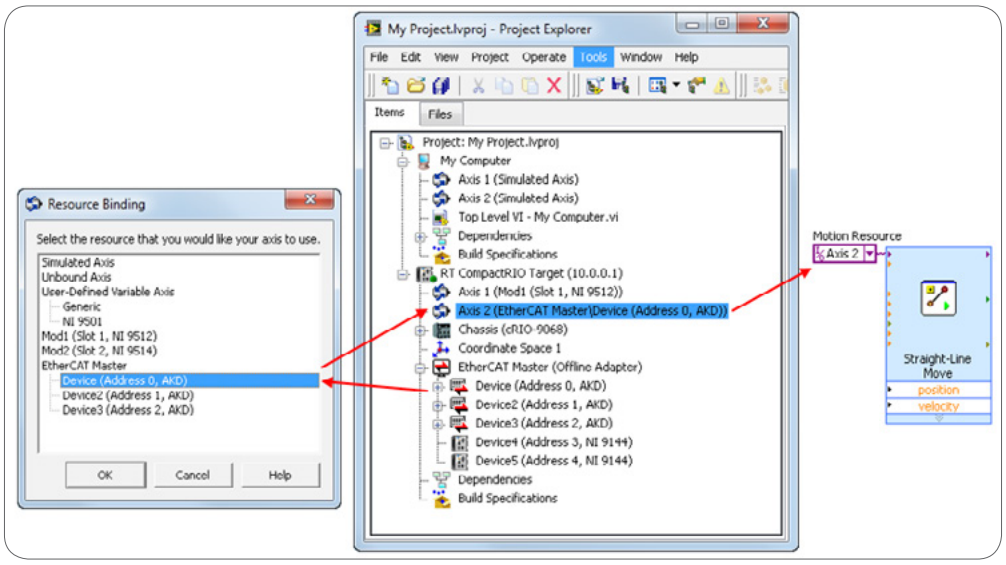


Figure 10.58. AKD EtherCAT Drives can be bound to an axis, and then directly used with the NI SoftMotion API

NI SoftMotion runs on the NI EtherCAT master (CompactRIO), generates trajectory information (setpoints), and sends them to the EtherCAT AKD slave drive, which runs the position, velocity, and torque control loops. You can configure the AKD drive for position mode, velocity mode, or torque mode. In the standard use case, the AKD is configured for position mode when operating with NI SoftMotion over EtherCAT.

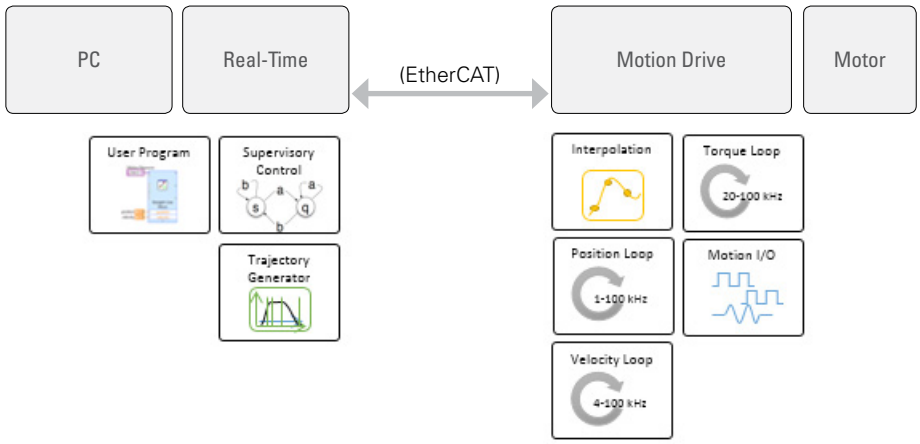


Figure 10.59. In the most common configuration, NI SoftMotion sends trajectory information (position commands) to the AKD, which then closes the control loops.

Benefits of building a motion system with CompactRIO and EtherCAT AKD drives:

- Simplest cabling and setup
- AKD autotuning feature
- No additional C Series modules required
- Fault code can be read from the AKD drive across the network

Disadvantages:

- No position compare functionality

- Encoder feedback to EtherCAT master is slower—same as the scan rate
- No control loop customization
- Position/velocity loop rates limited to 62.5 µs

See the video NI SoftMotion and AKD EtherCAT Brushless Servo Drives for a demonstration of the AKD setup experience.

Read the case study Master Machinery Builds Semiconductor Pick and Place Machine With CompactRIO and LabVIEW for an industry example of this configuration.

NI SoftMotion and Drive Interface Modules (NI 951x C Series)

These motion modules provide servo or stepper drive interface signals for a single axis to communicate from NI SoftMotion running on a CompactRIO chassis to an external drive. In addition, they offer a full set of motion I/O including inputs for a home switch and limit switches, incremental encoder inputs for position feedback, and digital input and digital output lines. NI 951x drive interfaces include a processor to run the spline interpolation engine and the patented NI step generation algorithm or servo control loop.

You can choose from three C Series drive interfaces:

Figure 10.60. NI C Series Drive Interfaces

- NI 9512 single-axis stepper or position command (P-command) drive interface with incremental encoder feedback
- NI 9514 single-axis servo drive interface with incremental encoder feedback
- NI 9516 single-axis servo drive interface with dual incremental encoder feedback

NI 951x modules were designed to simplify wiring by providing the flexibility to hook up all the I/O for motion control to a single module. To further ease connectivity, the digital I/O is software configurable to connect to either sinking or sourcing devices. The modules offer the following:

- Analog or digital command signals to the drive
- Drive enable signal that is software configurable as sinking or sourcing (24 V)
- Digital input signals for home, limits, and general digital I/O; all software configurable as sinking or sourcing (24 V)

- Encoder inputs and 5 V supply; configurable for single ended or differential
- Digital input and digital output for high-speed position capture/compare functions (5 V)
- LEDs to provide quick debugging for encoder states, limit status, and axis faults

To further simplify wiring, NI offers several options for connecting NI 951x drive interface modules to external stepper drives or servo amplifiers including the following:

- NI 951x Cable and Terminal Block Bundle—Connects an NI 951x module with 37-pin spring or screw terminal blocks
- D-SUB and MDR solder cup connectors—Simplify custom cable creation
- D-SUB to pigtails cable and MDR to pigtails cable—Simplify custom cable creation
- Direct connect cables to the analog AKD servo drive

Figure 10.61. Directly connect cables between an NI 951x module and the analog AKD servo drive.

NI C Series drive interface modules are supported in NI Scan Mode and in FPGA mode. In NI Scan Mode, you do not need LabVIEW FPGA to configure and run the module. This configuration is the easiest to set up and use, and it provides all the necessary functionality for standard use cases. If you need to add custom triggering and timing with other C Series I/O or modify the control, then use the modules in FPGA mode.

By default in LabVIEW 2013 and earlier, you can use only NI 951x modules in the first four slots in a CompactRIO chassis because these are the only slots compiled with the high-speed interface (HSI). See the KnowledgeBase article How Do I Use the NI 951x Motion Control Modules in Slots 5-8 (Non High Speed Interface Slots) of My CompactRIO Chassis? for instructions on compiling the entire chassis with HSI support. Some older CompactRIO chassis such as the NI cRIO-9072, NI cRIO-9073, and NI cRIO-9074 have FPGAs that are not big enough to support HSI for all slots.

The NI 9512 drive interface module can control stepper drives that accept step and direction signals as well as control brushless servo drives that accept P-commands. Thus, you can use this module to interface to brushless servo axes as well as take advantage of the tuning features of those servo drives.

When a servo drive is connected to an NI 9514 or an NI 9516, it is configured in torque mode, which means position and velocity loop moved from the drive and executed in the NI 951x module. Torque loop always runs in the drive.

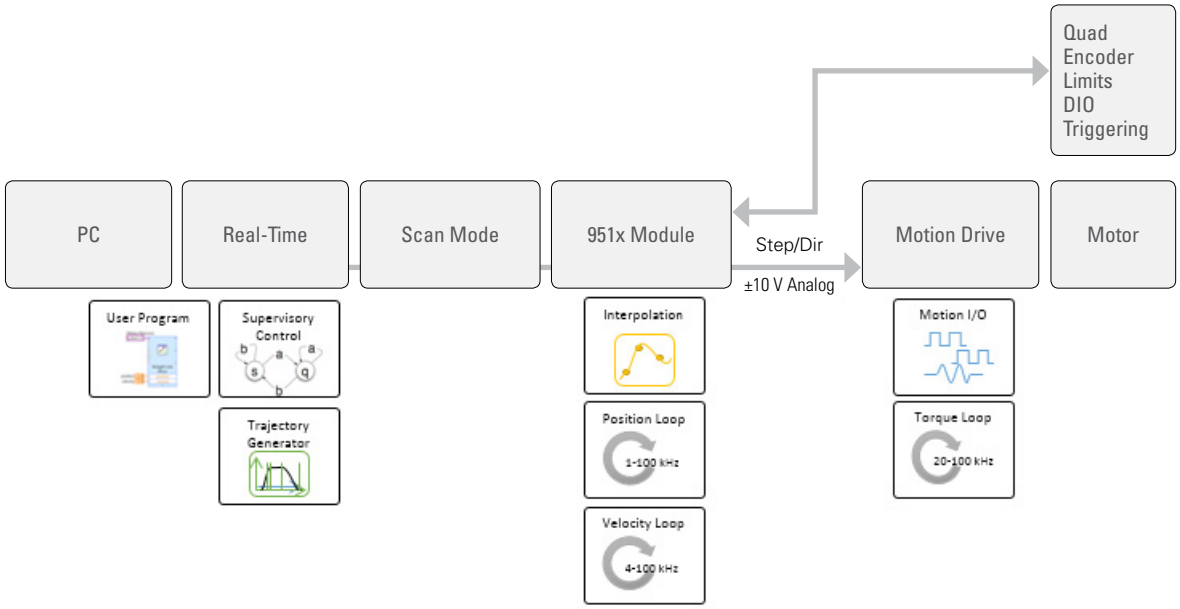


Figure 10.62. Using NI 951x modules in NI Scan Mode, spline interpolation, position, and velocity loops are executed on the module. Using NI 951x modules in FPGA mode, these loops are executed in LabVIEW FPGA in the backplane FPGA.

NI SoftMotion and Drive Modules (NI 950x)

NI 950x drive modules are motion drives implemented in the C Series form factor. They connect directly to motors to provide an embedded form factor.

Figure 10.63. All that you need to run this system is an external power supply. An NI 9502 and NI 9411 are connected to the AKM brushless servo motor (left). An NI 9503 is connected to the stepper motor (center), and an NI 9505 is connected to the brushed DC motor (right). You can control all three motor types and organize them in different combinations from the same CompactRIO chassis.

All drive firmware for these modules is implemented on the backplane FPGA using NI SoftMotion motor control IP blocks. These modules open up the ability to customize pretty much anything in an application because you implement all the control, including the torque loop, in LabVIEW FPGA. This means these drives require substantial FPGA resources and LabVIEW Real-Time/LabVIEW FPGA knowledge. Because all the control is open, the NI 950x drive modules provide a configurable off-the-shelf alternative to custom drive development. They save on cabling costs, so you can implement a configurable motion drive with a user-defined axis mix and I/O in a compact form factor.

Figure 10.64. NI 9503 (Stepper), NI 9502 (Brushless Servo), and NI 9505 (Brushed DC) Drive Modules

Because these drives are implemented in the CompactRIO C Series form factor, power dissipation is limited to 1.5 W. This spec is important because it says you have no limits on the other modules you can use in the CompactRIO chassis next to these drive modules. For instance, there are no thermal issues putting a high-accuracy 24-bit analog input module next to an NI 950x motion drive. However, this limits the power output of these drives, so they can control only small to mid-size motors. When selecting a platform, make sure that the drive modules can provide enough power with your selected motor to give you the torque you need in your application.

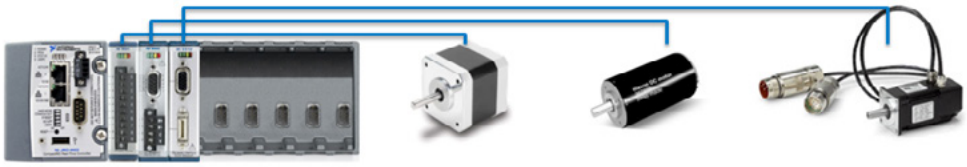


Figure 10.65. Connect motors directly to a CompactRIO system with NI 950x Drive Modules

- NI 9503 stepper, 3 A rms, 4.24 A peak current per phase
- NI 9502 brushless servo, 4 A continuous/8 A peak current
- NI 9505 brushed DC, 5 A continuous/12 A peak current

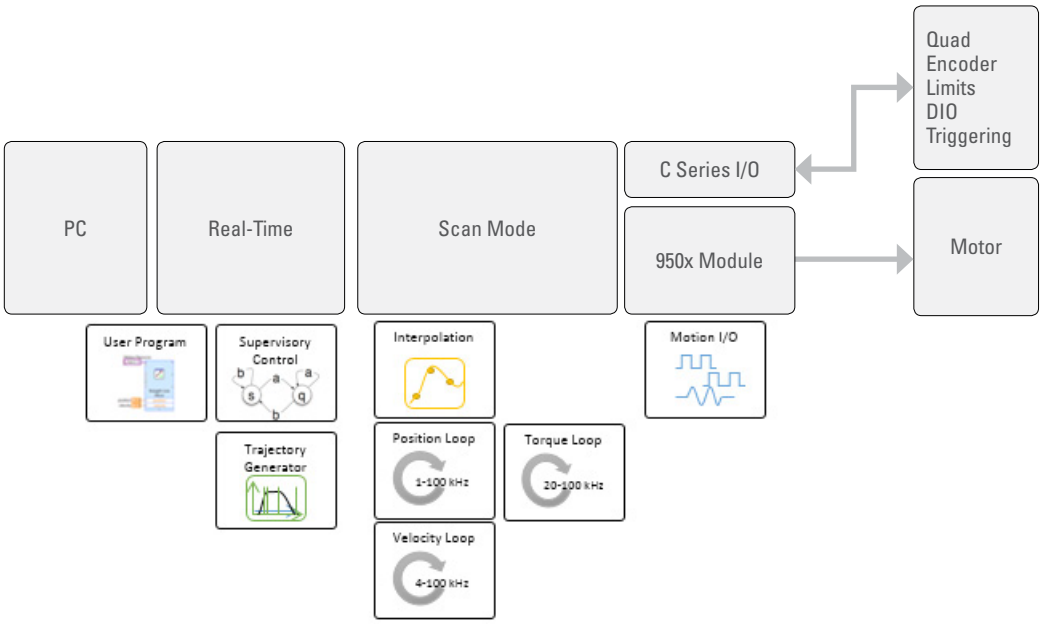


Figure 10.66. In the NI 950x drive module architecture, all control is implemented in NI SoftMotion on LabVIEW FPGA.

When using NI 950x drive modules, components that are typically executed on external drive firmware are pulled in from the drive and implemented in LabVIEW FPGA on the CompactRIO backplane instead. The NI SoftMotion shipping examples provide working starting points for NI 950x modules, and can be used with little modification to the motion IP and control loops for many motion applications.

Example Configurations

The following are some common/interesting ways to configure a CompactRIO motion system. They illustrate the flexibility and possibilities in assembling a configurable off-the-shelf system to meet a variety of application needs.

Ethernet RIO With C Series Motion Modules

Ethernet RIO expansion chassis are the only NI reconfigurable I/O (RIO) targets that do not require any additional software other than the LabVIEW development system and necessary driver. NI Ethernet RIO technology offers a simplified software experience directly out of the box. The LabVIEW Real-Time Module is not required to access the NI C Series I/O from a Windows-based system. The LabVIEW FPGA Module is not required to access the C Series I/O using NI Scan Mode.

Because NI 951x drive interface modules are supported in NI Scan Mode, you can add them to an Ethernet RIO chassis to create a distributable motion controller with stepper or servo motion combined with other C Series modules in the remaining slots. The system requires no LabVIEW Real-Time or LabVIEW FPGA programming; all you need is LabVIEW and the LabVIEW NI SoftMotion Module installed on your development PC.

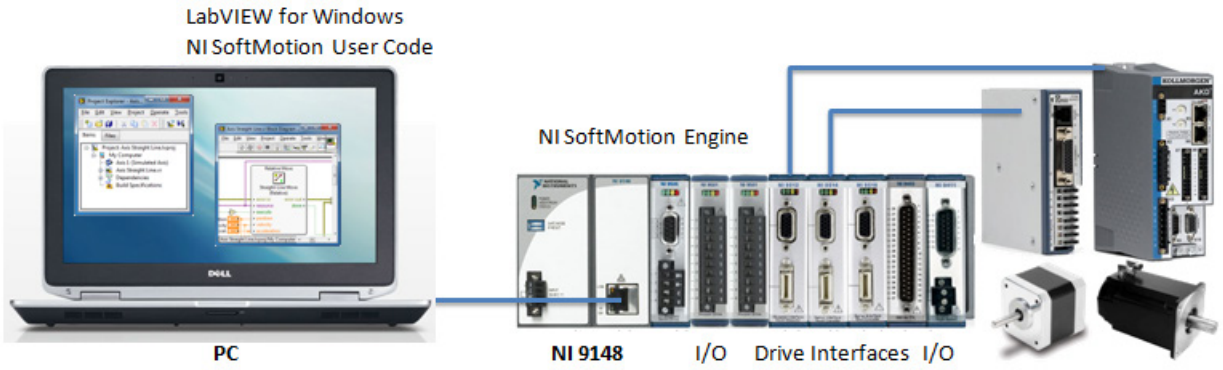


Figure 10.67. When using Ethernet RIO chassis and NI 951x drive interface modules, no software is needed beyond LabVIEW with the LabVIEW NI SoftMotion Module installed on the development machine.

When you run a high-level VI containing an application built with the NI SoftMotion API on the development PC, NI SoftMotion uses web services to deploy the necessary code to the Ethernet RIO chassis, where it runs all the necessary supervisory and trajectory tasks locally to maintain the determinism and performance needed for motion control applications.

Figure 10.68. This 4-axis stepper motion controller was built using the NI 9076 Ethernet RIO chassis and four NI 9512 drive interface modules.

Axes within the chassis are coordinated. Axes across multiple Ethernet RIO chassis on the same Ethernet network are not coordinated.

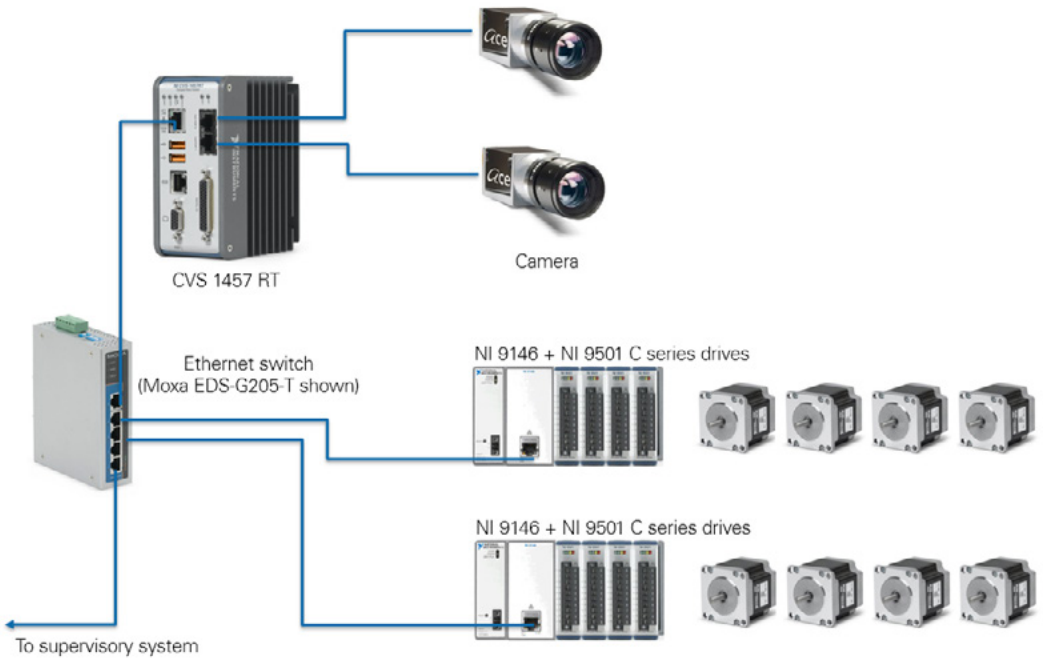


Figure 10.69. You can use the Ethernet RIO chassis to add a modular motion control unit to any application over an Ethernet network. This example shows an integrated vision and motion system for an inspection and positioning application.

In Figure 10.69, the NI 950x drive modules are used in the Ethernet chassis to create an embedded motion drive. This configuration requires the correct LabVIEW FPGA code to be running on the backplane to implement the drive firmware for these modules. However, once you develop the bitfile, you do not need LabVIEW FPGA to run or interact with the Ethernet RIO motion controller except to change the functionality of the drive firmware implemented in LabVIEW FPGA.

EtherCAT-Based System

The EtherCAT-based system can include EtherCAT AKD drives and any motion C Series module in the NI 9144 EtherCAT slave chassis. Any CompactRIO controller with two Ethernet ports can act as the EtherCAT master. Because of the determinism of EtherCAT, you can synchronize motion axes across multiple chassis at the scan rate of the EtherCAT master. EtherCAT provides for high-axis-count and mixed-axis systems. The axis count is limited only by the processing power of the EtherCAT master and the desired scan rate. It's fairly common to have more than a dozen axes in an EtherCAT system with an appropriate CompactRIO controller (NI cRIO-9024/9025/9068/9081/9082) as the EtherCAT master.

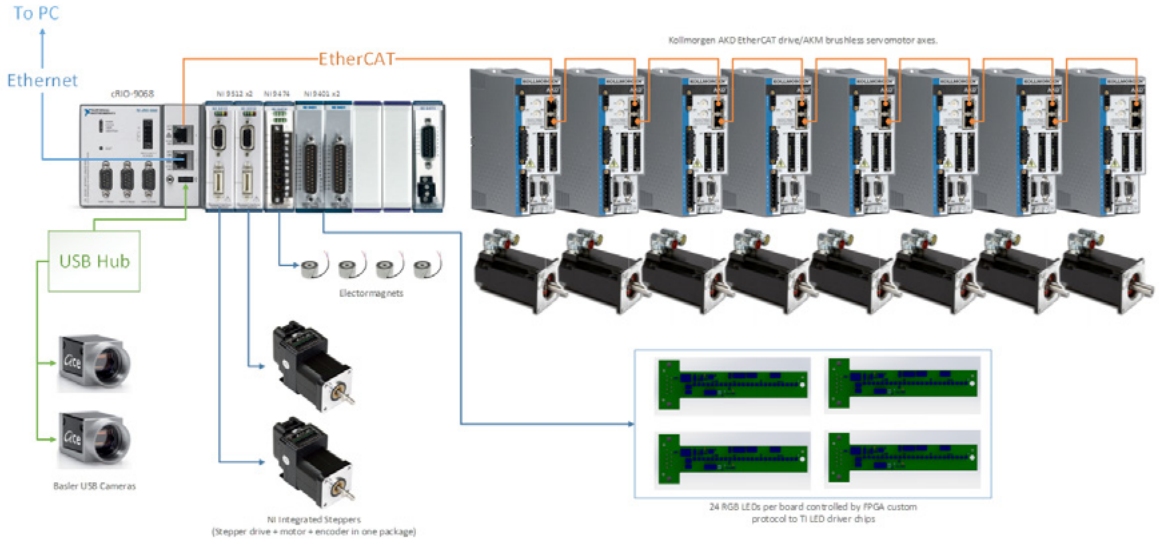


Figure 10.70. System Setup Used on the NI 3D Gears Demonstration System (The NI cRIO-9068 is controlling eight AKD servo drives over EtherCAT, two USB cameras, and a variety of I/O from the C Series slots including two stepper drives, four electromagnets, and a custom digital protocol controlling 96 RGB LEDs with a custom FPGA-based digital protocol.

In the Figure 10.70 setup, if there were insufficient slots in the CompactRIO chassis and more motion axes or I/O channels were required, you can add an NI 9144 chassis with the right mix of NI 951x drive interfaces, NI 950x drives, and C Series I/O modules.

Mixed-Axis Systems

Figure 10.70 features mixed-axis systems. With NI SoftMotion, you can synchronize axes of different types as long as they all use the same motion controller. For example, using an NI 9024 EtherCAT master, you could perform synchronized moves using NI 9512 stepper axes, NI 9514 servo axes, EtherCAT AKD axes, NI 9502 servo axes, or any other axis type. Axes in NI SoftMotion, though often bound to a particular hardware type, are not associated with hardware when using the high-level NI SoftMotion API. A straight line move is the same on an AKD axis as it is on an NI 9512 axis. It is only when the straight line move command gets to the communication interface that it is transformed into the signals that are expected by the hardware.

NI Drives and Motors

NI partners with Kollmorgen to provide a full suite of brushless servo drives and motors, gearheads, and linear actuators. Interfacing the AKD drive to CompactRIO through NI SoftMotion is discussed in an earlier section. All the motors shown in Figure 10.71 are compatible with the AKD drive and can be used in a CompactRIO system.

Figure 10.71. Kollmorgen Servo Product Offering Available Through NI

NI offers a full suite of stepper products including integrated motors, DC and AC stepper drives, and NEMA 8–34 frame-sized stepper motors with integrated encoder options.

Figure 10.72. NI Stepper Offering

These industry-leading drives and motors are fully compatible with NI controllers and drive interfaces.

Determining System Requirements and Components

- Controllers (number of axes, processing power, I/O count and synchronization, communication strategy)
- Motor and drive (comparison of motor types, power ranges, typical application uses)
- Feedback devices (I/O synchronization)

Start by selecting the right mechanical components and motors for your system. One of the most common applications involves moving an object from one position to another. A typical way to change the rotary motion of a rotary motor to a useful linear motion is by connecting the motor to a stage and using a leadscrew mechanism to move the payload. Stages are mechanical devices that provide linear or rotary motion useful in positioning and moving objects. They come in a variety of types and sizes, so you can use them in many different applications. To find the correct stage for your application, you need to be familiar with some of the common terminology used when describing stages. Some of the key items to consider when selecting a stage include the following:

- **Transmission gear ratio**—Determines the linear travel distance of the stage per rotary revolution of the motor.
- **Accuracy**—How closely the length of a commanded move compares to a standard length.
- **Resolution**—The smallest length of travel that a system is capable of implementing can be as small as a few nanometers.

- **Travel distance**—The maximum length the system is capable of moving in one direction.
- **Repeatability**—The repeated motion to a commanded position under the same conditions. Often this is specified in unidirectional repeatability, which is the ability to return to the same point from one direction, and bidirectional repeatability, which specifies the ability to return from either direction.
- **Maximum load**—The maximum weight the stage is physically designed to carry, given accuracy and repeatability.

Stage Selection

You can choose from a variety of stages to meet your application needs. You can narrow down these stages to two main types—linear and rotary. Linear stages move in a straight line and are often stacked on each other to provide travel in multiple directions. A three-axis system with an x, y, and z component is a common setup used to position an object anywhere in a 3D space. A rotary stage is a stage that rotates about an axis (usually at the center). Linear stages are used to position an object in space, but rotary stages are used to orient objects in space and adjust the roll, pitch, and yaw of an object. Many applications, such as high-precision alignment, require both position and orientation to perform accurate alignment. The resolution for a rotary stage is often measured in degrees or arc minutes (1 degree equals 60 arc minutes). Special types of stages include the goniometer, which looks like a linear stage that moves in an arc rather than a straight line, or a hexapod, which is a parallel mechanism that gives you movement in six axes to control—x, y, z, roll, pitch, and yaw. With a hexapod, you can define a virtual point in space about which the stage can rotate. While that is a benefit, the disadvantage is that hexapods are parallel and the kinematics involved are much more complex than those for simple stacked stages.

Backlash

Another stage attribute to consider when choosing a stage for your precision motion system is backlash. Backlash is the lag created when one gear in a system changes direction and moves a small distance before making contact with a companion gear. It can lead to significant inaccuracies especially in systems using many gears in the drive train. When moving on a nanometer scale, even a small amount of backlash can introduce major inaccuracy in the system. Effective mechanical design minimizes backlash but may not be able to eliminate it completely. You can compensate for this problem by implementing dual-loop feedback in software. The NI 9516 C Series servo drive interface module supports dual-encoder feedback and accepts feedback from two different sources for a single axis. To understand the circumstances for which you need to use this feature, consider a stage. If you monitor the stage position directly (as opposed to the position of the motor driving the stage), you can tell if a move you have commanded has reached the target location. However, because the motion controller is providing input signals to the motor and not to the stage, which is the primary source of feedback, the difference in the expected output relative to the input can cause the system to become unstable. To make the fine adjustments necessary to help the system stay stable, you can monitor the feedback directly from the encoder on the motor as a secondary feedback source. Using this method, you can monitor the real position of your stage and account for the inaccuracies in the drive train.

Motor Selection

A motor provides the stage movement. Some high-precision stages and mechanical components feature a built-in motor to minimize backlash and increase repeatability, but most stages and components use a mechanical coupling to connect to a standard rotary motor. To make connectivity easier, the National Electrical Manufacturers Association (NEMA) has standardized motor dimensions. For fractional horsepower motors, the frame sizes have a two-digit designation such as NEMA 17 or NEMA 23. For these motors, the frame size designates a particular shaft height, shaft diameter, and mounting hole pattern. Frame designations are not based on torque and speed, so you have a range of torque and speed combinations in one frame size.

The proper motor must be paired with the mechanical system to provide the performance required. You can choose from the following four main motor technologies:

- **Stepper motor**—A stepper motor is less expensive than a servo motor of a similar size and is typically easier to use. These devices are called stepper motors because they move in discrete steps. Controlling a stepper motor requires a stepper drive, which receives step and direction signals from the controller. You can run stepper motors, which are effective for low-cost applications, in an open-loop configuration (no encoder feedback). In general, a stepper motor has high torque at low speeds and good holding torque but low torque at high speeds and a lower maximum speed. Movement at low speeds can also be choppy, but most stepper drives feature a microstepping capability to minimize this problem.
- **Brushed servo motor**—This is a simple motor on which electrical contacts pass power to the armature through a mechanical rotary switch called a commutator. These motors provide a 2-wire connection and are controlled by varying the current to the motor, often through PWM control. The motor drive converts a control signal, normally a ±10 V analog command signal, to a current output to the motor and may require tuning. These motors are fairly easy to control and provide good torque across their ranges. However, they do require periodic brush maintenance, and, compared to brushless servo motors, they have a limited speed range and offer less efficiency due to the mechanical limitations of the brushes.
- **Brushless servo motor**—These motors use a permanent magnet rotor, three phases of driving coils, and Hall effect sensors to determine the position of the rotor. A specialized drive converts the ±10 V analog signal from the controller into three-phase power for the motor. The drive contains intelligence to perform electronic commutation and requires tuning. These efficient motors deliver high torque and speed and require less maintenance. However, they are more complex to set up and tune, and the motor and drive are more expensive.

Stage Drive Technology	Speed	Maximum Load	Travel Distance	Repeatability	Relative Complexity	Relative Cost
Stepper	Medium/Low	Medium/Low	High	Medium/Low	Low	Low
Brushed Servo	High	High	High	Medium	Medium	Low
Brushless Servo	Very High	High	High	High	High/Medium	High

Table 10.4. Comparison of Motor Technology Options

Because the motor and drive are so closely coupled, you should use a motor and drive combination from one vendor. Though this is not a requirement, it makes tuning and selection easier.