# Designing Automated Test Systems

*A Practical Guide to Software-Defined Test Engineering*

NATIONAL INSTRUMENTS

# Contents

# Preface

Defining a corporate test strategy is critical to reducing cost and maximizing the efficiency of your product development and manufacturing organizations. You should decide on a predominant test strategy based on where your organization is today as well as where it plans to be the next five to 10 years. At the highest level, a strategy is typically dominated by the volume and mix of your product portfolio. You can represent volume and mix in four quadrants, as shown in Figure 1.

I. It is difficult for large companies to have a common test strategy. Each division is like a separate company and has unique requirements. However, in a division, you can begin to form a common test strategy that typically falls into different quadrants. Standardization is a key strategy for balancing volume and mix. (See Hella KGaA Hueck & Cocase study on page 6.)
II. It is cost-prohibitive to build dedicated testers to support each product. Each tester should be flexible enough to support multiple products. (See Benchmark Electronics case study on page 6.)
III. Typically, a small test organization needs a handful of testers, each of which should be product-specific and optimized for cost. (See any small startup.)
IV. Corporations must optimize for continuous flow by employing test strategies such as parallel test that maximize capacity. (See Harris Communications case study on page 7.)

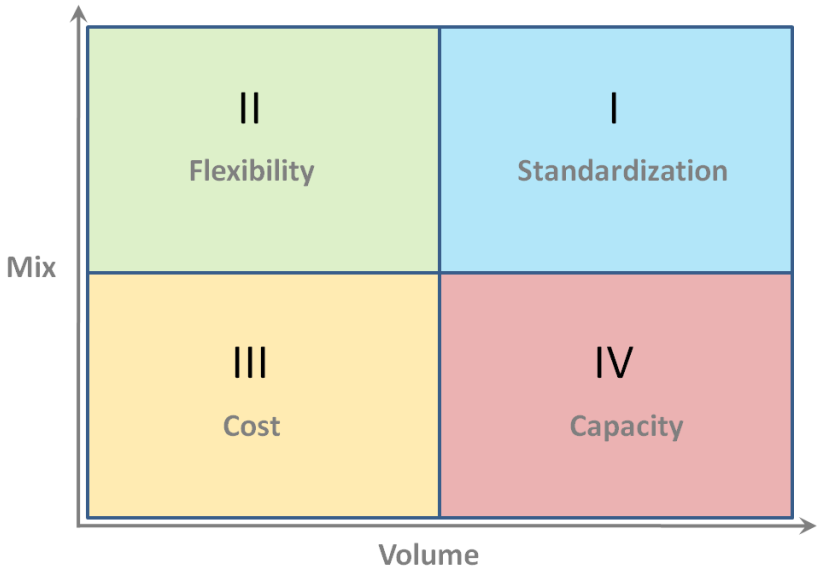



*Figure 1.*You can represent volume and mix in four quadrants and map them to a test strategy.

For more than two decades, National Instruments has collaborated with industry-leading companies to document test strategy best practices and techniques for building more effective automated test systems. NI works with a multitude of sources as well as members of the NI Automated Test Customer Advisory Board (AT-CAB) to capture these best practices. The AT-CAB community is a cross-section of industries that work to leverage constantly changing commercial technologies while maintaining long-term supportability of current test systems.

The adoption of software-defined test systems is the most significant trend among these industry-leading test engineering teams. Engineers are using software-defined test systems to achieve new levels of measurement performance and lower test costs. The quick return on investment from these benefits is contributing significantly to the mainstream adoption of the software-defined test system approach.

Thousands of companies are building software-defined test systems based on NI software tools and the open, multivendor PXI hardware standard. According to the PXI Systems Alliance, more than 100,000 PXI systems will be deployed by the end of 2009, and the number of deployed PXI systems is expected to double in the next decade. Below are three examples of large companies that have adopted the software-defined test system design approach, even though their respective company test strategies fall into different quadrants. The content for this guide has been developed using best practices shared by NI AT-CAB members and the expertise of the NI test engineering and product research and development teams.

## Hella KGaA Hueck & Co.

This **€**3.9 billion international automotive company serves the automotive lighting and electronics needs of leading vehicle makers worldwide. Hella created a standard universal tester (SUT) to serve as a homogeneous yet modular platform capable of testing a variety of products with minimal modification and changeover time between tests. Hella was able to deploy over 200 SUTs globally.
**Quadrant I - Standardization**

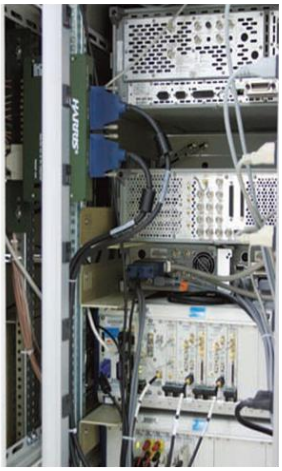
## Benchmark Electronics

Benchmark Electronics, a leading electronics contract manufacturer, created a standard test platform to serve a wide range of product functional test categories and to deliver the best instrument capability with a low cost and small production footprint. The tester is internally called the Target Tester. Benchmark has deployed over 25 test systems.
**Quadrant II - Flexibility**

## Harris Communications

The RF Communications division of Harris Communications is a $1 billion USD business unit. This division develops multiband tactical radios for military use. The Falcon III test system, on the other hand, is capable of testing up to eight radios in about the same time required by the previous system to test two radios, resulting in a 400 percent throughput improvement per tester.

**Quadrant IV - Capacity**



## NI Test Engineering Strategy

NI is a medium-sized ($500M to $1B USD) high-tech company whose products serve several markets including the design, industrial, and test and measurement markets. From a mix-versus-volume perspective, NI can be classified as a high-mix, low-volume organization (quadrant II) because it has more than a dozen primary product lines with more than 1,000 unique devices. Based on these characteristics, the corporate test strategy at NI emphasizes flexibility and reuse. By building flexible testers, NI test engineers can test all of the products within a product line on a single test station as well as quickly adapt the test stations to address the requirements of the more than 200 new products that NI releases each year. Each test station is used when performing regression testing and product validation/verification in design and for functional test in production.

The NI test engineering team has developed and implemented a common test software and hardware platform that can be scaled across multiple product lines. The strategy was to create and maintain a standard test development software environment with flexible capabilities that engineers can use to focus on developing tests rather than reinventing their own, unique test frameworks. This not only fostered test code reuse across new products and product lines but also provided common interfaces with enterprise systems to help improve quality tracking and consistency in test data storage.

To build the framework, the test engineering team used standard commercial off-the-shelf (COTS) technologies to maximize personnel and capital resources. They selected NI TestStand software to handle test management, development, enterprise integration, and operator interfaces and NI LabVIEW graphical programming software to develop the test modules. This software framework connects to a modular test hardware platform based on a PXI core platform and a combination of hybrid PXI and GPIB instrumentation. The hardware framework offers a common base of PXI modular instrumentation but also provides for unique configurations based on product line needs. Figure 2shows the block diagram of the modular, software-defined test architecture. The architecture, which is maintained by the NI test engineering team, offers commonality across all test systems.

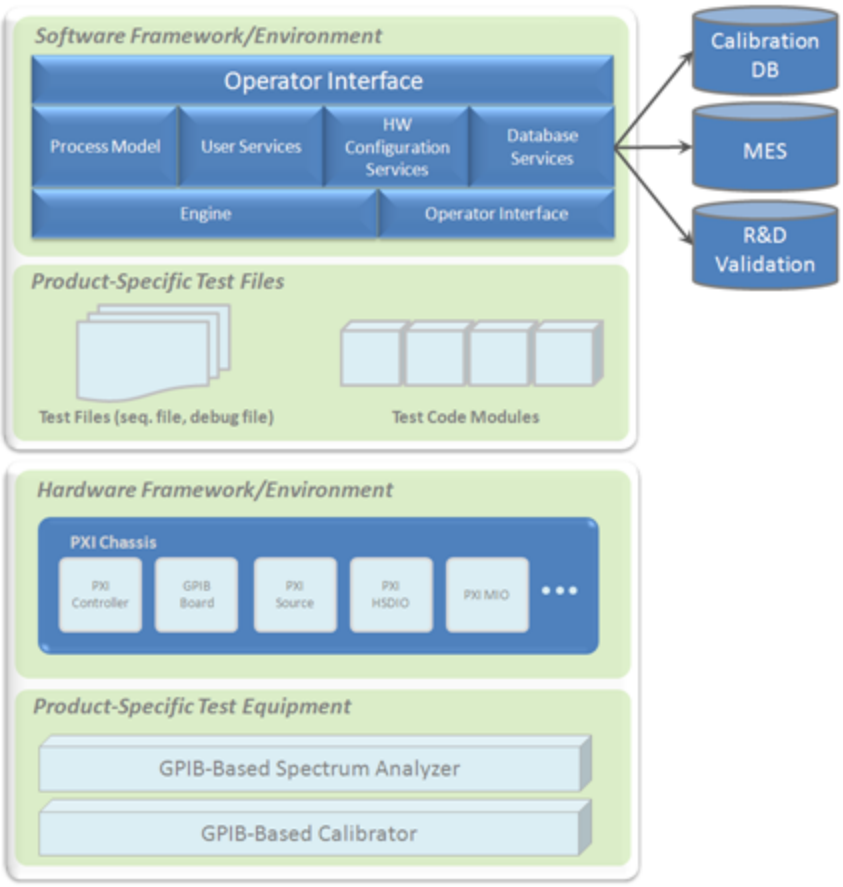



*Figure 2.*Modular Architecture Designed and Implemented by the NI Test Engineering Team

## Recommended Test System Development Process

This five-step guide details the recommended process for building software-defined test systems from start to finish. It presents these test engineering best practices in a practical and reusable manner and features specific examples used by industry-leading test engineering teams. It also references a scalable, software-defined production test system developed by the NI test engineering team for testing I/O modules for the NI CompactRIO platform, which is shown in Figure 3. CompactRIO is an advanced embedded control and data acquisition system designed for applications that require high performance and reliability.

*Figure 3.*The NI CompactRIO product family features more than 50 modules.

Figure 4 depicts the production test system from different viewpoints.

*Figure 4.*Multiple Views of a Software-Defined Production Test System

First the guide focuses on best practices for choosing your test system hardware. Topics in this section range from making instrument decisions for your test system to choosing your rack type and power distribution unit. Next, the guide examines how to connect your instrument to your device under test (DUT) by offering best practices for designing your switch network, choosing your mass interconnect system, and building a custom fixture. Figure 5 shows a close-up of a production test system switching, mass interconnect, and custom fixture.


*Figure 5.*A Close-Up of a Production Test System Switching, Mass Interconnect, and Custom Fixture

After discussing various best practices for choosing hardware, the guide delves into designing a strong software framework that you can use across multiple tests and products. Topics in this section range from making appropriate driver decisions to integrating code modules into a test executive (Figure 6), which is the software layer that handles the operations common for all test scenarios with respect to a given test system. Finally, the guide features best practices for validating, deploying, and maintaining your test system over its lifetime.

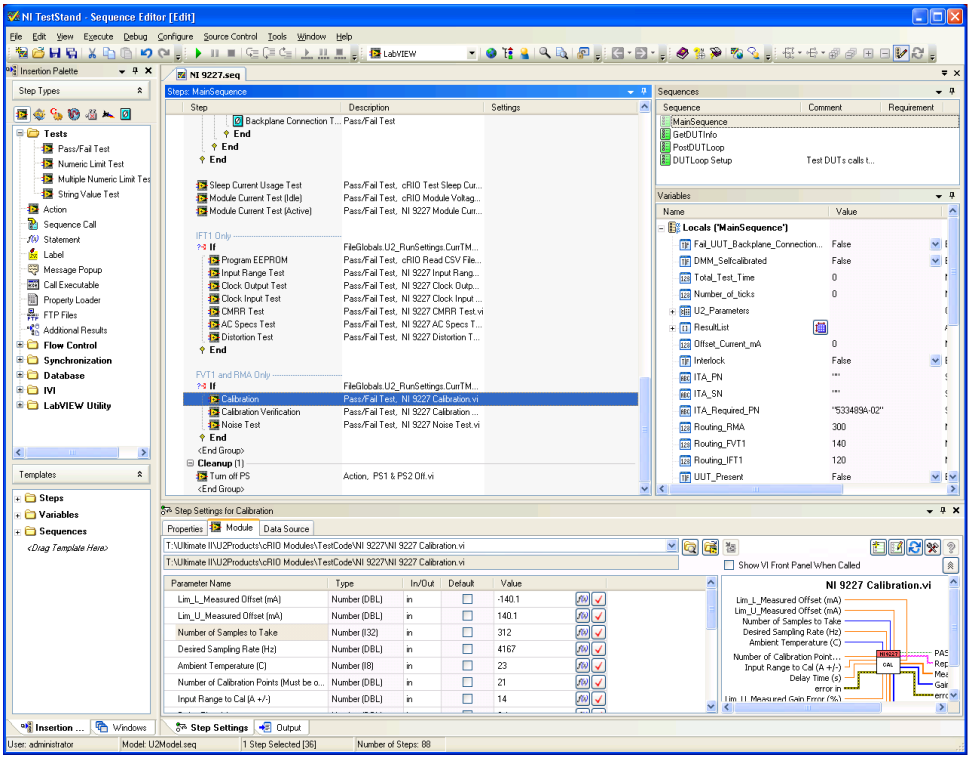



*Figure 6.* Use the NI TestStand Sequence Editor to develop automated test systems.

# Step 1: Identifying Measurement Needs

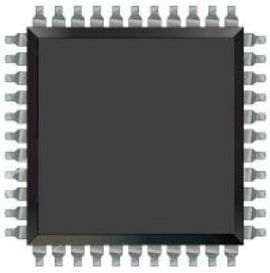
Your company has a test strategy in place. Your test architects have done an excellent job in putting together a solid software framework that takes your test engineers' needs into consideration. Now that you know your constraints as well as priorities, you are ready to start designing your test system. The first step is to determine the measurement requirements for your device(s) under test. This section outlines the various factors to consider when evaluating the measurement needs of your test system. It also examines the process the National Instruments test engineering team underwent to choose instrumentation for the automated test system used to test more than 50 NI CompactRIO I/O modules.

## 1.1 Identifying the Scope of Your Test System

The first step in identifying your test measurement needs is to determine the system's scope. Is the system testing a single product, an entire product line, or a series of product lines? Take a look at a simple example of how determining the scope can significantly change test system requirements.

### Scenario 1: Testing a Single Product

Assume that you are a test engineer working for a semiconductor company. Your immediate goal is to design a system that can test the rise time, nonlinearity (integral nonlinearity or INL and differential nonlinearity or DNL), and current leakage specifications of the digital-to-analog converter (DAC) shown in Figure 1.



**Digital-to-Analog Converter**

- Rise Time = 5 ns
- Resolution = 8 Bits
- Current Leakage = 10 µA

*Figure 1.* DAC under Test

To ensure that you test the device rather than the test system, you need a set of instruments with better specifications than the DAC under test. Thus, your test system must have a high-speed measurement instrument with a rise time that is faster than 5 ns or a bandwidth that is greater than 700 MHz (bandwidth = 0.35/rise time). In addition, you must fit the system with an instrument that has a current sensitivity greater than 10 µA. Finally, the system must have an instrument with resolution greater than 8 bits to appropriately measure the DAC code widths and perform the nonlinearity tests.

### Scenario 2: Testing an Entire Product Family

Now consider building a system that can test the rise time, INL and DNL, and current leakage specifications of the entire family of analog-to-digital converters (ADCs) shown in Figure 2.

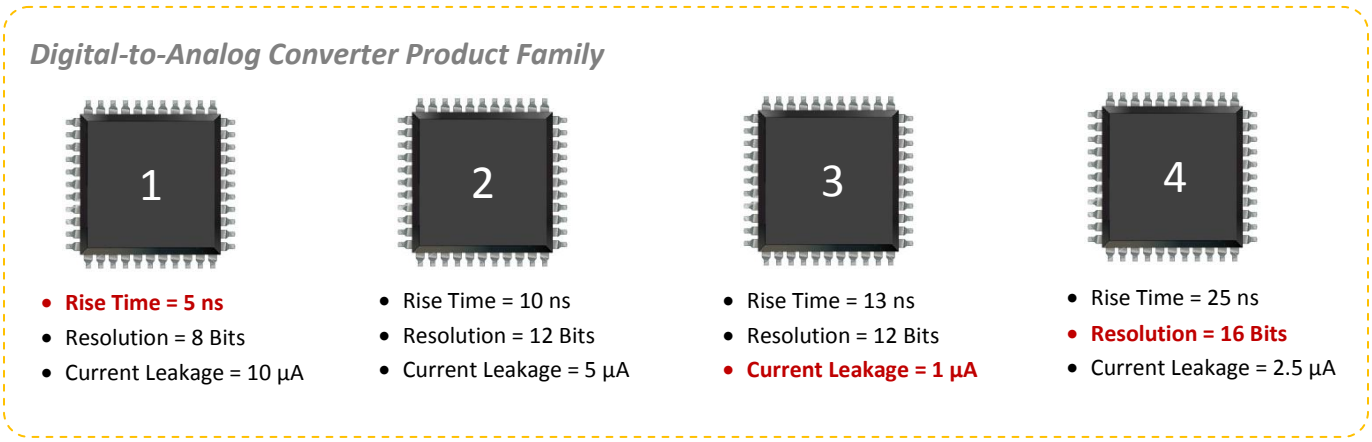



*Figure 2.* DAC Product Family under Test

To test the DAC product family shown in Figure 2, your system must incorporate instruments that have specifications superior to that of the entire product family. Thus, the instruments in the test system must have the following:

a. A rise time that is faster than 5 ns or a bandwidth that is greater than 700 MHz (bandwidth = 1/rise time)
b. Current sensitivity greater than 1 µA
c. Resolution higher than 16 bits

## Scenario 3: Testing Multiple Product Families

It is tempting to widen the test system's scope as much as possible to have a common platform for several programs; however, the following pitfalls can occur:

- To accommodate different product lines, the complexity of the core test system increases, thus increasing nonrecurring, recurring, and material costs
- Maintaining configuration control is difficult among a larger group
- Obsolescence issues increase
- Costs increase on high-production-rate product lines requiring multiple test systems even though a DUT may use only a small portion of the test system capabilities
- Designing test systems for a new product line becomes difficult because of the constraints to use only the existing capabilities of the system
- Keeping up with state-of-the-art technology grows more difficult as test capabilities start to stagnate

## Future Plans and Other Considerations

In addition to understanding your current tester needs, you need to evaluate its future requirements. Are you going to use the tester to test additional product families going forward? Will you add new products to the current product family? If you answered "yes," then you must also consider the measurement needs of these future additions. If you are certain that your test system needs will expand but are unsure of the measurement requirements of your future products, you must design your system

using a modular platform that is easily scalable. For example, you should make interfaces easily available to your test system such as USB, LAN, and GPIB so that you can quickly add new measurement capability to the system that is not available in the rack such as a USB-based modular instrument.

### *Other Considerations*

- Budget and timeline
- Expected life span of the test system
- Additional test requirements such as fault diagnostic capability
- Skill level of operators
- Product volume

## Avoiding Scope Creep

Ensure you understand the project vision and spend time documenting and determining the project objectives. Produce a project plan document that describes the test system deliverables. It is a good idea to document what is in scope as well as out of scope for absolute clarity. Verify the content of this document with the key stakeholder, spending time to walk them through it, and ask them to sign off on it. You should plan for some degree of scope creep in most projects; therefore, it is important to design a process to manage these changes. You can then implement a simple process of document, consider, approve, and assign resources. Use a change control form and change log from the start of the project and communicate the process for using these forms to the customer and project team. Attach a cost and time to each change so the customer is clear about its impact. Implementing a formal process helps ensure there's a clear business value for the requested change.

## Example

Put the concepts discussed previously into practice by examining a real-world example. As with other scenarios in this guide, this example is based on the automated test system for testing a variety of CompactRIO I/O modules. The scope of this system is to test a product family of more than 50 I/O modules for the CompactRIO platform, which is an advanced embedded control and data acquisition system designed for applications that require high performance and reliability. In addition, the test system must be scalable to test future CompactRIO module releases.

*Figure 3.* The NI CompactRIO I/O product family consists of more than 50 modules.

To design a system to test the entire family of CompactRIO I/O modules, NI engineers thoroughly evaluated the measurement requirements of all 50 modules. Based on this analysis, they compiled a comprehensive list of every measurement they needed to make and identified their most stringent measurement needs. For example, NI engineers determined that the test system required the ability to

source currents up to 10Aand measure voltages as low as 1 mV. Finally, they chose suitable instrumentation to address these stringent needs.

## 1.2 Choosing a Core Hardware Platform

After determining the measurement needs of your test system, you can begin architecting your hardware framework. Many test engineers jump straight into matching their measurement needs to instruments available on the market. A better approach is to first pinpoint a suitable test platform that can serve as the core or nucleus of your test system. You can choose from many platforms, most of which are based on one of the four most commonly used instrument backplanes/buses –PXI, GPIB, USB, and LAN. Because each of these buses has at least some advantages and limitations (as discussed in chapter 4 of the Software-Defined Test Fundamentals guide), you often have to build hybrid test systems based on multiple platforms. Even so, it is often a best practice to pick a prominent or core platform for your test architecture. This section outlines some of the factors you must consider when choosing a core platform for your test system.

### Processing Power and Data Throughput

Assess the worst-case computational power and throughput rates when selecting a controller.

### Scalability

Another factor is the ease with which you can scale or modify your system. This is especially important if your test system has the potential to change during the course of its lifetime. One example of this is if you are building a system to test a product family that is continually expanding. In such a case, you may need to add new functionality to the system without making significant changes that could force you to redesign your test rack.

### Measurements Diversity

The platform that serves as the core of your test system must be able to address a significant portion of your test system needs. Thus, if your system requires the ability to make low-level DC measurements along with high-speed rise time measurements, you must select a platform that is capable of accommodating mixed-signal instrumentation. In general, you should choose a core platform that accommodates at least 80 percent of your test system's measurement needs.

### Communication with Other Buses and Instruments

As mentioned previously, each instrument bus and platform has distinct advantages and disadvantages. By building hybrid systems based on multiple instrument buses, you can take advantage of the strengths of several different test platforms. A hybrid architecture also increases the flexibility of your test system by allowing you to choose from a larger pool of instruments on the market. Such flexibility is especially important if you are building a complex and dynamic test system that will change over time. The first step toward building a hybrid architecture is choosing a core platform that can communicate with instruments that are based on a variety of instrument buses.

## Timing and Synchronization

When designing a test system composed of multiple buses and platforms, you must ensure that your core platform can synchronize those instruments by sending triggers and sharing clocks.

## Lifetime

Another factor to consider is the lifetime of your test system. If you expect to use your system for several years, you should choose a platform that can stand the test of time. Sometimes products and platforms go end of life (EOL). It is often difficult to service and maintain products like these in a test system. For this reason, you must choose a proven platform for which products and replacements will be available for several years. For long military programs, consider vendor support agreements or lifetime buys of equipment that may be required.

## Example

These six criteria helped NI engineers select a core platform for the automated test system they used to assess the CompactRIO I/O modules. The following is an evaluation of test system needs based on the criteria:

1. **Processing Power and Data Throughput:** This system required a multicore processor and a high-speed data bus that could support the expected data analysis.
2. **Scalability:** Because the CompactRIO product line is continually evolving and growing, it was essential for the system to use a core test platform that is highly scalable to be able to add new functionality for future CompactRIO platform releases.
3. **Measurements Diversity:** The CompactRIO tester had to be capable of testing the large variety of I/O types on the more than 50 modules for the CompactRIO platform. These include ±80 mV thermocouple inputs, ±10 V simultaneous-sampling analog I/O, 24 V industrial digital I/O with up to 1 A current drive, differential/TTL digital inputs with 5 V regulated supply output for encoders, and 250 $V_{rms}$ analog inputs. Because of the variety of signal types and voltage ranges these modules need to address, the NI engineers also required the tester to make a wide range of measurements.
4. **Communication with Other Buses and Instruments:** The CompactRIO platform is continually growing and future product plans are hard to predict, so NI engineers needed a tester with a high level of flexibility to accommodate a vast range of instruments that are based on different buses.
5. **Timing and Synchronization:** Because there is a possibility for multiple instruments based on different buses to coexist in the system, the core platform needed to seamlessly synchronize these instruments.
6. **Lifetime:** The test system was designed to work for the entire lifetime of the CompactRIO product line, and thus required a core platform that is continually growing and whose components were likely to be serviceable and replaceable for several years.

Based on these six requirements, NI engineers chose the NI PCI eXtensions for Instrumentation (PXI) platform as the core of the test system. PXI provides several benefits that meet test system needs.

The most prominent reason for selecting PXI is its modular and scalable architecture. In PXI, all instruments share many components, such as the chassis, power supply, and controller, so adding new instruments is as easy as plugging a module into one of the empty slots in the chassis. This framework makes it simple to add new measurement capabilities to the system in a cost-effective manner. More importantly, the modular architecture of PXI makes it possible to incorporate new capabilities without changing the physical dimensions of the test rack.

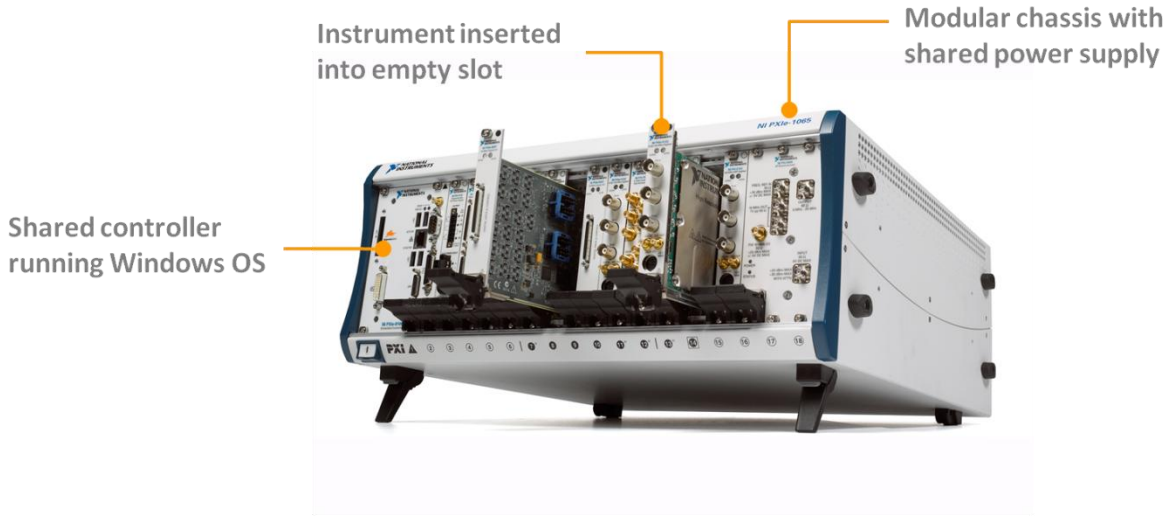



*Figure 4.* PXI provides a modular and scalable architecture.

In addition to being scalable, PXI is capable of addressing a diverse set of measurement needs. There are nearly 1,500 PXI products that provide the following functionality:

- Analog input and output
- Boundary scan
- Bus interface and communication
- Carrier products
- Digital input and output
- Digital signal processing
- Functional test and diagnostics
- Image acquisition
- Prototyping boards
- Instruments
- Motion control
- Power supplies
- Receiver interconnect devices
- Switching
- Timing input and output
- RF and communications

This diverse range of products made PXI suitable to serve as the core of the test system, which required a broad range of functionality to test the entire family of more than 50 CompactRIO I/O modules.

Another advantage of PXI is its ability to connect to multiple instrument buses including USB, Ethernet/LAN, and GPIB. This functionality helped increase the flexibility of the test system by enabling the addition of instruments based on various instrument buses. This feature of PXI was especially useful

when the tester's functionality required expansion to incorporate tests for the new NI 9227 5 ADC current input module.

The test method for the NI9227 5A current input module involves connecting a precision shunt in series with the input terminals of the module, and sourcing current values from -5 to 5A (entire range of the module) through the input terminal using a power supply. To ensure that the power supply is sourcing the right value, the resultant voltage drop is measured across the precision shunt using the NI PXI-4071 7½-digit digital multimeter (DMM).

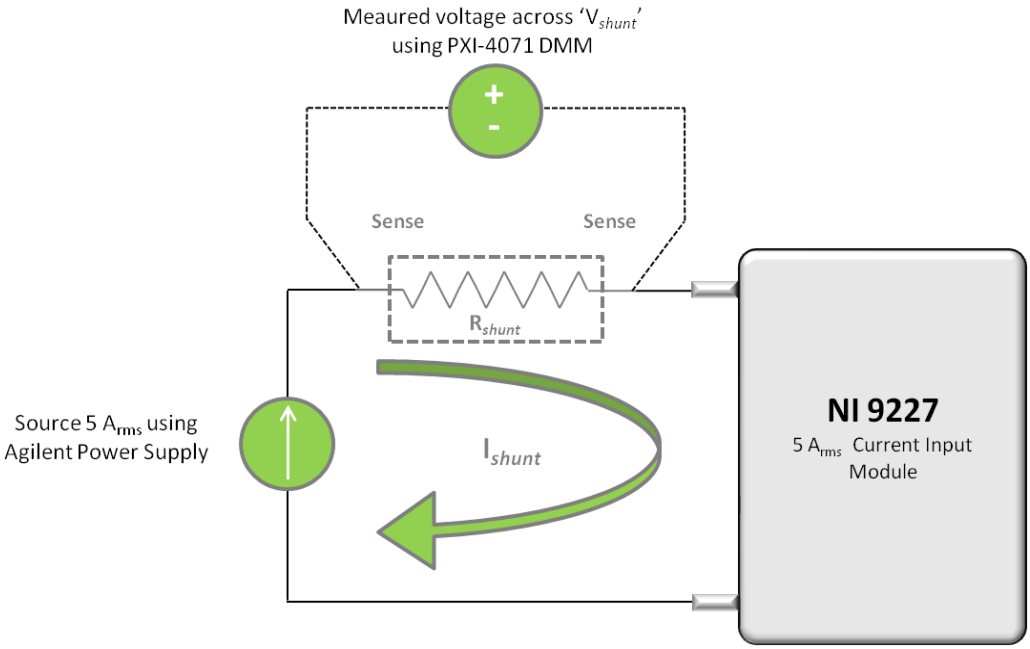



*Figure 5.* Circuit for Testing the NI 9227 Current Input Module.

The measured voltage is then converted to a current value using the following formula:

$$I_{shunt} = V_{shunt}/R_{shunt}$$

This value is then compared with the current that was meant to be sourced by the power supply. Once it is ensured that the power supply is sourcing the right value, the current sourced by the power supply is compared to the current measured by the NI 9227. The difference is entered as a calibration value on the NI 9227.

Because there is currently no PXI module that can source a current of 5 ADC, NI engineers chose the Agilent N6702A modular power system mainframe along with the N6754 DC power module, which is capable of sourcing up to 20 ADC to conduct the test.

The GPIB port on the PXI controller enabled the easy integration of the Agilent power supply into the test system.

Additionally, PXI provided the ability to synchronize the device with other instruments in the test system. For instance, calibrating the NI 9219 analog input module requires measuring the excitation voltage being sent from the module as well as the voltage measured by the input channels of the module simultaneously. For this test, NI engineers leverage the star trigger on the PXI chassis backplane to synchronize both DMMs. In addition to the star trigger, the PXI chassis backplane has several other timing and synchronization features including the following:

- 100 MHz differential system reference clock
- 10 MHz reference clock signal
- Star trigger bus with matched-length trigger traces to minimize intermodule delay and skew
- Trigger bus to send and receive high-speed timing and triggering signals
- Differential signals for multichassis synchronization

Finally, the PXI platform's consistently growing product portfolio along with data from the analyst firm Frost & Sullivan, which stated that PXI revenue in measurement and automation is expected to increase at a 17.6 percent compound annual growth rate (CAGR) through 2014, provided the certainty that servicing the tester throughout its lifetime would not be arduous.

These benefits made PXI the best platform to serve as the core for the CompactRIO module test system.

## 1.3 Determining the Required Instrumentation

Now that you have a better understanding for determining your test system measurement requirements and choosing your hardware platform, you are ready to start selecting the specific hardware instruments you need to conduct your measurements. This section features some best practices for choosing instruments for your tester.

### Basing Instrument Choices on Measurement/Stimulus Rather Than Instrument Type

Test engineers often choose an instrument based on type rather than need. For example, many engineers select DMMs to make high-accuracy measurements even though in many applications, the accuracy of a data acquisition board may be sufficient. Such decisions often result in higher costs, so you should choose your instrument based on your measurement need rather than the instrument type.

Following this practice was highly beneficial when NI engineers selected a method to calibrate the NI 9219 thermocouple module in the test system described in this guide. Typical calibration methods involve using expensive instruments that cost upwards of $50,000 USD. In this particular test system, however, the NI 9219 is calibrated using a Keithley source measure unit (SMU) and the NI PXI-4071 7½-digit PXI DMM.

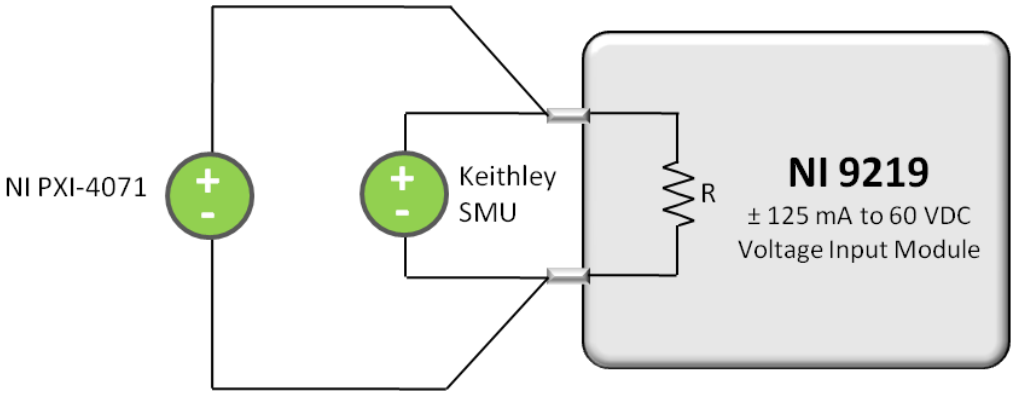



*Figure 6.* Circuit for Calibrating the NI 9219 Voltage Input Module

This is possible because the PXI-4071 DMM has an accuracy that is substantially higher than that of the NI 9219. Table 2 provides a comparison between the accuracy of the NI 9219 and the PXI-4071.

| | NI 9219 | | NI PXI-4071 (2 yr cal values) | |
|---|---|---|---|---|
| | *Gain Error (ppm)* | *Range Offset Error (ppm)* | *Gain Error (ppm)* | *Range Offset Error (ppm)* |
| **125 mV** | 3000 | 120 | 32 | 2 |
| **60 V** | 1000 | 20 | 22 | 0.8 |

[1]ppm = parts per million

*Table 2.* NI 9219 vs. NI PXI-4071 Accuracy Comparison.

As you can see from Table 2, the accuracy of the PXI-4071 is several times that of the NI 9219. In addition, because the PXI-4071 was already required for testing other CompactRIO modules, using it for calibrating the NI 9219 helped to reduce the overall cost of the test system substantially.

## Test Accuracy Ratio

Another best practice for choosing your test system instruments is to calculate the test accuracy ratio (TAR) to ensure that the accuracy of your measurement equipment is substantially larger than the accuracy of the component you are testing. If you do not meet this criterion, then you may see significant measurement error caused by both the device under test and the test equipment, making it impossible to know the true source of error. Because of this, engineers use TAR to determine the relative accuracy of the measurement equipment and the component under test. You can calculate TAR with the following formula:

*TAR =Desired Accuracy of the Component Under Test/Accuracy of Measurement Equipment*

Your TAR value should equal 4 or more, depending on the test you are performing and the test certainty you require. TAR was one method NI engineers used in determining the suitability of the PXI-4071 for

calibrating the NI 9219. Because the accuracy of the PXI-4071 is more than 10 times that of the NI 9219, it was deemed suitable for calibrating the NI 9219.

## Other Considerations

In addition to determining measurement needs and pinpointing the right combination of instruments to test the device under test using the TAR, you often need to make several decisions unique to each test system. In the case of the automated test system built for testing CompactRIO modules, NI engineers had to give special consideration to accommodating the measurement needs of the NI 9219 universal input module. The NI 9219 can operate in several different modes, including a full-bridge mode. When operating in this mode, it sources a specific excitation voltage to facilitate current flow through the bridge. At the same time, the module measures the voltage drop across the load that is the DUT. The measurement recorded by the NI 9219 is the ratio of the voltage drop across resistor R and the excitation voltage provided by the NI 9219.

*Note:* No current flows through the ADC because it is a high-impedance circuit.

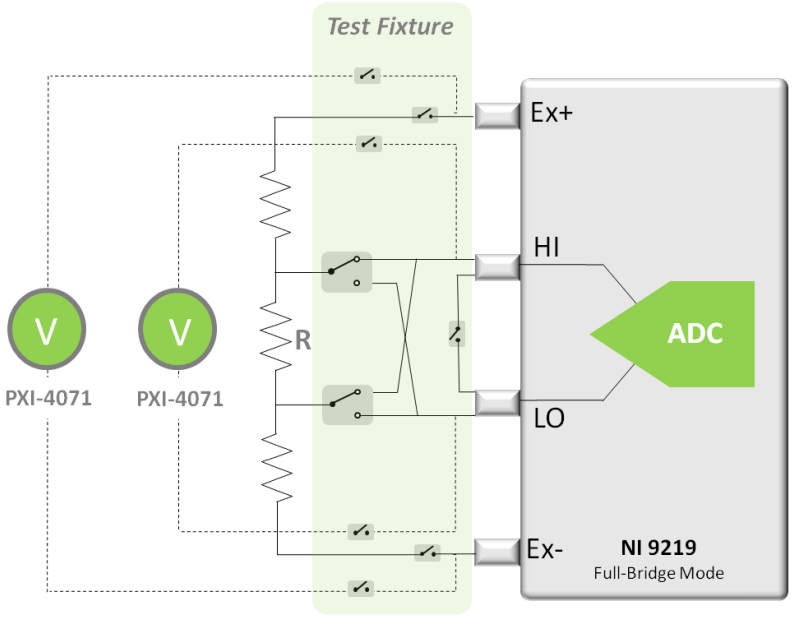



*Figure 7.* NI 9219 in Full-Bridge Mode

In such a measurement, there are two different sources of error. The first is the voltage measurement and the second is the voltage excitation. To test the NI 9219, both sources have to be measured simultaneously. The test station therefore uses two PXI-4071 7½-digit DMMs to measure these two sources of error at the same time.

# Step 2: Best Practices for Selecting Hardware

# Step 2.1: Choosing Rack Type, Size, and Power Distribution Unit

Choosing your hardware and designing your test system can be a daunting task. It is absolutely essential that you get this step right to avoid spending additional money during the lifetime of the test system. For instance, not leaving enough empty space in the test rack can lead to a complete test system redesign, which can cost thousands of dollars and result in months of downtime. Some key factors that you should consider when designing rack-mounted test systems include rack type, rack size, power distribution within the enclosure, fixturing, mass interconnect systems, and switching.

This section offers guidance on determining your rack type and size and examines important factors to keep in mind when choosing a power distribution unit for your rack.

## 2.1.1 Choosing the Rack

Rack-mount enclosures are typically used to store test equipment, including measurement instruments, switches, cables, fixtures, mass interconnect systems, power supplies, and cooling systems, in a limited amount of space. Some rack enclosures can protect your instruments under extreme conditions. Others can provide the necessary ventilation and cooling to prevent your test system components from overheating. You need to consider many factors when choosing a rack, including environment of use, portability, size, and cooling.

### Environment of Use

The type of rack you choose for your test system depends a lot on the environment in which you plan to use the system. For example, a test system that is being deployed on the manufacturing floor of a global cell phone company can be considerably less rugged and portable than a system that is going to be deployed for military purposes.

Consider the example of a system used for in-flight testing of Australian Army BlackHawk helicopters. A key requirement of the system was to acquire, process, and display up to 80parameters in real time including engine data; flight control data; rotor, air, time, space, and position data; and both day and night cockpit video and audio.

Because the system was being built for use on the helicopter, it required a high degree of ruggedness. Thus, the Royal Australian Air Force chose the PXI platform, which was mechanically designed for industrial environments. To add more protection, the PXI system was placed in a rugged rack casing that withstands a high degree of shock and vibration during flight.

*Figure 1.* The Royal Australian Air Force placed its PXI system in a rugged rack that withstands a high degree of shock and vibration during flight.

## Portability

Another factor when choosing your rack enclosure is portability. Some test systems are built to be stationary – chip testers usually fall in this category. Once a chip tester has been transported from the location where it was built to the manufacturing floor where it will operate, it is rarely ever moved again. In contrast, a test system built by the military to test radios on the battlefield is moving on an ongoing basis and must, therefore, be highly mobile.

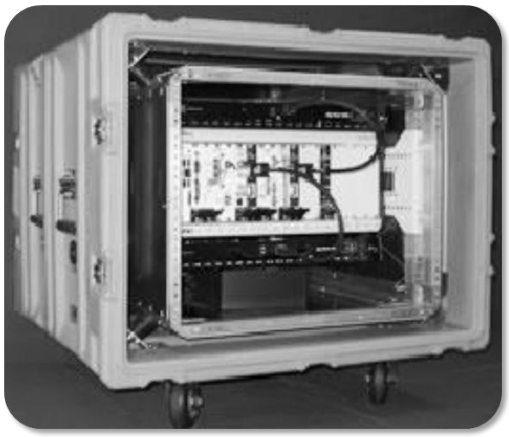


*Figure 2.* Portable rack enclosures are suitable for test systems that need to be mobile.

## Size

Rack size is an important consideration especially if you expect your system to expand in the future. Choosing a rack that is too small can cause you to run out of space and result in a complete redesign.

It is also important to choose a rack size that is based on the Electronic Industries Alliance (EIA) standard because most test instrumentation is built according to this standard. EIA racks are typically 19 in. wide. Their height is measured in “rack units” or “rack U,” where 1U is equal to 1.75 in. A rack’s internal height is measured from the tallest point of any side rail to the bottom chassis, its depth is measured from the insides of both front and rear doors, and its internal width is measured from side to side.

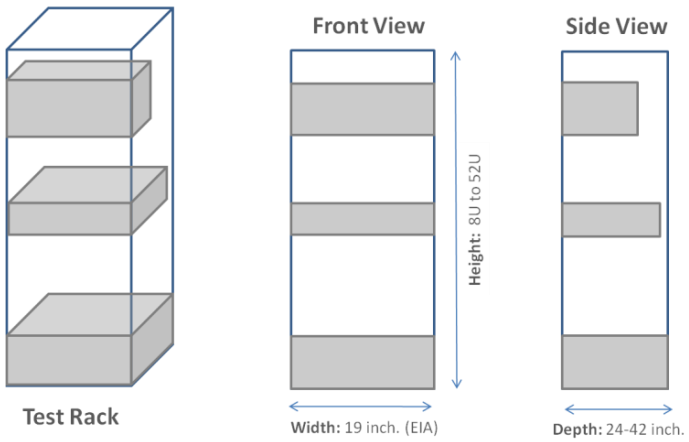



*Figure 3.* Typical Rack Enclosure Size

Racks can vary in height: 8U, 12U, 15U, 20U, 24U, 32U, 42U, 44U, 45U, and 52U. Rack depth, which is measured in inches, also varies and typically falls between 24 and 42 in. To determine an appropriate size for your rack, pay close attention to the following factors.

**Instrument and Accessory Size:** You must ensure that the rack enclosure you choose has sufficient space for all of your instrumentation. Additionally, you must allocate space for accessories such as keyboards, LCD monitors, integrated test adapters, test fixtures, and accessory drawers, which you can use to house spares and manuals.

**Floor Space:** Be sure that you can accommodate the rack's external dimensions in the floor space allotted for it. Also, check that it can be transported through your facility's doorways and elevators safely.

**Future Expansion:** If you plan to add more functionality to your test system to expand its capabilities, you need to leave some empty space in the rack to accommodate future test needs. A good rule of thumb for this is 20 percent of additional space.

**Ventilation:** Most test instrument vendors recommend that you maintain a certain amount of space above and below the instrument for hot air to escape. This prevents the overheating of test equipment. Typically, an NI PXI chassis with a rear air intake and top/side exhaust needs a minimum of 76.2 mm (3 in.) of clearance from the air intake on the rear of the chassis and 44.5mm (1.75 in.) of clearance above and on the sides of the chassis. Because different instruments and vendors have different requirements, you must ensure that you review the instrument specifications and plan for enough ventilation space when determining the size of your rack.

## Cooling

You must also ensure that your rack enclosure is capable of dissipating the heat generated within the system. The type of rack you choose depends on the cooling methods used in your environment. Typically there are two main types of options for cooling.

**Option 1:** If your test system is set to operate in a relatively clean environment, you can select a perforated or vented rack that can provide a channel for the hot air to escape the system. A perforated rack is also the best solution if the primary method of cooling in your test environment is ambient air cooling (fans, air handlers, blowers, and/or room air conditioning).An example configuration is a push-pull system with filtered blowers located at the very bottom of the rack that pull ambient air in and an exhaust fan at the very top of the rack to push heated air out. The combination of the two rapidly removes heated equipment air.

**Option 2:** If your test system operates in an environment where it is exposed to dust, you may want to choose a fully sealed rack. However, when using such an enclosure, you must make sure that there is sufficient cooling inside the rack (typically with a liquid cooling unit or rack air conditioner) to prevent components from overheating. Because air conditioners and cooling units are typically rated for different British thermal units (BTUs), you must add the total wattage of each device and convert the sum into BTUs (1 kW = 3412 BTUs). Depending on this value, you can choose a suitable cooling system.

## Other Rack Design Considerations

- Height and width – It must fit in doorways and elevators.
- Structure and wheel capacity – Power supply racks tend to get quite heavy.
- Style and color – You may want to blend well with existing equipment where the test system will be deployed.
- Adding attention indicators – Adding lights to alert the operator when to answer a test prompt or when there is a failure allows multiple test systems to be controlled by one operator.
- Work surfaces – Consider whether it will be deployed to a vehicle, a tarmac, or a factory floor.
- Operator ergonomics – Consider the average height of your operator.

## Real-World Example

Now examine the four criteria of environment, portability, size, and cooling more closely by considering a real-world example. As with other scenarios in this guide, this example is based on the automated test system for testing various CompactRIO I/O modules. For a complete overview of this system, refer to the Test Strategy and Example System Overview section at the start of this guide.

### *Environment*

The CompactRIO I/O module test system was designed for operation in NI manufacturing facilities. These facilities are similar to those at companies that manufacture electronics products; thus, they are air-conditioned and generally clean, with minimal pollution. As a result, the test system did not need to endure high vibration, pressure, or temperature, so a rugged rack enclosure was not required for the test system.

### *Cooling*

Cooling on the test system was achieved through an exhaust fan located at the top of the rack enclosure along with perforated or vented panels that enable hot air to escape. As mentioned earlier, the operating conditions for the system are relatively clean and dust-free; thus, there was no need to use a fully sealed rack. NI test engineers did not need an internal air conditioning system because the factory

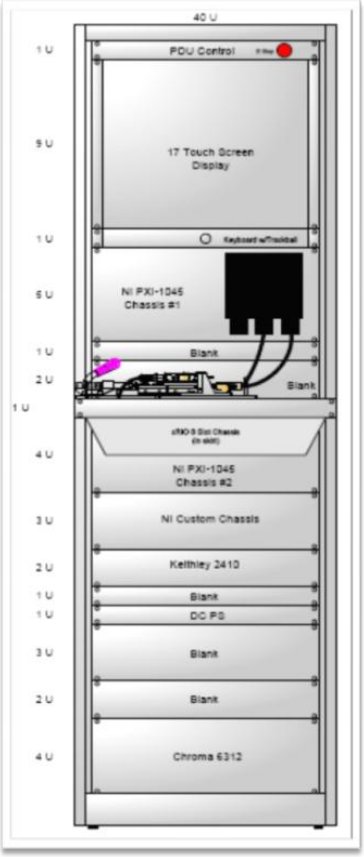
air conditioning along with the exhaust fan within the system was enough to keep all instrumentation from overheating.

### *Portability*

Because the system was designed to stay stationary on the manufacturing floor, it did not have to be extremely portable. Even so, the system did have to be transported from the location where it was built to the location where it is used. Test engineers therefore added wheels to the system. Additionally, they ensured that the system's size did not prevent it from fitting into the elevators and lobbies of the buildings between which it would have to be transported.

### *Size*

Instrument and accessory size, floor space, future expansion, and ventilation are the four main criteria for determining rack size. This subsection discusses how each of these considerations shaped the decisions of NI test engineers when they chose a rack size for the test system.




**Instrument and Accessory Size:** In determining the rack size, the total height of all instrumentation in the test system was analyzed. In addition to simply calculating the size of instrumentation, the various types of accessories as well as their sizes were measured. This was extremely important for pinpointing the size of the rack because some accessories took up a significant amount of rack space. For example, the touch panel monitor (see Figure 4) takes up 9U of rack space.

**Floor Space:** To optimize usage of floor space, NI test engineers were asked to fit the entire test system in a single 19 in. rack. To do so, they used the compact PXI platform. Modular instruments designed for the PXI platform provide the same quality of measurements as rack-and-stack instruments but in a more compact form factor.

**Future Expansion:** The CompactRIO product line is continually evolving and growing, so it is likely that the test system will grow over time. To accommodate future test needs, NI test engineers left sufficient space empty (this is shown as "blank" in the image on the left) in the rack.

**Ventilation:** The amount of ventilation or cooling space needed for each instrument in the rack was different. For example, both PXI chassis required at least 3 in. below and 1.75 in. above of empty space to dissipate heat. These requirements were factored into the rack size.

## 2.1.2 Designing Rack Layout

Placement of components in a rack depends mostly on personal preference. That said, there are some general guidelines you can follow to increase measurement accuracy, safety, and ease of use. Four of

the most important considerations are weight distribution, operator ease of use, signal integrity, and debugging.

## Weight Distribution

It is important to distribute the weight of your individual components evenly in the rack to ensure maximum stability. In general, it is always advisable to place your heaviest equipment, such as power supplies, at the bottom of the rack. This ensures a low center of gravity and thus increases stability. You must also make sure that the weight of all the equipment in the rack is evenly balanced from front to back and side to side.

Another factor that can impact test system stability is the placement of external shelves, which are sometimes used for supporting fixtures that house the DUT. If you are using shelves, do not place extensive weight on them. Additionally, place the shelves near the center of the enclosure (height-wise).

Also take into account the weight of the DUT when determining how to distribute weight evenly across the test system because your system must be stable with and without the DUT in place.

Finally, as discussed in the previous section, you may need to leave some empty space in the rack to expand the system for your future test needs. It is best to leave some empty space at the top to allow for the addition of lighter instruments and some space at the bottom to allow for the addition of heavier instruments. By doing so, you can build a stable rack enclosure.

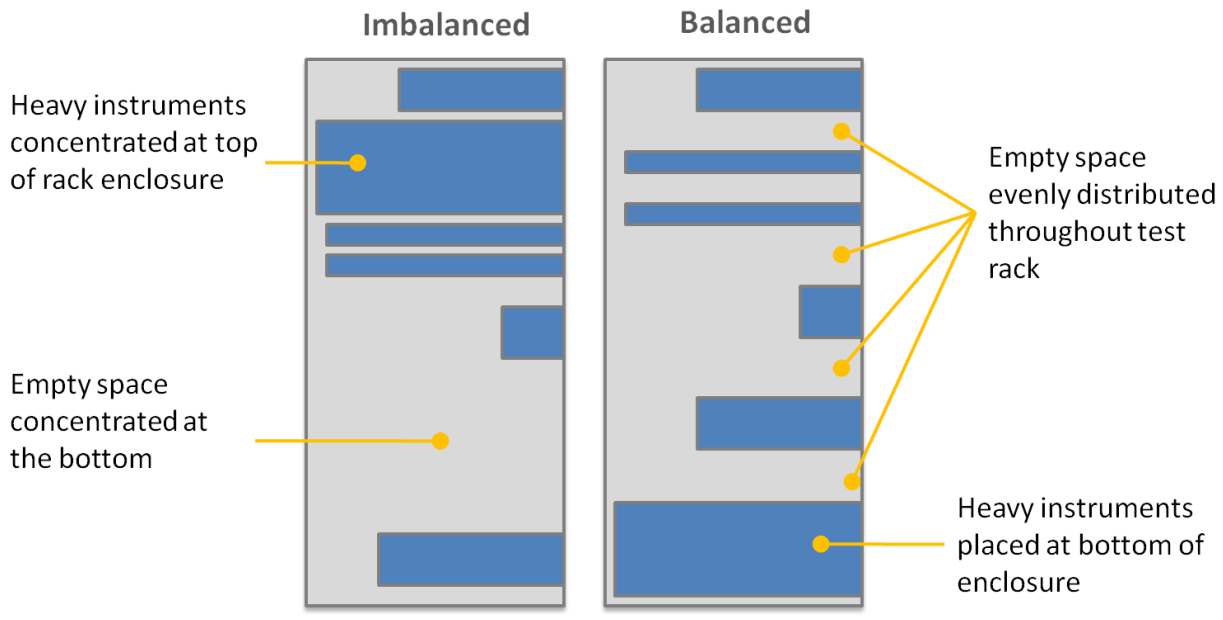



*Figure4.* Imbalanced versus Balanced Test Rack

## Operator Needs

When determining the layout of your rack enclosure, you need to know how the operator will interface with the system. For example, will the operator sit or stand when interacting with the system? The

answer to this question can help determine how high your monitor and workstation should be and where you must place your DUT.

You must also determine how much access you plan to give the operator based on your specific application requirements. For security purposes, many test applications in the defense industry require operators to enter certain codes to access the system. For such applications, you want the operator to have a keyboard and mouse. If this is the case for your system, you must ensure that there is enough workspace for left- and right-handed operators to use the mouse on either side of the keyboard. For most applications, however, operators are required to enter a minimum amount of information to test a product. In this case, you need to provide the operator with just a scanner to scan the product serial number and a touch screen panel.

## Signal Integrity

It is important to keep signal integrity in mind when laying out your test rack. Long cable lengths can often distort signals and cause measurement errors. If you need to measure low-level signals (currents, voltages, resistances, and so on), use minimal cable lengths. Consider an example where you are trying to measure the pin-to-pin leakage current of a specific integrated circuit. Typical values for leakage current on this device are less than 10 nA. If the total leakage current in the cables that connect the device to the measurement instrument is more than 5 nA, then your measurement has up to 50 percent of error. One way to eliminate errors due to cabling is to minimize cabling by placing your switching, measurement instrumentation, and DUT as close to each other as possible.

Cabling can also cause errors in radio frequency (RF) applications. RF signals incur voltage attenuation and power loss when traveling through components with different impedances. Although you can often purchase components that are impedance-matched, these components are never exactly the same. Thus, the longer the total cable length, the more loss the RF signal incurs. One way to reduce cable lengths is to place your RF switches as close to the RF analyzer or generator as possible.

## Debugging

If you are using an instrument with a digital display on the front for debugging purposes, it is important to place these instruments in a location where the display is easily visible.

## Real-World Example

Rack layout is integral to designing an efficient and user-friendly test system. NI test engineers put a lot of thought into the layout of the test system they built to test CompactRIO I/O modules. They paid special attention to weight distribution, operator needs, signal integrity, and debugging requirements. The following are examples of some decisions made when designing the layout of the rack enclosure.

**Weight distribution:** NI test engineers took great care to ensure that heavier instruments were placed at the bottom of the rack. For example, the Chroma power supply, which was the heaviest component in system, was placed at the bottom of the rack to ensure maximum stability. Test engineers also distributed empty spaces in the rack evenly across the rack so that the addition of new instruments for system expansion does not adversely affect the center of gravity of the rack. Additionally, all instruments were centered side to side and front to back to increase the stability of the rack. Finally, the

shelf used to hold the DUT fixture was placed at the center of the rack. With these decisions, engineers ensured a stable test rack design with a low center of gravity.

**Operator needs:** The intent was to provide operators with enough access to scan the device and enter simple test parameters; thus, NI test engineers incorporated a scanner and a touch screen panel. Figure 5 shows that the monitor is at the top of the test rack on the left because the operators need to stand when operating it.


*Figure 5.*Rack Layout of the System Used to Test CompactRIO I/O Modules

**Signal Integrity:** The test system is required to make several low-level (as low as a few millivolts when testing the NI 9219) and high-speed measurements. To maximize signal integrity, NI test engineers minimized cable by placing the switching modules as close to the measurement instrument as possible.

**Debugging:** Instruments for debugging purposes, such as the Keithley source measure unit, have been placed at the front of the rack. This provides test engineers with easy access to measurement readings.

### 2.1.3 Power Distribution

Typically in large test systems, instruments in the rack are plugged into terminal strips inside the rack. These individual strips connect to the main power distribution unit, or PDU, which is usually placed at the bottom of the rack to increase stability. Finally a single connection is made between the PDU in the rack and the power source on the wall.

There are many best practices for choosing PDUs for your test rack. Prior to choosing a PDU, however, you should first determine the total power that your test system needs by adding the total watts of the equipment plugged into the main power supply and selecting a power supply with a watt rating that is higher than the equipment load. To allow for future system growth and power-on inrush current, a good rule of thumb is that the test system load should be about 70 percent of the power supply watt capacity.

Now that you know how much power you need, you can consider the other important features your PDU must have. NI test engineers incorporated the following key features when choosing the PDU for the CompactRIO I/O module test system.

## Emergency Power-Off Button

It is always advisable to have a way to easily and quickly disconnect power to your system. With an emergency power-off (EPO) button on your test system, you can power down your entire test system from a single point by activating a push button.

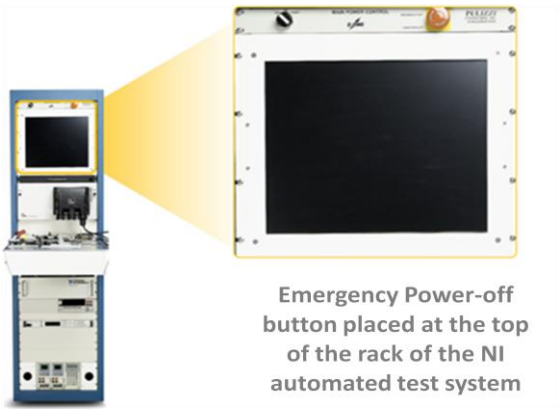



*Figure 6.* Emergency Power-Off Button on an NI Automated Test System

In some test systems, the primary method for turning off power is through a power strip that sits at the back of the test rack. This requires the operator to reach behind the test rack to turn off power. This is a task that can result in dangerous situations. While there are many different scenarios where cutting power to the test station may be important, one common scenario is when a test station goes into an unknown state while sourcing a high-power signal to the DUT. In such a case, you may need to cut off the power supply immediately to prevent damaging components. Additionally, you may not want your operator reaching inside the test station, where there may be open wires carrying high-power signals that could cause injury. An EPO button provides a secure and fast way for the operator to cut power to the test station.

## Circuit Breakers

Fitting your PDU with circuit breakers that protect specific load segments is another best practice. This technique ensures that an overloaded circuit does not affect other segments of the test system and thus increases system reliability.

## Sequential versus Parallel Power-Up Options

Unless you absolutely need to power all of the components in your test system at once, you should power your instruments sequentially. Many measurement instruments draw more power during startup than they do during operation. By turning on all of the instruments at once, you may draw too much power and blow a fuse.

It is also fairly common in test systems to use multiple PXI chassis, one of which is the master chassis that houses the embedded controller and the rest of which are the slave chassis that interface with the main chassis using a MXI interface. In these scenarios, it is important to turn on the slave chassis first because if you turn on the master chassis first, it may fail to recognize slave chassis components that are connected to it and cause the test program to throw errors.

*Figure 7.*PXI Chassis Daisy Chained Together

## Designing for Multiple Continents

Different countries have different voltage standards. For example, in the United States, AC voltage from the wall outlet is supplied at 110 to 120 V and at 60 Hz. In Europe, electricity from the wall is supplied at 220 to 230 V and at 50 Hz. The specifications are different for Asia as well. Furthermore, the plug adapters also vary by country.


*Figure 8.* Different continents and countries use different plug adapters.

If you expect your system to be used on different continents, you must ensure that it supports the different plug configurations, voltage levels, and frequencies in the countries of use.

Designing for use on different continents was an important consideration for NI test engineers because the CompactRIO module test system was built in North America for use on the manufacturing floor in the NI Hungary manufacturing facility. Consequently, test engineers chose a PDU that enabled them to select between 120V~Single Phase 50/60 Hz (North American rating) and 240V~Single Phase 50/60 Hz (European rating) using a voltage switch.

# Step 2.2: Switching, Mass Interconnect, and Fixturing Considerations

In addition to choosing a rack enclosure and power distribution unit, you need to determine how your DUT connects to the measurement instrument. The process for connecting your DUT to your instrumentation involves determining your switching architecture, choosing an appropriate mass interconnect system, and designing your test fixture.

It is also important that you analyze each of these three system components in the right order. Only after you determine the number of switches you need and the location where they will reside in the test system can you choose a suitable mass interconnect system and design an appropriate fixture that seamlessly mates your DUT to the rest of the system.

## 2.2.1 Switching

The first thing you need to consider is choosing your switching architecture. Adding switching or some form of signal routing mechanism to your test system is often necessary to expand the channel counts of the instruments or provide the necessary flexibility of measurements for optimal equipment usage and test throughput.

There are four main switching architectures:

1. No switching
2. Switching in test rack only
3. Switching in test fixture only
4. Switching in test station and test fixture

The following table outlines the strengths and weaknesses of all four architectures.

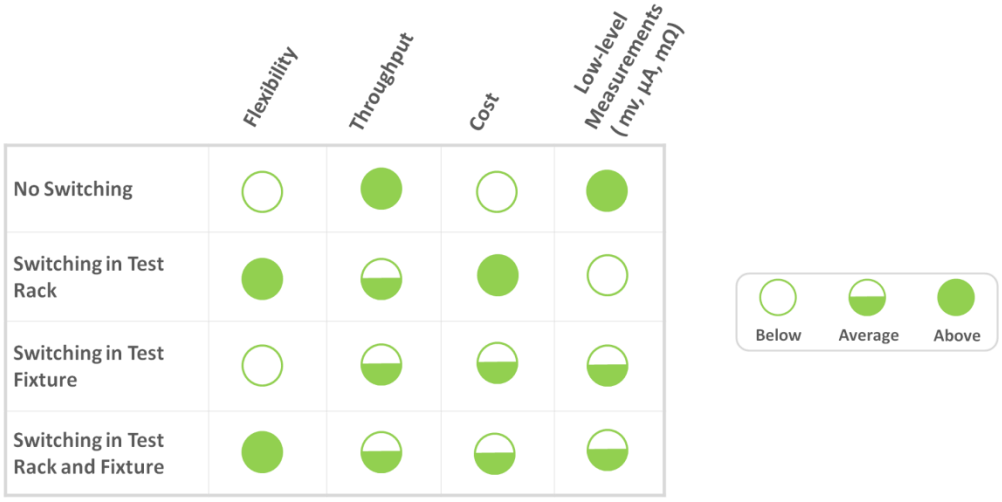



*Table 1.* Pros and Cons of Various Switching Architectures

This section describes in detail the pros and cons of each architecture as well as which architecture to use when.

## No Switching

In the first architecture, no switches are used to route signals between the DUT and the instruments in the test system. Such systems typically have a single channel on an instrument dedicated to every test point.

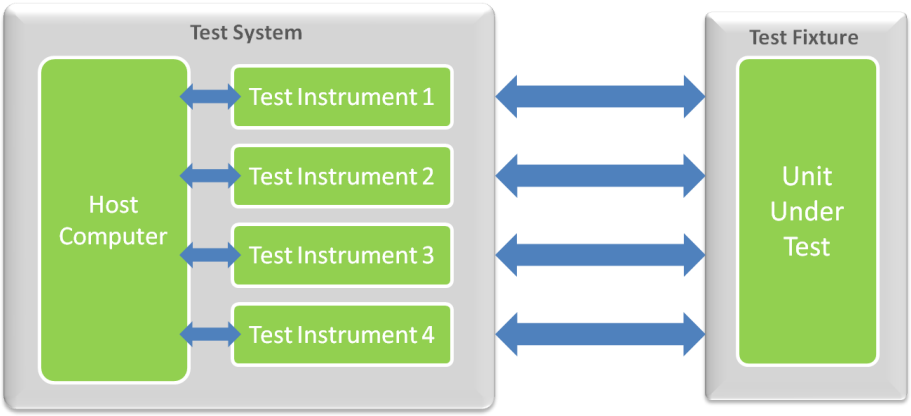



*Figure 1.* Test System without Switching

There are both advantages and disadvantages to using this architecture.

### *Advantages*

Cabling and switches can often degrade the integrity of your signal. By not using switches, you can provide your signal with a more direct path to the measurement instrument and thus improve your measurement accuracy.

In addition to improved measurement accuracy, you can achieve faster speeds. By having a dedicated instrument for each test point, you can make parallel rather and sequential measurements and thus increase throughput.

### *Disadvantages*

Having a dedicated instrument for each test point can prove to be extremely costly. Another disadvantage is expandability. You can very easily run out of space in your test rack if you build a test system without switching. This can cause you to completely redesign your test system, which may result in additional costs.

For instance, suppose that you have a test system that tests 20 thermocouples in parallel using 20 NI PXI-4071 DMMs. Now assume that your system needs to expand to test 40 thermocouples. To do so, you need to add 20 DMMs, which require 20 PXI slots. Alternatively, you can use a single PXI-4071 7½-digit DMM along with an NI PXI-2575 96-channel two-wire multiplexer to test all 40 thermocouples sequentially. The second solution uses only two slots.

## When to Build Systems without Switching

Building a test system without any switching is usually recommended if you are making either extremely sensitive measurements that would get distorted with the addition of cabling and switches or if you need to keep test time to a minimum. For instance, some semiconductor test applications have a single parametric measure unit dedicated to every pin on a chip (see Figure 2). This is because semiconductors are a high-volume business and test costs often make up a significant portion of the total manufacturing cost of a chip. You can significantly reduce test costs by minimizing test time. Additionally, in the semiconductor industry, testers are often built for specific chipsets or chipset families, so they are not usually expanded over their lifetimes.

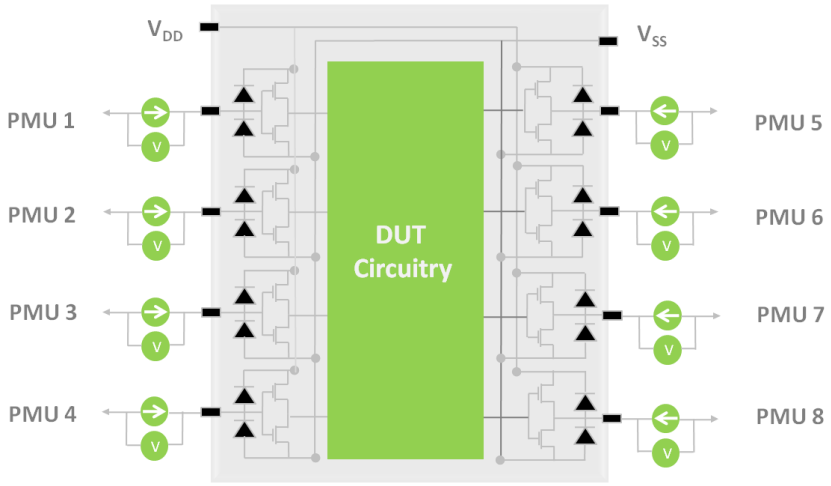



*Figure 2.* Some semiconductor applications use dedicated instruments to test each pin on the chip.

## Switching in Test Rack Only

The second architecture uses only commercial off-the-shelf (COTS) switches to route signals between the measurement instrument and the DUT.

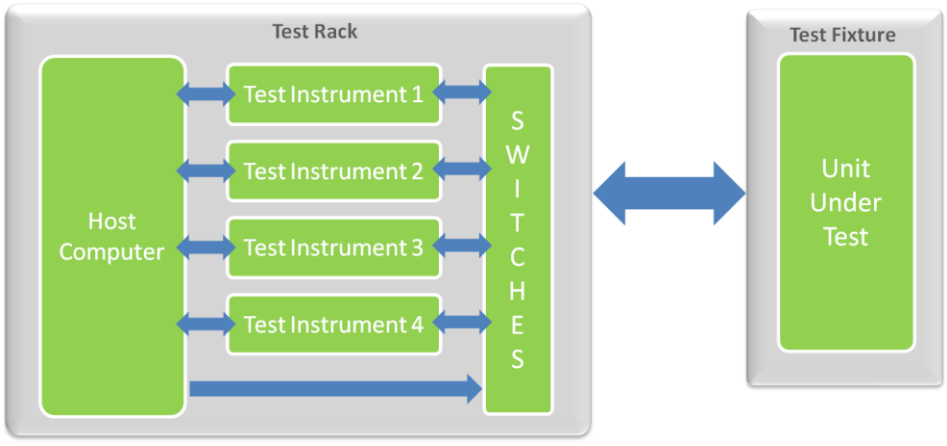



*Figure 3.* Test System with Switching in Test Rack

It is important to choose a COTS switching platform that can offer a broad range of functionality. Not doing so can result in higher expenditures due to test system redesign over the long run.

The PXI platform, for instance, offers more than 500 different types of switch modules that can route signals as high as 600V, 40A, and 26.5 GHz. NI alone makes 50 different switch modules that you can configure in more than 150 different topologies.

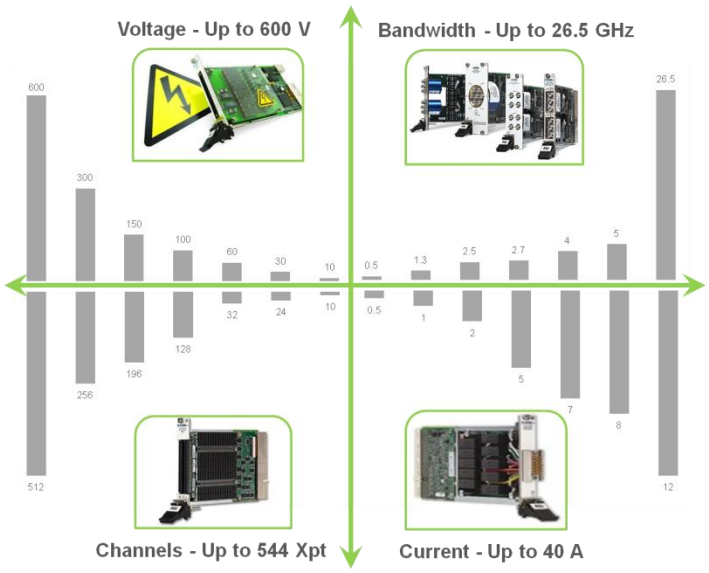



*Figure 4.* NI offers a wide range of PXI-based switch modules.

Using this architecture has several advantages and disadvantages, both of which are discussed below.

### *Advantages*

By using COTS switching solutions, you can save considerably on development time. Additionally, COTS switching improves test system scalability because you can now add switching by purchasing additional modules from a switch vendor rather than redesigning your entire test fixture.

Furthermore, each vendor provides solutions with their own unique advantages. NI switches, for example, have an onboard EEPROM that keeps track of the number of instances each time a relay on the module is actuated. With this feature, you can predict when a specific relay will reach the end of its mechanical lifetime and thus conduct predictive maintenance. The feature is especially useful when maintaining high-channel-count switch systems, which can be extremely time-consuming to debug manually.

You also can use NI modules to increase the throughput of your switch/DMM application by downloading a list of connections to the switch modules and cycling through the list using an event (trigger) without any interruption from the host processor.

### *Disadvantages*

Using switches can often slow down your test process because it requires you to take measurements from your test points on the DUT sequentially rather than in parallel, which is the case if you use dedicated instrumentation.

Placing all of the test system switching in the test rack also increases the total amount of cabling. In addition to using cables between the switch and the measurement instrument, you need cables between the DUT and the switch. This can cause error in sensitive measurements such as leakage current or low resistance measurements.

Sometimes it can be time-consuming and expensive to expand your switching system if it is in the rack. For instance, if you have 50 identical test stations located in different countries, you may have to spend a considerable amount of time upgrading the cabling and software in each of those stations. You may also have to update your documentation and localize it in a variety of languages.

## Switching in Test Fixture Only

The third architecture is to build switches into your test fixture. In this case, signals from the measurement instrument are switched to various test points on your DUT using individual relays placed on a printed circuit board (PCB) near the fixture or in the fixture itself. If you are using this architecture, you need a relay driver in your test station to control the individual relays using your test program. Alternatively, you can design an external circuit to drive your relays, but this requires some additional design work.

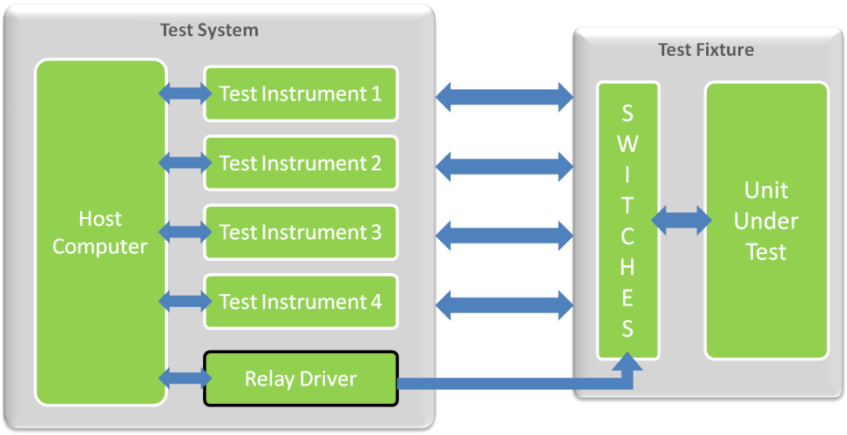



*Figure 5.* Test System with Switches in the Test Fixture

There are both advantages and disadvantages to using this architecture.

### *Advantages*

Building switching into your test fixture eliminates the need for cables between the DUT and the switches themselves. This helps reduce measurement error. Additionally, as mentioned before, you can reduce your test costs by using switching.

### *Disadvantages*

As discussed previously, using switches can often slow down your test process. Additionally, building custom switching into the test fixture requires PCB design expertise, so this is not an option for everyone. Finally, because this architecture often requires building custom components, scaling your switching PCB to accommodate more test points may not be easy. Thus, it is also not the best option if you expect your test system to change over time.

Another disadvantage of this architecture is the cost associated with designing a custom board for specific safety and compliance standards. If you are testing a high-voltage device, you likely need to build a switching fixture that adheres to various regulations such as UL, CE, and VDE. It can be challenging to design a PCB filled with relays that complies with the creepage and clearance requirements of these various standards. In such cases, using COTS switches can help reduce costs. Many COTS vendors certify their modules to comply with a wide range of standards. For instance, all NI modules that have a voltage rating greater than 60VDC or 30VAC and 42.2$V_p$ are considered high-voltage modules. These modules are built to adhere to the following standards:

*Figure 6.* NI switch modules meet a wide range of safety and compliance standards.

## Switching in Test Fixture and Test Station

The last architecture features switching in the test station as well as the test fixture. Using this paradigm, you can leverage the benefits of both COTS switching solutions and simultaneously minimize errors in sensitive measurements by placing switches closer to the DUT and in the test fixture.

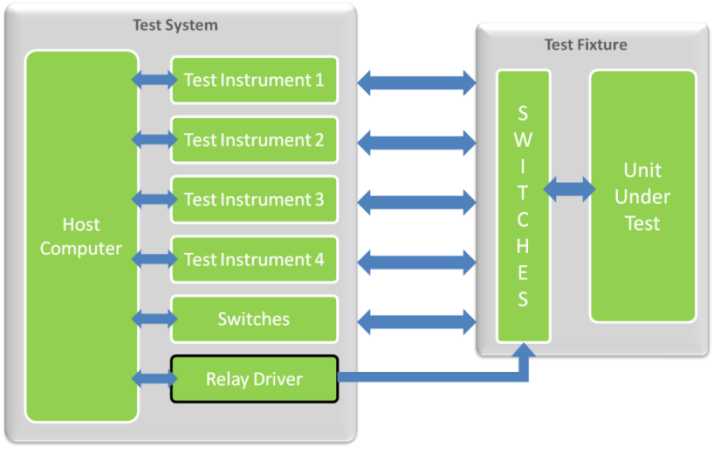



*Figure 7.* Test System with Switches in the Test Rack and Fixture

This architecture provides the following advantages and disadvantages.

### *Advantages*

By placing switches in both your test rack as well as your test fixture, you can build a switching system that can be scaled easily and that adds minimum errors into your measurement. Using this architecture, you can place those switches that are being used to route sensitive signals in the test fixture, and all the remaining switches, in your test rack. In addition to scalability, using COTS switching helps you take advantage of vendor-specific features such as the relay count tracking feature in NI PXI switch modules.

### *Disadvantages*

Using switches can often slow down your test process because it requires you to take measurements from your test points on the DUT sequentially rather than in parallel. Building custom switching into your test fixture can also be time-consuming and require a considerable amount of PCB design expertise.

## Real-World Example

Most NI test systems feature switches in the test fixture and test station. Test engineers can use this model to build test systems that have high flexibility, low cost, and low measurement error. This is important for many applications such as calibrating the NI 9219.

The N 9219 is an analog input module that can take very low-level measurements. The following is an example circuit that is used to calibrate the NI 9219 in full bridge mode. This module has two ranges in the full-bridge mode: ±62.5 mV/V and ±7.8 mV/V. To test both of these ranges, a full-bridge is simulated using three resistors, with resistor R being the load resistance of the DUT. An excitation voltage is supplied across the resistor network and the voltage drop across R is measured by the ADC. Next the two terminals of resistor R are switched to test the negative voltage range on the NI 9219. It is also important to note that no current flows through the ADC because it is a high-impedance circuit. The measurement taken is a ratio of the voltage drop across resistor R and the excitation voltage provided by the NI 9219.

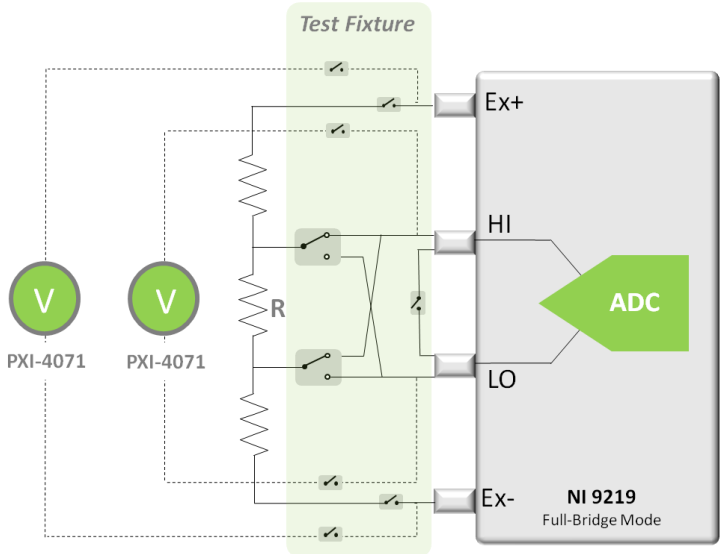



*Figure 8.* DMMs and loads are routed to the NI 9219 using switches in the test fixture.

Calibrating this device is a three-step process. First, the three resistors shown in the circuit in Figure 8 are switched in to the NI 9219. Next, two PXI-4071 DMMs are used to simultaneously measure the accuracy of the excitation supplied by the NI 9219 as well as the voltage seen by the device. Finally, the difference between the measurements made by the DMMs and the NI 9219 is entered as calibration values on the EEPROM of the NI 9219.

The switching for this procedure is fairly complicated. Because the measurements being made can be less than 10 mV, the test setup must be fairly noise-free. As a result, cable lengths must be kept to a minimum. To ensure minimum noise, switching on the test fixture is used to connect the DMMs and the loads to the NI 9219.

Unlike the NI 9219, most other CompactRIO modules that are tested by the system are not designed for measuring low-level signals. For these DUTs, it is more cost-effective to use COTS switches in the test station than to build custom switching into the test fixture.

## 2.2.2 Mass Interconnect

*(Content and images are courtesy of Virginia Panel Corporation.)*

A mass interconnect system is a mechanical fixture designed to easily facilitate the connection of a large number of signals either coming from or going to a DUT. For automated test systems, this usually entails some mechanical enclosure through which all signals are routed from instruments (typically in a rack) to the DUT, making it easy to quickly change DUTs or to protect the cable connections on the front of the instruments from repeated connect/disconnect cycles.

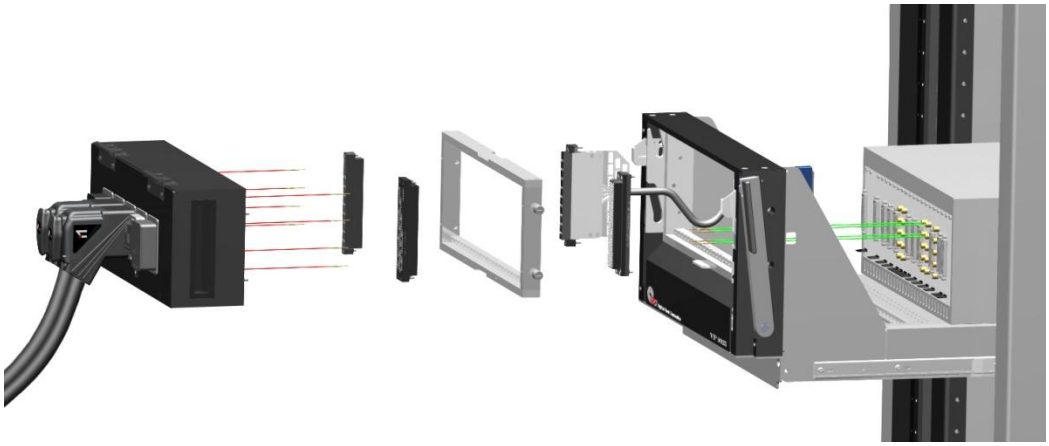


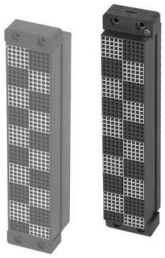
*Figure 9.* Typical Mass Interconnect System

### Components of a Mass Interconnect System

*Receiver:* The receiver is the “instrument side” of the mass interconnect system. It includes housing connector modules that access each of the instrument, switch, power, or vacuum connections.

*Mounting Hardware*: Mounting hardware holds the receiver on the front of the rack. It is typically mounted on the front side of the rack with easy access to the test operator. Oftentimes, the mounting hardware has a hinge to allow easy access to the instruments or cables behind the receiver.

*Receiver Modules:* The receiver modules expose the instrument, power, or vacuum connections through the appropriate pins or other connectors inside the receiver. These are typically specified according to the density, bandwidth, current, or special-purpose requirement of the signals being passed through them.



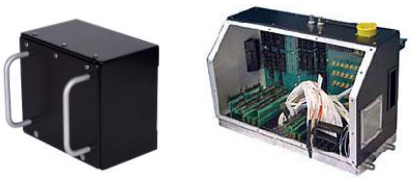
*Cable Assemblies/Interface Boards:* Cable assemblies/interface boards offer a means of connecting the front of the test instruments to the receiver modules. With cable assemblies, standard cables are connected from the instruments either directly to pins in the receiver modules or to interface boards connected to the receiver modules.

Interface Boards: There are also interface boards that connect directly from mass connectors (DIN, D-Sub, SCSI, and so on) on the front of the instruments to the receiver modules. These offer the advantage of minimizing the signal length between the receiver module and the instrument but are rigid and therefore require precise, inflexible mounting locations.

*Interchangeable Test Adapter (ITA):* The ITA is the main mechanical mate to the receiver. It is the “DUT side” interface that houses the modules/cables that interface to the connections on the DUT. In many test systems, one receiver system can be connected to several different ITAs, with each ITA having a different enclosure, cables, and DUT attached to it for rapid modification of test system requirements.

*ITA Cables/Modules:* The ITA modules are mounted inside the ITA in much the same way the receiver modules are mounted inside the receiver. They provide the main interconnect with the different signals coming through the receiver and expose those connections through cables, PCBs, or other connections inside the enclosure.

*Enclosure:* The enclosure is the mechanical housing around the ITA and the corresponding ITA cables/modules. It is often a physical platform on which to set the DUT, as in a bed-of-nails application.



Umbilical Cables: Umbilical cables are used to connect the ITA enclosure to the DUT. They typically have a high-cycle life connector on the ITA enclosure end of the cable and a mating connector on the other end that is compatible with the connector mounted to the DUT.

With these components and the custom test fixture (see next section), the test operator can connect all of the test points in the system to the DUT in a quick, orderly, and repeatable fashion.

## Choosing Your Mass Interconnect System

Follow the below steps to determine which components are necessary to complete your system based on your I/O requirements. The steps shown here can help you properly choose the equipment that best suits your needs.

**Step 1. Determine the System's I/O Requirements:** To choose a suitable mass interconnect system, you must first determine the following about your system:

**A.** Number of signal points
**B.** Number of power points
**C.** Number of coaxial points
**D.** Number of other contacts
**E.** Cable assembly modules
**F.** Wired adapters

Choosing the signal, power, and coaxial points helps you determine the modules you need. Using cable assemblies also narrows down your module choices.

**Step 2. Select the Receiver:** Determine the size of the receiver by how many modules you need plus room for expansion. As a rule of thumb, 20 percent of the module positions should remain open for future growth.

**Step 3. Choose Receiver Accessories:** These include the receiver mounting panel, vertical hinged mounting frame (VHMF), plug and play mounting plate, receiver protective cover, slide kits, and chassis mount kits.

**Step 4. Determine the ITA:** Select the interchangeable test adapter (ITA) based on the size of the receiver you have chosen.

**Step 5. Select the ITA Modules and Contacts:** The modules for the ITA are the mating modules for the receiver modules you selected in Step 1. The ITA contacts mate to the receiver contacts and are dictated by your system's I/O requirements.

**Step 6. Determine the ITA Accessories:** These include the ITA protective cover, ITA patch cords, and the ITA enclosure.

### Real-World Example

Using some of the steps above and the help of representatives from the Virginia Panel Corporation (www.vpc.com), NI test engineers chose a suitable mass interconnect system for their CompactRIO I/O module test system.

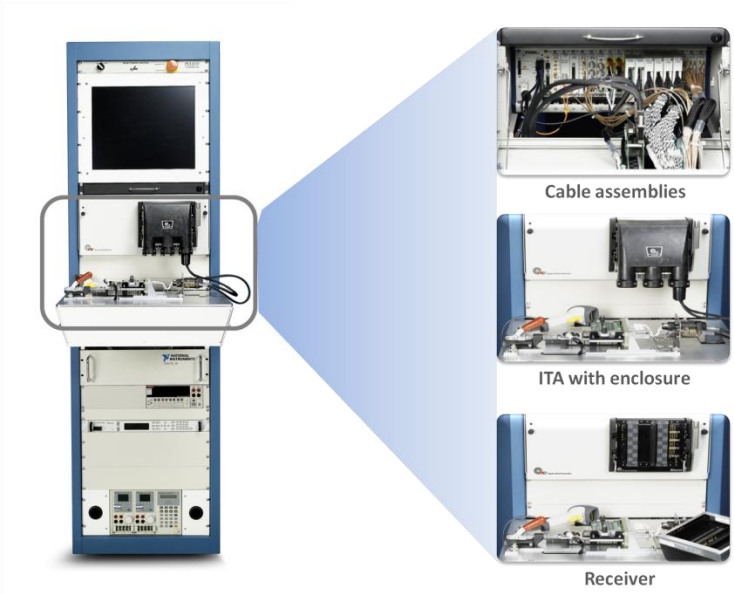



*Figure10.* Mass Interconnect System on the NI Automated Test System

In addition to Virginia Panel Corporation, there are several vendors, such as Mac Panel Corporation (www.macpanel.com), that provide mass interconnect products and solutions for the PXI platform. It is important that you research solutions from these and other vendors so you can choose the solution that fits your test needs best.

## 2.2.3 Fixturing

A test fixture is a device that provides an interface between the test station and the DUT. Fixtures are often custom designed to meet the needs of the specific product that is being tested. This section covers some best practices to help you design suitable test fixtures for your system.

### Understand the Fixture Usage

When designing a test fixture, you need to know how it will be used because the needs for design, verification and validation, and production are different.

**Production**

If a fixture breaks in production, it can cause significant downtime, which results in lost production output and increased test cost. As a result, for production it is important to build fixtures that are rugged.

In addition to being rugged, the fixture must be easy to use and ergonomic. Specifically, you must try to build a fixture that requires minimal interaction from the operator. Using a fixture where the operator has to spend considerable time aligning the DUT to the fixture or connecting cables from the instruments to the fixture can seriously impact throughput and increase test costs. It can also create more room for errors in your test process.

Finally, for production, you must build a fixture that can easily be duplicated and deployed in new test stations.

The following is an example of a test fixture for production test of a specific CompactRIO module.

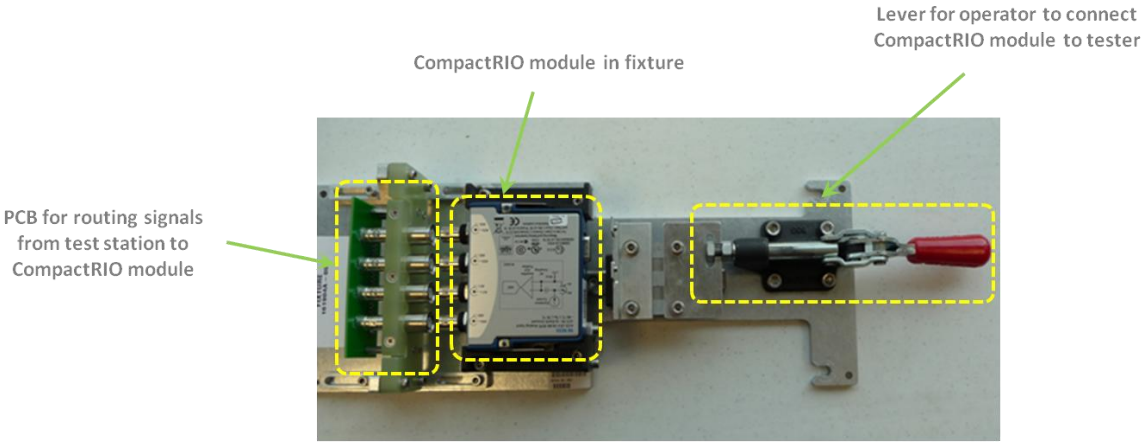



*Figure 11.* Test Fixture Used to Connect CompactRIO Module to the Test Station

This fixture requires minimal interaction from the operator. To test this module, the operator simply has to insert the module into the correct place on the fixture and push a lever to make a connection between the module and the test station. This setup is both fast and simple.

**Verification and Validation (V&V)**

V&V is the process of checking whether a specific product meets design specification by testing a statistically representative set of the product. The test data collected during the V&V process is also used to determine the test limits for the production test system. While only a small set of products undergoes V&V, almost all products that are sold on the market pass through production test. As a result, fixtures designed for V&V are required to make fewer connections between the DUT and the test instruments. Fixtures built for V&V can therefore be less rugged than those built for production.

**Design**

During the design phase of a product, the hardware engineer must have the flexibility to connect and disconnect one or more cables or pins from the DUT to the measurement instrument to test and troubleshoot all elements of the design. It is more important for a test fixture in the design phase to be highly flexible rather than streamlined or rugged.

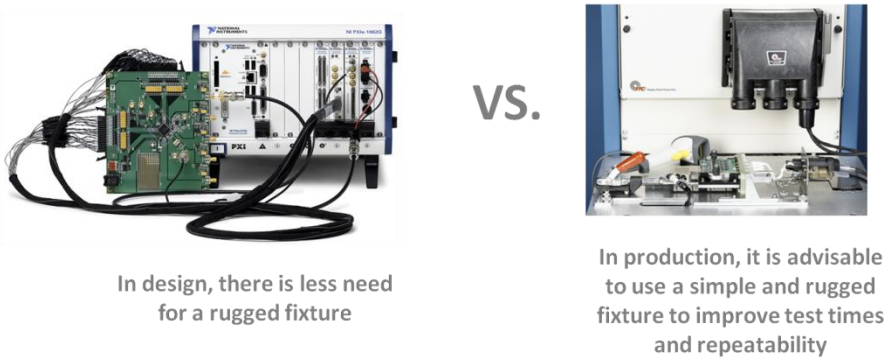



*Figure 12.* Fixtures for debugging product designs have different requirements than the fixtures used in production.

## Use Proper Wiring

Wires can be sources of noise and error. Some wires offer more insulation than others, so they inject fewer errors into measurements. In addition to choosing the right wires to connect your instrument to the mass interconnect receiver, it is important to use appropriate wiring in your custom fixture. It is particularly important to use shielded twisted-pair wires to reduce the noise in your test fixture and leverage proper grounding techniques when making low-level measurements.

Some test instrument manuals offer specific suggestions for which wires to use. If this is the case, you need to follow these suggestions from the test station to the test fixture. For instance, the help file for the NI PXI-4070 6½-digit DMM recommends cables that have tape-wrapped polytetrafluoroethylene (PTFE) insulation to guarantee low leakage.

## Use PCBs for Interconnects on Production Fixtures

Some products require more complex test procedures than others. In the NI test system, for example, the NI 9219 analog input module must be connected to the Keithley source measure unit, two PXI-4071 7½-digit DMMs, a dynamic signal acquisition board, and several load resistors. Using wires to connect the NI 9219 to the instruments in the fixture is a cumbersome process because it requires a significant amount of manual labor on the part of the technician. The total labor needed increases exponentially for every additional test system built. Also, when using cables, the technician must ensure that all cables have the same length and gauge to minimize differences in measurements from one test point to another. This can be a challenging task.

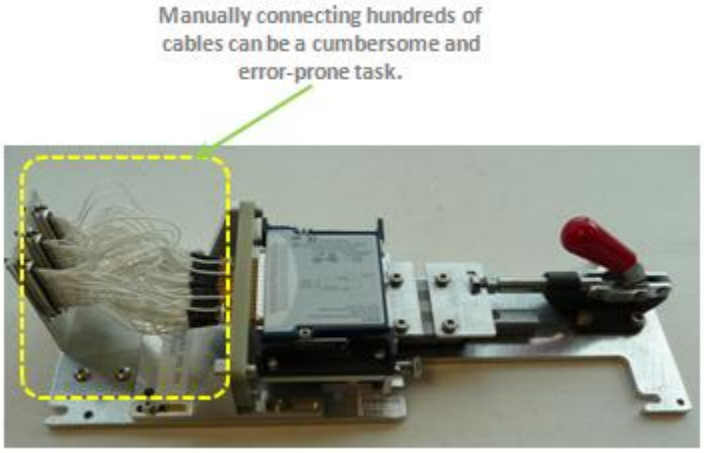



*Figure 13.*This test fixture uses cables to connect a CompactRIO module to instruments.

If you are going to build several duplicates of a given test fixture, it is beneficial to layout a PCB to route the signals from the ITA to the DUT on the fixture. While this may take some upfront investment, it can significantly cut down on manual labor costs and help to decrease error.

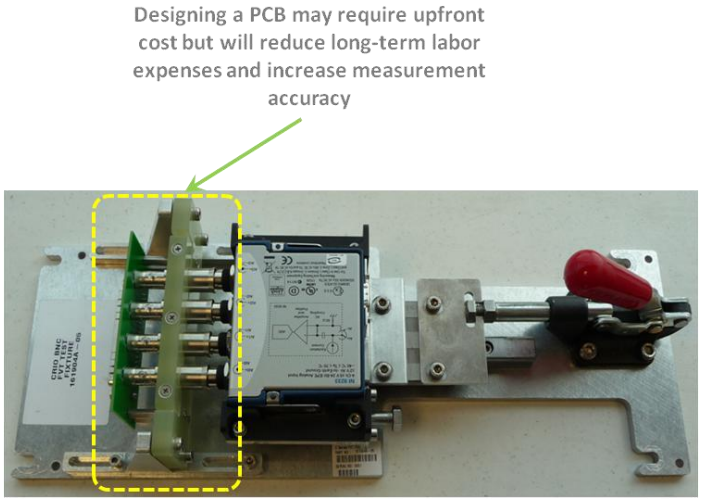



*Figure 14.*Test fixture that uses a PCB to connect CompactRIO module to instruments.

## Make as Many Connections as Possible

You need to build a fixture that minimizes operator interaction by making as many connections as possible. Building a fixture that requires the operator to make additional cable connections can result in operator errors and additional costs due to increased test times.

### Create a Preventive Maintenance Plan for the Test Fixture

You must also ensure that you have a maintenance plan in place for your test fixture. This plan should include inspection and/or replacement of connectors, cables, pogo pins, relays, and other related components at regular intervals of the test station's lifetime. You can also base your intervals on usage.

## 2.2.4 Next Steps

After choosing your rack, PDU, instrumentation, switching, mass interconnect, and fixturing, you should compile your findings in the form of a bill of materials (BOM). This makes the process of ordering your parts and submitting budget requests simpler. It also makes it simpler to replicate the test system. In addition to compiling a BOM, you must document how all of your system components connect to each other. You can achieve this by creating a series of system schematic diagrams, as illustrated in Figure 15.

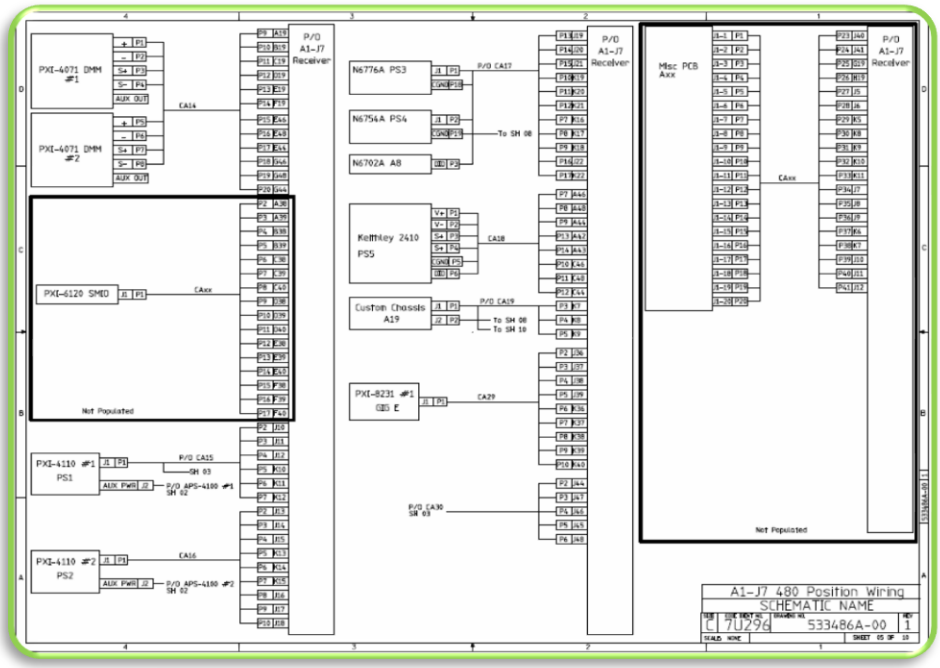



*Figure 15.* Example of a Test System Reference Schematic

# Step 3: Best Practices for Designing Software

# Step 3.1: Test Executive Best Practices

The role of software is intrinsically important in designing and building a test system because the top three layers of the recommended test architecture are software-oriented, as shown in Figure 1.For more background on the five-layer architecture, refer to the executive summary of the Software-Defined Test Fundamentals guide**.**

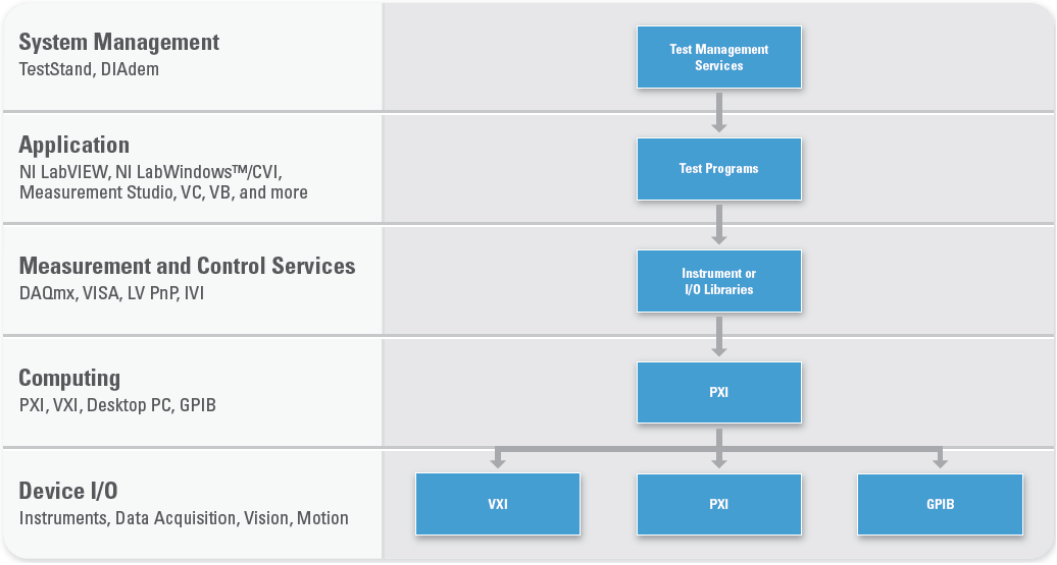



*Figure 1.* Five-Layer System Architecture for Developing Test Systems

This section describes the system management/test executive layer and offers best practices for using a test executive within your organization. The following bullets define the role of a test executive and how it interacts with the other two software layers.

- **Test Executive** –Use this to run a series of tests (for example, code modules) quickly and consistently in a predefined progression. Operations that are common for all devices, such as reporting and serial number entry, should be handled by the test executive, and DUT-specific operations should be developed in code modules. By allowing the test executive to handle the common operations, you are saving time for your developers because they do not have to write the same code for multiple devices.
- **Code Modules**–Implement tasks such as measurement I/O, high-speed looping, and analysis in code modules. Application development environments (ADEs) are used to develop code modules, so you should emphasize developing highly modular and reusable code modules. For more information, read the chapter on **Code Module Development Guidelines**.
- **Instrument Driver**–Control a programmable instrument with this set of software routines. 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 test program development time by eliminating the need to learn the programming protocol for each

instrument. For more information on hardware drivers, read the chapter on **Choosing Your Instrument Driver Paradigm**.

Test engineering teams that have a large number of test systems often implement a test framework to achieve another level of commonality. For example, some tasks/operations are specific to a single DUT, while others are repeated for every device tested on a test station and still others are repeated for every test station enterprise-wide. The following list provides a more detailed breakdown of the operations performed at each level:

| DUT-Specific | Test Station-Specific | Enterprise-Specific |
|---|---|---|
| • Instrument configuration | • Instruments selection | • User management |
| • Test limits | • Rack configuration | • DUT identification |
| • Results analysis | • Operator interfaces | • Test flow control |
| • Test code | • Variables/parameters | • Reporting and data storage |

You can use these guidelines to define the roles and responsibilities of your test engineering development team members. For example, the NI test engineering team has defined three primary roles: framework developer, test station architect, and test developer.

- **Framework Developer—** Defines and creates the software framework for an entire company or within a business segment. The framework defines all of the common elements that every test station can leverage including user management, serial number scanning, database integration, and reporting. Framework developers must have extensive knowledge of the technology behind the test framework and sufficient proficiency in one or more development languages to write operator interfaces and tools. They often need to know other technologies or applications, such as databases or analysis tools. Framework developers need only minimal knowledge of the DUT because test developers implement device-specific code. However, they must have a broad enough understanding of the DUTs to identify appropriate features for the test framework.
- **Test Station Architect —** Defines the common test station for a product line based on the five-layer architecture (see the executive summary of the Software-Defined Test Fundamentals guide). Test station architects should have a good understanding of the test framework. They are also responsible for defining all of the test station hardware elements, including choosing the modular instrumentation based on the measurement requirements of the product line, power distribution, and racks. Test station architects are typically responsible for developing the deployment and maintenance strategy for the test station.
- **Test Developer —** Develops test sequences for individual DUTs. The developer creates the code modules to test individual aspects or components of a device and the logic to combine and sequence these individual elements into a cohesive test routine. Test developers must have the technical ability to create test routines and write test modules in one or more development languages. They should be experts on the specifications of the DUT and possess enough understanding of the test framework to write test routines that use the framework.

Figure 2 shows a diagram of how these three roles work together. Depending on the size of your company, these may be the same person. However, at a minimum, you should consider thinking of them as virtual roles because they can help you develop test systems efficiently.

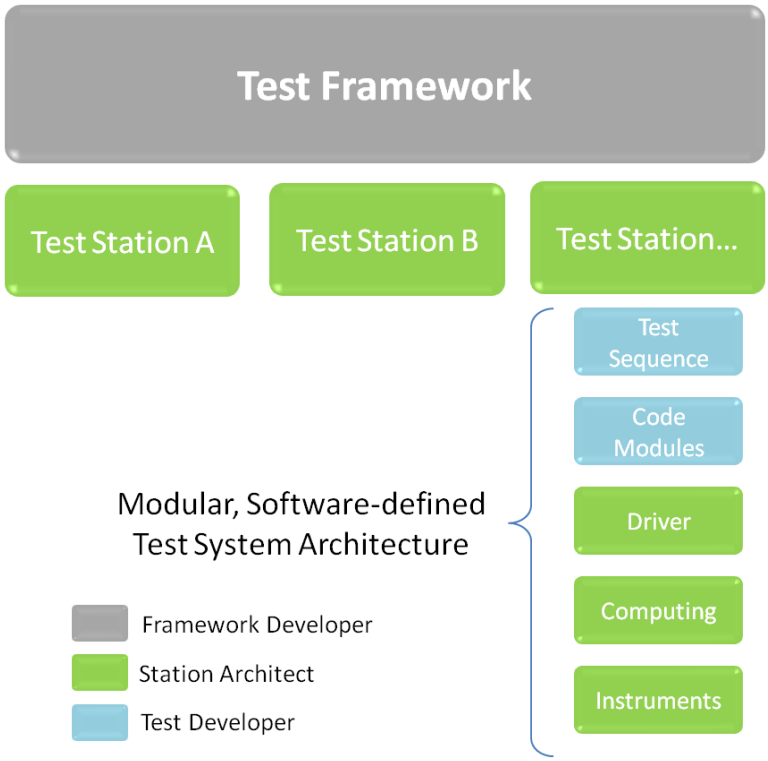



*Figure 2.* Typical Test Engineering Roles and Responsibilities

### 3.1.1 Test Executive Considerations

Some test engineering teams have developed an in-house test executive because previous off-the-shelf test executives did not adequately meet their wide variety of testing requirements. These in-house test executives direct many valuable resources away from the core competencies of the company and can be difficult to support and maintain. Today, the open software architecture of NI TestStand is a popular choice for many test engineering organizations.

NI TestStand is a ready-to-run test executive that is designed to deliver the cost benefits of a purchased product while maintaining the flexibility of a custom-developed solution. It is a high-speed, multithreading test engine that is capable of the parallel execution of test modules written in any test language, such as LabVIEW, LabWindows/CVI, Visual Basic, and Visual C++. The engine is designed to meet the most rigorous throughput requirements of high-volume production systems. In addition, with the NI TestStand sequence development environment, you can define advanced sequences with conditional looping, branching, and execution flow to handle the most complex testing scenarios. You also can customize NI TestStand to meet the needs of your organization. A framework developer, or a team of framework developers, can customize NI TestStand and then distribute it to test station

architects. For more information on developing a test framework, view the NI TestStand Advanced Architecture Series.

The following three sections describe the best practices for using a test executive.

## 3.1.2 Sequence Reuse/Sections

Consider reusing a sequence when there are few specification or functionality changes. You can use conditional statements to choose which test to run or limit to use when there are just a few changes. As the number of changes increases, you should use a second test sequence for clarity and simplicity. For example, NI engineers reuse a sequence file for a product featuring a subset of functionality from another product because they can typically skip steps within the main sequence.

### *Sequence Sections*

You should separate your sequence into three sections: Setup, Main, and Cleanup. The Main section, typically owned by the test developer for a particular DUT, contains the bulk of the steps in a sequence, including the steps that test the DUT, as shown in Figure 3. The test system architect is typically responsible for defining the test executive template for the product line test, which includes developing both the Setup and Cleanup portions of the test sequences. Because these portions of the test sequence are typically common for every DUT that runs on the test system, the test engineers responsible for each DUT can focus on developing code for testing the DUT and not configuring the tester.

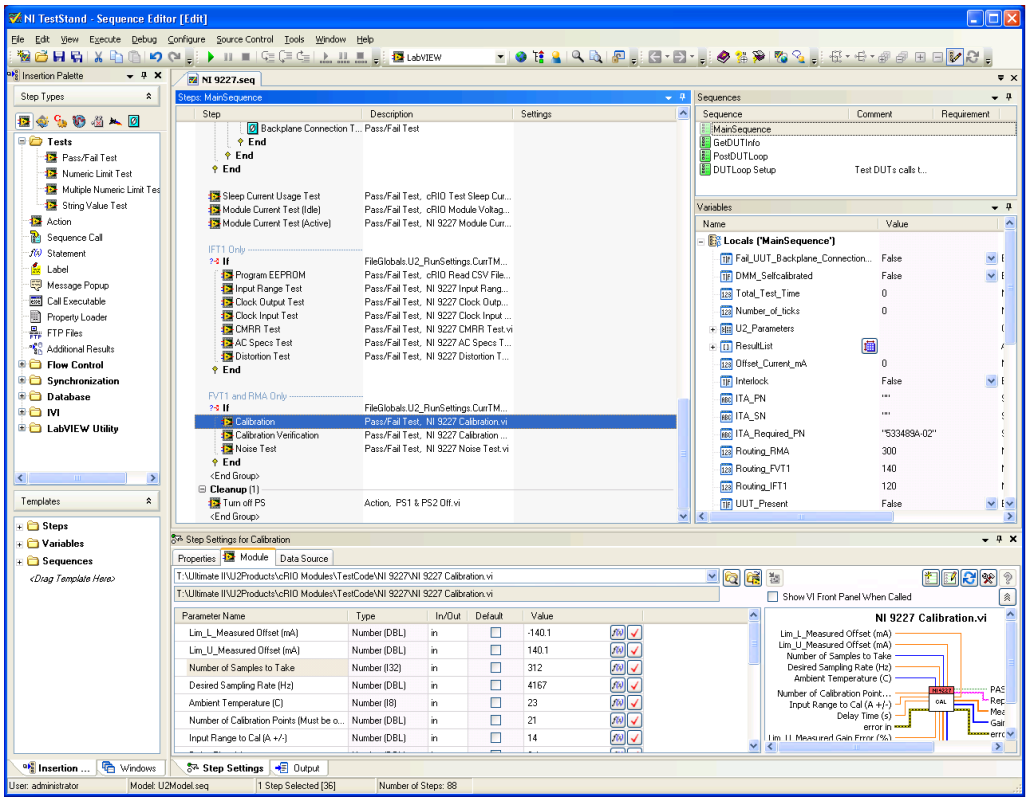



*Figure 3.* Screenshot of the Main Section of an NI TestStand Sequence

The Setup section of your sequence usually contains code modules that initialize instruments, self-calibrates instruments (if required), and checks that the proper ITA, fixture, and DUT are connected. Figure 4 is a screenshot of the setup procedure. In this example, the DMM is self-calibrating and the serial number of the ITA is being read.

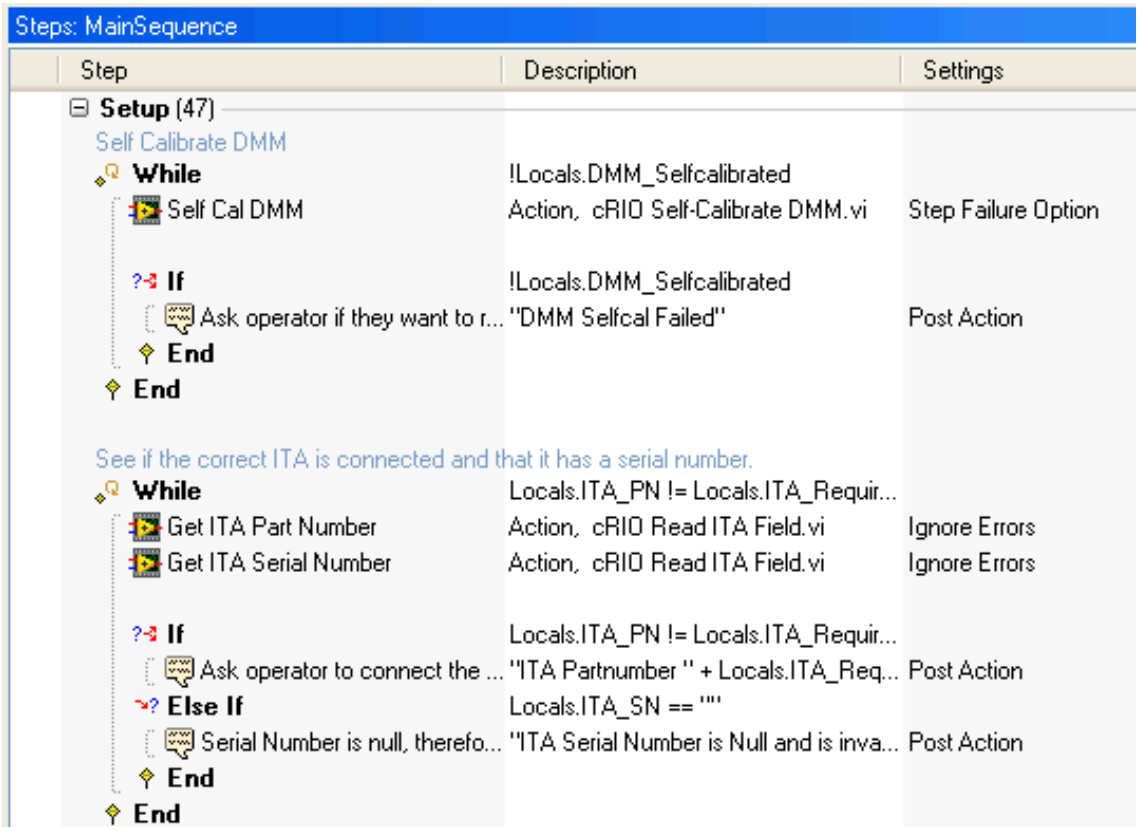



*Figure 4.*A Partial Screenshot of a Setup Sequence in NI TestStand

The Cleanup section typically contains steps that power down or restore the initial state of instruments, fixtures, and the DUT. It is good practice to leave the instruments in a known state. Hence, the Cleanup sequence should execute regardless of whether the main sequence completes successfully or a run-time error occurs in the main sequence. For example, when a step in the Setup or Main sequences generates a run-time error, the flow of execution should jump to the Cleanup group. The Cleanup steps always run even when some of the Setup steps do not run. If a Cleanup step causes a run-time error, execution continues to the next Cleanup step.

### 3.1.3 Operator Interface Guidelines

An operator interface is an application that provides a graphical user interface (GUI) for executing sequences on a production station. Operator interfaces should be as light as possible – avoid putting test management or operation code into the operator interface. They should also have a consistent look and feel across product lines to minimize training effort. The following are eight guidelines you should consider when designing your operator interfaces.

1. ***Hide unnecessary menus and toolbar buttons.***

When creating a user interface, you should limit the user interaction to only necessary functions. Hide all unnecessary or potentially problematic buttons and toolbars. For example, you do not want an operator to close a pop-up window in the middle of running a test sequence. This could cause a false failure and waste valuable test time. In this case, hide the window close button.

2. ***Use progress bars to communicate the length of an operation.***

Progress bars are useful to give the operator an idea of how long an operation takes. Use them only with operations that take a significant amount of time. Overuse of progress bars decreases their impact and causes user frustration. Also be sure to link progress bars to a useful measure such as percent completion or the amount of time left for an operation. Inform the user of which unit you are using as the measure of progression and the goal of the progression.

For example, if you have a bar indicating percentage complete based on time, than the name should indicate that the bar shows percent complete, and the time remaining should be shown. This allows operators to use their time wisely while waiting for an execution to complete.

3. ***Make pop-ups and UI windows modal.***

A modal window stays on top of all other windows on the screen. It is useful for user interfaces because it forces users to address an issue immediately and does not allow them to perform other tasks. The modal behavior of a window is set in the window appearance dialog box of the VI properties and in NI TestStand through an API call. Figure 5shows a screenshot of a modal dialog box.

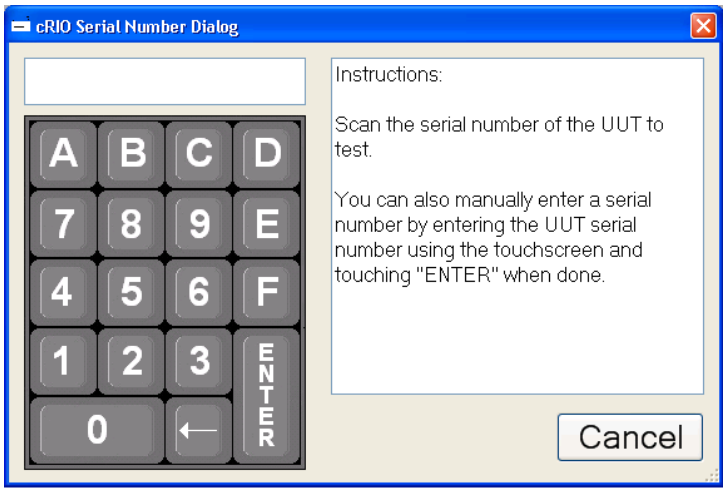



*Figure 5.* Example of a Modal Dialog Box

4. ***Dynamically manipulate front panel controls.***

Provide the user with more feedback on the front panel by dynamically manipulating front panel controls. You can achieve this in LabVIEW using property nodes. The following examples show how to effectively use these controls.

- *Initialize values in pop-ups*–Clear controls waiting for inputs from operators so they are ready for inputs. Reinitialize indicators to default values to avoid confusion. For example, not refreshing a graph to the default value can give the false impression that data was collected.
- *Key focus*– Ensure that when the user enters data via the keyboard or a scanner that the data goes into the proper control.
- *Hide unused controls/indicators*– Avoid confusing the operators by displaying too many indicators or controls. Change viewable content in real time if the pop-up is used for multiple user interactions.
- *Modify window properties* – Change the window size, the window position, or the background.

### 5. *Enhance the front panel with graphics.*

Graphics such as connection diagrams, schematics, and setup pictures make pop-ups more informative and useful. They can clearly indicate concepts that are sometimes difficult to explain with words. They also demonstrate behaviors in a more concise way. Because pictures are language-independent, they work well with localized code.

Adding a graphic to text can augment the message but only if done properly. You should use a focused picture with an appropriate resolution and size for the window and monitor on which it is displayed. It should include only relevant parts. Figure 6shows a screenshot of a dialog that pops up if operators using the NI manufacturing test station do not make the proper connections to the DUT.

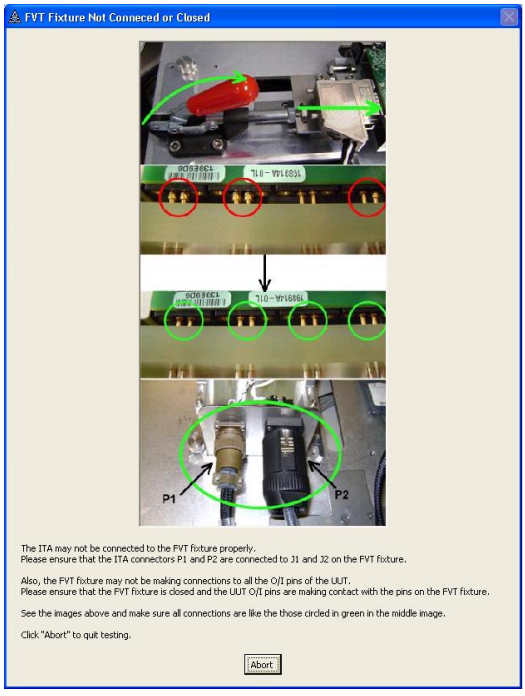



*Figure 6.* An Example of an Operator Dialog with Graphics

6. ***Set up key navigation and keyboard.***
   Key navigation gives operators the option of using the keyboard to enter values or move key focus rather than using a mouse. For key navigation to succeed, you should clearly note the shortcut keys on the front panel itself.

   Keyboard shortcuts are also used for debugging purposes. For example, you can use a keyboard shortcut to access a stop button not visible on a pop-up window. This gives the developer options for debugging code later. The drawback is that operators could use the shortcut accidentally and cause false failures; hence, you should disable this feature when it is deployed to production.

7. ***Configure the front panel to fit on the screens of most users.***
   Be sure that the front panels for the operator interface can scale on the monitors used in manufacturing. You may want to note the desired screen size and resolution in your test code or in the test code documentation.

8. ***Localize front panel and pop-up code.***
   Localize front panel and pop-up windows for all manufacturing facilities that need to run the code. To make pop-ups useful across manufacturing facilities, avoid the use of written text, or create text versions for each language used. Using pictures helps avoid language barriers in understanding setup configurations. Remember, however, that not all images translate from language to language. For example, using a picture of a tree log to represent a test failure log does not translate correctly to someone who does not speak English. If you must use words in a pop-up picture, then consider having versions for each manufacturing facility. You can query the manufacturing location to determine which picture to use. In LabVIEW, you can use property nodes to change the text on front panels.

   NI TestStand provides the flexibility to easily define different languages. For instance, if an NI TestStand operator understands Spanish better than English, you should modify the shipping definitions for English to account for additional languages. NI TestStand stores all user-viewable strings in resource string *.ini files in <TestStand>\Components\NI\Language\English\.NI TestStand uses the definitions in these files to populate dialogs, menus, and even operator interface elements. Because NI TestStand stores these resource strings using tags and constant name variables, you can easily modify any of these values to be reflected in NI TestStand.

## 3.1.4 Defining Variables/Parameters

Variables are data storage constructs that you can use to store and share information. Variables can apply globally to a sequence file or locally to a particular sequence. You can create and use variables and properties and monitor the values of the variables and properties. How variables are used depends on your test executive. In NI TestStand, you can define variables in the following ways to share data among steps of a sequence or among several sequences:

- **Use local variables** to store data relevant to the execution of the sequence. Each step in the sequence can directly access local variables. In NI TestStand, you can pass local variables by

value or by reference to any step in the sequence that calls a subsequence or any step that calls a code module that uses the LabVIEW, LabWindows/CVI, C/C++ DLL, .NET, or ActiveX/COM adapter.

- **Use sequence file global variables** to store data relevant to the entire sequence file. Each sequence and step in the sequence file can directly access sequence file global variables.
- **Use station global variables** to maintain statistics or to represent the configuration of a test station. You can access station global variables from any sequence or step on the test station. Unlike with other variables, NI TestStand saves the values of station global variables from one NI TestStand session to the next. Limit the number of station global variables you use in your test sequence. These variables are locally stored on each computer, so editing or updating them within your sequence requires the station globals file to be saved and copied to all test stations. This can be problematic if you have numerous test stations; therefore, it is not recommended for use in manufacturing.

Use the Variables pane in the sequence editor to show and modify local variables, parameters, sequence file global variables, and station global variables. You can also use the Variables pane to show sequence context and run state information when executing sequences.

### *Standard Variable Naming Convention*

If your team has not already defined a standard naming convention, consider developing one at a test engineering team level. However, many organizations have seen the benefits of standardizing on a naming convention across the entire organization (R&D, Manufacturing, and Quality).For example, NI has a standard naming convention that is heavily used throughout all NI organizations to generate high-level quality and parametric reports from the test data stored in a data management database and in the manufacturing quality data database. The data mining tool helps mine the test data that was generated using the NI test framework during the production or V&V testing phase. The following is an example of an NI naming convention for variables of scalar and array parameter limits for a parameter named ABC. You can have the following limits:

| **Limit** | **Naming Convention** |
|---|---|
| Lower limit | Lim_L_ABC |
| Upper limit | Lim_U_ABC |
| Base | Lim_B_ABC |
| Absolute tolerance | Lim_TA_ABC |
| Percentual tolerance | Lim_TP_ABC |

Limits are defined as follows:

- Parameter > Lower limit
- Parameter < Upper limit

If your parameter must fall inside a range, you have three options:

| **Option** | **Example** | **Result** |
|---|---|---|
| Specify a lower limit and an upper limit. | Lim_L_<parameter> = -10<br>Lim_U_<parameter> = +5 | -10<Parameter<+5 |
| Specify an absolute tolerance. (*In this option, the BASE is optional. If a BASE limit is not found, BASE is assumed to be equal to zero (=0).* ) | Lim_B_<parameter> = 200<br>Lim_TA_<parameter> = 10 | 200-10<Parameter<200+10<br>190<Parameter<210 |
| Specify a base and a percentual tolerance. | Lim_B_<parameter>=200<br>Lim_TP_<parameter>=5 | (200 x (100-5))/100<Parameter<(200x(100 + 5))/100<br>190<Parameter<210 |

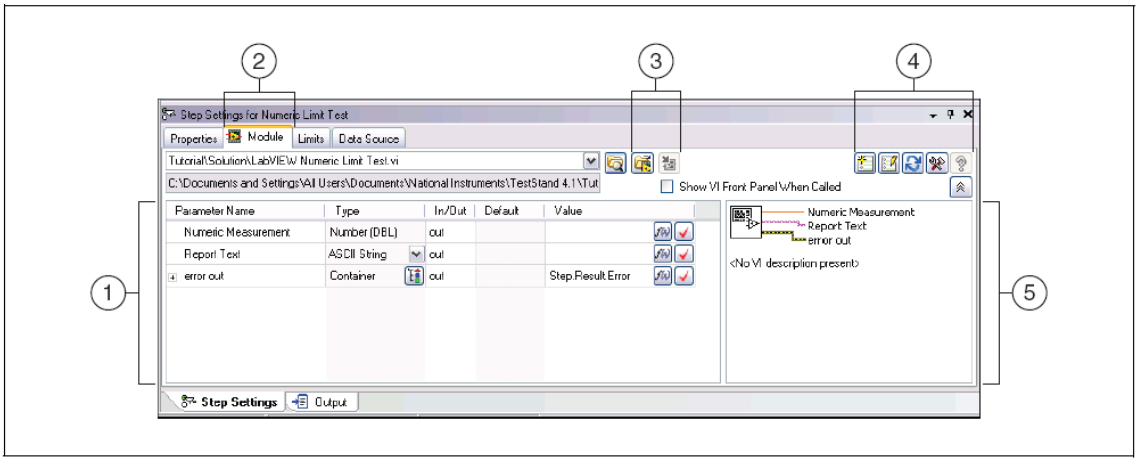
# Step 3.2: Code Module Development Guidelines

A test executive often gives you the flexibility to call many types of programming languages and pass data freely between the test executive and the code modules. This flexibility does not override the need for uniformity in calling and developing code modules. Regardless of which ADE you use, the following guidelines can help you create modular, maintainable test code.

## 3.2.1 Isolate Code Modules from Test Executive Operation

As a general rule, the test code should be as independent as possible from the test executive. Developing modular tests that perform a single measurement and perhaps some analysis leads to more readable and reusable code. To facilitate the separation of measurements from test executive operation, NI TestStand provides module adapters that you can use to pass data freely between the test executive and the code modules. Module adapters can load and call code modules, pass parameters to code modules, and return values and status information from code modules. The NI TestStand module adapters support code module types including LabVIEW, LabWindows/CVI, and C/C++.

When you edit a step that uses a module adapter, NI TestStand displays the Module tab on the Step Settings pane, where you specify the code module for the step and specify parameters to pass when you invoke the code module. NI TestStand stores the link to the code module, the parameter list, and any additional options as built-in properties of the step. ADE-specific adapters can open the ADE, create source code for a new code module in the ADE, and display the source for an existing code module in the ADE. For each adapter, you have a panel to link NI TestStand parameters to code module I/O. Figure 1 is a screenshot of the LabVIEW Adapter configurations within NI TestStand.




1 VI Parameter Table
2 Parameters Table Control
3 Express VI Buttons
4 Source Code and Help Buttons
5 VI Context Help Image and Description

*Figure 1.* LabVIEW Adapter Configurations in NI TestStand

### 3.2.2 Encapsulate Commonly Used Functions

Consider a function that is designed to bring back a waveform of data and make measurements to characterize the waveform. You then always want to run a post function routine that makes the measurements on the acquired waveform. In NI TestStand, you can encapsulate these functions in a custom step type. For the developer using this custom step type, the analysis routine executes transparently because the functionality is implemented in the step type.

Figure 2 shows the four built-in Test step types and an Action step type. The design of each step type controls how the step behaves. You can configure these step types to call code from LabVIEW, C DLLs, ActiveX, or other programming environments. The Action step type calls a code module, in which you can specify the input values and determine what to do with the output data. The other four step types share the same behavior, but you can also use them to analyze the resulting data using string or numeric comparisons. These step types provide the foundation for common test systems.

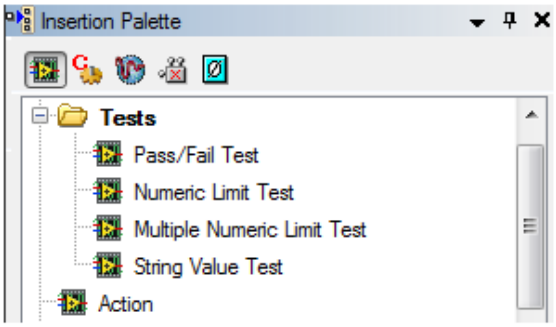



*Figure 2.* Built-In Steps for Calling Code Modules

### 3.2.3 Create Code Module Templates

Creating code module templates helps your test code look similar from test system developer to developer. In addition, with templates, you can develop code faster because you have a common starting point. NI TestStand offers code templates for LabVIEW, LabWindows/CVI, DLL, and more. The code template shown in Figure 3 defines the skeleton code for a particular step type that NI TestStand creates when the sequence developer uses the Create Code option for that step. You can also define multiple templates for a single step type, in which case a menu appears from which the developer can choose a template to use. For the DLL Flexible Prototype Adapter, you can use a code template to define the default mapping between the DLL prototype and the properties of the step.

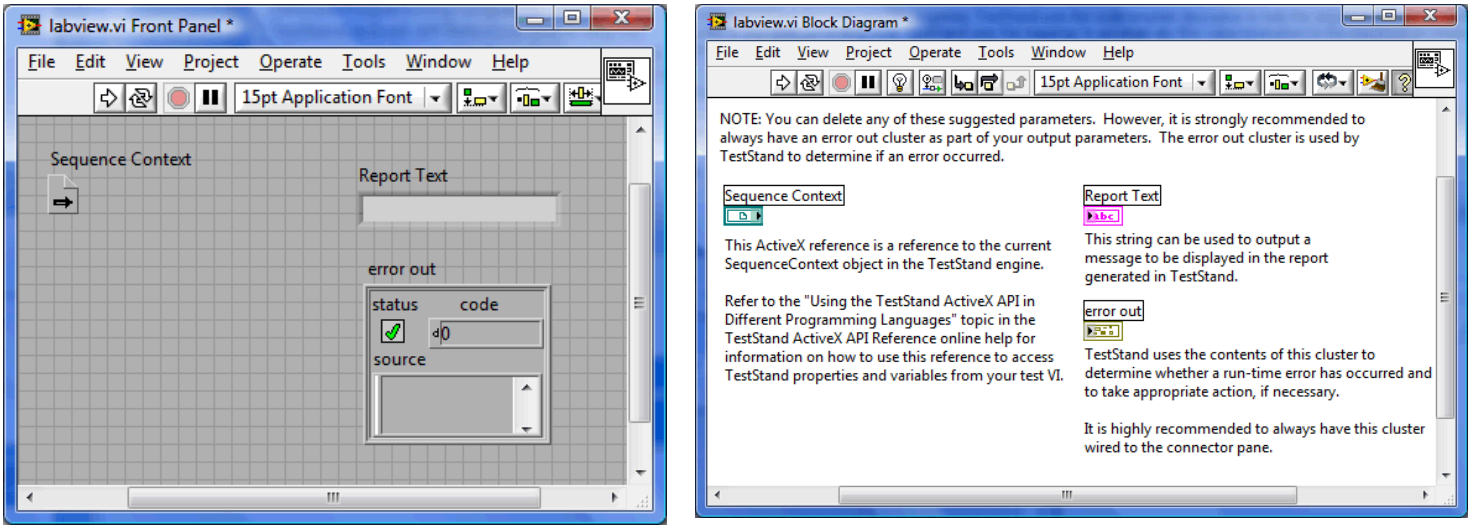



*Figure 3.* Example of the LabVIEW Template

### 3.2.4 Implement a Strategy for Error Handling

An important element in any test system is error trapping and handling. As discussed earlier, the Cleanup step group is useful for handling errors and, if necessary, shutting down a test system gracefully. In addition, a certain amount of error trapping must be done in the test modules themselves. All of the standard NI TestStand step types include an output structure for error information. This structure features a Boolean value to signal when an error occurs, a numeric value for an error code, and a string for an error message. In NI TestStand, a run-time error is not a test failure; instead, it indicates that there is a problem with the testing process itself and that testing cannot continue. When a test module traps an error, the module then reports this error condition back to NI TestStand. To be useful to the individual running the test, the error needs to display an informative message. One way to handle this error is to establish a set of error codes and associated messages. You can use a function to look up the message based on a code and pass both the code and message back to NI TestStand when an error occurs.

Another aspect of error handling is the ability to shut down gracefully or attain a known state when an operator detects some incorrect condition and manually shuts down the test system. You can stop execution in two ways. When you *terminate* an execution, all the Cleanup steps are executed and the sequence continues to run. Depending on the sequence model, it might continue with the next DUT or generate a test report. When you *abort* an execution, the Cleanup steps do not run and the sequence stops executing. You abort an execution when your execution must cease as soon as possible and you are not necessarily concerned about the resulting state of the system.

Whether you choose to terminate or abort, NI TestStand does not automatically stop executing in the middle of test module code. To enable a test module to shut down automatically when a sequence is terminated or aborted, you must use the NI TestStand API to check the status of the execution. The

LabVIEW Get Monitor Status.vi and the LabWindows/CVI CancelDialogIfExecutionStops function provide ways to monitor the execution state. In other ADEs, you can use the NI TestStand API Execution method GetStates to check the status of an execution to see if it has been terminated or aborted. It is especially important to use these functions when displaying a dialog box or running a particularly long test so you can exit these routines in response to an operator's attempt to stop the execution.

### 3.2.5 Document Code

Code module developers must provide good documentation, as shown in Figure 4. Without it, modifying the code is more time-consuming and error-prone. The following are some suggestions for comments on the LabVIEW block diagram.

- Use comments on the block diagrams to explain what the code is doing. LabVIEW code, in some cases, can be challenging to understand even though it is graphical. The free label located on the **Functions** palette has a colored background that works well for block diagram comments.
- Do not show labels on function and subVI calls because they tend to be large and unwieldy. A developer looking at the block diagram can find the name of a function or subVI by using the **Context Help** window.
- Use free labels on wires to identify their use. Labeling wires is useful for wires coming from shift registers and for long wires that span the entire block diagram.
- Use labels on structures to specify the main functionality of that structure.
- Use labels on constants to specify the nature of the constant.
- Use comments to document algorithms. If you use an algorithm from a book or other reference, provide the reference information.
- Insert images, for example, an image of the flowchart showing how the code should operate.

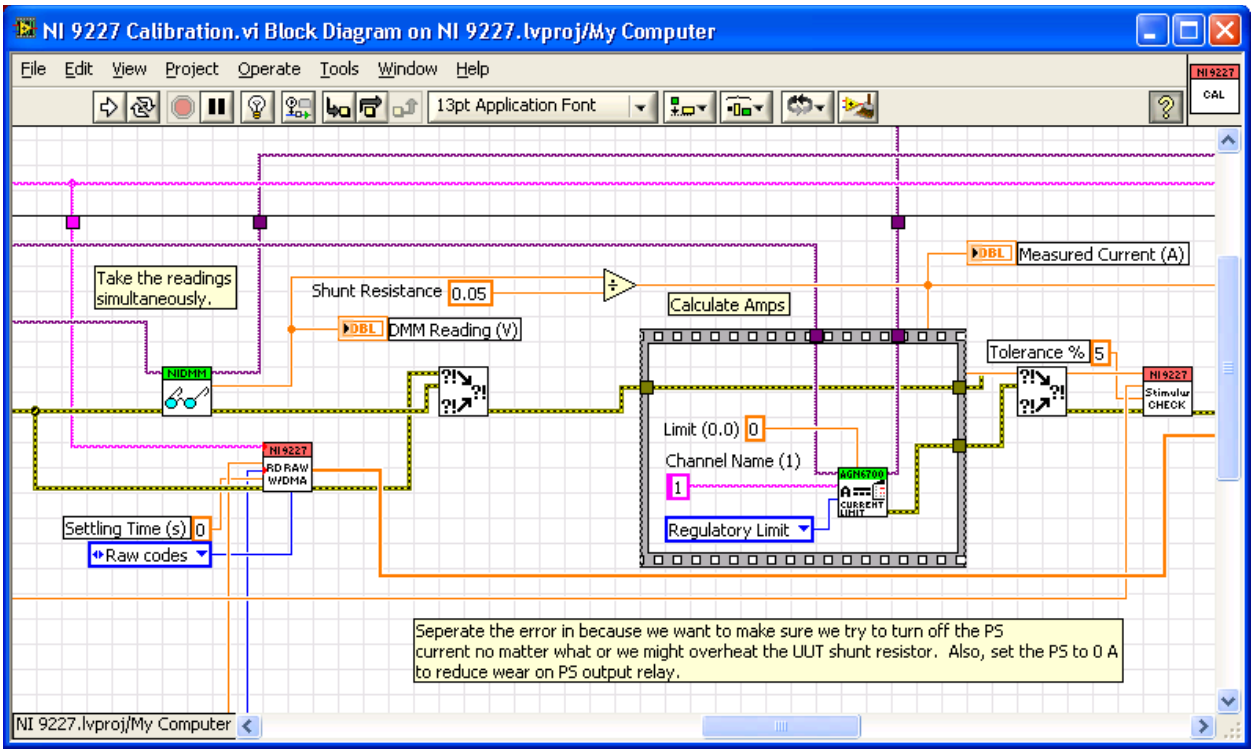



*Figure 4.* Example of Documenting a LabVIEW Code Module

### 3.2.6 Use Source Code Control

It is rare for a single engineer to work alone when developing a large test system. For the vast majority of applications, a team of developers or engineers work together on the same set of code. However, without proper infrastructure and planning, group development can introduce several pitfalls and inefficiencies. Common high-level group development challenges include the following:

- Tracking and identifying changes to code
- Distributing code modules to development teams
- Detecting and resolving conflicting changes to source code
- Providing centralized access to all versions of code
- Integrating changes to the master code depot

It is nearly impossible to coordinate all modifications to software components through simple electronic and verbal communication. In many cases, developers attempt ad hoc change control processes, whether they realize it or not, by saving multiple versions of a code module on disk as changes are made. However, this technique leads to confusion about the history of the code and adds even more time to the development cycle. Without a defined process for change control, bug fixes get lost and mismanaged, the differences from one version to the next are not easily distinguishable, and changes made by one developer are not appropriately taken into account by the rest of the development team. This leads to deficient code and inefficient development. Additionally, simultaneous editing of the same files by multiple developers is a common cause of lost work because changes can result in overwritten

data. Infrastructure to prevent these scenarios and notify developers that they need to combine or merge changes is critical.

Source code control (SCC) programs are designed to manage the shared development of test code. An SCC program tracks revisions and protects against two or more users making changes to the same test code at the same time. Using an SCC program is beneficial for a single developer and almost essential when development is shared among several engineers. You can check files and projects in and out of your SCC system from a LabVIEW project or NI TestStand workspace. Specifically, National Instruments has tested LabVIEW and NI TestStand with the following SCC providers: Microsoft Visual SourceSafe, Perforce, MKS Source Integrity, and IBM RationalClearCase.

# Step 3.3: Choosing Your Instrument Driver Paradigm

Developing your instrument driver strategy does not have to be a tedious process, but you should have a good understanding of your instrument control options before you finalize your test strategy. For example, using the driver that is shipped with each of your instruments may not always be the best solution for your test system. In an ideal world this might work, but in reality not all instrument drivers are created equal. Most test systems use instruments from multiple vendors and may not be interoperable, or you might have to program a legacy instrument that does not have a driver supported in your ADE. It is also important to document an official driver strategy when you are the one developing the tester because you may not be the one maintaining it or developing the test code for each DUT. This section provides background information on the different instrument driver approaches.

## 3.3.1 Instrument Driver Options

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 test program development time by eliminating the need to learn the programming protocol for each instrument. You can choose from three standard approaches for programming your instruments: Plug and Play instrument drivers, IVI instrument drivers, and Direct I/O. Each approach offers advantages for different use cases and application needs, as shown in Figure 1. Plug and Play instrument drivers are the most common instrument control method because they offer the ideal mix of instrument accessibility (typically 80 percent of the instrument functions are available) and they deliver ease of use through a straightforward programming paradigm that abstracts the low-level instrument function calls. Use the following descriptions of each approach to help you make educated decisions about your instrument driver strategy.

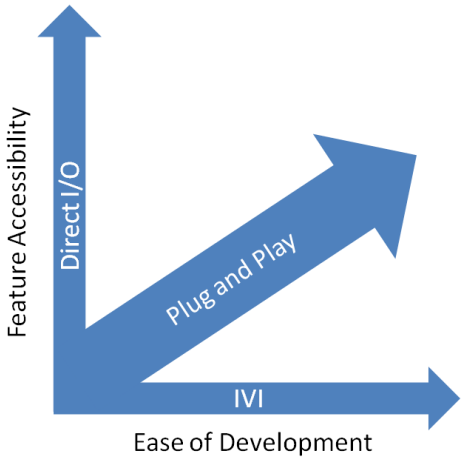



*Figure 1.*Each approach offers advantages for different use cases and application needs.

### 3.3.2 Plug and Play Instrument Driver

A Plug and Play instrument driver is a set of functions used to control and communicate with a programmable instrument. Each function corresponds to a programmatic operation such as configuring, reading from, writing to, and triggering the instrument. Plug and Play instrument drivers comply with programming guidelines. Because these drivers maintain a common architecture and interface and they include application examples, you can quickly and easily connect to and communicate with your instruments with little or no code development. Moreover, with the standard programming model of Plug and Play instrument drivers, you can easily add instruments to your test system without worrying about learning new communication protocols or spending time understanding new programming paradigms.

Plug and Play instrument drivers provide source code native to the development environment. With access to source code, you can modify, customize, optimize, debug, and add functionality to the instrument driver. Source code also makes Plug and Play instrument drivers operate cross-platform, so you can use them in any OS that works with LabVIEW or LabWindows/CVI software.

To obtain instrument drivers for NI products (for example, NI-SCOPE, NI-DMM, and so on) or for third-party products (for example, Agilent 34401A digital multimeter), visit the Instrument Driver Network. The Instrument Driver Network makes available drivers for more than 8,000 instruments for over 275 third-party vendors and for LabVIEW and LabWindows/CVI.

### 3.3.3 IVI Instrument Driver

Updating a test system can be a costly and time-consuming task. What initially seems like a simple change to a test system can lead to a considerable amount of re-development, re-verification, and re-documentation. Interchangeable Virtual Instruments (IVI) drivers, which are maintained by the IVI Foundation, are the most sophisticated instrument drivers with their key feature being multivendor instrument interchangeability.IVI provides a standard API for the eight most widely deployed instrument types, including DMMs and oscilloscopes, so you can write a test application based on an Agilent scope and program it using the IVI Scope class driver. If for some reason you need to replace the Agilent scope with a Tektronix scope, you can do so without changing your code. In addition, the IVI specifications offer base, extended, and instrument-specific API options; range checking; simulation; and other features that make upgrading instruments easier.

The IVI Foundation defines two architectures for IVI drivers, one based on ANSIC and the other on Microsoft Component Object Model (COM) technology. Although the technologies underlying IVI-C and IVI-COM are different, the implementation technology by itself should not be your primary concern. Instead, you should focus on two key issues: (1) the *longevity* of the architecture on which the instrument driver is based and (2) the *usability* of instrument drivers in your ADE.

***Architectural Longevity:*** The issue of architectural longevity is particularly important to users of IVI drivers. Interchangeability is one of the most beneficial features of IVI, and a primary reason for achieving interchangeability is to make it easier to replace instruments in systems that must last 10 to 20 years. Having a common API for instrument drivers is not sufficient if the architecture on which those

instrument drivers are based changes every few years. It is for this reason that the IVI-C architecture is preferred. ANSIC is a long-established standard that is expected to remain available on all platforms. Conversely, COM is not freely available on all platforms and has already been superseded by .NET.

***Usability:*** One of the key motivations behind IVI-COM drivers is the desire to develop one driver that works automatically in all development environments. Unfortunately, this comes at the expense of usability. IVI-COM drivers present an interface style that is optimal only in Microsoft Visual Basic 6.0, which was superseded by Microsoft Visual Basic .NET. Ideally, instrument drivers should present an optimal interface for each ADE in which they are used. For example, instrument drivers should be presented as a set of LabVIEW VIs for use in LabVIEW, a set of C++ classes for use in Microsoft Visual C++, and a set of .NET classes for use in Microsoft Visual Basic.NET and Visual C#.

Although IVI-C drivers are the best choice to build an interchangeable test system that can stand the test of time, if all that is available for your instrument is an IVI-COM or VXIplug&play instrument driver, then you should use it. Your goal is to be able to easily communicate with your instruments and to not be tied to one driver technology.

Before determining if a driver technology is right for your application, you should ask yourself some of the following questions:

- Does my test system need to last longer than 10 years?
- Am I sure I want to sole-source my instrumentation?
- Will the eight IVI instrument classes meet my application requirements?
- Do I have to revalidate my entire test system when I make a change to an instrument driver (common for mil/aero/medical)?
- Am I maintaining a system that is currently using instruments that will probably be obsolete before my test system?

If possible, use IVI if it satisfies your requirements because developing a user-defined IVI Class driver is often a large investment for an organization. However, the investment of many companies and the stability of the standard over many years can reduce your need to design, develop, and maintain a driver, saving time and money.

### 3.3.4 Direct I/O

Direct I/O is a low-level method for programming instruments. Standard Commands for Programmable Instruments (SCPI), pronounced “Skippy,” is an extension of the IEEE 488.2 standard that defines a standard programming command set and syntax for Direct I/O. Engineers typically use this method only when Plug and Play or IVI instrument drivers are not available. You may also want to use Direct I/O because of the following:

- You are updating an old system that is based completely on SCPI
- You need to send only a few commands to your instrument
- You do not need to distribute a set of instrument commands to other developers

The modern Direct I/O communication standard to instruments is through the Virtual Instrument Software Architecture (VISA) API. VISA is an industry standard communication protocol managed by the IVI Foundation that provides bus- and platform-independent instrument communication. For example, the VISA command to write an ASCII string to a message-based instrument is the same whether the instrument is serial, GPIB, or USB. Thus, VISA provides interface independence. This can make it easy to switch interfaces and gives the users who must program instruments for different interfaces a single language they can learn.

If there is not an instrument driver available, you should take advantage of the interactive, direct I/O capabilities built into the software development environments. NI ADEs offer the Instrument I/O Assistant, built-in VISA and bus-specific interfaces, and several debugging tools within Measurement & Automation Explorer (not covered in this document) including NI Spy, Interface Bus Interactive Control (IBIC – for GPIB), and VISA Interactive Control (VISAIC).

# Step 4: Assembling the Test System

This section provides five simple steps to follow when assembling a test system to help you get up and running as quickly as possible.

## 4.1 Assembling Components

Begin the chassis and instrument installation by printing out the test system design material developed by the station architect. This material is developed during Step 2 when you are designing the automated test system hardware. Station architects should provide a system schematic, bill of materials (BOM), and mechanical drawings that technicians or test engineers can use to assemble the system to specification. For example, Figure 1 illustrates all of the connections on the PXI embedded controller. Figure 2 shows an example of a mechanical drawing for a test system that outlines the spacing and the location where you should mount each DIN rail.

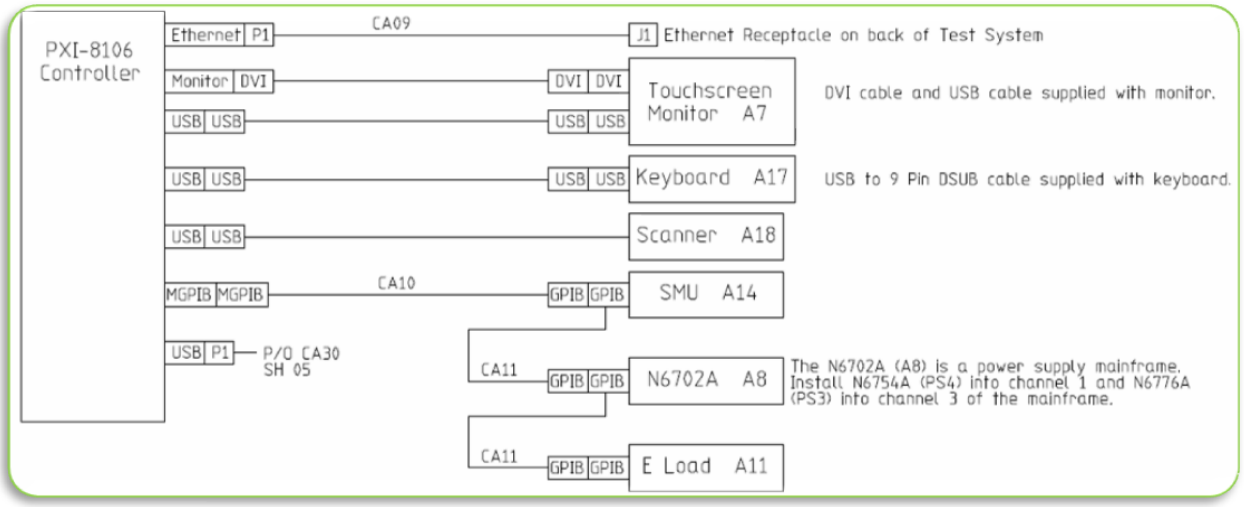



*Figure 1.*Example of Documenting the Connections on a PXI Embedded Controller

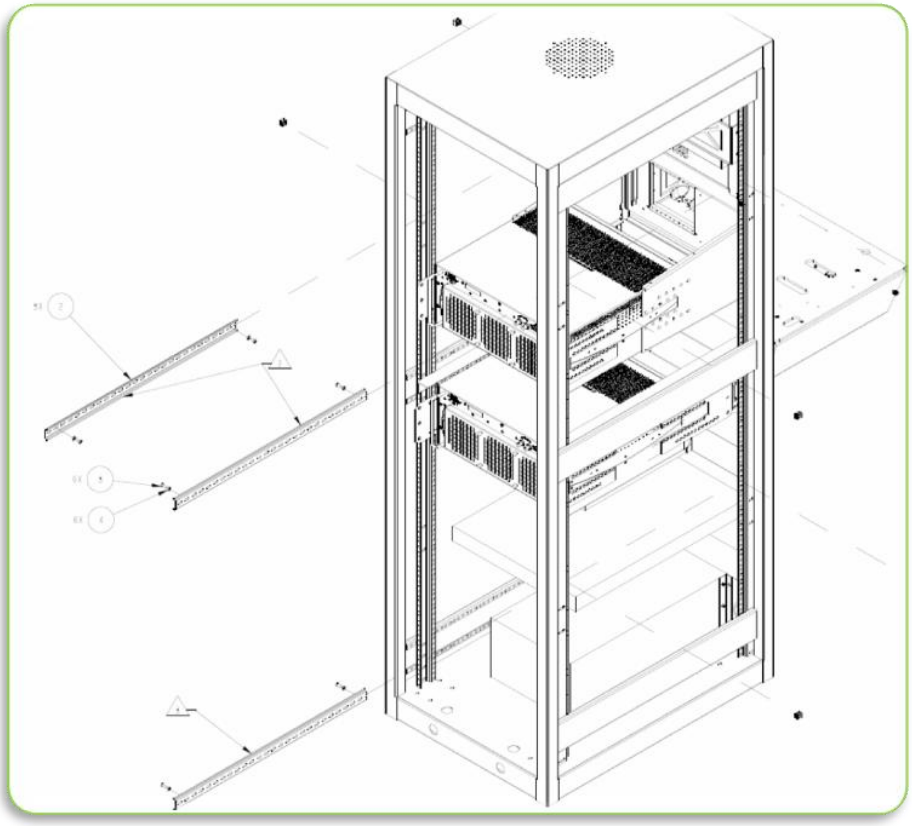


*Figure 2.*Example of a Mechanical Layout

## 4.1 Building and Routing Cables

When building cables for your test system, use cables recommended by the instrument manufacturer because these cables typically provide the best measurement results. For example, for NI 4070 6½-digit DMMs, National Instruments recommends selecting a cable with characteristics similar to the Belden 83317E; refer to the Belden CDT Incorporated Web site at www.belden.com for information about this cable. This recommended cable is tape-wrapped polytetrafluoroethylene (PTFE) insulation, which guarantees low leakage. Copper conductors with silver plating perform well for low-voltage measurements because of their low thermal characteristics. Use a cable with at least 90 percent shielding effectiveness.

| **Belden #83317E Teflon Insulated Cable** | | | | |
|---|---|---|---|---|
| **Type of Insulator** | **AWG** | **Conductor Material** | **Resistance per Foot** | **Capacitance per Foot** |
| TFE | 26 | Silver-plated copper, stranded | 39 mΩ | 35.5 pF |

*Table 1.*Recommended Cable for NI 4070 6½-Digit DMMs

### *Cable Length Considerations*

It is important to keep signal integrity in mind when laying out your test rack. Long cable lengths can often distort signals and cause measurement errors. To determine the proper cable length you need, measure the distance from the front face of the instrument to the module location in the receiver with the mounting frame in the "open" position. Allow for bends in the cable for proper cable routing with extra length in the cable to prevent tight bends when the enclosure is closed.

To determine the appropriate cable length for instruments not directly behind the receiver, measure the vertical distance from the receiver to the instrument and then measure the distance from the front of the rack to the receiver with the slides fully extended, taking into account any bends and special routing considerations. Cables routed to instrumentation not placed on the instrument bracket should be secured to the strain relief on the instrument bracket. You can incorporate a "maintenance loop" in your cable lengths to ensure you have enough slack to repair the cable or replace the instrument. If cable lengths are too perfect, you may have difficulty accessing the instrumentation. Figure 3 is an example of a well-planned cable layout.

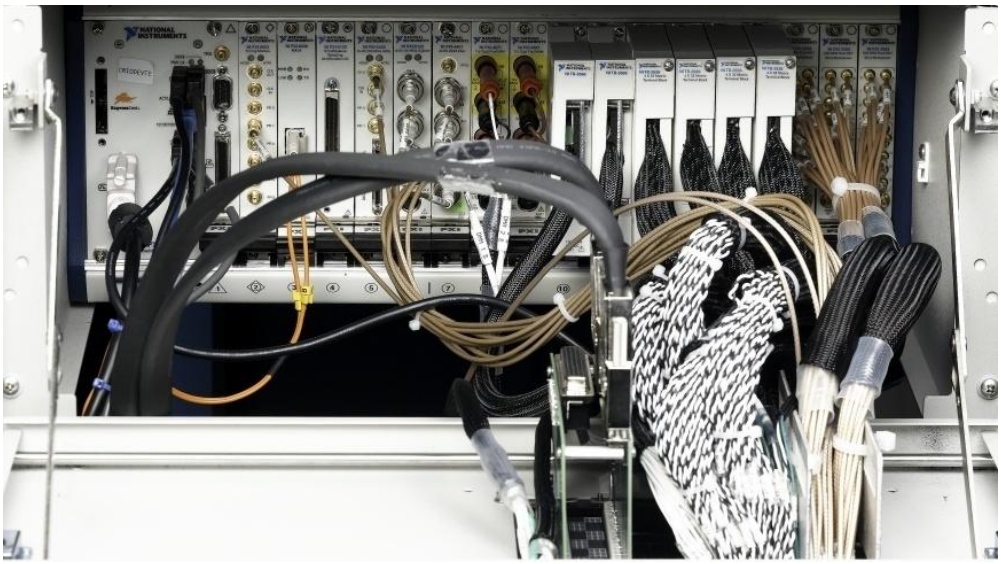


*Figure 3.* An Example of a Well-Planned Cable Layout

### *Bundling and Securing Cables*

Because of the large number of cables in test systems, you need to strategically bundle individual cables using sleeving and/or wire ties. Sleeving helps protect cable assemblies, hoses, and wire harnesses from chafing, cutting, and abrading. It also offers heat protection and defends against chemicals or abrasion. You can choose from several types of sleeving material based on your system requirements, including a standard, lightweight expandable sleeve or a fiberglass sleeve. Fiberglass sleeves are used for high temperatures and voltages up to 300 V as well as for the thermal protection of insulated wires against damage from external heat sources. Sleeving does take some planning because you want to make sure

to group common signal types. Figure 4 shows an example of grouping DMM cables into a single-sleeved harness labeled CA14.

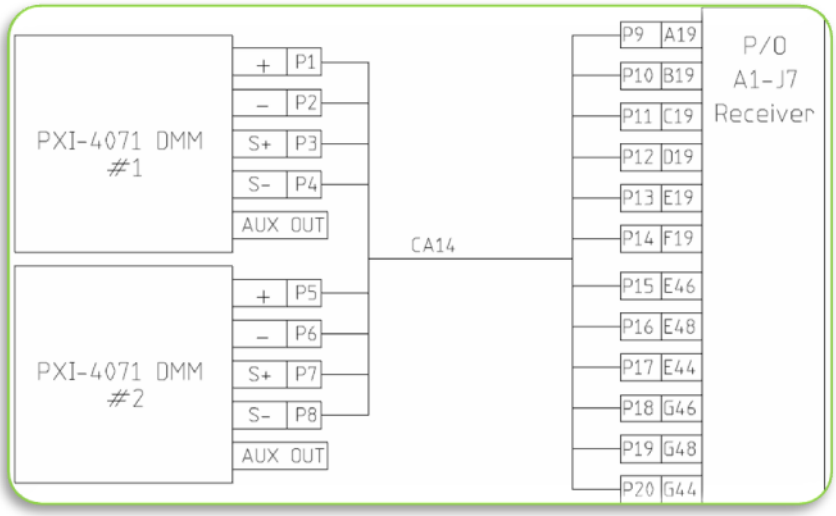



*Figure 4.* An Example of How NI Details Cable Assemblies That Should Be Sleeved Together

To find the correct size of braided sleeving for your application, begin by measuring the diameter of the cable bundle you are covering. Depending on how you want the braided sleeving to fit around the cables, you can select a size in one of two ways. If you want a snug fit, opt for braided sleeving with a diameter that is slightly smaller than that of your cables. If you want a loose, flexible fit, select sleeving with a diameter that is equal to or slightly larger than the diameter of your cables. Whatever you choose, take into account that braided sleeving shortens a bit when it expands.

Strain relief is critical in developing a reliable and consistent tester. You can choose from several strain relief options. The most common strain relief method is to attach strain relief plates directly to the receiver or ITA modules. Wire or cable bundles are secured to the plate with wire ties, which prevents forces that exert on the cables while transferring directly to the contact termination from causing damage and loss of signal integrity. You can also use a second type of strain relief on the receiver side of a tester: cable tie-down bars. These bars are commonly included in the instrument bracket or cable tray kits used with slide-mount receivers.

Lastly, you should label each cable and each sleeved bundle of cables with clear heat shrink to protect the labels. Figure 5 shows an example of how to implement these best practices.

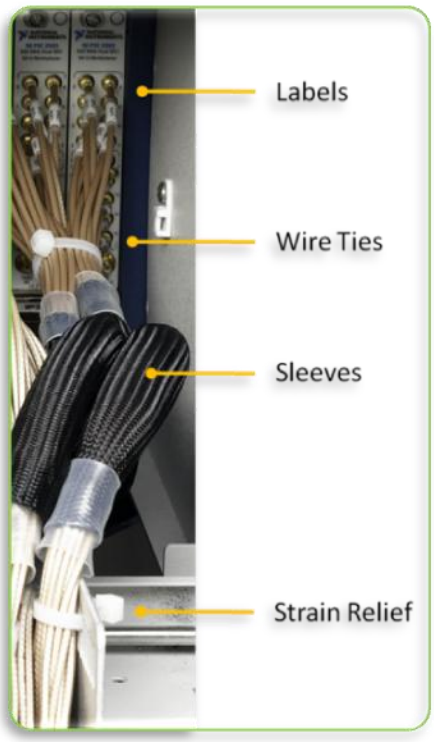



*Figure 5.*A Close-Up of a Cable Assembly

### *Proper Grounding and Routing of Signals*

Securing a common earth ground for your test system is critical because it prevents ground loops and provides a common zero. A typical method for providing a common ground is to attach a grounding block to the ground lug on the power distribution unit. Then terminate everything (for example, shields, computer, and rack) to this single grounding block, as shown in Figure 6. You should also route power lines and signal lines separately to minimize/prevent noise and crosstalk.

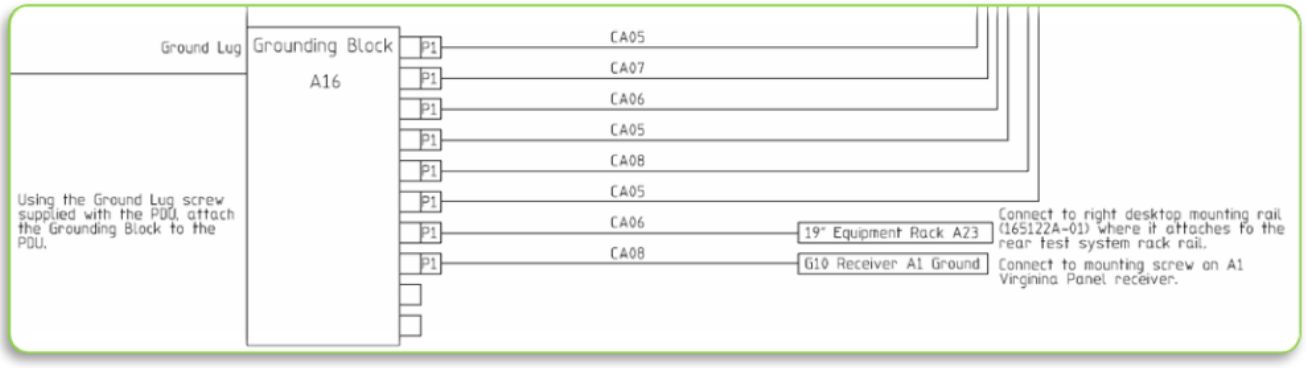



*Figure 6.*An Example of How to Document a Common Grounding Block in a Schematic

## 4.2 Installing and Activating Software

Before installing any software, you should research the installation requirements for each piece of hardware and software you plan to use in the test system because most vendors recommend a software installation order. A key task for the station architect is to resolve all conflicts and system dependences. In the end, the station architect should document the software installation order. A completed software installation document includes details such as software version number and installer location (for example, an internal directory or installer CD).The following is an example of an installation order:

1. Operating system
2. Antivirus software
3. Application software
4. Software add-ons to application software (for example, LabVIEW FPGA Module and LabVIEW Database Connectivity Toolkit)
5. Hardware drivers
6. Specific software add-ons for hardware (for example, LabVIEW Sound and Vibration Toolkit)
7. Test executive
8. Test framework
9. Additional software (touch screen drivers)

Your license strategy is another consideration when installing test system software. For example, some NI software products provide debug and deployment licenses so you can resolve issues with systems and applications in production and you can license the intellectual property provided by the development license, respectively. For example, NI TestStand provides three licensing options to meet your specific needs – Development System, Debug Deployment Environment, and Base Deployment Engine.

### *Debug License*

A debug license for a particular software product provides the permission to install that software on one computer. You may use the software only for debugging an existing application created with a purchased development license. You typically use debug licenses when an application is deployed, and you may need to make minor changes to the code periodically. Debug licenses are available for LabVIEW, LabVIEW modules, LabWindows/CVI, and NI TestStand. The LabVIEW and LabWindows/CVI debug licenses also include permission to install any relevant add-ons needed for debugging. The NI TestStand debug license includes debugging permission for LabVIEW, LabWindows/CVI, and relevant toolkits.

### *Deployment License*

A deployment license is required for distributing applications that use specific software products. You must purchase one deployment license for each target computer on which the application is distributed. The following are a few examples of NI software products that require a deployment license: NI TestStand, NI Switch Executive, LabVIEW Real-Time Module, LabVIEW Data logging and Supervisory Control Module, LabVIEW Sound and Vibration Toolkit).Note: The supported products list may not be exhaustive. Consult the product documentation for complete information on available licenses.

## 4.3 Configuring and Validating in MAX

The NI Measurement & Automation Explorer (MAX) configuration utility is automatically installed with NI software and device drivers (see Figure 7).With MAX, you can:

- Configure your NI hardware and software
- Execute system diagnostics
- View devices and instruments connected to your system (via GPIB, USB, and Ethernet)
- Update your National Instruments software

If you expand the Configuration tree within MAX, you can configure both PXI and traditional instruments.

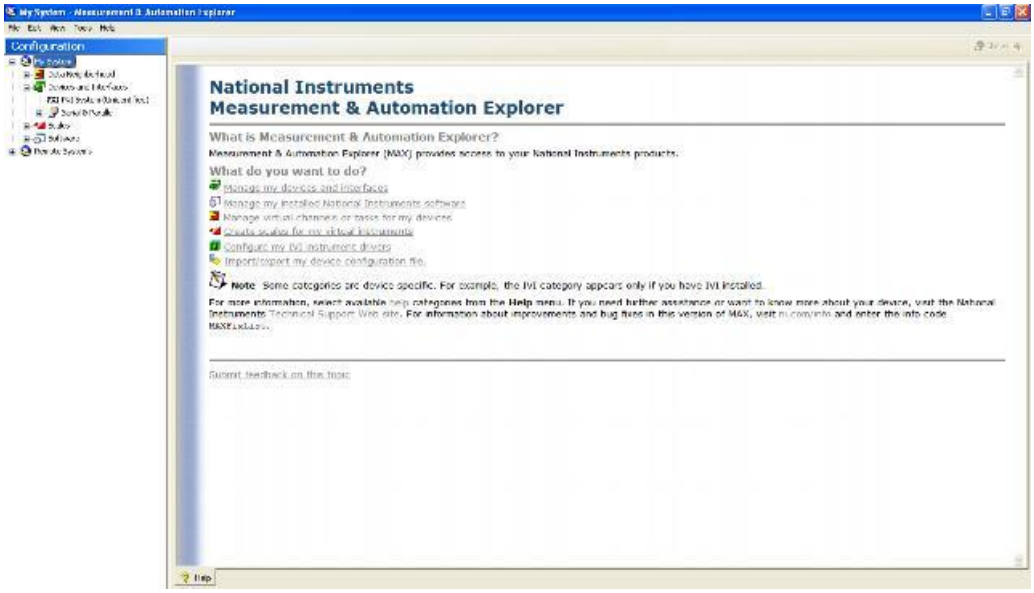



*Figure 7.* Screenshot of MAX

### *Configure Your PXI System*

The steps required for configuring any PXI system entry are the same, regardless of whether you are using an embedded controller or a remote controller (for example, MXI-4). To configure your PXI system, follow these steps:

1. Launch MAX (find the icon on the desktop of your computer). Ensure that you are working with MAX 3.x.
2. Click on the **Devices and Interfaces** section in the Configuration tree to expand it.
3. If the PXI system controller has not yet been configured, it is labeled as **PXI System (Unidentified)**. Right-click on this entry and click on **Identify As** from the menu. Choose the controller model that you are using.

   Note: If you are using a remote controller, such as MXI-4 or an NI PXI-8360, to control your chassis, you should select **External PC**.

4. Right-click on the chassis that appears under the PXI system entry and select **Identify As** from the menu. Choose the chassis that you are using.
5. If you are using multiple chassis in your system, then repeat step 4 until you have identified all of the chassis in your PXI system.
6. Only after you have defined the controller and the chassis that you are using in your system does MAX save the correct PXISYS.INI file in the Windows system folder.

### *Configure Traditional Instruments (GPIB, USB, LAN, and so on)*

To confirm that your GPIB devices are connected properly, expand the **Devices and Interfaces** subdirectory below **My System**. Then select your GPIB controller. This example uses an NI PCI-GPIB controller. If you are using a USB, serial, or Ethernet controller, the name may be slightly different. Click on **Scan for Instruments**, as shown in Figure 8. If your GPIB device is SCPI-compliant, the name and address appears in the main window, as shown in Figure 9.

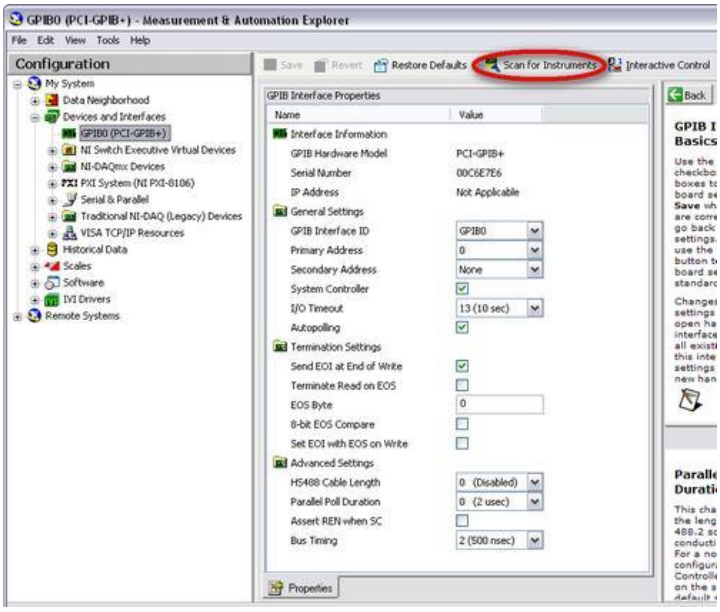



*Figure 8.* Scanning for Instruments in MAX

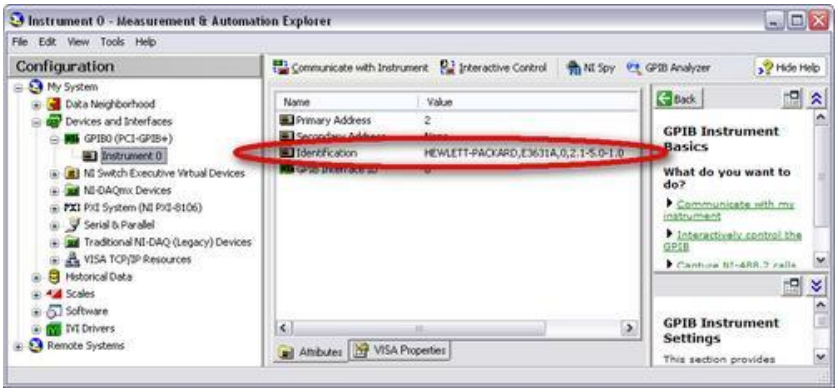



*Figure 9.* GPIB Device Found

## 4.4 Validating the Tester

As discussed in Step 2 of this guide, you should consider how you are going validate the test system, once it has been completely assembled. You can choose from several techniques to simplify test system validation. One best practice is to route all of your resources to the receiver on your mass interconnect system. By hardwiring components of your test system to each other (for example, switching to instruments), you can significantly increase your system validation efforts. Once you have all of your resources available on your receiver, you can easily conduct a manual point-to-point test.

To conduct a manual point-to-point test, you must close a specific channel on your switch module and manually probe both ends of the switch on the receiver to ensure that a short has been created. Instead of conducting a manual validation of your test system, you can build a self-test fixture (consisting of several loopbacks or shorts) that mates to your receiver and automates the process.

In addition to validating your hardware, you should validate your software. You can do this by stepping through your code using probes and breakpoints and ensuring that each section of your code does what it is supposed to do. Figure 10 shows a LabVIEW routine that was developed to debug the software for all the switching in the NI production test system.

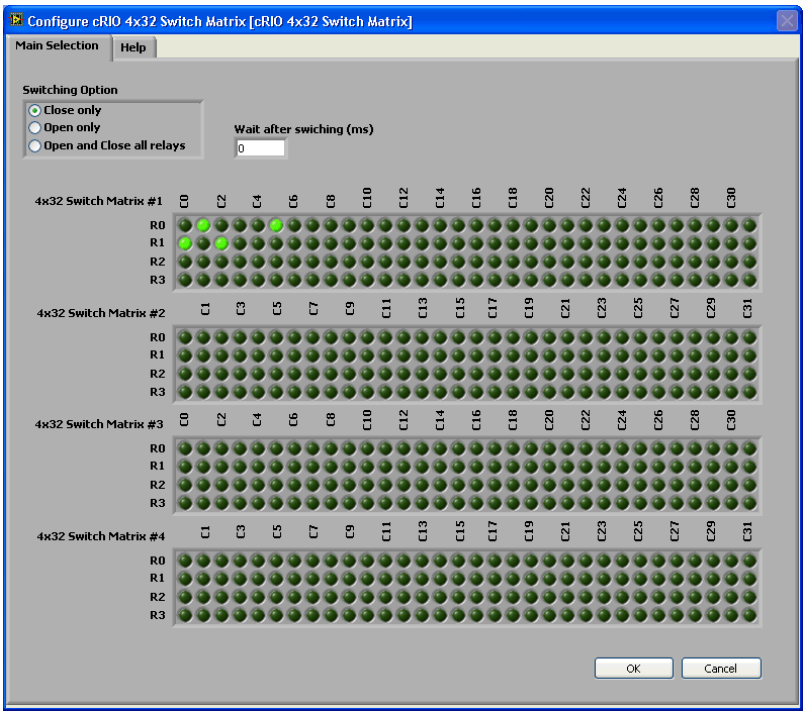



*Figure 10.* LabVIEW Routine for Validating Switching Software in the Test System

# Step 5: Deploying the Test System

Now that you have analyzed the product line requirements for your test system, researched and implemented the best hardware for your system, designed your test system software, fabricated the system, and validated the system, the final step is deploying the test system. Because you have taken special care and consideration in the development of your ATE system, the deployment phase should be a smooth process. This chapter outlines the steps for deploying your test system.

## 5.1 Deploying the Test System Software

An effective software deployment strategy provides the confidence and reliability needed to ensure an effective software deployment. Software deployment presents several questions that need to be addressed such as the following:

- *What method should I use to gather all of my required files?*
- *How do I distribute those files to my test stations?*
- *How much control do I need or want over my distribution?*
- *How do I go about updating those test stations?*
- *Should I make this deployment process manual or automatic or even both?*

To address these issues, this section describes several methods for effectively deploying your software. First consider the methods for creating a deployment image.

### System Replication

A deployment image contains a directory of files to be installed to the target computer. You can manually create this deployment image or use the NI TestStand Deployment Utility.

- **Manual** – Manually gathering all files needed for distribution is a tedious job, but it provides the developer full user control over the files that are deployed to the target computer. One of the most common problems developers experience when deploying test software is missing or incorrect versions of file dependencies. A file dependency is a secondary file that a file requires to compile, load, or run correctly. Normally dependencies come in the form of DLLs, .NET assemblies, or subVIs. It is extremely important that you identify exactly which dependencies your test software requires as well as their versions. This can be challenging and, in many cases, you may need a third-party product to determine the explicit dependencies of a file.
- NI **TestStand Deployment Utility**–This deployment utility greatly simplifies this process by using workspaces and project files to collect all of the files required to successfully distribute your test software to a target computer. The NI TestStand Deployment Utility is tightly integrated with LabVIEW facilitating the deployment of the VIs that make up your test system. The deployment utility focuses on facilitating the collection of all necessary code. For example, the deployment utility analyzes all of the LabVIEW code that it deploys to determine its complete hierarchies, including all subVIs, DLLs, external subroutines, run-time menus, LabVIEW Express configuration diagrams, and help files that your VIs may reference. It then packages these VIs and their

hierarchies to ensure that they are executable on systems that do not have the LabVIEW development system installed.

- **Hard Drive Imaging Software (in other words, Ghost)** –The previous options help you create a deployment image of your code. With imaging software, you can make a complete copy of the hard drive at a very low level. You need a disk imaging utility to do this. This image contains all of the software including OS, drivers, and applications on that PC as well as all user settings. This is what you deploy to your test system(s). In most cases, an image works only if you are deploying to a PC with the exact same hardware and chipsets.

Now that you have created the deployment image, the next step is to actually deploy this image to your test machines by using one of these four methods: file copy, source code control, installer, and shared drive.

### *File Copy Method*

The file copy method consists of copying and pasting the deployment image either directly to a test station's local drive or to a network drive. Copying the deployment image to a local drive allows the test station to become self-sufficient. However, the downside to using this option is the fact that distributing updates to all the separate test stations is time-consuming. Copying the deployment image to a network drive reduces distribution time and greatly assists in providing updates to the test software. Because this method is solely network-based, common network problems such as network status (up, down), speed of accessing network components, and accessing files already in use must be factored in. Another difficulty when deploying to a network drive is determining if the test system fails because of an update of the files in the network or because of an actual test failure. When test software developers send updates to the network, normally notices do not get sent out. Thus it is difficult to narrow down the test failure.

The file copy method in general requires very little Windows knowledge and provides the greatest level of control over distributing test software. Upgrades can be easily installed by simply copying over the updated files. Because most test software applications usually incorporate the use of software/hardware drivers, you must ensure that these drivers are present on the target computer. The file copy method requires you to install the software/hardware drivers separately, which increases the effort required to successfully deploy your test software.

### *Source Code Control Method*

Using a source code control (SCC) system to assist in distributing your deployment image can be very beneficial. In this case, the source code control server maintains a centralized master copy of the deployment image and allows clients (test stations) to sync up and use the test software. Even though in this case the SCC software is networked-based, local copies of the deployment image are downloaded to each client machine, which addresses the possible event of network failure. Upgrading the client machines to the latest test software is easy because clients are connected directly to this SCC server and, therefore, are using the latest revisions of the test software. Additionally, this SCC software allows clients to revert to older revisions in case the newest revisions are corrupted or incorrect. This ensures test stations continue to run regardless of unexpected events. To use SCC software, you need some

background knowledge and experience with SCC. Similarly to the copy-and-paste method, this method requires you to install the software and hardware drivers separately.

### *Installer Method*

The deployment methods mentioned earlier are based on a more user-controlled deployment approach where the developer performs most of the labor. However, as software applications continue to grow in complexity, a more automated approach to deployment may be a better option. With installers, you can integrate installer technology with the deployment image to create one easy-to-use installer package that is distributed by any convenient means. Installers provide the additional benefit of bundling supporting software such as hardware drivers, documentation, licenses, and configuration files with your software application into a single package. In addition, installers work directly with the Windows OS to keep track of user-created files and use the modify/repair/uninstall features of Windows installers.

Installers provide multiple benefits but also present some challenges such as distributing minor updates and usability. It is difficult to simply deploy minor changes to a target system because a whole new installation package must be rebuilt. You also must consider the usability difficulties involved with using common installer packages. Most installer applications are not user-friendly. Windows installers in general provide extreme flexibility and maximum freedom. However, they force the user to know more about Windows technologies. The NI TestStand Deployment Utility reduces the common difficulties associated with normal installers. The easy-to-use graphical user interface for creating installers is extremely simple to use and provides a flexible and customizable environment to include various components. You need no previous installer knowledge to create an NI TestStand installer. Incorporating National Instruments and third-party drivers is easy with the deployment utility.

### *Shared Drive Method*

The shared drive deployment architecture concentrates file storage on a shared network drive rather than a local hard drive. In this architecture, test stations map to a folder on a single network drive that contains the necessary files to execute a test solution. In this approach, test station configuration is standard and it does not vary between test stations. All test files are stored on the shared drive and are controlled through regular network access. Figure 1 illustrates the major components of the shared drive architecture.

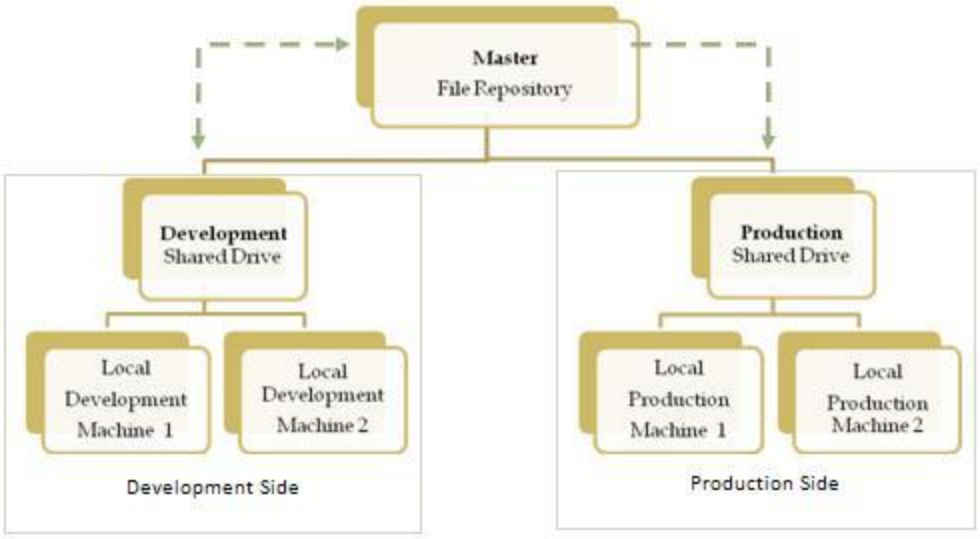



*Figure 1.* Shared Drive Deployment Architecture

The Master File Repository contains the current version of each file along with a copy of older versions of each file. You can use an SCC application for file management functionality.

The development side includes the components required to support the file development, test, and debugging needs of the test development group. The Development Shared Drive is updated by the Master File Repository, which provides the versions of the files that test developers need. Test developers use local development machines to develop and debug test files, and then update the Master File Repository with new files or new versions of existing files. With the shared drive, test developers can test their changes and make them available to other developers without committing them to the Master File Repository until they are verified.

The production side includes components that support the execution of files on production machines. The Production Shared Drive is updated by the Master File Repository to use a specific version of the files – usually the most current version. Production machines can then access this Production Shared Drive to run these test files. On the production side, you cannot load changes to the files from the Production Shared Drive into the Master File Repository. This reduces the risk of accidental changes to a file that can cause tests to fail. Depending on the resources available, it is possible to develop other models that combine the functionality of multiple machines into one. For example, you can combine the **Master File Repository** and the **Development Shared Drive** or you can set up test stations to access the current version of all files directly from the **Master File Repository**.

The design of a shared drive deployment strategy does not end with deciding to use a shared drive for deployment. You must also make tactical implementation decisions based on your specific business needs. Some key business considerations include whether all test systems are running the same version of the test software, which test file resources must be deployed to test stations and which resources should be stored on the shared drive, how to implement change management control, how to handle incremental updates, and how to manage required differences between development and production

test stations. To learn more about this topic, read the NI TestStand Shared Drive Deployment Reference Architecture whitepaper.

## 5.2 Activating the Software

The last step in deploying the test system software is to change the computer name on each of your test systems so they have unique names and to activate all of your software licenses. When you deploy the image from the development PC, the computer name from that PC is deployed to all of the test stations. You need to change these names so there are not conflicts if your test stations are connected to a network. As discussed earlier, you also need to activate all of the software licenses on the test systems. Because you already have determined your licensing strategy, this should be an easy process. Depending on the software you have installed and whether you chose a corporate license or single-seat license strategy, you may have to manually activate each license.

With activation, you can download evaluation software and simply activate the fully featured version once purchased. You also can add features or migrate to higher product levels without reinstalling software. For example, a system deployed on the manufacturing floor contains an NI TestStand Deployment License. If something goes wrong with that system and it needs debugging, you do not need to install additional software. Simply apply a new activation code to enable the NI TestStand Development System to fix the problem.

NI has made activation quick, easy, and flexible by providing several activation methods. It takes only a few minutes of your time and does not require you to formally contact NI directly. When you install the product, the NI Activation Wizard is immediately launched and guides you through the entire activation process. There are six methods available to activate your product. When you activate, the NI Activation Wizard displays each of these options and gives you any information required to activate.

## 5.3 Deploying the Test System Hardware

Now that you have examined the process for deploying your test system software, explore the considerations you need to make when deploying the test system hardware. Before you reach the deployment phase, you need to make sure that the facilities housing the test systems meet all of your requirements. Consider the following:

1. **Space**

   You need to make sure there is enough space on the test floor for all of your test systems and that each test system has adequate space for the hardware and operator. Before deploying the hardware, make sure the facility doors are big enough for the test system to fit through. If this is overlooked and the test system is too large, you need to disassemble and reassemble it, which causes unnecessary delays.

2. **Power**

   Make sure the facility has the appropriate power supply for the test systems. Your test system may require 110, 220, or 240 V to power it. This should be known and planned for ahead of time. Also make sure there are adequate power drops for each system.

### 3. Networking

If you are using a network to access the test code, deploy the image, or access the test systems remotely, then you need network drops for each test system. This should be planned for before you deploy the test system. Again, overlooking this requirement can add time to your test system development and result in schedule slips.

### 4. Environment

It is also important to know the environmental specifications of all of the hardware in your test system and to ensure that the environment meets all of these specifications. These include temperature, humidity, shock, vibration, altitude, and pollution degree. For example, if one of your instruments is limited to an operating temperature of 40 °C, you need to make sure that instrument does not exceed the temperature specification. This may require adding more cooling fans to the facility or surrounding the test system.

### 5. Safety

You need to take safety precautions to protect the operators and systems. Moving parts may require enclosures or fencing and high-voltage components need adequate spacing. Safety should be a primary concern –action should be taken to make sure the environment is as safe as possible and all safety standards are followed.

### 6. Maintenance

Because your test systems are now deployed and running, it is important to put the preventive maintenance (PM) plan that you have already developed into place and make sure that it is on a schedule. The PM includes instructions on cleaning the test equipment and components, replacing items that may fail or wear out, calibrating your instruments, and monitoring your system. The PM plan should also include a station document that explains step-by-step how to correctly build the OS, install drivers, and software. In addition, it should include a list of required accounts and passwords, ways to back up data generated by the tests, and instructions on how to change software and employ revision control.

As defined by your PM plan, make sure that you clean the test system and components periodically. The cycle you choose to clean fan filters and other components should be determined by the cleanliness of the environment. If your test systems are in a dusty environment, then they need to be cleaned more regularly. It is also a good idea to stock spares. Consumables include anything that has a high usage rate or can wear fast. Examples are fan filters, pins and connectors, power supplies, hard drives, and so on. You want to be able to replace items that fail as fast as possible to minimize the downtime of your system. Make sure you are calibrating the instruments that require calibrating at the timer intervals outlined in your PM plan. Instruments that are not calibrated properly can result in faulty tests. Finally, if you have chosen to add features in your test code to monitor your test systems, make sure you have a plan in place to react if a failure happens. This plan should be outlined in your PM plan, and operators should be aware of it to ensure minimal downtime in the event of a failure.