The Test-System Development Guide series is designed to help you quickly design a test system that produces reliable results, meets your throughput requirements, and does so within your budget.

**Table of contents**

Introduction to Test-System Design *AN 1465-1* 2

Computer I/O Considerations *AN 1465-2* 14

Understanding Drivers and Direct I/O *AN 1465-3* 24

Choosing Your Test System Software Architecture *AN1465-4* 38

Choosing Your Test-System Hardware Architecture and Instrumentation *AN 1465-5* 58

Understanding the Effects of Racking and System Interconnections *AN 1465-6* 76

Maximizing System Throughput and Optimizing System Deployment *AN 1465-7* 88

Operational Maintenance *AN 1465-8* 104

Agilent Technologies
This application note is part of the Test-System Development Guide series, which is designed to help you quickly design a test system that produces reliable results, meets your throughput requirements, and does so within your budget.

This first application note in the series, Introduction to Test-System Design, covers test-system philosophy and planning and discusses how test is used in three sectors: R&D, design validation and manufacturing.
Pathways to exceptional test

Functional test is fundamental to the electronics world. In the past, test has been treated as a necessary expense, but enlightened companies have realized that test can be a significant asset. For example, rather than using the test system to simply verify the limits of the device under test (DUT), you could use it to:

• find the weaknesses of the device—before your customers do
• predict failures or out-of-spec trends in production
• search for the boundaries of the design—to stretch specifications or search for something you didn’t know the product could do
• verify the long-term characteristics of the product
• optimize a production process
• test for environmental limits
• find the weaknesses in a competitor’s product

Test can be used simply as a gating factor for “good” or “bad” devices, or it can be used to gain a competitive advantage. This application note is the first of a series that will give you an overall view of how tests are made, techniques to optimize tests, and a number of methods you can use to your advantage. Other application notes in the series will cover topics such as hardware architecture, choosing instruments, software architecture, computer I/O and connectivity, assembling a test system, maximizing throughput, and optimizing deployment and maintenance.

A systematic test-system design process as outlined in these application notes will assist you to quickly design a test system that produces reliable and repeatable results, meets your throughput requirements, and does so within your budget. For further information regarding test-system design, refer to the book Test-System Design, A Systematic Approach by Tursky, Gordon, and Cowie (Prentice Hall, 2001). Much of the information in this application note was derived from this book.

This application note covers the three primary sectors of the product life cycle that require test: R&D, design validation, and manufacturing.

The earlier a product weakness is discovered, the less expensive the consequences. That’s one reason why the role of test changes with the stage of the product life cycle. When a product is first developed, the role of test is to verify that the design concept is viable. This calls for quick measurements, usually with hands-on use of discrete test instruments. Sometimes there is a need to load measurement data into an Excel spreadsheet for use in a lab report or for further analysis.

Excel is the most common software analysis tool for the R&D engineer. The connection is usually simple: a PC connected via GPIB or USB to an instrument or a small set of instruments. Simple software, such as Agilent Intuilink, finishes the connection.

Textual software generally provides an effective programming environment for manufacturing test, as it enables the engineer to extract the highest throughput from the test system. In manufacturing, repeatability and reliability become paramount concerns. Again, if the same equipment can be used for all three test situations (R&D, design validation, and manufacturing), then the R&D engineer can more readily assist with any problems that may arise during manufacturing test.

The process of designing and integrating systems used for electronic test requires more than simply coding instrument commands to automate the measurements made on the R&D bench. The instruments are only one part of the complete test system; cables, software, test-plan documentation, and fixturing are equally important. The latter are especially prevalent in a manufacturing environment.
Test-system considerations

There are many factors to consider when developing a test system. The three main driving factors are test requirements, development time, and test cost. The factor that is most important will drive the other two. For example, if the test requirement is for a very accurate measurement, as in R&D or design validation, you must be willing to take a bit more time and spend more to achieve the required accuracy. On the other hand, the manufacturing manager would not be pleased if the test system were to perform more tests than required, or perform them at a higher-than-needed level of accuracy, due to the obvious impacts on test-system cost and throughput.

Before the process to design a test system can begin, you must have a good understanding of the test application. This goes beyond simply understanding the device you are testing, as you must also be aware of other factors such as the skill level of the test system operator, the operating environment, and any standards requirements.

Planning your test system

Creating a comprehensive test plan allows you to take a big-picture view of the project and forces you to focus on meeting the objectives and requirements for the test system. The result is a considerable time saving in the development process.

Even in the R&D environment, there are times when it is useful to create a test plan, so that you can document and compare results after each design cycle. You must also consider the future for any test system you create today. It may be reasonable to create a dedicated and somewhat inflexible test system on some high-volume projects, but it is usually more appropriate to create a system that has the flexibility to adapt to future needs.

The test plan describes more than just the requirements of the DUT. It should also cover other areas of the test such as the level of experience required of the test system operator, calibration and maintenance requirements, physical limitations, and throughput requirements.

So the first step in creating a test system is to seek out and compile all the information needed to create an overall test plan. Important information includes:

- functional and parametric tests to be performed
- DUT design validation criteria
- format and usage of test results, including sharing data throughout the enterprise
- number of tests
- DUT pin counts
- physical constraints such as size, environment, and available power
- heat buildup and power dissipation
- how the test system will be verified, maintained, and calibrated
- RF environment
- accuracy and resolution requirements
- throughput goals
- development time constraints
- software-development and runtime environment
- cost constraints
- continuity constraints with existing legacy systems

Among the decisions involved in determining the design of a test system, the most obvious is what it is you must test. This is usually defined in a test specification. The test specification should include a complete list of the product functions to be verified, operating parameters to meet, and any regulatory standards to adhere to.

Accuracy

System accuracy is a critical specification of any test system, and the overall test plan should include both the accuracy requirements of the test and the recommended margin. As a minimum, the test equipment should have twice the accuracy specified for the DUT. To maintain this margin requires that the operating temperature be maintained closely and that calibration cycles be followed faithfully.

Often, it is more cost effective to buy test equipment with a 10X accuracy margin so that calibration and maintenance requirements can be relaxed without affecting accuracy. In the “10X” case, you may even increase the product yield, since the product can come closer to its specification tolerance limits because you can count on the accuracy of the test system. Whatever the accuracy required, you must have confidence that you can rely on the results.

Obviously, a calibration and maintenance plan is important for achieving the required test accuracy.
When determining instrument requirements, resolution must be specified as well as accuracy. Accuracy defines how close a measurement agrees with a standard value. Resolution indicates the smallest change that can be measured. There may be times when the absolute accuracy over an extended period is not as important as the resolution to measure small changes over the short term. Switching, fixturing, and cabling also add noise and crosstalk that can increase uncertainties.

**Throughput**

Throughput requirements will direct the necessary system capacity. Throughput is normally more important in the manufacturing environment than during design validation and rarely a concern in R&D. However, some complex designs require lengthy testing to be validated before going into production. A significant delay during R&D or design validation can cause a product launch to be delayed, and be costly in terms of missed market opportunity.

Downtime seriously degrades test-system throughput and can have a significant impact on product shipments. Predicting and preparing for wear-out mechanisms can reduce downtime. Further, using diagnostics or built-in test can help determine when the test system is about to fail. Such preventative maintenance procedures can result in big savings when they identify a test system failure before many DUTs are erroneously tested. In all cases, whether in R&D, design validation, or manufacturing, you should consider how you will handle downtime, either with spare test equipment or with a known path to repair or rental.

The overall test plan is a good place to describe what diagnostics the test system will require. It is easy to overlook test-system diagnostics as time consuming and costly to develop. Diagnostics are an important tool for maintaining throughput by reducing the downtime to repair failures. On most systems, a well-thought-out diagnostics approach will shorten test-system deployment time as well.

Developing and following a calibration and maintenance plan in conjunction with the diagnostics is another way to prevent system failures that disrupt test-system throughput.

**Results**

All tests must produce results. Sometimes this is merely a simple pass/fail indication, but often test results must be analyzed and archived. These requirements must also be defined in the overall test plan. If the test sequence is short, a few minutes or so, it is simpler to perform all data analysis after the test is over. However, if the test sequences are lengthy, some intermediate data analysis is recommended so that failing functions can be detected early enough to halt the test and avoid wasted time.

**Hardware/software decisions**

Once the requirements of the test system have been established in the test plan, then it is time to outline the design of the test system itself. The question is: What to consider first—software or hardware? In the past, the hardware provided the lead in test-system development. The test instruments that met the accuracy and throughput requirements were defined first, and then software was created to automate the test system.

But today, software can often be more expensive to develop than the cost of the hardware, so if test system cost is a driving factor, it is important to make sure that a new system can use as much existing software as possible.

The choice of programming languages may be based primarily on the experience of the programmer. Some find graphical languages such as LabVIEW or Agilent VEE easy to use. Others believe that textual languages such as C++ or Visual Basic are easier to use, especially for complex test programs. If it is important to use existing textual test code, then a multi-language development environment like Microsoft® Visual Studio .NET is a definite advantage. For a thorough examination of test-system software options, see the application note *Test System Development Guide: Choosing the Test-System Software Architecture (AN 1465-4).*

In any case, it is critical to ensure that drivers exist for the selected equipment. If the required drivers and support are not available, the anticipated advantages provided by the selected language may not materialize. Driver issues are discussed in detail in the application note *Test System Development Guide: Understanding Drivers and Direct I/O (AN 1465-3).*

**Control decisions**

A major consideration for a test system is the level of automation to build into the system to control the test process. Manual control requires that a human operator make all of the test connections, set the instruments, and then record the data. Increasingly, even in simple R&D setups, most engineers prefer to use instruments under the control of a PC in order to have a record of the test.
Once the testing becomes more complex or repetitive, a fully automated test system is in order. A fully automated test system takes care of signal switching, measurement, recording, and even analysis of the results for pass/fail determination. Once the DUT is in the test fixture, the test system takes over and runs all of the tests. This is the ultimate in terms of test speed, reliability, and repeatability, but it is also the most expensive and time consuming to develop.

The type of control, either manual, semi-automated, or fully automated, should be determined early as it will influence which instruments you select. As shown in Table 1, many factors will affect which control method is most suitable for your application:

- cost
- volume of tests
- test speed
- future uses
- upgrade path
- operator experience

Table 1. Comparison of test system control options

<table>
<thead>
<tr>
<th></th>
<th>Manual</th>
<th>Semi-automated</th>
<th>Automated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instrument cost</td>
<td>Depends. Can be higher than automated, since R&amp;D typically needs more accuracy than production specs.</td>
<td>Similar to manual.</td>
<td>Depends on requirements. If space is paramount, cardcages can be used, but they are typically more expensive than standalone rack &amp; stack instruments.</td>
</tr>
<tr>
<td>Development cost</td>
<td>Very low, just hook up and go</td>
<td>Low or high depending upon how much is automated</td>
<td>High</td>
</tr>
<tr>
<td>Operator experience</td>
<td>Very high, often experienced engineers</td>
<td>High as the manual portions of the system may require an engineer</td>
<td>Low</td>
</tr>
<tr>
<td>Development time</td>
<td>Low</td>
<td>Low to high</td>
<td>High</td>
</tr>
<tr>
<td>Flexibility</td>
<td>High, changes can be made easily.</td>
<td>Medium, some portions can easily be changed.</td>
<td>Low, changes require significant effort.</td>
</tr>
<tr>
<td>Throughput</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>Repeatability</td>
<td>Varies with expertise</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>System calibration</td>
<td>Rare. Usually only each instrument is calibrated.</td>
<td>Some system calibration may be possible.</td>
<td>Full system calibration is possible.</td>
</tr>
<tr>
<td>Self-check diagnostics</td>
<td>Individual instruments only, not system diagnostics</td>
<td>Individual instruments only, not system diagnostics</td>
<td>Common</td>
</tr>
<tr>
<td>Ease of instrument reuse</td>
<td>High</td>
<td>Medium</td>
<td>Low if card cage, medium if stand alone instruments</td>
</tr>
<tr>
<td>Potential for human error</td>
<td>High</td>
<td>Medium</td>
<td>Low</td>
</tr>
</tbody>
</table>

Manual control

A test system based on manual control depends entirely on the operator for all test functions (Figure 1). Connections between the DUT and instruments are made manually with test leads or cables. R&D engineers may follow procedures that are completely undocumented, but when using a manual control system for other test requirements, each instrument is normally manually operated by following a documented procedure. The results of each test are then manually recorded. This is a very
flexible approach as it allows changes to the test system to be made very easily. On the other hand, it is a very slow method of testing and has significant problems with repeatability. For example, the engineer may make readings one time with the voltmeter at full scale, while the next reading might be at 1/10 of full scale, resulting in a slightly different answer.

Manual control is often the least expensive test-system control option to set up, since it may not include such items as a system switch, expensive software, or test fixtures. Also, the time and cost required to set up the test are very low. However, the instrument cost for manual control varies. Often, the R&D application calls for a more accurate measurement than the equivalent measurement needed in manufacturing and therefore requires rather expensive instruments.

The cost to conduct the test is usually very high. Manual control generally requires a skilled operator to follow the labor-intensive test procedures. System self-testing is almost impossible, and complex and frequent calibration is often required due to the high accuracies needed. Typically, only the individual instruments are calibrated and not the entire system. As a result, inexperienced engineers may believe that the overall system accuracy is better than it actually is.

Repeatability is a concern with manual test systems. There are many opportunities for operator error to go unnoticed. These errors creep in when the operator is attaching cables, setting instruments, recording results, and even when transferring the results to other documents.

Even with these limitations, the manual approach can be useful. With due diligence while conducting the test and techniques such as using the same cables to increase repeatability, the manual approach can produce reasonably reliable results. Another advantage of manual control is the ease in which the test system can be reconfigured or the instruments used for other projects.

Additionally, a skilled engineer conducting the tests is constantly comparing the results against expectations, thereby providing a form of continuous verification of the test system. An incorrectly operating fully automated test system could continue to test for hours, days, or even weeks without detecting the problem, resulting in the shipment of incorrectly tested products.

Use manual control when:

- cost of automation outweighs benefits
- speed of test is not critical
- test requirements may change regularly
- the delay to create an automated system is unacceptable
- skilled operators are available
- the instruments need to be easily disassembled for use elsewhere

Semi-automated control

Semi-automated control is the most common type of control approach used for test systems, and is useful in R&D, design validation, and manufacturing test (Figure 2). Test systems using this control approach have manual portions for flexibility where it is needed and automation where it makes sense. Those sections of the test system that are expected to change often or would be too expensive to automate can be manual. Those sections that will not change or would benefit from automatic data recording can be automated.
A semi-automated test system might require the operator to manually connect the DUT, provide instructions to the operator for the procedural steps, and automatically record the results. For example, a semi-automated system might have an oscilloscope and an RF source that are under computer control, with a power supply under manual control. The engineer would vary the voltage to the DUT via the power supply, run a set of tests at this voltage level, and then manually change the voltage and run another set of tests.

Semi-automated control is often much faster than manual control and produces a more repeatable and reliable result. This method of control can take advantage of simplified software development with Agilent’s VEE or Visual Studio .NET for quickly creating the required automation.

The most common type of test equipment includes a fully functional front panel and a computer interface that allows both manual and automated use. This is a major benefit, even when automating, as you can always go back to a manual approach if you need to measure other parameters, troubleshoot the system, or conduct an experiment. These standalone instruments are beneficial when developing a fully automated test system for manufacturing as it is common to start with a semi-automated system and then increase the level of automation as experience and production volume increases.

Use semi-automated control when:

• automation benefits will outweigh added costs
• test volume does not require full automation
• some flexibility in the test system is required
• reasonably repeatable results are required
• skilled operators are available or close by
• a move to full automation is anticipated but not yet required

Automated control

Fully automated test systems are the domain of complex design validation testing or the manufacturing test environment (Figure 3); they are rarely used in R&D. All of the instruments, signal switching, and connections to the DUT are controlled by computer. In some automated test systems, an operator may be required to manually install the DUT into a test fixture as a single action, but others have an automated handler to insert and remove the DUT from the test fixture.

Full automation is the most expensive control method in terms of software development time, but it also results in the highest throughput and most repeatable and reliable measurements by nearly removing the human-error factor from the test. The skill level required of the operator is usually much reduced.

Full system calibration and diagnostics are easier to implement in an automated system where software can reconfigure the test system to allow it to test and calibrate itself against an external traceable reference. Full system calibration can even calibrate the cables and connections instead of just the individual instruments.

Figure 3 A fully automated test system requires minimal operator interaction.
There must be compelling reasons to justify an automated test system. Not only is the initial development cost high, but any changes or upgrades to an automated system can be very expensive. The compelling reason for the expense is usually the high-volume requirements of manufacturing test, but there are times during R&D and design validation when the required accuracy is very high or the test is very complex, making it necessary to automate the test to remove potential human errors or speed up the test process.

Use fully automated control when:

- high-volume manufacturing requires automation
- reducing test time is critical
- test requirements are known and stable
- cost per test outweighs test-system development cost
- time is available for development
- skilled operators are not available
- accuracy or complexity requirements dictate automation

**Planning for the future**

When making test-system design decisions, you should keep future needs in mind. Upgrades are a fact of life for a test system. They can be very expensive and time consuming but are often unavoidable. Naturally, any upgrades must justify the expense and effort required. Some of the reasons for upgrades are to:

- accommodate changes in design of the DUT
- conduct additional tests
- obtain higher accuracy
- obtain higher throughput
- eliminate redundant tests
- rearrange the test sequence to detect failures earlier
- improve analysis
- automate more of the test
- decrease the skill level required to operate the test system
- replace obsolete equipment
- change reporting requirements
- upgrade the operating system
- conform to new standards
- add newly developed models
- repeatability is important

A few moments considering the future can have a significant impact on future options. For example, when selecting instruments for a manual system, there is usually very little added cost to select instruments that have computer interfaces. You may not need the interface today, but computer control is not possible without it (and could be costly, difficult, or even impossible to add at a later date).

Using open standards will increase the likelihood that test system components will be useable in the future. Proprietary interfaces have a habit of disappearing or not supplying the drivers you need for future software options. Using proprietary measurements made by specific equipment in a test system from manufacturers that do not supply future upgrade paths could make an entire test system obsolete if that exact instrument is no longer available.

Following proper software design techniques resulting in well-written software that is easily understood, maintained, and modified is an obvious requirement for future upgrades. Good documentation is also critical to the future of a test system: Chances are you will not be the one that is tasked with future modifications.

**Conclusion**

Although test-system development is a complex task that can include many aspects of electronic and mechanical design, following a systematic approach and partnering with quality test equipment manufacturers will enable you to enhance your success while lowering the cost and time it takes to create the test system.
Case study: testing power supplies

This case study is an example of how a test system can evolve from R&D to design validation to manufacturing. Many of the same instruments are used in all three areas with the major difference being the type of control used. This is a common practice as the knowledge gained in each phase of product development is transferred to the next.

Manual control

When developing a product such as a power supply, the R&D engineer will create a test system as required to explore options and verify results. The test bench in Figure 1 is typical of such use. Many instruments are within reach and it is easy to rearrange them as needed. All of the connections to the DUT are made manually and each instrument is manually operated. This is an example of a test system with manual control.

The flexibility to quickly move from measurement to insight to next measurement, whatever that next measurement might be, is obvious. Standalone test instruments readily lend themselves to this usage model. The high level of skill required of the operator is also important. There is significant opportunity for error and confusion with a manually controlled system. R&D engineers are in their element at such a bench, but it falls short on reliability and repeatability when compared to other control methods.

The block diagram of Figure 4 shows the interconnection of the instruments for some of the tests used during the R&D phase of power supply development. Some of the standard tests measure output-voltage accuracy, output noise, load regulation, line regulation, and output programming speed.

Figure 4 Block diagram of a manually controlled test system used for R&D
The test system diagrammed in Figure 4 is just one example of a manual setup for testing some aspects of the design. Other R&D engineers would have other manual setups on their benches to test for other parameters. In this case, the total R&D manual test system is actually distributed throughout the benches of the entire design team.

More-specialized tests will also be conducted at this stage. Loop gain (Bode plot) is used to evaluate the stability of the control loops used to regulate the output voltage and current of the power supply. Load transient response is measured by applying a load-current step change and monitoring the output voltage on the scope, also giving insight into the stability of the control loops. Voltage and current stress on the components are also measured so power can be calculated to ensure that no parts are over stressed. The temperature of individual components may also be measured.

As these measurements are made, the test system is rearranged, the cables are attached as required, the instruments are manually controlled, and the results are noted. Often, the exact configuration is not recorded, making an exact repeat of the measurement difficult. The cable connections are often made with probes and clip leads in a manner that is quick but not reliable. Even so, the advantages to a skilled operator far outweigh the problems associated with manually controlling a test bench (Figure 1).

Semi-automated control

The design is “complete.” Now it needs validation, so the test requirements are somewhat different. In this case, the same instruments are used, but a computer is added for semi-automated control. The block diagram of Figure 4 remains the same, but now a computer is connected to some of the instruments (Figure 2).

Many of the same measurements are made during design validation as were made during R&D. But now, more of them can be made to fully validate the design. For example, the output accuracy of the power supply under test can be checked at a variety of operating conditions. The input voltage, load current, and even the ambient temperature can be varied to ensure proper regulation of the output voltage and that the output noise is within requirements. The same tests can be conducted on multiple prototypes to ensure that the design is consistent across units. Further, these tests can be completed much faster and include automated data recording, enabling statistical analysis.

The repeatability and reliability of semi-automated control along with automated data gathering are a significant enhancement to manual control. By selecting instruments that include computer interfaces, automating portions of the test system is much easier. In many cases, the automation is merely a matter of having a computer perform the commands and read the results that were done by an operator.

Automated control

The move to a fully automated test system may require additional instruments, as shown in Figure 3. The computer now controls all of the instruments as well as the reconfiguration of the interconnections for various tests. The digital multimeter, scope, and loads are still used, but now switches are employed to connect the DUT to the instruments. As the tests are performed, the computer uses the switches as required.

The block diagram of Figure 5 includes connections to the DUT and measurements that test the power supply in the manufacturing environment. The number of tests performed may approach those conducted during R&D and design validation but they are normally not as thorough. Manufacturing tests are often performed only at one operating point that is considered to be a worst-case condition. This maximizes the amount of information gained about the DUT in the minimum time.

The speed, repeatability, and reliability of the fully automated system can be significantly better than that of other test system control methods. Also, the skill level of the operator can be less. But the time and expense to create the system and make any changes usually makes automated test systems only feasible for manufacturing uses.
Figure 5. Block diagram of a fully automated test system

Build your own test system

Let Agilent Technologies help you streamline system development and lower the true cost of test. Our world-class measurement expertise and support can deliver accurate results in a hurry. Agilent’s instruments are optimized for use in systems, and our open industry software and I/O standards take the hassle out of creating test code.
Related literature

Agilent 33220A 20 MHz Function/Arbitrary Waveform Generator,
Data Sheet, pub. no. 5988-8544EN

Test-System Development Guide:
- Introduction to Test-System Design
  (AN 1465-1) pub. no. 5988-9747EN
- Computer I/O Considerations
  (AN 1465-2) pub. no. 5988-9818EN,
- Understanding Drivers and Direct I/O
  (AN 1465-3) pub. no. 5989-0110EN
- Choosing Your Test-System Software Architecture
  (AN 1465-4) pub. no. 5988-9819EN
- Choosing Your Test-System Hardware Architecture and Instrumentation
  (AN 1465-5) pub. no. 5988-9820EN
- Understanding the Effects of Racking and System Interconnections
  (AN 1465-6) pub. no. 5988-9821EN
- Maximizing System Throughput and Optimizing Deployment
  (AN 1465-7) pub. no. 5988-9822EN
- Operational Maintenance
  (AN 1465-8) pub. no. 5988-9823EN
- Using LAN in Test Systems: The Basics
  (AN 1465-9) pub. no. 5989-1412EN
- Using LAN in Test Systems: Network Configuration
  (AN 1465-10) pub. no. 5989-1413EN
- Using LAN in Test Systems: PC Configuration
  (AN 1465-11) pub. no. 5989-1415EN
- Using USB in the Test and Measurement Environment
  (AN 1465-12) pub. no. 5989-1417EN
- Using LAN in Test Systems: Applications
  (AN 1465-14) (available in February 2005)

To discover more ways to simplify system integration, accelerate system development and apply the advantages of open connectivity, please visit the Web site at www.agilent.com/find/systemcomponents. Once you’re there, you can also connect with our online community of system developers and sign up for early delivery of future application notes in this series. Just look for the link “Join your peers in simplifying test-system integration.”

Agilent Email Updates

www.agilent.com/find/emailupdates
Get the latest information on the products and applications you select.

Agilent Open Connectivity
Agilent simplifies the process of connecting and programming test systems to help engineers design, validate and manufacture electronic products. Agilent’s broad range of system-ready instruments, open industry software, PC-standard I/O and global support combine to accelerate test system development. More information is available at www.agilent.com/find/openconnect.

By internet, phone, or fax, get assistance with all your test & measurement needs

Online assistance: www.agilent.com/find/assist
Phone or Fax

United States: (tel) 800 829 4444
(fax) 800 829 4433

Canada: (tel) 877 894 4414
(fax) 800 746 4866

China: (tel) 800 810 0189
(fax) 800 820 2816

Europe: (tel) (31 20) 547 2111
(fax) (31 20) 547 2390

Japan: (tel) (81) 426 56 7832
(fax) (81) 426 56 7840

Korea: (tel) (82 2) 2004 5004
(fax) (82 2) 2004 5115

Latin America: (tel) (550) 752 5000

Taiwan: (tel) 0800 047 866
(fax) 0800 286 331

Other Asia Pacific Countries:
(tel) (65) 6375 8100
(fax) (65) 6838 0252
(e-mail) tm_asia@agilent.com

Microsoft is a U.S. registered trademark of Microsoft Corporation.

Product specifications and descriptions in this document subject to change without notice.

© Agilent Technologies, Inc. 2004
Printed in the USA December 28, 2004
5988-9747EN
This application note is part of the Test-System Development Guide series, which is designed to help you quickly design a test system that produces reliable results, meets your throughput requirements, and does so within your budget.

The first application note in the series, *Introduction to Test-System Design*, covers test-system philosophy and planning and discusses how test is used in three sectors: R&D, design validation and manufacturing.

This application note, *Computer I/O Considerations*, is the second in the series. It describes the advantages of using computer-industry standard I/O and explores the advantages and disadvantages of GPIB, USB and LAN interfaces for rack-and-stack test systems.

See the list of additional application notes in the series on page 9.

Table of contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>2</td>
</tr>
<tr>
<td>Proprietary I/O versus industry-standard I/O</td>
<td>2</td>
</tr>
<tr>
<td>GPIB interfaces</td>
<td>3</td>
</tr>
<tr>
<td>USB interfaces</td>
<td>3</td>
</tr>
<tr>
<td>Configuring USB systems</td>
<td>4</td>
</tr>
<tr>
<td>Simple connections</td>
<td>4</td>
</tr>
<tr>
<td>LAN interfaces</td>
<td>4</td>
</tr>
<tr>
<td>Connection methods</td>
<td>5</td>
</tr>
<tr>
<td>Remote access</td>
<td>5</td>
</tr>
<tr>
<td>Instrument communication and operation over LAN</td>
<td>6</td>
</tr>
<tr>
<td>Which I/O interface should you use?</td>
<td>6</td>
</tr>
<tr>
<td>Real data rates</td>
<td>6</td>
</tr>
<tr>
<td>Comparing costs</td>
<td>7</td>
</tr>
<tr>
<td>Ease of implementation</td>
<td>8</td>
</tr>
<tr>
<td>Conclusion</td>
<td>8</td>
</tr>
<tr>
<td>Glossary</td>
<td>9</td>
</tr>
</tbody>
</table>

Agilent Technologies
Introduction

Whether you plan to use your rack-and-stack test system for R&D, design validation or manufacturing, you are likely to program and control your system with a PC. For decades, the IEEE-488 bus, commonly known as the general-purpose instrumentation bus (GPIB), has been the standard interface for connecting test instruments to computers and for providing programmable instrument control. GPIB is still a common and useful technology, but now other I/O options are available. This application note explores the various I/O options and helps you decide which interfaces make the most sense for your test system.

Proprietary I/O versus industry-standard I/O

Most of today's PCs offer built-in Universal Serial Bus (USB) and Ethernet-based local area network (LAN) connections. These industry-standard PC I/O technologies are much faster than previous PC I/O technologies such as RS-232, and therefore are much more suitable for automating and controlling test-and-measurement instruments. IEEE-1394, or FireWire interfaces, while not as ubiquitous as USB and LAN ports on today's computers, also are readily available.

Using these industry-standard interfaces for communicating with your test instruments can save you time and money and reduce headaches as you build your test system. Some benefits of using industry-standard I/O are immediate and obvious. For example, with USB, you don't have the additional expense of purchasing an I/O card, and you don't have to dismantle your PC to install the card.

There are other less obvious advantages to industry-standard I/O as well. Because the computer industry employs thousands of engineers who work on improving the throughput rate and data integrity of these interfaces, they are likely to continue to improve more rapidly than proprietary interfaces. Using industry-standard I/O also makes it easy to interchange instruments in your system with instruments from a variety of manufacturers.

Proprietary interface cards, such as MXI-2 and MXI-3 from National Instruments are expensive, with typical price tags of about US$1,500. You have to open up your PC housing to install them. And if you don’t have an open expansion slot, you need to consider replacing your computer.

Because of the inherent advantages of industry-standard I/O and customer demand for it, instrument manufacturers have begun adding LAN and USB interfaces to their test equipment. For example, the Agilent 33220A arbitrary waveform/function generator, introduced in early 2003, includes LAN, USB and GPIB interfaces.

If you want to use your existing GPIB instruments in a rack-and-stack test system, you don’t necessarily need to use GPIB as your interface. Agilent also offers converters—USB/GPIB and LAN/GPIB—that allow you to use your GPIB-equipped test instruments with USB- or LAN-equipped PCs, eliminating the need to install a GPIB card in your PC. National Instruments also offers a FireWire/GPIB converter. In the next section, we will look at GPIB and the two main industry-standard interfaces, LAN and USB, and explore the applications where each is most appropriate. (FireWire interfaces are used primarily for VXI test systems. You will find more information about VXI in Application Note 1465-4, Test-System Development Guide: Choosing your Test-System Architecture and Instrumentation.)

GPIB interfaces

GPIB is the most common interface for programmable test-and-measurement equipment, and it is still one of the best choices if you want to maximize throughput for a variety of block sizes. GPIB is a parallel bus that includes control lines, handshake lines, and 8 bi-directional data lines—specifically designed for instrument communications and control. GPIB supports up to 14 devices that can be connected to your PC. You can use either a star or a daisy-chain configuration for connecting multiple instruments (see Figure 1), but cable length is limited to 2 meters (times the number of devices) up to a maximum length of 20 meters.

Figure 1. You can configure a GPIB bus in either a daisy-chain or star topology, or you can intermix these two configurations.
You can achieve data transfer rates of more than 500 KB/s on a GPIB bus if you limit bus cable length to 1 meter (times the total number of devices), up to a maximum length of 15 meters. Longer cable lengths reduce the maximum data transfer rate to less than 500 KB/s.

When you use GPIB, configuring the instrument I/O bus is a relatively easy task. However, each instrument on the bus needs to have a unique address. This requirement means you may have to manually change an instrument’s address when you configure your system.

GPIB has other drawbacks, too. GPIB cables and connectors are rather large and bulky and they are relatively expensive. And because GPIB isn’t a standard built-in PC interface, you have to open your PC housing and install an interface card in one of your PC’s expansion slots.

To communicate with instruments over GPIB, you need to install an I/O software package. Plug and Play drivers, IVI-COM drivers, and VISA/SICL (Virtual Instrument Software Architecture and Standard Instrument Control Library) are examples. These packages support popular languages such as C and C++, Microsoft® Visual Basic 6.0, Visual Basic .NET, National Instruments’ LabView, and others.

**USB interfaces**

USB was originally intended as an alternative to the RS-232 serial interface and the Centronics parallel interface, an older and still widely used standard I/O interface for connecting printers and certain other devices to computers. USB is suitable for a range of computer peripherals, from slow devices, such as mice and keyboards, to high-performance devices such as scanners, printers, and cameras. Now, USB is finding its way into test-and-measurement instrumentation, too.

USB is a serial interface bus that includes two power wires and a twisted pair to carry data. USB is capable of data transfer rates of about 12 Mb/s for v1.1, and up to 480 Mb/s for v2.0. In addition, v2.0 is fully backward-compatible with v1.1. The only difference is the data transfer rate. If you decide to use USB v2.0 to obtain faster throughput, you need to be aware that the USB bus cannot transfer data faster than the specified throughput rate of the test instrument to which it is connected. For most GPIB-based instruments, the throughput rate will not exceed the USB v1.1 rate. In other words, using USB v2.0 will not increase the throughput speed if the instrument is not able to provide throughput at the faster rate. Also see the “Gating factors on data rates” sidebar on page 6.

USB is capable of supporting up to 127 devices on a given interface. If you use a GPIB-based system, you must ensure that instrument addresses are unique, but USB provides this function automatically. When USB devices are manufactured, they are given unique identifiers based on the manufacturer, the instrument serial number, and the product number. When the device is powered up and connected to a controller, the controller detects its presence automatically, and if the host-side software drivers are loaded, the instrument will be ready to communicate on the bus. USB devices also are “hot swappable,” which means you don’t have to shut down your PC to plug in or unplug an instrument.

With USB, the computer schedules and initiates all transactions. If you are using a Windows NT® operating system, you will find that it does not support USB connections.

**Configuring USB systems**

USB cables and connectors are considerably smaller than their GPIB counterparts. However, device-interconnect configurations for USB are somewhat different from those usually seen in GPIB-based systems. Many USB instruments are equipped with a single USB connector, so you cannot daisy-chain multiple devices together. Instead, you need to use a hub to connect the devices to your computer, as shown in Figure 2. Not all test-and-measurement USB drivers are designed to work with hubs, so it is a good idea to check with the manufacturer.

Hubs provide expansion capability for USB, permitting multiple devices to be connected to a single USB port. These hubs are transparent to a controller, and you can cascade them.
up to five deep. Using hubs in your system offers several advantages. For example, many USB hubs include LED status lights that indicate which port is connected. Also, a hardware failure at the interface to one instrument, such as a shorted line, is unlikely to cause an entire bus to fail. This makes troubleshooting an I/O interface fault in a large system with many instruments a much easier task than having to disconnect each device in turn, as required in a GPIB-based system.

Simple connections
Connecting USB instruments to a PC controller is also a simple task. USB is especially useful with laptops, since typically they do not have the PCI slots required to install GPIB PCI cards. Most of the PCs produced within the last few years have several USB ports already built in (these will probably be USB v1.1-compliant). PCs that support USB 2.0 are beginning to appear in the marketplace; some even have ports on both the rear and the front of the PC as a convenience. For those PCs that don’t have built-in USB ports, PC interface cards for both USB v1.1 and USB v2.0 are available from several manufacturers. If you use USB v2.0, you will want to use either the Windows 2000 or Windows XP operating system, since both provide support for USB v2.0.

As with GPIB, communications with instruments via USB requires the installation of an I/O software package. Plug and Play drivers, IVI-COM drivers, VISA/SICL, and IntuiLink software—supporting C/C++, Visual Basic 6.0 and Visual Basic .NET—are available with USB support.

LAN interfaces
You also can connect your test-and-measurement instruments to a PC via a LAN interface. Ethernet LANs are almost universally available at industrial and commercial sites, and most PCs found in these facilities are already connected to a LAN. As a result, Ethernet-based LAN interfaces for test equipment are likely to become even more common than USB connections. Ethernet-based LANs commonly support data rates of 10 Mb/s to 100 Mb/s, and some even operate at up to 1000 Mb/s.

USB and LAN interfaces share a number of features. They both operate in serial mode, and both use relatively small and inexpensive cables and connectors (especially when you compare the connector costs to those of GPIB).

You will probably need an Ethernet hub to interconnect multiple LAN instruments in a test system. But, Ethernet hubs are readily available today—and are relatively inexpensive. Many provide network status, or activity indication with a series of LEDs.

Ethernet-based LAN devices typically need to be configured to operate properly on a network. However, instruments that support Dynamic Host Configuration Protocol (DHCP) provide the capability for test instruments to configure themselves automatically to operate on a network—if these services are available on the network. Unlike other interconnection methods, using LAN interfaces requires knowledge of computers and the Local Area Network connections. You may need help from your local IT group to set up your LAN connection.

Connection methods
You can connect LAN-enabled instruments several different ways. They may be connected directly to a site LAN (a workgroup LAN, intranet, or enterprise LAN), or they may be connected to a private LAN.

In private-LAN configurations, your PC and your test instruments are connected to each other via a LAN, but they are not connected to a site LAN. The simplest private-LAN configuration consists of a controller and only one instrument. See the first illustration in Figure 3. You also can connect multiple instruments in a private LAN, as shown in the second illustration in Figure 3.

Figure 3. Single and multiple instrument configurations can be connected to private LANs and site LANs.
If you plan to use your site LAN, rather than a private LAN, you need to be aware of two potential drawbacks:

1. Traffic on your site LAN can slow down your measurements.
2. If you are using a LAN interface for controlling your test system, it is possible that a faulty instrument could damage or disrupt the network, particularly when the instrument is turned on and tested for the first time. Controlling your test instruments via a private LAN is the safest approach, since it limits the range of potential disruption and access.

For all setups, you can connect instruments to the LAN either with a conventional LAN cable or through a wireless adapter. Wireless routers and hubs also are available, as are wireless USB-to-LAN interfaces. See Application Note 1909-3, Creating a Wireless LAN Connection to a Measurement System.

Remote access
A site LAN has the potential for permitting any controller on the LAN to access instrumentation—either intentionally or unintentionally. If the site LAN can be accessed from physical locations outside of your facility, then others can access your instrumentation. This open access can be a valuable asset because it lets you remotely control instruments and systems almost as easily as if they were next door. You can use remote access capability to diagnose system and instrument faults from faraway locations. Multiple engineers can share the expensive test instruments and systems from remote locations.

However, this open access also can be a disadvantage. For example, if the site LAN is connected to the outside world to provide Internet access, you face a serious risk of exposure to undesired system accesses. Firewall software and/or using a router which requires specific device addressing rather than a switch or hub can provide protection.

If you want remote access to your test equipment, but security and controlled access are a system requirement, then you need a lockout feature. Some instruments, such as the 33220A function/arbitrary waveform generator mentioned earlier, provide this feature via an Allow List.

An Allow List is a list of remote LAN addresses that are permitted to communicate with the instrument. Any controller that attempts to access an instrument whose address is not on the Allow List is rejected. This feature provides a level of system security for those instances where your system is connected to a site LAN and is at risk for inadvertent access.

Instrument communication and operation over LAN

Instrument communication over an Ethernet-based LAN requires a software driver package if I/O is to be performed via Plug-and-Play, IVI-COM or VISA/SICL. It’s also possible to use the TCP/IP's sockets or telnet to perform instrument I/O directly without a host-side driver. In fact, I/O operations using sockets provide the fastest data transfer rates, since the host-side driver is bypassed.

You can operate some LAN-enabled test instruments via a virtual front panel that appears on your PC screen. Typically, the display looks and acts like the actual instrument itself (see Figure 4), and you use your mouse to actuate buttons as if you were actually pushing front-panel buttons. The virtual instrument display mimics that of the actual instrument, that may be thousands of miles away.

Instrument communication and operation over LAN

Instrument communication over an Ethernet-based LAN requires a software driver package if I/O is to be performed via Plug-and-Play, IVI-COM or VISA/SICL. It’s also possible to use the TCP/IP's sockets or telnet to perform instrument I/O directly without a host-side driver. In fact, I/O operations using sockets provide the fastest data transfer rates, since the host-side driver is bypassed.

You can operate some LAN-enabled test instruments via a virtual front panel that appears on your PC screen. Typically, the display looks and acts like the actual instrument itself (see Figure 4), and you use your mouse to actuate buttons as if you were actually pushing front-panel buttons. The virtual instrument display mimics that of the actual instrument, that may be thousands of miles away.

Remote access
A site LAN has the potential for permitting any controller on the LAN to access instrumentation—either intentionally or unintentionally. If the site LAN can be accessed from physical locations outside of your facility, then others can access your instrumentation. This open access can be a valuable asset because it lets you remotely control instruments and systems almost as easily as if they were next door. You can use remote access capability to diagnose system and instrument faults from faraway locations. Multiple engineers can share the expensive test instruments and systems from remote locations.

However, this open access also can be a disadvantage. For example, if the site LAN is connected to the outside world to provide Internet access, you face a serious risk of exposure to undesired system accesses. Firewall software and/or using a router which requires specific device addressing rather than a switch or hub can provide protection.

If you want remote access to your test equipment, but security and controlled access are a system requirement, then you need a lockout feature. Some instruments, such as the 33220A function/arbitrary waveform generator mentioned earlier, provide this feature via an Allow List.

An Allow List is a list of remote LAN addresses that are permitted to communicate with the instrument. Any controller that attempts to access an instrument whose address is not on the Allow List is rejected. This feature provides a level of system security for those instances where your system is connected to a site LAN and is at risk for inadvertent access.
Which I/O interface should you use?

To decide which I/O interface or interfaces you use in your test system, you will need to consider many factors. These include data transfer rates and block sizes, and costs for cables, routers, hubs, and PC I/O cards. Other factors include I/O driver availability, and programming requirements, as well as the need for possible remote system access.

Keep in mind that you do not have to choose a single I/O interface. Systems incorporating multiple interfaces are particularly useful if you have a mixture of older GPIB instruments and newer instruments with other interfaces built in. Today’s advanced software tools that include VISA technology eliminate the need to talk to different kinds of I/O in different ways. A minor change to a single line of code is typically all that is required. However, do not mix interfaces on a single instrument—the input and output must be on a single interface—and make sure your software drivers know which instrument is using which interface.

To see an example system that incorporates multiple interfaces (RS-232, FireWire, USB, GPIB and LAN), see the application note Test System Development Guide: Choosing Your Test-System Architecture and Instrumentation (AN 1465-5).

Real data rates

You will notice that individual I/O bus specifications for data transfer rates usually give only the theoretical maximum transfer rate. The actual data transfer rate that can be achieved for any given system depends on a number of factors. These factors include PC microprocessor speed, PC software and driver overhead, I/O card hardware, and instrument-specific hardware and firmware.

These variables make it difficult to predict the actual data transfer rate that might be expected for any given system configuration. Table 1 shows a relative comparison of data transfer rates for several data block sizes among GPIB, USB v1.1, USB v2.0, and LAN interfaces. These data were compiled using the Agilent Model 33220A function/arbitrary waveform generator and a Hewlett-Packard Kayak PC with an 800 MHz processor running on a Windows XP operating system.

For small data-block sizes of a few hundred bytes, there is no appreciable difference in bus speed, but the higher-performance buses (USB v2.0 and LAN) demonstrate a marked improvement in the time required to transfer large data blocks.

The differences in data transfer rates between small and large data blocks for any given interface are largely due to variations in the latency, or software overhead, required for each of the interfaces prior to the start of the actual data transfer.

If you’re looking for high throughput in a test system, don’t be swayed by the perception that high-speed interfaces will always get you there. In most test systems, the use model is one of “Close a channel; measure a point”, then “Close another channel; measure another point.” In this case, block transfer rate is meaningless. What really counts is the time to make that first measurement. This fact is one of the reasons that GPIB has lasted so long as an interface. It is quite good at executing this use model.

Gating factors on data rates

The data rates of a test system are determined by the slowest device/firmware/software in the system.

For example:

1. A high-speed instrument with integrated LAN controlled with an older 485 computer will be limited by the computer processor speed and possibly memory depth.

2. A USB2 interconnect will operate at a USB1 rate if the instrument and computer do not also support USB2.

3. An Instrument with a data transfer rate of 33K bytes/second will not transfer data any faster with USB, LAN or a computer that is able to transfer data at 1M bytes/second.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Function change</th>
<th>Frequency change</th>
<th>4K arb</th>
<th>64K arb</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIB</td>
<td>99 ms</td>
<td>2 ms</td>
<td>20 ms</td>
<td>340 ms</td>
</tr>
<tr>
<td>USB 1.1</td>
<td>100 ms</td>
<td>4 ms</td>
<td>10 ms</td>
<td>185 ms</td>
</tr>
<tr>
<td>USB 2.0</td>
<td>99 ms</td>
<td>3 ms</td>
<td>8 ms</td>
<td>100 ms</td>
</tr>
<tr>
<td>LAN (socket)</td>
<td>100 ms</td>
<td>3 ms</td>
<td>8 ms</td>
<td>110 ms</td>
</tr>
</tbody>
</table>
For a detailed look at data transfer rates of two different block sizes over the various interfaces, see Application Note 1475-1, Modern Connectivity—Using USB and LAN Converters (see sidebar below). One of the benefits of having an instrument that supports multiple interfaces is the ability to easily compare the actual data transfer rate for each of the I/O interfaces in a given application. This permits you to select the interface that offers the optimum performance.

If the application program’s I/O calls are written with a driver interface that provides a common set of programming commands independent of the interface, such as Agilent’s VISACom, then it becomes a simple matter to direct the I/O calls to any of the three interfaces.

**Comparing costs**

Today, many companies are looking for ways to lower the cost of test. If this is true of your organization, implementation cost will be an important consideration in selecting an I/O interface for your test system.

New PCs typically have several USB ports built in, but GPIB and LAN interfaces usually require a card that you must purchase separately. GPIB cards typically cost several hundred dollars and LAN cards usually sell for US$10 to US$50.

Also, if you plan to use USB or LAN interfaces to connect multiple instruments in your system, you will need hubs. These hubs can cost from US$25 to US$100 each, depending on features and the number of ports they support.

You also need to consider the cost of the cables for your test system. GPIB cables are relatively expensive, ranging in price from US$25 to US$100 each, depending on their length. USB cables, on the other hand, range from US$8 to US$30. LAN cables are usually the least expensive, typically costing less than US$10. Some can be found for as low as US$3.

You can make useful cost comparisons by assuming that all test instruments are able to support any of the three interfaces and computing the interface cost for your proposed test system. Today, few test instruments actually do support all three, since the industry is just beginning to provide instruments equipped with multiple computer-industry-standard interfaces. However, the I/O interface converters mentioned earlier permit GPIB-only instruments to be connected to USB- and LAN-based interfaces. For example, the Agilent 82357A USB/GPIB interface enables your PC to communicate with GPIB devices via the PC’s USB port. Similarly, the Agilent E5810A LAN/GPIB gateway provides a means to connect GPIB devices to a LAN. (See sidebar and Figure 5.) These converters can save you the cost of replacing your existing GPIB test instruments if you decide you want to use industry-standard I/O. However, these converters are appropriate only for applications where measurement speed is not critical, as they do slow the data transfer rate.

Let’s look at an example of a test system designed to test the Agilent 33220A function/arbitrary waveform generator. The test system consists of a controller, a local printer, seven rack-and-stack instruments, a fully loaded 13-slot VXI mainframe, and support for testing three 33220A waveform generators.

![The Agilent E5810A LAN/GPIB Gateway and the 82357A USB/GPIB Converter.](image)

**Using USB and LAN I/O Converters**

If you are considering using an I/O converter, be sure to read Application Note 1475-1, Modern Connectivity—Using USB and LAN I/O Converters: What is the Best Input/Output Interconnect for Your System? This application note compares the Agilent 82350B GPIB PC card, the 82357A USB/GPIB converter, and the E5810A LAN/GPIB gateway in terms of controller and operating system requirements, set-up steps, data transfer rates, allowable distances from instruments to the PC, etc. These details will help you choose the best interconnection method for your application.

www.agilent.com/find/buildyourown
As Table 2 shows, GPIB is the most expensive scheme to implement. Even with the added costs of USB and LAN hubs, their reduced cable costs and higher overall speed performance makes them more attractive alternatives.

From a systems perspective, hubs also offer some I/O interface operational feedback that is lacking with GPIB systems. Also, the much smaller USB and LAN cables and connectors take up much less rack space, making system cabling easier. They also weigh less.

**Ease of implementation**

USB is the simplest I/O to implement, and GPIB is also relatively straightforward, as long as you don’t mind the hassle of opening your PC and installing an interface card. Using LAN interfaces typically requires more LAN knowledge and configuration effort. For many system developers, the advantages of LAN far outweigh the added development time required. You need to evaluate your own situation to decide if that is true for you.

For low-frequency test systems, Agilent offers the N1908A Test Automation Kit that simplifies the process of automating measurements. The kit handles driver issues, launches your preferred software environment and ensures reliable measurements. It is an inexpensive way to cut weeks off your system development time. The kit comes with an Agilent 82357A USB/GPIB converter, but the Test Express software also works well if you are using a LAN/GPIB gateway.

**Conclusion**

With the new generation of test instruments offering a choice of interfaces, you need to decide which interface is best suited for your test system. Comparing costs, data transfer rates and ease of implementation will help you choose the interface most appropriate for your application. For R&D applications, where the number of instruments in a system is usually small and a quick and easy interface setup is desired, USB is usually the best choice.

For design verification and manufacturing, USB 2.0 and Ethernet-based LAN are good choices, although LAN is typically the better of the two alternatives for larger systems because of its data-throughput performance, cost, remote access, and ease of system assembly.

The added flexibility, remote system access and control, performance on a par with USB, captive cable connectors (which aren’t found on USB), and the capability for wireless operation offered by the LAN approach can make LAN the most attractive choice for many systems applications.

**Get help configuring your I/O interfaces**

Configuring an interface to connect your PC to an instrument or system can be a daunting task for someone who is not well versed in the intricacies of PCs, I/O technologies, and I/O interface configuration. In the past, this was especially so for LAN-based I/O that required a system to be connected to a site LAN. Fortunately, step-by-step guides such as Agilent’s USB/LAN/GPIB Interface Connectivity Guide are now available to help you to configure your I/O interfaces. The USB/LAN/GPIB Interface Connectivity Guide describes in detail how to connect instruments to various interfaces, and how to configure your PC. It also includes programming examples and interface troubleshooting tips. You can view the guide at [http://we.home.agilent.com/upload/cmc_upload/connectivity_guide.pdf](http://we.home.agilent.com/upload/cmc_upload/connectivity_guide.pdf)

**Table 2.** Typical costs for GPIB, USB, and LAN interfaces

<table>
<thead>
<tr>
<th>Interface</th>
<th>Single instrument</th>
<th>12-instrument system</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIB</td>
<td>PCI card + cable</td>
<td>$500</td>
</tr>
<tr>
<td>USB 1.1</td>
<td>Cable</td>
<td>$10</td>
</tr>
<tr>
<td>USB 2.0</td>
<td>PCI card + cable</td>
<td>$60</td>
</tr>
<tr>
<td>LAN</td>
<td>PCI card + cable</td>
<td>$30</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interface</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIB</td>
<td>Ubiquitous interface on test instruments, Maximizes throughput for all block sizes</td>
<td>Expansion slot required, Must open PC housing to install card, Relatively expensive, Limited cable lengths permitted between computer and instruments</td>
</tr>
<tr>
<td>USB</td>
<td>Quick, easy setup, Low cost, Good data-throughput performance</td>
<td>Does not work with Windows NT</td>
</tr>
<tr>
<td>LAN</td>
<td>Good data-throughput performance, Low cost, Remote access makes it easy to control system from remote location</td>
<td>Requires LAN knowledge to set up</td>
</tr>
</tbody>
</table>
Glossary

C# (pronounced “C sharp”) — a new C++-like, component-oriented language that was built to run on the .NET Framework

Dynamic Host Configuration Protocol (DHCP) — a protocol for assigning dynamic IP addresses to devices on a network

FireWire — a high-speed serial bus defined by the IEEE 1394 standard

Hub — a common connection point for devices in a network. A hub contains multiple ports.

Interface — a connection and communication media between devices and controllers, including mechanical, electrical, and protocol connections.

IVI — interchangeable virtual instruments — a standard instrument driver model allowing you to swap instruments without changing software. Learn more at http://www.ivifoundation.org/

IVI-COM — IVI-COM presents the IVI driver as a COM object in Visual Basic.

Router — a device that connects any number of LANs and determines where packets go

SICL — Standard Instrument Control Library — software used for I/O application programming

VISA — virtual instrument software architecture

VXI — VXI is a standard, open architecture for cardcage test systems. The VXIbus (VMEbus eXtensions for Instrumentation) was developed by a consortium of test-and-measurement companies to meet the needs of the modular instrument market.

References

2. Connectivity Resources: www.agilent.com/find/ADN
3. Connection, communication, and control of test instruments from a PC: www.agilent.com/find/connectivity
4. Agilent IntuiLink: www.agilent.com/find/intuilink

For information on how to streamline system development and lower the true cost of test, please visit www.agilent.com/find/buildyourown

Other Agilent literature

Data sheets
• Agilent N1908A Test Automation Kit, pub. no. 5989-0000EN
• Agilent Connectivity Suite, pub. no. 5988-5756EN
• Agilent 82357A USB/GPIB Interface for Windows, pub. no. 5988-5028EN
• Agilent E5810A LAN/GPIB Gateway, pub. no. 5988-5810EN
• Agilent 82350A PCI GPIB Interface, pub. no. 5966-2720E
• Agilent 33220A 20 MHz Function/Arbitrary Waveform Generator, pub. no. 5988-8544EN

Application notes

Test-System Development Guide:
• Introduction to Test-System Design (AN 1465-1) pub. no. 5988-9747EN http://cp.literature.agilent.com/litweb/pdf/5988-9747EN.pdf
• Understanding Drivers and Direct I/O (AN 1465-3) pub. no. 5989-0110EN http://cp.literature.agilent.com/litweb/pdf/5989-0110EN.pdf


Understanding the Effects of Racking and System Interconnections (AN 1465-6) pub. no. 5988-9821EN http://cp.literature.agilent.com/litweb/pdf/5988-9821EN.pdf


Using LAN in Test Systems: Applications, AN 1465-14 (available in February 2005)
Application notes, continued

- **Simplified PC Connections for GPIB Instruments.**
  AN 1409-1, pub. no. 5988-5897EN
- **Creating a Wireless LAN Connection to a Measurement System.**
  AN 1409-3, pub. no. 5988-7688EN
- **Modern Connectivity—Using USB and LAN Converters.**
  AN 1475-1, pub. no. 5989-0123EN

Manuals

- **USB/LAN/GPIB Interface Connectivity Guide** available at

To discover more ways to simplify system integration, accelerate system development and apply the advantages of open connectivity, please visit the Web site at [www.agilent.com/find/systemcomponents](http://www.agilent.com/find/systemcomponents). Once you’re there, you can also connect with our online community of system developers and sign up for early delivery of future application notes in this series. Just look for the link “Join your peers in simplifying test-system integration.”

---

Agilent Email Updates
[www.agilent.com/find/emailupdates](http://www.agilent.com/find/emailupdates)
Get the latest information on the products and applications you select.

**Agilent Open Connectivity**
Agilent simplifies the process of connecting and programming test systems to help engineers design, validate and manufacture electronic products. Agilent’s broad range of system-ready instruments, open industry software, PC-standard I/O and global support combine to accelerate test system development. More information is available at [www.agilent.com/find/openconnect](http://www.agilent.com/find/openconnect).

By internet, phone, or fax, get assistance with all your test & measurement needs

**Online assistance:** [www.agilent.com/find/assist](http://www.agilent.com/find/assist)

**Phone or Fax**

**United States:**
(tel) 800 829 4444
(fax) 800 829 4433

**Canada:**
(tel) 877 894 4414
(fax) 800 746 4866

**China:**
(tel) 800 810 0189
(fax) 800 820 2816

**Europe:**
(tel) (31 20) 547 2111
(fax) (31 20) 547 2390

**Japan:**
(tel) (81) 426 56 7832
(fax) (81) 426 56 7840

**Korea:**
(tel) (82 2) 2004 5004
(fax) (82 2) 2004 5115

**Latin America:**
(tel) (55) 752 5000

**Taiwan:**
(tel) 0800 047 866
(fax) 0800 286 331

**Other Asia Pacific Countries:**
(tel) (65) 6375 8100
(fax) (65) 6836 0252
(e-mail) tm_asia@agilent.com

Microsoft, Windows, and Windows NT are U.S. registered trademarks of Microsoft Corporation.

Product specifications and descriptions in this document subject to change without notice.

© Agilent Technologies, Inc. 2004
Printed in the USA December 9, 2004
5988-9818EN
This application note is part of the Test-System Development Guide series, which is designed to help you quickly design a test system that produces reliable results, meets your throughput requirements, and does so within your budget. This application note answers common questions about the use of drivers and direct I/O to send commands from a PC application to the test instrument. It discusses how the driver came about, what the different software layers do in a system to help the instrument communicate to the PC, which drivers are compatible with various software languages and I/O software, and references for further study. See the list of additional application notes in the series on page 13.
Introduction

In a September 2001 survey, Test & Measurement World published a summary of engineers' worst headaches. Instrument drivers topped the list. Instrument manufacturers and various trade groups have been working on driver standards for some time, in an attempt to alleviate the frustrations of engineers who need to automate measurements and create test systems on a deadline. As a result of these efforts, we might expect finding and using appropriate drivers to be dramatically easier, but at the moment, complexities and incompatibilities are still troublesome.

This application note answers common questions about the use of drivers and direct I/O to send commands from a PC application to the test instrument. It discusses how the driver came about, what the different software layers do in a system to help the instrument communicate to the PC, which drivers are compatible with various software languages and I/O software, and references for further study.

With new insight into these topics, you should be able to choose, install and use drivers more easily and reduce the amount of time you spend getting your instruments and computer applications to talk to each other.

History

By computer standards, 1970 could be considered the mists of antiquity. That's when instruments were connected via imaginative schemes to devices resembling computers. One popular I/O format involved connecting a large cable to the instrument. Each line on the cable represented a function or range, and the line was simply grounded at the proper time. The device, say a voltmeter, would return a value using binary coded decimal (BCD) 1-2-4-8 format, or a quaintier 1-2-2-4 format. Needless to say, the programming syntax of instruments at this time was anything but standardized. However, since everything was hard-wired, the process was straightforward and immediate.

GPIB

In 1971, development began on a standard hardware interface. The idea was to be able to trigger multiple instruments at once and still allow both slow and fast instruments to "talk" on the same bus without conflict. The first products to use this bus were released in 1972. The same year this new bus was dubbed Hewlett-Packard Interface Bus (HP-IB). In 1975, IEEE adopted it as a standard with little modification, and IEEE-488 was born. A variant of the original interface is now popularly known as General Purpose Interface Bus (GPIB).

With GPIB and a desktop computer (actually at the time it was called a 'desktop calculator'), the need arose for a common communication language. Limited processing power in the 'calculators' demanded a simple syntax, so ASCII commands were chosen. A DMM might be sent what was affectionately termed "R2D2 code". Here's an example:

"F1R2T1"

The command means "Go to the dc volts Function, the 1 volt Range and Trigger a reading." Different manufacturers had unique ways to interpret the command strings, based on their instruments' capabilities. If you had to replace a product with one from another manufacturer, or even a new-generation product from the same manufacturer, it could mean completely rewriting the entire program. Later versions of IEEE 488 elevated the standard from being a hardware-only standard to one that also specified protocol.

Figure 1. Early instrument control utilized hard-wired commands.
SCPI

In 1989, seeing a need for more clarity and interchangeability that was available with simple ASCII, Hewlett-Packard introduced a programming language known as Test & Measurement Systems Language (TMSL). Within less than a year, nine T&M manufacturers had met to generate a universal approach to instrument control, using TMSL as the basis. The outcome was Standard Commands for Programmable Instruments (SCPI).

Today, SCPI is still the most-used form of instrument control. In SCPI, the instrument programming syntax became much more robust and predictable. SCPI defined a strict hierarchy, and every command was associated with a concomitant response. These were defined for source, sense and switch devices. Here’s an example of SCPI code:

```
CONF:VOLT:DC 0.3,0.003
```

This command tells the instrument to configure itself to get ready to read a 0.3 volt dc signal with 3-millivolt resolution. It should be obvious from this statement that SCPI commands require some intelligence on the other end of the wire, as not every voltmeter has a 0.3 V range. The commands need to be parsed by the voltmeter and this parsing adds a small layer of delay time to the system.

One advantage of SCPI is that the list of commands typically covers 100% of the instrument’s programmable functions, no matter how arcane. For a friendly tutorial on SCPI, go to: http://ftp.agilent.com/pub/mpusup/pc/iop/hpibtut/ib5_scp.html.

Figure 2. Compared to “R2D2” code, SCPI commands standardize programming and make life easier for the programmer. SCPI commands can access virtually any programming function in the instrument, but the parser does add small delays to the process.
**The I/O Software... SICL and VISA**

Instrument commands aren’t the whole story. It takes more “layers” of software to communicate with a computer. Before you send the instrument a command, you need to define the I/O path, route the information through the proper I/O card, find out where the instrument is on the bus and speak to the instrument in the syntax of the I/O you’re using. Assuming the GPIB I/O card in the computer is at address 7 and the DMM is at address 22 on the bus, the simple BASIC command might be:

```
ASSIGN @Dvm to 722

This tells the computer where to send the command

OUTPUT @Dvm;

“TRIG:SOURCE:INT”
```

sets the trigger source to internal

The above will work with a GPIB interface, but if you try the same thing using RS-232, the syntax is very different. Switching between GPIB and RS-232 would require rewriting some code.

**SICL**

That’s where Standard Instrument Control Library (SICL) I/O software comes in. SICL was developed by HP to make software as I/O-independent as possible. It adds a layer on top of the instrument code. The layer checks to see what I/O is used and alters the syntax accordingly. The code looks the same, regardless of I/O type. All you have to do is use one line of code to declare the I/O type at the beginning of the program.

SICL is not the only I/O software available today. **AGILENT VISA**, **NI-VISA** and **NI-488** and **VISA-COM** (from Agilent) perform similar functions. That’s a dizzying array of choices, so for now let’s concentrate on **VISA**. While SICL software was created to communicate with Agilent interfaces only, VISA was created to work industry-wide.

**VISA**

In the late 1980’s, there was a move to build standardized card cage instruments. This movement led to a software and hardware standard known as **VME Extensions for Instrumentation (VXI)**. Based on the VME standard, VXI made special modifications for software, shielding, triggering, power supplies and analog performance. VXI was adopted by hundreds of instrument manufacturers who produced a wide variety of plug-in cards. VXI’s interchangeability at the card level brought about the need for common I/O software, similar to HP’s SICL, but implemented as an industry-wide standard. Largely derived from the SICL library, the VISA syntax was born.

---

**Figure 3.** SICL I/O software reduces a test engineer’s programming burden by making it easier to change I/O types (USB, GPIB, USB, VXI, RS-232, etc) without recoding the program. SICL adds a software layer, which has a small effect on system speed.
Virtual Instrument Software Architecture (VISA), was created by the VXIplug&play Foundation to standardize I/O software across physical interfaces and between various vendors. In most cases, test systems are not solely VXI, but rather hybrids of VXI and Rack & Stack architectures, so it was not enough to create I/O software exclusively for VXI. For that reason, the VXIplug&play specifications were extended to include traditional standalone instruments as well as both types of VXI instruments.

Today’s two main suppliers of VISA are Agilent Technologies and National Instruments. (In 2000, the same people from HP Test & Measurement who were involved in instrument connectivity were split from HP in the new venture now known as Agilent Technologies.)

VISA I/O software uses common terminology and syntax to connect to and control instruments. A VISA library supports complete control of instrument across the physical interfaces GPIB, RS-232, USB, LAN and VXI.

The VISA library provides the capability of SICL, in a way that conforms to industry standards. A program written to work with Agilent’s VISA library will work with implementations of VISA from other vendors. For those accustomed to using SICL, Agilent’s implementation of VISA is provided along with its SICL libraries. (Since the introduction of VISA, programming based on the SICL library has gradually been phased out in favor of the industry-standard VISA library.)

To program a new test system, the test engineer installs the appropriate I/O library along with the application programming language. VISA was originally developed to be used with C and C++, but can also be called from any language that can call arbitrary Windows dynamic-link libraries (DLLs), including Microsoft® Visual Basic. Agilent provides header files to facilitate the use of VISA in Visual Basic .NET and C#. These can be downloaded from http://www.agilent.com/find/iolib.

Figure 4. VISA is the most popular form of I/O software. Drawing heavily on the work done for SICL, VISA was created to serve multiple T&M suppliers and be a universal standard. VISA-COM is a new variant of VISA.

---

1 VXI has two types of instruments, based mostly upon their local intelligence. “Message-based” cards can react to a high-level message, and usually have on-card parsing. “Register-based” cards are just what the name implies... cards that have directly-programmable registers. Message-based cards do more, but are inherently slower, since they must interpret complex commands.
PC industry adds language independence

As I/O development was proceeding in the T&M industry, the PC industry was making big strides in I/O-independence and language-independence. In 1994, Microsoft stated: “The Component Object Model (COM) is a software architecture that allows components made by different software vendors to be combined into a variety of applications. COM defines a standard for component interoperability, is not dependent on any particular programming language, is available on multiple platforms, and is extensible.”

In February, 2001, Microsoft introduced .NET, their 3rd generation of component technology. .NET has been applied to their integrated development environment, Visual Studio® .NET, as well as MS Office, other applications, operating systems and web services.

All this is well and good, but should the Test & Measurement industry embrace PC Operating Systems?

Detractors point out the frequent operating system upgrades in the PC industry relative to T&M languages. However, from Figure 5, it can be seen that COM, which is integral to .NET components, has been around longer than most T&M standards. It seems only logical to take advantage of the investments Microsoft has made to create this paradigm shift. With 3,000 engineers working for three years on the first version of .NET, Microsoft’s investment is twenty times that of the leading T&M language. Similar correlations apply to software. Visual Basic has over 6,000,000 users and C/Visual C++ has 1,000,000 users worldwide. This will result in an unprecedented body of software the average engineer will be able to leverage.

The most important immediate benefit for the test engineer is that, using Visual Studio .NET, engineers are reporting 20-30% less development time to create their test programs. They are delighted in their ability to pull in legacy code from languages such as C, Visual C++, VEE and Visual Basic into the .NET environment.

To incorporate this programming language independence, Agilent initiated a VISA-COM standard as a companion to the VISA specification. VISA-COM software makes VISA services available in a language-independent COM component architecture. What does that mean? It means not only are you free to pick from popular I/O configurations, but now you also have the freedom to choose from a list of software languages like C++, C# and VB.NET. With Agilent’s T&M Programmer’s Toolkit product acting as a T&M “face” for .NET, you can access all this from a single environment.

When using Agilent VISA-COM, you also need to install Agilent VISA. Agilent I/O libraries are shipped along with Agilent software and I/O products.

Figure 5. PC Software Overtakes T&M Software in interchangeability. The millions of people using Visual Studio software will afford the engineer an unprecedented pool of available intellectual property.

[Diagram showing increasing software interchangeability from 1980's to 2010's, with labels for T&M, PC software industry, Microsoft: COM, Microsoft: .NET components, VISA-COM components, IVI-C, VXIplug&play, SICL, and Microsoft: ActiveX.]

2 Dr. Dobb’s Journal, Microsoft Corp. December, 1994.
What is a driver?

It’s about time we explained what a driver is; after all, that’s the title of this application note. By now, we know this much: The computer has an operating system, say Windows® XP, under which there is an Application Development Environment (ADE) like Visual Studio.NET. Some language, say C#, is used to program commands for the instrument, and those commands are passed to the I/O software, which then passes them via a physical interface to the instruments’ internal microprocessor. The microprocessor decodes those commands using its internal I/O structure, and the instrument carries out the commands.

To make all this practical, you need to write some code. If you are a programmer, you must either memorize or look up the Direct I/O SCPI commands related to the particular instrument being programmed. If you intend to code in a proprietary language, then you need to know how those commands fit. For simple applications, this approach works well, but as application complexity increases, using direct I/O quickly can become difficult and time consuming. Programming a direct communication path usually requires you to know a specialized computer programming language and its programming environment, and be familiar with proper command sequences and interrelationships between commands. You also need to know how to load and configure various I/O libraries and parse instrument responses that may be in the form of binary data or screen graphics. Whether you have these competencies or not, when today’s product design cycles are measured in months rather than years, it doesn’t make sense to spend several of those months coding a new test system, unless very high volume production is the goal.

The driver is...  
The driver is a high-level, intelligent, instrument-specific or instrument class-specific piece of software intended to make programming simpler and shorten development time. In the T&M world, it facilitates communication to an instrument by guiding the user through the steps. Its user interface can take many forms. A driver could be a list that pops up when you hit the next “dot” in Visual Basic, or it could be as elaborate as a “panel driver” that displays a virtual front panel on the screen of your computer to help you set up the instrument.

Figure 6. Agilent’s T&M Programmers Toolkit using a VXIplug&play WIN32 power supply driver in VB.NET after being wrapped by the Driver Wizard.

Figure 7. A tiny but interesting program, written in VEE. With its intuitive interface, VEE is the fastest T&M graphical language to learn. Fill in the boxes, and the VEE panel driver generates code for you. See http://we.home.agilent.com/upload/cmc_upload/tmo/downloads/E206HPVEE_TESTENGR_EVAL.pdf
Even if you have never programmed an instrument in a test system, you have probably used a driver. Digital cameras, external hard drives and printers—all require a driver to talk to the PC. If you’ve upgraded a PC, you may have found that the old printer driver no longer works with the new operating system, and you need to go to the Microsoft website to find a new one. Or you may find that the printer doesn’t work exactly the same way it did under the old operating system. Similar issues exist in T&M equipment.

**Driver coverage**

A simple DMM may have only 25 commands, while a more complex instrument may have hundreds. You can imagine how expensive it is to write an intelligent driver that anticipates all the possible permutations of instrument setup, triggering, sourcing and measurement. And that’s why you’ll seldom see a driver that covers every command in the instrument.

Instrument manufacturers take their best guess at the commands you are likely to use and craft the driver accordingly. A typical IVI driver covers about 40-60% of the instrument’s command list. This may sound like a small number, but consider this: Agilent surveyed customers who used our 3852A Data Acquisition/Switch Unit. It was a complex instrument with over 300 distinct commands available. By poring over our customers’ code, we found they rarely used more than 5% of the available commands. This is an extreme case, but it tells you that 40%-60% coverage is a good start.

**Figure 8.** The driver is, among other things, a programming aid that works between the PC application and the I/O software. It can save enormous amounts of development time and prevent mistakes, but can also slow system performance by adding another layer of software.

**Figure 9.** If you are using a driver and need to access instrument functions the driver doesn’t have, you can send direct SCPI or ASCII commands, or go through the driver with pass-through commands to control the instrument directly. This gives you the convenience of drivers, with the 100% coverage of direct I/O. To avoid command conflicts, this technique requires in-depth knowledge on the part of the programmer.
Generations of drivers

There are three basic generations of drivers: Proprietary T&M drivers, Traditional T&M drivers, and Component PC drivers (Figure 10). These represent the past, present, and future of driver technology. In the past, instrument drivers were custom-designed to function with a vendor's own application development environment (ADE). A considerable body of legacy application programs uses these proprietary drivers, but for new development, engineers today have better choices.

When you need to accelerate test system design and deployment, Agilent recommends the new IVI-COM driver and the VXI plug&play WIN32 driver for instrument control. The only Component PC driver built on PC standard architecture is the new IVI-COM driver. This standard is being led by Agilent and other instrument companies. A component driver built on COM works in all popular PC languages and most T&M languages, uses the most popular types of I/O, can be used in the latest .NET technologies and is backward-compatible.

What is IVI?

Notice the word “IVI” is sprinkled around the chart in Figure 10. In 1998, test and measurement companies formed the Interchangeable Virtual Instrument (IVI) Foundation to address the high cost of developing and maintaining test system software and being able to evolve technology more rapidly, by the use of better drivers. The foundation comprises end-user test engineers, equipment manufacturers and system integrators with many years of experience building test systems.

IVI classes:
The goal of hardware interchangeability led IVI to the concept of instrument classes. The idea is as simple as it sounds: If you use a spectrum analyzer, it certainly would save time if you could program every instrument in the spectrum analyzer class the same way, no matter who built it. Both the specification and any specific driver that implements it are called an IVI Class Driver (IVI-C Class or IVI-COM Class).

As of this writing, the IVI Foundation has defined the following instrument classes: DC Power Supply, Digital Multimeter (DMM), Function Generator/Arbitrary Waveform Generator, Oscilloscope, Power Meter, RF Signal Generator, Spectrum Analyzer and Switch. Others are under development.

This work makes it much simpler for the engineer to program instruments from separate suppliers, when those instruments conform to a particular “class”.

When should I use a driver?

Use an instrument driver if:

• A driver is available that works with your development environment and I/O software, and supports the majority of instrument features you want to use.
• You want easy access to commonly used instrument functions because the instrument commands are typically organized in a hierarchical structure
• You need to simplify the process of developing and maintaining your code over time, because there is a single point of interface to update or change
• Software interchangeability is important to you.
• You need to simplify maintaining the system when instruments need to be exchanged.

Use direct I/O if:

• You have instrument programming experience or access to programming experts
• You need to use instrument features not supported by the available drivers (the other 40–80% of the instrument capability)
• You need the absolute maximum in system throughput speed
• You need to control the exact configuration of the instruments in your system
• You have a large volume of legacy SCPI-based code.

Figure 10. The three generations of drivers represent varying degrees of language independence. IVI-COM is the newest and the one supporting the widest variety of software environments.

### Instrument driver families

<table>
<thead>
<tr>
<th>Component PC (based on PC standards)</th>
<th>Traditional T&amp;M (based on T&amp;M standards)</th>
<th>Proprietary T&amp;M (specific to one language)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IVI-C (via NI)</td>
<td>WIN</td>
<td>VEE Panel Drivers</td>
</tr>
<tr>
<td>VXIplug&amp;play</td>
<td>LabWindows/ CVI Plug&amp;Play</td>
<td>LabVIEW Plug&amp;Play (VXIplug&amp;play GWN)</td>
</tr>
</tbody>
</table>

3 For additional information, you can visit the IVI Foundation website at www.ivifoundation.org
Conclusion

If the project you are pursuing is not complex, there are often situations where you don’t even know you are using a driver. Indeed, that is the ultimate goal of T&M companies... to keep this process entirely transparent. In the meantime, if you do get embroiled with issues of driver selection, note there can be tradeoffs between speed of development and speed of execution. The industry is working through these issues by instituting faster I/O and software aids, such as tools to keep track of instrument states. The whole idea is to give you both fast programming times and fast throughput.

If you choose to use a driver, computer industry-standard IVI COM drivers and a Visual Studio .NET-compliant development program such as the Agilent T&M Programmers Toolkit give you significant leverage. The T&M applications you develop will show significant hardware and software interchangeability, while being easily maintainable and extensible. The intellectual property you create during the development process will be widely transferable to other projects.

For downloads or more information on drivers, I/O software, connectivity and application software, join us at the Agilent Developer Network: www.agilent.com/find/adn.

Appendix

Resources

Where do I get drivers and driver tools?
Instrument vendors typically provide drivers on a CD with new products and offer their most up-to-date instrument drivers on their Web pages. Table 1 lists some of the primary sources.

Third-party software and systems integration companies that support the test-and-measurement industry can provide driver development tools and services. One such company is Vektrex (www.vektrex.com).

Agilent offers its own drivers on the Web at www.Agilent.com/find/ADN, but it does not post drivers written by others. Because you are at the mercy of whoever created the driver, it is a good idea to use a driver supplied by the same vendor who made the equipment.

Tools

Mixing I/O hardware and I/O software from different suppliers
Want to use Agilent I/O cards with NI LabVIEW software?

Want to use NI I/O cards with Agilent VEE?

Need to install Agilent VISA and NI-VISA side by side?

Help is available for all these scenarios. Go to: ftp://ftp.agilent.com/pub/mpusup/pc/binfiles/iop/m0101/readme/trouble/niinfo.htm

Table 1. Sources of driver software

<table>
<thead>
<tr>
<th>Instrument/Tools vendor</th>
<th>Finding Driver Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agilent</td>
<td>Agilent drivers are available through the Agilent Developer Network Web site. Go to <a href="http://www.agilent.com/find/ADN">http://www.agilent.com/find/ADN</a> and click on “Downloads.” Drivers are listed by type of driver, and by instrument model number.</td>
</tr>
<tr>
<td>Vektrex</td>
<td><a href="http://www.vektrex.com">http://www.vektrex.com</a> Tools for developing IVI-COM drivers</td>
</tr>
<tr>
<td>Data Translation</td>
<td><a href="http://www.datx.com/support/">http://www.datx.com/support/</a> Registration is required to download drivers.</td>
</tr>
<tr>
<td>iOtech</td>
<td><a href="http://www.iotech.com/ftp.html">http://www.iotech.com/ftp.html</a> Listed by instrument type</td>
</tr>
<tr>
<td>National Instruments</td>
<td><a href="http://www.natinst.com/idnet">http://www.natinst.com/idnet</a> Allows you to search by instrument vendor, instrument type, etc.</td>
</tr>
<tr>
<td>Racal</td>
<td><a href="http://www.racalinst.com/downloads">http://www.racalinst.com/downloads</a> After registering on this site, you get a listing by instrument of the types of available drivers.</td>
</tr>
<tr>
<td>Tektronix</td>
<td><a href="http://www.tek.com/site/sw/search/?wt=247&amp;link=/site/sw/search/">http://www.tek.com/site/sw/search/?wt=247&amp;link=/site/sw/search/</a> Search by product category or model number (drivers co-mingled with software and firmware)</td>
</tr>
<tr>
<td>Rohde &amp; Schwarz</td>
<td><a href="http://www.rohde-schwarz.com/">http://www.rohde-schwarz.com/</a> Look under “Shortcuts” for “drivers”</td>
</tr>
</tbody>
</table>
Agilent T&M Programmer’s Toolkit (Agilent W140A-TK1)
Want to use IVI-C drivers in Visual Studio .NET? Among many other capabilities, Agilent’s T&M Programmer’s Toolkit (see www.agilent.com/find/toolkit) can create managed wrappers around your existing IVI-C and VXIplug&play drivers. The wrapper is a native .NET class and fully object-oriented. The T&M Toolkit ships with more than one hundred pre-generated wrappers and its powerful wizard helps you to easily create others. As Figure 11 shows, the Toolkit wizard will:

• Automatically find all your installed drivers
• Find the installed drivers for your instruments, or allow you to download a driver from the Web
• Create a managed wrapper around the raw C-language DLL
• Add the appropriate Project Reference into the project
• Insert sample code to create the driver for your instrument at the proper hardware address

The Wrapper Wizard makes the test engineer’s life easier:

• An example of how to call a method on the driver
• IntelliSense help supports all the driver’s properties and methods, including help on each method parameter
• Driver call errors are automatically translated into standard .NET exceptions
• Automatic translation of parameters that have only a small range of possible values into a true enumeration, including IntelliSense help on each possible value
• Fully object-oriented implementation of the wrapper makes it intuitive to use

Figure 11. Agilent’s IVI-C Driver Wrapper Wizard makes it easy to use IVI-C compliant drivers in VS .NET.

The wrapper class generated by the wizard will invoke the functions in the driver DLL to perform the work for each method.

This wizard searches for VXIplug&play and IVI-C drivers which have been installed on this machine. VXIplug&play and IVI-C drivers are installed according to their respective specifications. NI COM drivers do not need wrappers.

Select the VXIplug&play or IVI-C Driver to Wrap.

Note: IVI-C drivers not conforming to the 2002 (VI) specifications may appear in the list as VXIplug&play.

Figure 12. Toolkit saves time. It searches for instruments, talks to them regardless of I/O type, shows all choices for the next function call, writes the VB .NET or C# commands for that function, and gives you context-sensitive help—all in one environment.

www.agilent.com/find/systemcomponents
Agilent Test Automation Kit
(Agilent N1908A)
www.agilent.com/find/kit
The average test system takes 360 hours to configure, test and verify. The Test Automation Kit can save up to 100 of those hours by:

• A USB-to-GPIB converter to simplify installation
• loading all the Instrument Drivers/I/O libraries (included)
• providing a real device and wiring harness for independent verification
• stepping the engineer through the setup process, using Test Express software
• calling any familiar programming language and automatically installing the proper drivers for the instruments present
• a library of over 200 examples in various languages, to use as a head start
• and providing two hours of expert test consulting
Agilent N1908A Test Automation Kit Lit # 5989-0000EN.

Agilent Developer Network
www.agilent.com/find/adn
The Agilent Developer Network is the place to go for

• Drivers
• Downloads
• Discussions

...Instrument Connectivity from Agilent... It simply works.

Glossary

ADE (application development environment) — An integrated suite of software development programs. ADEs may include a text editor, compiler, and debugger, as well as other tools used in creating, maintaining, and debugging application programs. Example: Microsoft Visual Studio.

API (application programming interface) — An API is a well-defined set of software routines through which application programs can access the functions and services provided by an underlying operating system or library. Example: IVI Drivers

C# (pronounced “C sharp”) — new C-like, component-oriented language that eliminates much of the difficulty associated with C/C++.

Direct I/O — commands sent directly to an instrument, without the benefit of, or interference from a driver. SCPI Example: SENS:e:VOLTage:RANGe:AUTO

Driver (or device driver) — a collection of functions resident on a computer and used to control a peripheral device.

DLL (dynamic link library) — An executable program or data file bound to an application program and loaded only when needed, thereby reducing memory requirements. The functions or data in a DLL can be simultaneously shared by several applications.

Input/Output (I/O) layer — The software that collects data from and issues commands to peripheral devices. The VISA function library is an example of an I/O layer that allows application programs and drivers to access peripheral instrumentation.

IVI (Interchangeable Virtual Instruments) — a standard instrument driver model defined by the IVI Foundation that enables engineers to exchange instruments made by different manufacturers without rewriting their code.

IVI COM drivers (also known as IVI Component drivers) — IVI COM presents the IVI driver as a COM object in Visual Basic. You get all the intelligence and all the benefits of the development environment because IVI COM does things in a smart way and presents an easier, more consistent way to send commands to an instrument. It is similar across multiple instruments.

Microsoft COM (Component Object Model) — The concept of software components is analogous to that of hardware components: as long as components present the same interface and perform the same functions, they are interchangeable. Software components are the natural extension of DLLs. Microsoft developed the COM standard to allow software manufacturers to create new software components that can be used with an existing application program, without requiring that the application be rebuilt. It is this capability that allows T&M instruments and their COM-based IVI-Component drivers to be interchanged.

.NET Framework — The .NET Framework is an object-oriented API that simplifies application development in a Windows environment. The .NET Framework has two main components: the common language runtime and the .NET Framework class library.

Plug and Play drivers — (also known as universal instrument drivers) are an important category of proprietary drivers. Plug and Play driver standards were originally developed for VXI instruments, and were known as VXIplug&play standards. When these standards were adapted for non-VXI instruments they became known simply as “Plug and Play” drivers. Library functions are in accessible C-language source and you can call them from programs written in VEE, BASIC, LabVIEW or LabWindows/CVI.

SCPI (Standard Commands for Programmable Instrumentation) — SCPI defines a standard set of commands to control programmable test and measurement devices in instrumentation systems. Learn more at http://www.scpiconsortium.org. See “Direct I/O” for example.
SICL — Standard Instrument Control Library (SICL) is a library of I/O function calls primarily implemented and supported by Agilent. Some of these are core functions that are common across all physical interfaces (GPIB, RS-232, etc.), while others are specific to the interface. The SICL library provides very complete and flexible control of instruments. SICL is optimized for use from C-language and C++ application programs, but can also be used from Visual Basic and other environments that can call arbitrary Windows DLLs. SICL provides complete access to GPIB, RS-232, LAN, VXI message-based, and VXI register-based products.

Universal drivers — another name for Plug and Play drivers

VISA (Virtual Instrument Software Architecture) — The VISA standard was created by the VXIplug&play Foundation. Drivers that conform to the VXIplug&play standards always perform I/O through the VISA library. Therefore if you are using Plug and Play drivers, you will need the VISA I/O library. The VISA standard was intended to provide a common set of function calls that are similar across physical interfaces. In practice, VISA libraries tend to be specific to the vendor’s interface.

VISA-COM — The VISA-COM library is a COM interface for I/O that was developed as a companion to the VISA specification. VISA-COM I/O provides the services of VISA in a COM-based API. VISA-COM includes some higher-level services that are not available in VISA, but in terms of low-level I/O communication capabilities, VISA-COM is a subset of VISA. Agilent VISA-COM is used by its IVI-Component drivers and requires that Agilent VISA also be installed.

VXIplug&play — A hardware and software standard that allows interoperability between VXI instruments made by different manufacturers. Learn more at http://www.vxipnp.org

Related literature

Data sheets

- W1140A Software and Connectivity, pub. no. 5988-5756EN
- N1908A Test Automation Kit, pub. no. 5989-0000EN

Application notes

Test-System Development Guide:

- Introduction to Test-System Design (AN 1465-1) pub. no. 5988-9747EN
- Computer I/O Considerations (AN 1465-2) pub. no. 5988-9818EN
- Understanding Drivers and Direct I/O (AN 1465-3) pub. no. 5989-0110EN
- Choosing Your Test-System Software Architecture (AN 1465-4) pub. no. 5988-9819EN
- Choosing Your Test-System Hardware Architecture and Instrumentation (AN 1465-5) pub. no. 5988-9820EN
- Understanding the Effects of Racking and System Interconnections (AN 1465-6) pub. no. 5988-9821EN
- Maximizing System Throughput and Optimizing Deployment (AN 1465-7) pub. no. 5988-9822EN

- Operational Maintenance (AN 1465-8) pub. no. 5988-9823EN
- Using LAN in Test Systems: The Basics (AN 1465-9) pub. no. 5989-1412EN
- Using LAN in Test Systems: Network Configuration (AN 1465-10) pub. no. 5989-1413EN
- Using LAN in Test Systems: PC Configuration (AN 1465-11) pub. no. 5989-1415EN
- Using USB in the Test and Measurement Environment (AN 1465-12) pub. no. 5989-1417EN
- Using LAN in Test Systems: Applications, AN 1465-14 (available in February 2005)
- The IVI Open-Architecture Driver Specifications: An Overview for System Designers, (AN 1409-4) pub. no. 5988-7939EN
- Operational Maintenance (AN 1465-8) pub. no. 5988-9823EN
- Using LAN in Test Systems: The Basics (AN 1465-9) pub. no. 5989-1412EN
- Using LAN in Test Systems: Network Configuration (AN 1465-10) pub. no. 5989-1413EN
- Using LAN in Test Systems: PC Configuration (AN 1465-11) pub. no. 5989-1415EN
- Using USB in the Test and Measurement Environment (AN 1465-12) pub. no. 5989-1417EN
- Using LAN in Test Systems: Applications, AN 1465-14 (available in February 2005)
- The IVI Open-Architecture Driver Specifications: An Overview for System Designers, (AN 1409-4) pub. no. 5988-7939EN
This application note is part of the Test-System Development Guide series, which is designed to help you quickly design a test system that produces reliable results, meets your throughput requirements, and does so within your budget.

The information presented here will help you choose the direction for your software based on the application you have in mind and the amount of experience you have. We will explore the entire software development process, from gathering and documenting software requirements through design reuse considerations.

The complete list of application notes for this series is available on page 19.

Table of contents
Introduction  2
Gathering and documenting software requirements  3
Programming and controlling your instruments  5
Collecting and storing test data  6
Designing the user interface  9
Choosing the development environment  10
Working with open standards  12
Developing a test sequence  14
Planning for software reuse  15
Summary  17
Glossary  18
Related literature  19
**Introduction**

This white paper will help you understand the tools required to design, develop and deploy the software component of your test system (see Figure 1).

**Gathering and documenting software requirements**—Before gathering and documenting your software requirements, finalize your test system hardware design. Once finalized, start working with your R&D and manufacturing teams to collect the information you need to create software requirements specifications (SRS).

**Programming and controlling your instruments**—The control of instruments is rapidly evolving from proprietary test and measurement standards to open, computer-based industry standards. This trend affects the hardware that connects the PC to the instrument as well as the software and drivers that control the instrument.

**Collecting and storing test data**—Data collection is the science of obtaining, moving and formatting data. The integrity of your test system depends on obtaining the right data at the right time.

**Designing the user interface**—One of the most important (and easily overlooked) aspects of test systems is the Graphical User Interface (GUI). This is what the test engineers, operators and technicians see when they interact with your software.

**Choosing the development environment**—The software environment and tools you choose will have a significant impact on the overall cost of your test system. When choosing your software environment, consider more than just the purchase price of the software. Also, consider how easy it is to learn and use the software, how hard it is to connect to other languages, devices or enterprise applications, as well as support and maintenance costs. Over the life of a test system, software support and maintenance costs alone can exceed hardware costs.

**Working with open standards**—Today, the industry trend is to move away from closed, proprietary development environments. More and more people are embracing open, industry-standard development environments as their platform of choice for test-system development projects. Making the right choice now will give you the flexibility and capabilities you need in the future.

**Developing a test sequence**—Test executives are applications designed to run a series of tests quickly and consistently in a pre-defined order. Of the 93% of test-system developers who use test equipment, approximately 37% use a commercial test executive for test sequencing, while the remaining 56% use a “home-grown” test executive.

**Planning for software reuse**—Designing for code reuse means you and your co-workers won’t have to re-create your software components every time you start a new project. Instead, you can build up a company knowledge base of best ideas, best practices, and software components. This knowledge base will bring uniformity and consistency to your company’s product testing functions.

This application note will provide you with a solid overview of the test-system software architecture as outlined above. For more in-depth information, refer to the sources listed throughout this document. Now, let’s get started with the first phase of choosing your test-system software architecture—gathering and documenting your software requirements.
Gathering and documenting software requirements

The Software Requirements Specifications (SRS)\(^1\) is a prioritized list of required test-system software capabilities and information on the software’s external interfaces, performance requirements, system attributes and design constraints. Typically, some requirements “musts” are essential and others “wants” can be traded for time (e.g., to meet a project deadline).

The IEEE Society identifies the following areas you should address in your SRS:\(^2\)

- Functionality—What is the software supposed to do?
- External interfaces—How does the software interact with people, the system’s hardware, other hardware and other software?
- Performance—What is the speed, availability, response time and recovery time of various software functions?
- Attributes—What are the portability, correctness, maintainability and security considerations?
- Design constraints—What industry standards do I need to follow? Do I need to use a specific language? What about internal policies for database integrity, resource limits and operating environments?

Ideally, the SRS will describe WHAT you need the software to do, not HOW the software will do it. In other words, you can look at the software as a “black box” that controls a set of external resources such as instruments, a computer monitor and other components (see Figure 2).

The SRS will include implementation details only if those requirements are imposed externally. For example, your company may require that a portion of the system be implemented in a specific programming language.

A good SRS should answer the following questions:

1. What measurements and tests are required to exercise the device under test (DUT)?
2. How will the measurements and tests be performed given the available instruments and devices?
3. What types of data need to be collected?
4. Where will the data be stored?
5. What are the external constraints (i.e., performance and time specifications)?
6. How will the operators, test engineers and technicians interact with the software?

Within the product development lifecycle, the R&D department should provide a formal list of testing requirements to the test-development department. The System Requirements Specifications, also referred to as a Project Requirements Specification, refers to the system as a whole and therefore is different from the Software Requirements Specifications. Furthermore, the manufacturing department will have their own requirements, such as safety standards. It is the combination of R&D and manufacturing specifications that determine the hardware requirements of a test system and provide the basis for the Software Requirements Specifications.

It’s important to note that trying to build or design software while the test system hardware is still in a state of flux typically results in additional software re-work and re-design. This is one of the challenges you will face in the real world of test-system development!

---

Figure 2. Scope of the SRS

---

1 May be referred to as an ERS or simply as “the requirements.”

Figure 3 provides an SRS template and a requirements example. As shown in the template, SRS is more than requirements. Document within the SRS what the software is meant to do and provide definitions for the terms you are using. Document the external constraints imposed upon you and the external resources you have available. Describe your users in detail and the modes of operation for each user class. Finally, include appendices and an index. Once you’ve completed these tasks, you’re ready to describe the specific requirements.

The requirements example (user interface of a test sequencer) is a snippet from a larger set of requirements divided by function. The words “MUST” and “HIGH WANT” are a way of ranking the relative importance of the requirements. You can break up requirements into more manageable hierarchies based on function, program mode, or some other classification system that will make the requirements section easier to navigate.

IEEE says that requirements must be correct, unambiguous, complete, consistent, ranked for importance, verifiable, modifiable and traceable. You can see that the above format meets a number of those goals, but some additional practices are necessary to meet them all. If you refer to requirements in more than one place, you will need to cross-reference them using a unique number (3.4.3, for example) so that if a requirement changes, you will know where to fix it elsewhere in the document.

Each written requirement needs to be verifiable and unambiguous to ensure the test program behaves as expected. As you write the SRS, refer to the System Requirements Specifications whenever possible. This is called backward-traceability, helping to explain why certain requirements are included and not just an arbitrary restriction.

The SRS must describe what testing resources (instruments) are required (e.g., the type of voltmeter, switches, computer monitor, etc.) and whether any factory resources are needed (e.g., a results database). In addition, you need to define within the SRS the data collection method, user interface requirements, performance constraints and, most importantly, the specific DUT test requirements. For example, if you need to perform a specific resistance measurement and you know you have an Agilent 34401A multimeter, the SRS would specify a single-sample 4-wire measurement including a description of the proper switching path, thus ensuring access to the pins on the DUT.

In order to accurately describe the test-system software user interface requirements, you should develop specific use cases for the different users of the test system (e.g., operators, test engineers, managers, etc.). Use cases are scenarios describing the users’ interactions with the software.

Taking the time to develop well-written requirements specifications up front will save you time later in the development process. The SRS process forces you to think about the scope of your project and helps to identify poorly understood areas of your software. This means you will spend less time re-writing and re-testing software due to confusion over what was truly required in the first place. A well-written SRS will help ensure that the project portion you want to contract out or redistribute will not require re-work on your part.

### Example SRS template

**Table of contents**

1 Introduction
   1.1 Purpose
   1.2 Scope
   1.3 Definitions, acronyms and abbreviations
   1.4 References
   1.5 Overview

2 Overall description
   2.1 Product perspective
   2.2 Product functions
   2.3 User characteristics
   2.4 Constraints
   2.5 Assumptions and dependencies

3 Specific requirements

Appendices

Index

### Example requirements

3.4 User interface functionality:

3.4.1 (MUST) The UI allows the user to create, modify, run and debug sequences.

3.4.2 (MUST) The UI allows users to view and export, load and store sequence run result data.

3.4.3 (MUST) The UI represents sequences in a hierarchical manner, which may be expanded or collapsed to view or hide internal details of the sequence.

3.4.4 (HIGH WANT) The UI can represent shared (used several places) sequences separate from the main sequence hierarchy.

3.4.5 (HIGH WANT) The UI will use graphical icons to denote variations in state of sequence items.
Programming and controlling your instruments

When designing your test-system architecture, you need to think about how your PC will communicate with different instruments. The two most important factors are 1) how to physically connect the PC to other instruments, and 2) what software will you use to control and communicate with other instruments.

Physically connecting the computer to other instruments

For decades, the IEEE-488 bus, commonly known as the general-purpose instrumentation bus (GPIB), set the standard for connecting test instruments to computers and for providing programmable instrument control. While GPIB is still a common and effective instrument interface technology, PC-based standards such as USB and LAN tend to be more cost effective solutions (see Table 1).

USB is the best choice for R&D applications where the number of instruments in a system is usually small and a quick and easy interface setup is desired.

USB 2.0 and Ethernet-based LAN are good choices for design verification and manufacturing applications where data-throughput performance, cost, remote access, and ease of system assembly are top priorities.

Given the choice between USB 2.0 and Ethernet-based LAN, most people choose LAN because of its inherent flexibility and remote system access and control capabilities. In addition, LAN performance is on par with USB—it has captive cable connectors (which aren’t found on USB), and LAN has the capability for wireless operation.

The I/O software layer

Once you’ve decided how your computer and instruments will be physically connected, you need to decide what I/O software you will use to control and communicate with the instruments (see Figure 4).

The I/O software is the layer of software that sits between the software application and the instruments’ physical interfaces. Once again, you have two choices. You can write directly to the instrument (Direct I/O) or you can use an instrument driver. Even though standard instrument drivers are popular because they are easier to use, they only express a subset of the instrument’s functionality.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Theoretical Interface Speed</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIB</td>
<td>• 8 Mb/s transfer rates</td>
<td>• Ubiquitous interface on test instruments&lt;br&gt;• Maximize throughput for all block sizes&lt;br&gt;• Low cost</td>
<td>• Expansion slot required&lt;br&gt;• Must open PC housing to install card&lt;br&gt;• Relatively expensive&lt;br&gt;• Limited cable lengths permitted between computer and instruments</td>
</tr>
<tr>
<td>USB</td>
<td>• USB 1.1 12 Mb/s&lt;br&gt;• USB 2.0 480 Mb/s</td>
<td>• Quick, easy setup&lt;br&gt;• Low cost&lt;br&gt;• Good data-throughput performance</td>
<td>• Does not work with Windows® NT.&lt;br&gt;• Not available on most deployed instruments</td>
</tr>
<tr>
<td>LAN</td>
<td>• 10/100/1000 Mb/s transfer rates</td>
<td>• Good data-throughput performance&lt;br&gt;• Low cost&lt;br&gt;• Remote access makes it easy to control system from remote location</td>
<td>• Requires LAN knowledge to set up&lt;br&gt;• Not available on most deployed instruments</td>
</tr>
</tbody>
</table>
So, how do you decide? Here are a few factors to consider.

You may want to use Direct I/O if:

- You need to use instrument features not supported by the available drivers (the other 40% to approximately 80% of the instrument’s capabilities). You can often use a combination of direct I/O and instrument drivers in this case. Some drivers make it even easier by providing a direct I/O connection for such scenarios.
- You have instrument programming experience or access to programming experts.
- You need the absolute maximum in system throughput speed.
- You need to control the exact configuration of the instruments in your system.
- You have a large volume of legacy SCPI-based code.

You may want to use an instrument driver if:

- A driver is available that works with your development environment and I/O software, and supports the majority of instrument features you want to use.
- You want the ease of use gained by an easy-to-understand hierarchical organization of instrument functionality provided by drivers.
- You want to simplify the process of developing and maintaining your code over time so there is a single point of interface to update or change.
- You need to simplify maintaining the system when instruments need to be exchanged.
- Development time is a paramount concern.

If you choose an instrument driver, consider using an industry standard IVI-COM (Component Object Model) driver together with a Visual Studio .NET-compliant development environment (such as the Agilent T&M Programmers Toolkit). IVI-COM drivers have the following advantages.

1. It works with all popular PC languages and most T&M languages.
2. It uses the most popular types of I/O.
3. It can be used in the latest .NET technologies.

By using IVI-COM drivers in the development of your test-system architecture, you’ll save time and have a higher degree of hardware and software interchangeability. You also will find that your software is easier to maintain and is more extensible in the future.

**Collecting and storing the test data**

Data collection is the science of identifying, collecting, formatting and distributing important information about the behavior of your test system and the devices it tests (see Figure 5). Quality data collection is the foundation for controlling your manufacturing and test processes—the ultimate goal of a manufacturing test engineer. Quality data also can be used to support many functions throughout your organization and support products throughout their development lifecycle.

**Figure 5. Overview of data collection process**

---

3 IVI-Component drivers are based on Microsoft’s Component Object Model (COM). IVI-C (NI) drivers are based on C dll’s.

4 A recent survey conducted by Agilent Technologies found that test programmers experience a 20%-30% reduction in test program development time when using Visual Studio .NET.
Communicating results of a test sequence is one use of test data. Test data also may be used to ensure regulatory standards are met, document performance standards, or provide traceability for the DUT. Given these applications and others, you may want to collect more data than your R&D or manufacturing colleagues request.

In addition to external data requirements, recorded data can be used to debug a test sequence in ways debugging runs cannot. Debugging means slowing down and subtly changing the behavior of your test sequence. This means a defect you see in a normal run may not show up in a debugging run (and vice-versa). One way to reduce the burden of diagnosing test software, and its associated DUT, is to always collect the data you need to debug a problem. You will need to balance the benefits of collecting extra data with the costs in performance and time for your test software.

Just as important as the standard types of data (e.g., test limits, measured values and pass-fail judgments) are the contextual data. Contextual data are used to communicate everything relevant to the DUT’s testing environment. This includes the test-system configuration, software version, driver versions and other factors.

The more variables you record, the more correlation points you and your colleagues can analyze during debug. For example, in one particular manufacturing test situation, a DUT would fail in the afternoon. The test engineer was able to correlate the time of day to the time of the failure and use that information to look more closely at a photoelectric component of the DUT.

It turned out that sunlight would strike that component directly at certain times of the day, causing the component to charge a capacitor and cause the test to fail. A DUT may fail due to the temperature variations or relative humidity. Capturing contextual information and measurement conditions can save days of effort.

You want to ensure the writing or formatting of your data does not affect the behavior of your test system. Today’s PCs use a variety of caching techniques that can dramatically affect how long it takes for a given file or network I/O command. If the time it takes to cache your data varies between each test run, you will get inconsistent test results. For that reason, it’s a good idea to keep your data in RAM until the end of your DUT testing and then do your formatting and data transmission.

Data is useless unless it can be understood. Features of good data include:

- **Identifiable**—information to identify the circumstances surrounding the data and the condition in which it was collected.
- **Searchable**—regular structure or fields that are uniquely identifiable, making it easy for a script or software tool to identify and compare across multiple records or datasets.
- **Transformable**—raw data must be interpreted and displayed (insight is the goal). This means that software algorithms can perform operations on some or all of the fields of your data and create a new data format or data visualization based on your original data.
- **Permanent**—data must remain available and comprehensible. Relational databases tend to be the best choice for long-term storage of data as these databases are highly searchable. If your company does not already have a database for manufacturing information, you may want to consider a database solution. This decision depends on your company’s data storage policies, practices and budget.

Table 2 lists some common data file formats and relevant characteristics.

---

**Table 2. File data format comparisons**

<table>
<thead>
<tr>
<th></th>
<th>Binary</th>
<th>Unformatted text</th>
<th>Comma-separated variables (.csv)</th>
<th>XML (Extensible Markup Language)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Identifiable</strong></td>
<td>Only with special tools</td>
<td>Only for small data sets</td>
<td>Needs good column format design</td>
<td>No major issues</td>
</tr>
<tr>
<td><strong>Searchable</strong></td>
<td>Only with special tools</td>
<td>Difficult and error-prone</td>
<td>No major issues</td>
<td>Excellent, but requires XML expertise</td>
</tr>
<tr>
<td><strong>Transformable</strong></td>
<td>Only with special tools</td>
<td>Difficult and error-prone</td>
<td>No major issues</td>
<td>Excellent, but requires XML expertise</td>
</tr>
<tr>
<td><strong>Permanent</strong></td>
<td>Only with special tools</td>
<td>Only for small data sets</td>
<td>No major issues</td>
<td>No major issues</td>
</tr>
<tr>
<td><strong>Example: spreadsheet analysis</strong></td>
<td>Only with special tools</td>
<td>Not importable</td>
<td>Supported by Excel, others</td>
<td>Excel 2003 format available</td>
</tr>
</tbody>
</table>

---

Binary formats have the fundamental issue of not being self-describing. In addition, you need to acquire a separate software application to interpret the data. Depending on the software application you use for interpreting the data, you also may be limited in the number of transformation functions.

Text files are hard to search and transform, and are not very identifiable. Since plain text files do not have regular fields, a text search for the number 12, for example, could return the hour twelve, the limit value 12, or the DMM measurement 12.

Comma-separated value (dot-csv) text formats are a good choice since they are easy to import into Microsoft Excel. With Microsoft Excel, it’s easy to make a table of results with the rows containing the results and each column containing a unique description. Another advantage is most data analysis software can easily read this format. The downside of this format is that it cannot store hierarchical data or easily parse data sets. You must decide up front as to the number and types of columns, with each column containing one unique data field.

XML is self-describing, very transformable, and has excellent search characteristics. There is an XML language called Extensible Stylesheet Transforms (XSLT) that can apply arbitrary algorithms to convert your XML data into new XML formats, HTML, or simple text formats. A number of data analysis programs, including Microsoft Excel 2003, can import XML data. If you fail to output your data in the right XML format for an analysis tool, you can write a relatively small XSLT that will convert all your XML data into the desired format. XSLT also provides a powerful search feature, making it much easier to identify data values or data structures.

The manufacturing test industry has already begun adopting XML. Some test executive applications support XML data logging. There is a standard called IPC 2547 that defines an XML format for communication of manufacturing test data.

Figure 6 is an example of a standard test run in XML format. You will still want to know the test sequence ID, the variant of the test, if the test limits are modifiable on the “PowerTest” and the hardware configuration of the test system.

If this were a .csv file, we would have to create a field for every record to answer those questions. Using XML, we can insert a record type called <TestSequence ID=“32”> and fully describe the current test sequence in that record. We can then add an XML attribute called “IDREF” to refer to that test sequence record in our <TestRun> records.

In summary, the data format you choose will have a large impact on its value over time. You need to consider how easy or difficult it will be for someone else to read and interpret the data once you are no longer involved in the project.

---

6 Extensible Markup Language: http://w3.org/xml.
9 IPC 2547: http://webstds.ipc.org/2547/2547.htm
Designing the user interface

When a user logs into a test system, what they see should depend upon their user class. The user class could be an operator, test engineer, technician, or service and calibration engineer. A well-written SRS will define the commands and/or menu selections available to each user class. You will want to provide each user class with only the capabilities and information they need to do their job. The more choices you provide, the greater the possibility for confusion and mistakes.

To ensure security, you can create a unique login for each of the users. Each user login should be linked to the appropriate class.

You can verify that your GUI meets the users’ needs with a methodology called “User-Centered Design”, or UCD, which consists of prototyping and storyboarding. In general, a test system’s GUI should be able to:

1. Customize its behavior based on the user class.
2. Provide or allow input of detailed information about the DUT.
3. Provide information about the state of the system.
4. Provide operations for controlling the system’s state and potentially its configuration.
5. Display the DUT testing results.

For an operator, the interface you design should always show the state of the test system (e.g., running a test, paused or stopped). For example, you could use a large color-coded graphic on the PC monitor in conjunction with lights mounted on the test system. The operator also will need a way to control the state of the test system as well as a way to input DUT information (unless this is done automatically via a bar code scanner).

As a general rule, the test program you design will require the following.

1. Commands for starting and stopping the test sequence.
2. Commands for sending test results to various kinds of printers (defect report ticket, etc.).
3. Control of the behavior of the test sequence (i.e., picking a DUT variant from a drop-down list).
4. A way to display a more detailed description of test results. The quality of a test results message can help in providing a quick diagnosis of a user error or a recurring hardware problem and may ultimately eliminate the need for a test engineer to visit the factory floor.

The user interface shown in Figure 7 was designed for an operator in a low-to-mid-mix/high-volume test application. The operator starts by logging into the test system, selecting the name and version of the testplan and entering the DUT information. The test status portion of the display is a little less prominent and visible than recommended for a manufacturing test environment, which may necessitate the addition of test status lights to the test system.

![Figure 7. Low-mix, high-volume user interface](www.agilent.com/find/systemcomponents)

---

The system message field displays the test result information as well as instructs the user on what to do next. To help the test engineer during the debugging process, the system message field also can display error messages.

The user interface shown in Figure 8 was designed for a high-mix, very low-volume testing situation (e.g., cell phone base stations). It also can be used for test sequence development or debugging. The class of user for this interface is highly skilled and possesses detailed knowledge of the purpose and function of the available tests, the DUT, and the test system configuration. An unskilled test operator would not be able to use this interface effectively.

The two GUIs were created with the same test software, though they vary considerably in complexity. The operator GUI in Figure 7 hides unnecessary choices and information critical to the software developer.

Choosing the development environment

The next step in choosing your test-system software architecture is to select a software development environment. The software environment and tools you choose will have a significant impact on the overall cost of your test system. When choosing your software environment, consider more than just the purchase price of the software. You need to consider how easy it is to learn and use the software, how hard it is to connect to other languages, devices or enterprise applications, as well as support and maintenance costs. Over the life of a test system, just software support and maintenance costs can exceed hardware costs.

You have a number of options when it comes to software development environments, from writing everything yourself in a language such as C, C++, C#, VB, VB .NET, VEE or LabVIEW, to using an off-the-shelf test executive with pre-written third party tests. The software environment you choose needs to accomplish two goals: 1) meeting your time-to-first-test requirements and 2) meeting your test-throughput requirements. How fast can you get your test system up and running, and how can you get the greatest throughput?

Software development environments can be grouped into two categories: graphical or textual. Graphical environments, such as Agilent’s VEE Pro 7.0 (see Figure 9) or LabVIEW, are considered easy for engineers to learn and use, largely because of engineers comfort with the schematic environment. In addition, it is easier to modify small to medium size graphical programs versus textual programming languages. Historically, textual programming languages ran faster in the manufacturing environment and yielded higher throughput. Today, there is less difference between the runtime speeds of a graphical environment and a textual environment. Even though graphical environments are easier to use than textual environments, textual environments are used more commonly in manufacturing test systems. Only about 22% of the half-million-plus users who write code for test and measurement equipment use a graphical programming language.

Even though graphical environments are easier to use than textual environments, graphical environments are used more commonly in manufacturing test systems. Only about 22% of the half-million-plus users who write code for test and measurement equipment use a graphical programming language.

Graphical or textual programming?

Before you can decide on which development environment is best for your application, it’s important to understand the use model of each in greater detail.

Figure 8. Software developer’s interface
Graphical programming is accomplished by manipulating images, called icons or objects, and the lines that connect these images. The images represent pre-made commands while the lines represent the program flow, control points, and/or how data are generated and consumed. The icons and interconnecting lines are contained within the integrated development environment (e.g., the software program).

Many graphical programming environments provide the ability to create compiled or packaged programs that do not need the programming environment to run. There are several graphical programming environments targeted at test and measurement engineers. These programs tend to have extensive I/O and instrument drivers, and T&M-specific math and graphing operations.

Some of the advantages of graphical programming languages over textual languages are as follows:

1. **No complex syntax**—The program instructions, typically presented as a group of icons connected by lines, are more immediately understandable.

2. **Easier to visualize the paths of execution and interaction**—Multiple concurrent activities rely on what is called a data-flow model, where a command needs to have all its data available before it will execute. This is easier than using multi-threaded programming techniques in textual programming languages such as C++ or Java.

3. **Can use real life metaphors**—The icons representing the commands can use metaphors (images) that represent real-world equivalents of the actions carried out by the icon. Most test engineers find graphical programming to be more intuitive and user-friendly than textual programming.

![Figure 9. Agilent VEE Pro 7.0 graphical programming environment](image-url)

**Agilent VEE Pro 7.0 and T&M Programmers Toolkit**

**Agilent VEE Pro 7.0**
- Description—easy to use, powerful graphical instrument programming environment
- Applications—data acquisition, design, low volume manufacturing test
- Purpose—graphical program creation to acquire and analyze instrument data
- Features—easy test-system control, sequencing, support of Microsoft .NET framework, MATLAB® analysis and visualization, full support of ActiveX

**Agilent T&M Programmers Toolkit**
- Description—test code development (in VB .NET, C++ or C#) integrated into Visual Studio .NET
- Applications—design characterization, design validation, manufacturing
- Purpose—writing complex programs with a variety of drivers in a PC standard environment
- Features—instrument I/O and communication, test code debug, data collection, display and analysis, support for IVI-C, IVI-COM, VXI plug&play drivers

[www.agilent.com/find/systemcomponents](http://www.agilent.com/find/systemcomponents)
4. **Rapid prototyping**—With the intuitive nature of a graphical programming language, it can be easier to quickly build a prototype of your system. The prototyping capability is less useful when dealing with a large test system, but prototyping can aid development of systems of any size.  

5. **Easier to share and learn existing programs**—Using real-life metaphors as visual cues can make it easier to share and learn existing programs and increase productivity.  

Textual programming languages use special words and syntax to represent the program’s operations and flow. Most, but not all textual programming languages are based on open standards. This means you will have a choice of vendors when it comes to your programming environment and software tools. Textual programming languages have a much larger set of third-party drivers, tools, and add-ins because they are based on open standards and are more widely used than graphical languages. This benefits the test engineer. Some of the advantages of textual programming languages over graphical languages are as follows:

1. Textual programming languages are better suited for creating larger, more comprehensive programs.  
2. For larger programs, textual programming languages are easier to navigate and comprehend. A person can observe only about 50 graphical objects at a time before the information becomes too complex or too small to see. If a user is forced to move around in a program to see all its objects, he or she can lose track of the control and data lines and find it difficult to understand the overall flow of the program. With that said, you can improve the understandability of large graphical programs by breaking up the program’s large operations into smaller sub-operations. This is called functional decomposition and is achieved by putting a series of commands into a “black box.” You then send commands to the functional block and receive its output as appropriate. Make sure the graphical program you use supports this functional decomposition if you plan on working with larger programs in a graphical environment.  
3. While the use of a textual programming language can improve overall system throughput, it’s the time spent during instrument operations that will have a greater impact. For example, you’ll see a negative impact on performance in a test system where the DUT to instrument switching is inefficient, independent of which programming language is used (graphical or textual).  
4. You also have a greater choice of development environments with textual programming languages. For example, there are few graphical programming languages that have development environments provided by multiple vendors. This means that today’s graphical languages are less likely to have the advantages created by competition between vendors.

Graphical programming tends to be easier to learn and comprehend while textual programming is more pervasive and open. Table 3 summarizes the differences between the two programming environments.

**Table 3. Graphical versus textual programming**

<table>
<thead>
<tr>
<th></th>
<th>Graphical</th>
<th>Textual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Free and open</td>
<td>Few open standards, less extensible</td>
<td>Dominated by open standards, very extensible</td>
</tr>
<tr>
<td>Rapid prototyping</td>
<td>Excellent T&amp;M prototyping features</td>
<td>Some code wizards, (T&amp;M Programmers Toolkit, for example) but slower</td>
</tr>
<tr>
<td>T&amp;M support</td>
<td>Several graphical environments targeted at T&amp;M, many drivers</td>
<td>Several T&amp;M-specific 3rd-party tools available, many drivers</td>
</tr>
<tr>
<td>3rd-party tools</td>
<td>Hundreds</td>
<td>Tens of thousands</td>
</tr>
<tr>
<td>Learnable and shareable</td>
<td>Easy to pick up and use programs programs are easy to share</td>
<td>Only small or very-well-designed</td>
</tr>
</tbody>
</table>

Development environments for open-standard programming languages have a greater feature set and are less expensive than their proprietary counterparts. Simply stated, an open-standard environment tends to create greater competition, which in turn tends to drive down prices and create innovation.

Open-standard languages generate a lot of interest from both software tool vendors and open-source developers. Both of these groups spend considerable time understanding the needs of the test-system programmer and, as a result, develop both free and for-pay tools and applications to meet those needs. A good example is the tremendous number of C and C++ libraries available on the market, both from vendors and from end-users. These libraries save development time and money given that it is faster and less expensive for a developer to buy the domain-specific software (such as mathematical analysis libraries) than create it from scratch.

Open standard environments also have a time-to-market advantage, as most proprietary environments cannot quickly take advantage of emerging technologies. Emerging programming technologies are developed with the most common open standard programming languages in mind. It takes longer for a vendor to release a new version of proprietary software that takes advantage of new technology.

**The .NET framework**—The .NET Framework is an open, multi-platform, multi-vendor set of software technologies for programming computers. The C# language has been submitted to a standards body as an open language. The underlying .NET “Common Language Infrastructure” technology, also an open standard, is available in multiple operating systems, including Microsoft’s Windows and Linux.

The .NET technology has excellent support and applicability to both web development and PC software development environments. The .NET technology has many of the advantages of Java language without many of Java’s drawbacks. For example, the .NET technology eliminates programmer memory leaks, makes software deployment easier, and provides a rich Application Programming Interface (API) for system and GUI development. The .NET technology is fully compiled via a Just-In-Time (JIT) compiler. The JIT compiler takes the OS and platform-independent code and creates optimized, machine-level code for the target platform.

While there is some additional overhead required to load the .NET framework runtime, programs written with .NET are comparable, or run faster, than their C/C++ counterpart. The reason programs can run faster in the .NET environment is due to the inefficiencies inherent in the linker operation of older languages.

A survey of programmers and a number of case studies have shown significant improvements in productivity via the .NET environment over the programmers’ old environment.

The .NET Framework (the collection of API services and helper code used by the .NET languages) is not the same thing as Visual Studio .NET. Visual Studio .NET is Microsoft’s development programming environment with support for the .NET technologies. As shown in Figure 10, there are multiple .NET development environments and programming languages available from a number of different vendors and supported on multiple platforms.

The best-known .NET languages are C# and Visual Basic (VB) .NET. C# is a lot like Java in structure and features, but its syntax is meant to be an evolution of C++. A C++ programmer familiar with object orientation and exception handling could easily move to the C# programming environment.

---

19 http://www.microsoft.com/net/casestudies
VB .NET is an upgrade to Visual Basic 6. Engineers with existing VB 6 applications must use an upgrade wizard to migrate to VB .NET. Once the upgrade process is complete, access to .NET applications and the additional power and flexibility provided by .NET can be achieved.

Microsoft’s C++ language also has been enhanced to include a new version called Managed C++. Managed C++ makes it easier to execute calls within the .NET software. Microsoft provides the original unmanaged C++ in Visual Studio .NET as well.

One significant advantage of .NET over older programming technologies is its extensibility. Microsoft engineered .NET so that it avoids a lot of the DLL installation frustrations Windows programmers experienced in the past. There are already a large number of third-party tools for .NET. Many of these third-party controls (i.e., advanced graphing visual controls) are useful to test-system programmers. Additionally, several test and measurement vendors, including Agilent Technologies, National Instruments, and Measurement Computing, have released .NET-compatible tools. For a complete list of released .NET-compatible tools, refer to Microsoft’s .NET partner web site at www.vsippartners.com.


Developing a test sequence

In a survey of more than 2,500 test and measurement equipment users, 93% of the respondents said they use multiple test instruments and/or are connecting their test instruments to a PC. Of that, 37% said they use a commercial test executive for test sequencing. The remaining 56% of these respondents use internal or “home grown” software for test sequencing.

A test executive is a software application designed to run a series of tests quickly and consistently in a predefined sequence. If any of the tests within the test sequence fail, then the DUT fails. Over the years, test executives have improved considerably both in terms of flexibility and capabilities. First-generation test executives were language-specific and not powerful enough for a mission critical manufacturing environment. Second-generation test executives, such as Agilent’s TxSL and NI’s TestStand are more powerful but more expensive. They also lack the flexibility required for a low-volume, high-mix manufacturing environment.

Each of the tests within the test sequence is a separate module. Commercial test executives come with a standard set of test modules and allow the user to create additional test modules from scratch (as well as customize existing test modules). Test executives control the data to and from the test module and, after collecting and analyzing all of the data, determine if the DUT passed or failed.

One reason for using a test executive is it provides a structured framework for manufacturing test systems. Test executives work best in medium- to low-mix, and medium- to high-volume manufacturing test environments.

Test executives are written so that sequence design, individual test design, and test limits and configuration management are treated as separate tasks. Keeping the three tasks separate results in greater flexibility, higher quality, and an increased opportunity for code reuse. It is the test executive that provides the infrastructure and helper services required to connect each of the separate tasks into a complete program.
One of the most important features of a test executive is its test sequencer. As shown on the left side of Figure 11, the test sequencer is a sequence of tests that can be manipulated in design mode. Various test executives provide different levels of flexibility in this sequence, such as “test looping.”

At a minimum, test executives should perform the following tasks.

1. Capture the results (and any extra data) using their own data collection model.
2. Keep track of the test limits and test setup data, passing the setups to the tests at execution time.
3. Provide limit checkers.
4. Provide run-time analysis of the test results (pass or fail reporting).

Additionally, test executives may include a software repository for maintaining the test modules (and for encouraging the reuse of tests). With a software repository, the test engineer can look for a specific test by doing a search within the test module repository. If all the engineers in a company settle on one test executive, it then becomes possible to share test modules between different product and manufacturing groups.

Test executives may use a switching model that makes it possible to map the physical layout of the test system’s control and data lines (and any switch boxes) to the DUT and instrument’s I/O pins. This allows the test engineer to think in terms of logical connections between instruments and the DUT, rather than worry about how the system is wired.

Finally, some test executives include tools for building the operator interface. While this feature tends to be less flexible than using one of the development environments discussed earlier, it does provide a fast and simple alternative.

Planning for software reuse

Aside from the use of standard libraries and operating system API’s, most software reuse tends to be opportunistic. A typical reuse scenario is when a programmer encounters a problem and remembers a similar problem handled by a co-worker. The programmer searches through the old source code of previous programs to find the desired code. If the code is found, the programmer decides how and if the software can be adapted to the current test situation. After modifications are made, the software must then be re-verified. Most of the time, adapting software in these situations is faster than creating software from scratch.

The scenario above could have been improved with a systematic software reuse approach. The advantages of a systematic approach is in the reduced time it takes to search, find, verify, and adapt test code for new test situations. A systematic reuse approach requires following specific coding and architectural styles, as well as adherence to standardized company policies and practices.

Discussing all of the considerations for implementing a complete company-wide systematic reuse program is outside the scope of this paper, but there are decisions you can make to help achieve a more systematic approach for yourself, your team, and even your company. Reuse considerations should begin after you’ve gathered system requirements and before you begin the software development.

Professional test executive or custom software?

How do you decide if you should create your own test executive or buy an off-the-shelf version? Here are a few factors you will need to consider.

1. The first thing to look at is whether you need a test executive at all. If you don’t have a relatively fixed sequence of tests, test executives are probably not right for you.
2. If your company has an internal test executive, or more likely, several internal test executives, you’ll need to investigate their quality, features, availability of support, and the collection of tests or other auxiliary software available for them.
3. If you find a reasonable choice, it doesn’t hurt to look at the cost of porting existing code over to use a professional test executive.
4. You may decide to use a professional test executive because of its support, quality or features.
5. A professional test executive most likely will have better outsourcing characteristics. Third-party software contractors and consultants may already have experience with such a test executive, and third-party libraries may be available.
6. A professional test executive should include a complete set of documentation.

If you choose to go with a professional test executive, make sure it’s from a company that provides high-quality service and support.
The design reuse process

The first step in the design reuse process is to complete a domain analysis. This is accomplished by 1) systematically analyzing the functions and parts of your software domain, and 2) using this information to develop a software architecture with well-defined component types and algorithms.

Next, you will want to look for natural boundaries in your software. One software design practice of finding and documenting the natural boundaries is known as Design Patterns. To find the natural boundaries, look to those areas where one type of activity or data set links with another type of activity or data set. These areas can then be grouped into separate modules and documented accordingly. Once documented, the same type modules can then be swapped for one another.

Once you have identified, collected and documented your modules, components and or individual parts, you will need to thoroughly test them before they go into the repository (or are passed on to your co-workers). This will save you and your co-workers from problems later in the process.

Finally, reusable components are reusable only if your co-workers know they exist. You need a repository (such as a relational database) for your modules anywhere in your team, division, or company (if appropriate) can browse and search for them based on what the components are and what they do.

A design reuse example

A good model for design reuse of individual test modules is the test executive—here’s why.

1. Some test executives break test software up into swappable tests, sequencers, limit checkers, test sequence and test limit data.

2. Test executives rely on the concept of modules. For example, you can have a module that provides the ability to perform a single pass or fail judgment, including the sequencer data type, the sequence execution operation, and the test types.

3. Test executives allow reuse of tests in different test sequences with no change to the test code. The sequencer provides the necessary data to the tests to customize their operation for the current test sequence.

4. Test executives keep the tests in separate modules or files from the test sequencer or test executive application. This allows you to easily swap tests in and out without recompilation.

5. Some test executives allow you to write your own custom limits checkers or sequencers.

All of these modules are able to interoperate because test executives use well-defined application programming interfaces (APIs) for each module. The modules are placed on natural boundaries between different types of data and functions within the test executives.

You can achieve similar reuse success in your own code with good architecture influenced by the natural boundaries of your software’s functions, types and data. To accomplish this, put information that changes frequently, such as the limits for a test, into a Data File. Put less flexible elements, such as a test class, into Types or “classes.” Functions, or “procedures,” should be reserved for the least flexible elements.

Design reuse and .NET

While the definitions of the boundaries of your software domain are not specifically influenced by the programming language or software environment, some environments are better than others in helping to keep your software modular and swappable.

.NET provides software tools that make it easier to develop a formal software reuse program within your department or company. Since .NET is object-oriented, it’s good at representing boundaries between different types of objects, such as tests or sequencers. Nonobject-based languages, such as C, require you to keep track of which functions apply to which objects, without much context-sensitive help or compile-time error checking.

.NET also includes improved versioning and deployment features. In addition, .NET has the ability to tell Windows that you will only accept a certain version of an external library. This eliminates one of the common frustrations with earlier versions of Windows where you rely on an external library (DLL), but then the DLL changes and your software no longer functions correctly.

---


21 This is a good example of a design pattern specific to the test and measurement domain.
Design reuse benefits

In summary, the reasons for implementing a design reuse program include improved software quality, increased software development efficiency, and better use of expert knowledge.

Design reuse improves quality in a couple of different ways. First, software errors are reduced as a result of the extra architectural analysis, improved system design, and flexibility and transparency. With good reuse policies implemented throughout the organization, you have access to thoroughly tested and verified components, reducing the opportunities for creating new defects.

Design reuse increases software development efficiency by reducing duplication of effort. Components need to be designed, implemented and tested only once. Good reuse practices make it easier to reuse an existing component as opposed to re-writing or even re-creating a new component.

Design reuse takes advantage of an organization’s expert knowledge. For example, most software developers spend time specializing on a particular set of skills and will write components based on those skills. With time, the set of available components for reuse becomes the set of the best knowledge of your organization. The company’s expert skills and deep knowledge will be evident in a rich set of reusable software components.

These benefits are not theoretical. The Software Engineering Laboratory at the National Aeronautics and Space Administration’s (NASA) Goddard Flight Center achieved significant benefits by implementing software reuse in the development of software products in its Flight Dynamics Division. According to the software engineering lab, NASA realized a 35% reduction in the effort needed to deliver a line of code, a 53% increase in daily productivity, and an 87% increase in code quality.22

Design reuse summary

Systematic design reuse across your company requires that your management value the extra efforts required by designing for reuse. Failure to invest and do the job right the first time will lead to frustration and wasted time down the road. One or more repositories of software components must be made available to all the engineers who will need them. You also need to be aware of any copyright or patent limitations of the code you plan to reuse. For example, if your software is written under contract with another company, they may have exclusive rights to that code.23

Summary

Before you begin writing code for your test system, you need to make a number of important decisions about the system’s software architecture. You will want to start by creating a detailed software requirements specification that defines what you want the system to do and how it should operate. The SRS should include an outline of how you will gather, store, analyze and present your data as well as how end users will interact with your system.

Another important decision you need to make upfront is which programming environment and language you will use for writing your code. Using a standards-based environment such as Visual Studio .NET maximizes your flexibility and helps you prolong the useful life of your software. By combining Microsoft’s Visual Studio .NET with Agilent’s T&M Programmers Toolkit, you can wrap objects written in a variety of languages such as Agilent VEE Pro 7.0. This allows you to pull them forward into your new programming environment, making the most of your legacy code investment.

Whether you choose a graphical or textual environment will depend on the size and complexity of your system, your skill set, your company standards, and the size of your programming team. The decision usually comes down to which environment—graphical or textual—will make you more productive. Textual environments are almost always the best choice for creating code for large, high-throughput manufacturing test systems because they offer the most power and flexibility, and they allow faster throughput.

Finally, you need to decide whether to use an off-the-shelf test executive or write your own test routines. Test executives can speed up your test-system development and lower your costs but will require an up-front training investment. If you are only performing a few tests, you may want to consider writing your own code.

---


Appendices

Glossary

ActiveX—A standard method for encapsulating COM-compliant software modules so they can be used in standard PC applications. ActiveX controls can be used in any ActiveX-compliant application, regardless of where they were created.

ADE (Application Development Environment)—An integrated suite of software development programs. ADEs may include a text editor, compiler, and debugger, as well as other tools used in creating, maintaining, and debugging application programs. Example: Microsoft Visual Studio

API (Application Programming Interface)—An API is a well-defined set of software routines through which an application program can access the functions and services provided by an underlying operating system or library. Example: IVI Drivers

C# (pronounced “C sharp”)—New C-like, component-oriented language that eliminates much of the difficulty associated with C/C++.

COM — See Microsoft COM.

Direct I/O—Commands sent directly to an instrument, without the benefit of or interference from a driver. SCPI Example: SENSE:VOLTage:RANGe:AUTO

Driver (or device driver)—A collection of functions resident on a computer and used to control a peripheral device.

DLL (Dynamic Link Library)—An executable program or data file bound to an application program and loaded only when needed, thereby reducing memory requirements. The functions or data in a DLL can be simultaneously shared by several applications.

IDE (Integrated Development Environment)—See ADE.

Input/Output (I/O) layer—The software that collects data from and issues commands to peripheral devices. The VISA function library is an example of an I/O layer that allows application programs and drivers to access peripheral instrumentation.

IVI (Interchangeable Virtual Instruments)—A standard instrument driver model defined by the IVI Foundation (http://www.ivifoundation.org) that enables engineers to exchange instruments made by different manufacturers without rewriting their code.

IVI COM drivers (also known as IVI Component Drivers)—IVI COM presents the IVI driver as a COM object. You get all the intelligence and all the benefits of the development environment because IVI COM does things in a smart way and presents an easier, more consistent way to send commands to an instrument. It is similar across multiple instruments.

Libraries—Files containing reusable software operations or functions meant to be used by other programs. They can be C based libraries, Visual Basic libraries, .NET libraries, COM libraries, or based on other software technologies.

.NET Framework—The .NET Framework is an object-oriented API that simplifies application development in a Windows environment. The .NET Framework has two main components: the common language runtime and the .NET Framework class libraries. New frameworks can be added by anyone.

Plug and Play drivers (also known as universal instrument drivers)—An important category of proprietary drivers. Plug and Play driver standards were originally developed for VXI instruments and were known as VXIplug&play standards. When these standards were adapted for non-VXI instruments they became known simply as “Plug and Play” drivers. Library functions are in accessible C-language source and you can call them from programs written in C, Basic, VEE, LabVIEW, or LabWindows/CVI.

SCPI (Standard Commands for Programmable Instrumentation)—SCPI defines a standard set of commands to control programmable test and measurement devices in instrumentation systems. Learn more at http://www.scpiconsortium.org. See “Direct I/O” for example.

Universal drivers—Another name for Plug and Play drivers

VISA (Virtual Instrument Software Architecture)—The VISA standard was created by the VXIplug&play Foundation. Drivers that conform to the VXIplug&play standards always perform I/O through the VISA library. If you are using Plug and Play drivers, you will need the VISA I/O library. The VISA standard was intended to provide a common set of function calls that are similar across physical interfaces. In practice, VISA libraries tend to be specific to the vendor’s interface.
VISA-COM—The VISA-COM library is a COM interface for I/O developed as a companion to the VISA specification. VISA-COM I/O provides the services of VISA in a COM-based API. VISA-COM includes some higher-level services not available in VISA, but in terms of low-level I/O communication capabilities, VISA-COM is a subset of VISA. Agilent VISA-COM is used by its IVI Component drivers and requires that Agilent VISA also be installed.

VXI plug & play—A hardware and software standard that allows interoperability between VXI instruments made by different manufacturers. Learn more at http://www.vxipnp.org

XML (eXtensible Markup Language)—A subset of SGML constituting a particular text markup language for interchange of structured data. The Unicode Standard is the reference character set for XML content.

Related literature

Data sheets
Agilent VEE Pro, 7.0 pub. no. 5988-6302EN

Application notes

Test-System Development Guide:
• Introduction to Test-System Design (AN 1465-1) pub. no. 5988-9747EN http://cp.literature.agilent.com/litweb/pdf/5988-9747EN.pdf
• Understanding Drivers and Direct I/O (AN 1465-3) pub. no. 5989-0110EN http://cp.literature.agilent.com/litweb/pdf/5989-0110EN.pdf
• Choosing Your Test-System Software Architecture (AN 1465-4) pub. no. 5988-9819EN http://cp.literature.agilent.com/litweb/pdf/5988-9819EN.pdf
• Choosing Your Test-System Hardware Architecture and Instrumentation (AN 1465-5) pub. no. 5988-9820EN http://cp.literature.agilent.com/litweb/pdf/5988-9820EN.pdf
• Understanding the Effects of Racking and System Interconnections (AN 1465-6) pub. no. 5988-9821EN http://cp.literature.agilent.com/litweb/pdf/5988-9821EN.pdf
• Maximizing System Throughput and Optimizing Deployment (AN 1465-7) pub. no. 5988-9822EN http://cp.literature.agilent.com/litweb/pdf/5988-9822EN.pdf
• Operational Maintenance (AN 1465-8) pub. no. 5988-9823EN http://cp.literature.agilent.com/litweb/pdf/5988-9823EN.pdf
• Using USB in the Test and Measurement Environment (AN 1465-12) pub. no. 5989-1417EN http://cp.literature.agilent.com/litweb/pdf/5989-1417EN.pdf
• Using LAN in Test Systems: Applications, (AN 1465-14) (available in February 2005)

Simplified Instrument Communication and Programming Using Textual Programming Languages (AN 1409-2), pub. no. 5988-6617EN

Other resources

Agilent Developer Network (ADN) http://agilent.com/find/adn

To discover more ways to simplify system integration, accelerate system development and apply the advantages of open connectivity, please visit the Web site at www.agilent.com/find/systemcomponents. Once you’re there, you can also connect with our online community of system developers and sign up for early delivery of future application notes in this series. Just look for the link “Join your peers in simplifying test-system integration.”
This application note is part of the Test-System Development Guide series, which is designed to help you quickly design a test system that produces reliable results, meets your throughput requirements, and does so within your budget. This application note explores the hardware architecture decisions and design choices you must make before you begin building your system to ensure that it provides you with the performance and flexibility you need. It also discusses issues you should consider as you select instruments for your system.

See the list of additional application notes in the series on page 17.

Table of contents
Introduction 2
System architecture 2
Instrumentation type: rack-and-stack or card-cage? 3
Making a choice 4
The computing subsystem 6
Switching 6
Mass interconnects 8
Power sources 8
DUT-specific connections 9
Other architectural considerations 9
Choosing test instruments for your system 10
Identify your needs 10
Development time 11
Measurement speed 12
Choosing a vendor 12
Example test system 13
Conclusion 16
Glossary 17
A low-frequency test system is essentially a group of subsystems that work together to test a particular device or range of devices. You need to make important decisions about each of the subsystems before you begin ordering test instruments or building your system. The way these subsystems communicate and interrelate has a huge effect on the cost, performance, maintainability and usability of your system. The time you spend upfront defining the architecture of your system is likely to save you time later that you might spend debugging software and tracing down the cause of faulty measurements. Ultimately, careful planning will help you ensure accurate testing of your DUT.

When you design a test system, you need to consider many of the same issues that architects consider when they design buildings: esthetics, safety, heat, size, cost, future expansion, optimal location of parts, and so on. Once you have decided how to approach these high-level issues, your test requirements will guide you in designing a system for the range of devices you expect to test.

This application note explores the system architecture decisions and design choices you must make to ensure your test system provides you with the performance and flexibility you need. It also discusses issues you should consider as you select instruments for your system.
Instrumentation type: rack-and-stack or cardcage?

There are two major types of instruments for test systems, rack-and-stack and cardcage. Rack-and-stack instruments are standalone test instruments that can be used independently. For test systems, they are frequently stacked in a rack (hence the name) to save floor space, and typically, engineers use external PCs to control them.

Cardcage instruments

Cardcage instruments, as their name implies, are modular test instruments on plug-in cards. You insert the cards in a cardcage, or mainframe, and control them either with an embedded controller (a plug-in card that is a PC) or an external PC.

VXI is a standard, open architecture for cardcage systems that allows instruments from different manufacturers to operate in the same mainframe. The VXIbus (VMEbus eXtensions for Instrumentation) was developed by a consortium of test-and-measurement companies to meet the needs of the modular instrument market. The VXI standard was patterned after the VMEbus standard, but it was defined specifically as a new platform because VME did not meet the needs of the instrument community, particularly with respect to noise rejection and triggering. VXI instruments typically offer more performance and speed than other instrument types.

Another cardcage architecture is PXI (PCIbus eXtensions for Instrumentation). While PXI cards are very small, they typically lack the accuracy and performance of VXI or rack-and-stack instruments. If you are considering using a PXI system, be sure to investigate whether you will need to purchase additional signal-conditioning equipment. Also, PXI is based on a PC backplane with no electromagnetic interference (EMI) or cooling specs, and therefore it is not as well suited to be a quiet measurement environment. See the sidebar on page 5 to compare attributes of PXI, VXI and rack-and-stack systems.

Another cardcage architecture is compact PCI (CPCI). CPCI technology is the basis for PXI, although PXI adds triggering options not available in PCI. CPCI and PXI cards can be interchanged to some extent. CPCI cards tend to be used in industrial PCs, because they are rack mountable and more rugged than other card types.

![VXI mainframe with modular test instruments on plug-in cards](image-url)
Racked instruments

Racked instruments can take up more space than cardcage instruments, but typically they are less expensive because they are produced in higher volumes. It is easy to find high-quality, high-reliability standalone instruments that are suitable for use in systems. Lately, test-equipment manufacturers have been putting more thought into how their standalone test instruments work in a system environment, making rack-and-stack architecture easier to implement. Agilent, for example, offers “system-ready” test instruments that incorporate standard protocols and optimized features like shielding, filtering, high-speed I/O and on-board intelligence and memory.

There are many benefits of using system-ready rack-and-stack instruments in your test system. For example, they can reduce your system development time because troubleshooting a system is easier when you use instruments that are capable of standalone operation. You can use an instrument in standalone mode to run preliminary checks to ensure you are getting good test results before you have the entire system set up. You cannot do the same with cardcage instruments, so it is more difficult to differentiate between hardware and software problems.

In some organizations, using a standard set of racked instruments throughout the product lifecycle can lower the barriers to effective communication and cooperation among organizations with different responsibilities. For example, R&D engineers may use benchtop instruments as they develop and fine-tune product designs. When they turn to design validation testing—or in the case of larger organizations, when they turn their pre-production prototypes over to the design validation department—it is helpful to use the same instruments, even though the tests are more likely to be automated or semi-automated at the design validation stage. If it is the same engineer doing the validation testing, he or she is already familiar with instrument operation and already trusts the test results the instrument generates. If R&D and design validation are handled by different engineers or different organizations, using the same test instruments can facilitate effective communication and shared problem solving. You get the same benefits if you use the same test system architecture when the product moves to manufacturing.

Making a choice

The decision you make about which instrument architecture to use will be influenced by several factors. If you are building a system from scratch, you will want to look at overall system performance and cost. However, if you already have a collection of either rack-and-stack or cardcage instruments, reusing them and adding to your collection may be more cost effective than starting over. Also important is whether you have access to rack-and-stack or cardcage systems-building expertise. If all the expertise in your company is with cardcage architecture, it may not make sense to switch to rack-and-stack, even if the equipment cost is less. If you decide to stay with an existing cardcage setup for your system, you may want to consider migrating to a hybrid system, adding rack-and-stack instruments to gain the capabilities or performance you need. You will need to evaluate the specific circumstances to make the best decision.

Another factor to consider is the cost of maintaining your system. Look into typical repair costs and the cost of keeping spare parts and extra instruments/cards on hand.

In the “Choosing instruments for your system” section of this application note (see page 10), you will find more detailed information about choosing the right instruments for your system.
Comparison of instrumentation types

<table>
<thead>
<tr>
<th></th>
<th>Rack and stack</th>
<th>VXI</th>
<th>CPCI</th>
<th>PXI</th>
<th>See notes:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standalone use</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>1</td>
</tr>
<tr>
<td>Accuracy</td>
<td>****</td>
<td>***</td>
<td>**</td>
<td>**</td>
<td></td>
</tr>
<tr>
<td>Price</td>
<td>$$</td>
<td>$$$</td>
<td>$$$</td>
<td>$$$</td>
<td></td>
</tr>
<tr>
<td>Burst speed</td>
<td>**</td>
<td>****</td>
<td>****</td>
<td>****</td>
<td>4</td>
</tr>
<tr>
<td>Single-point measurement speed</td>
<td>**</td>
<td>****</td>
<td>**</td>
<td>**</td>
<td>5</td>
</tr>
<tr>
<td>GUI response time</td>
<td>****</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>6</td>
</tr>
<tr>
<td>Footprint</td>
<td>**</td>
<td>**</td>
<td>****</td>
<td>****</td>
<td>7</td>
</tr>
<tr>
<td>Ease of use and integration</td>
<td>****</td>
<td>**</td>
<td>**</td>
<td>****</td>
<td>8</td>
</tr>
<tr>
<td>Shielding</td>
<td>****</td>
<td>***</td>
<td>**</td>
<td>**</td>
<td>9</td>
</tr>
</tbody>
</table>

1. Standalone use
With an internal PC, a cardcage can operate standalone, but you need a monitor if you require an operator GUI. Cost of an embedded PC is several times that of a standard PC. In any case, card cages generally require some form of computer communication in order to be useful, while rack-and-stack instruments can be used to check out the system without a computer present.

2. Accuracy
Cardcages have power supplies that must be shared among several subsystems. Rack-and-stack instruments are optimized to one use, so they are designed to have the right power supply for the job at hand, and analog circuitry that is not subject to cage-imposed restrictions. Rack-and-stack instruments are designed to minimize magnetic interference so they are less likely to induce currents that would disrupt sensitive instruments. As a result, rack-and-stack systems typically outperform cardcage systems in terms of accuracy, crosstalk, noise, etc.

3. Price
Cost of a bench-top system is usually much lower when instruments are not rack-mounted. When instruments are rack-mounted, system cost typically is comparable with the cost of card-cage systems, with one exception: When a cardcage is sized such that the application consumes all the slots in the cage, the cardcage system is typically more cost-efficient. However, a full cardcage also eliminates the potential benefit of allowing for expansion.

4. Burst speed
Burst speed is the speed at which the instrument can move a large amount of data from a single channel across some bus or I/O port to the computer. Burst communication is used in data acquisition more than it is used in functional test. Cardcages typically shine in this arena, although recent improvements in I/O speed have blurred the boundary between backplane and external I/O.

5. Single-point measurements
Single-point measurement speed is the time it takes to make a single measurement, switch channels and then make another measurement. This is the predominant mode used in functional test. You’ll find more information about test-execution speeds in the “Measurement speed” section on page 12.

6. GUI response time
When a cardcage communicates to the PC, the PC must often do double duty as it processes the data and also updates the GUI. In some rack-and-stack instruments, these operations happen in parallel, giving the operator more real-time update capability. This is especially true with an oscilloscope, where lack of immediate feedback can be annoying.

7. Footprint
PXI and CPCI systems have the smallest footprints. However, many instrument functions are not fully realizable in PXI, so engineers typically adopt a hybrid approach of rack-and-stack plus PXI instruments. Once you have a rack for part of your system, you use the same amount of floor space as you would for a full rack-and-stack system, so you lose the space-saving advantage offered by the small form factor of the PXI cards.

8. Ease of use and integration
If a racked system has been designed to accommodate a reasonable amount of expansion space (a good idea to plan for unforeseen future needs), adding instruments to a rack is not a lot more complicated than adding an instrument to a cardcage. A more important consideration is the ease of adding additional cables to an existing architecture. For example, whether you use a cardcage or several racked instruments, their inputs and outputs are usually connected into a switching system or a mass interconnect. If the system has been designed to handle such new instruments, integration will only take a few minutes. If the system has to be redesigned to handle the new instrument, it can take days.

9. Shielding
Dedicated rack-and-stack instruments are typically well shielded. Since they are designed for a specific purpose, they are frequently more noise-free than their card-cage counterparts. VXI has specific shielding specifications, and these are lacking in PXI and CPCI. While it is possible to shield PXI, the implementation is left up to the vendor, so placing a new vendor’s product in a slot may result in some unwanted interference with nearby instruments.
The computing subsystem

Before you consider the questions surrounding the computing subsystem, you need to decide whether you will control your system manually, semi-automatically or with a fully automated control system. These issues are addressed in the first application note in the Test System Development Guide, Application note 1465-1, Introduction to Test-System Design. The information in this computing subsystem section is for test engineers who have decided to use either automated or semi-automated control.

For systems that use rack-and-stack test instruments, you will most likely use an external PC that is cabled to the instrumentation. For test systems that use card-based instruments, you need to decide whether to use an embedded PC (one that fits inside an instrumentation cardcage) or an external PC. At first glance, the embedded PC may seem like a good choice. It fits inside an existing cage, so it uses rack space efficiently, and it is directly connected to the backplane, so data transfer speeds are excellent. Unfortunately, embedded PCs cost a lot more than external ones, and typically they do not have room to hold many modern peripherals. The technology used in embedded PCs tends to lag the technology of the general computer industry, so embedded PCs often are a generation behind in processor type and speed.

If you use an external PC, you will get more computing power for your money. In addition, many external PCs come with industry-standard interfaces like USB, LAN and FireWire built-in. If you use a PC with these interfaces, you can lower the cost of your test system by using test instruments that support these interfaces, or shorten setup time by using USB/GPIB or LAN/GPIB converters. This topic is covered in detail in Application Note 1465-2, Test-System Development Guide: Computer I/O Considerations.

In manufacturing environments, cost is typically a critical concern, especially when you are implementing hundreds of identical test systems. The lower initial cost of external PCs typically makes them a better choice for manufacturing test systems, and the fact that they are typically less expensive to service than embedded controllers adds to their appeal.

Another major computing consideration is the choice of software and application-development and runtime environments. Computing subsystem decisions related to software are covered in Application Note 1465-3, Test System Software Architecture.

Switching

Switches, or relays that interconnect system instrumentation and loads to your DUT, are an integral part of most test systems. Choosing the proper switch type and topology will impact the cost, speed, longevity, safety and overall functionality of your test system. For a thorough examination of switching in test systems, see Application Note 1441-1, Test System Signal Switching.

The types of relays you choose for your switching subsystem are important, as they affect the type of circuits and systems you can test. Reed relays and FETs are the best choice for high-speed systems, and of the two, reeds have higher voltage and current ratings. Reed relays are excellent choices to connect measurement instruments and low-current stimulus to the DUT. They are very fast (typically about 0.5 to 1.0 ms), although they can have a higher thermal offset voltage than armature relays. Use armature relays (which typically switch in 10-20 ms) for higher-current loads. When you use armature relays, group your tests so the relays stay connected to perform as many readings as possible at one time. Because armature relays are relatively slow, you will want to avoid connecting and disconnecting them multiple times.

Switching topologies can be divided into three categories based on their complexity: simple relay configurations, multiplexers and matrices. The best one to use depends on the number of instruments and test points, whether connections must be simultaneous or not, required test speed, cost considerations and other factors.

A matrix arrangement of reed relays provides an excellent way to allow any instrument to be connected to any pin on your DUT, and it permits easy expansion as you add new instruments to your system or more pins appear on your DUT. Matrices use more relays than multiplexers, so they tend to cost more. If you don’t need to connect multiple instruments to any pin, a multiplexer is a suitable solution. If you have a 1 x 20 multiplexer for example, you can connect a test instrument to 20 pins, but you can’t hook anything else to those 20 pins. With those same 20 relays in a matrix, you can connect four instruments to five pins in any combination.
In manufacturing test and design validation systems you often need banks of general-purpose relays of varying current capability. You can use such relays to connect DUT inputs to ground or to a supply, or through resistors to simulate dirty switches. You also can use them to provide ways to disconnect output loads in order to allow parametric tests on output transistors, as shown in Figure 3.

You also need to think about where to place and how to arrange your switches. While relay cards can be placed in a cardcage that is intended for high-performance instruments, it is a waste of valuable real estate. The high-speed backplane in a modular cage is more suited to the control of high-speed instruments, not simple relays. If you place relays in a separate box that is tuned for that purpose, it will be easier to expand the high-performance instrumentation while allowing room separately for denser relay cards, more relay cards or a bigger switchbox. It also makes a clearer delineation between the instrumentation and the switching subsystems, which makes it easier to keep your system organized.

Placing the DUT interface panel (mass interconnect or feedthrough panels) in front of a switching subsystem that has the plug-in cards facing the interface panel accomplishes two goals: 1) It minimizes rack space, because the switchbox and mass interconnect are in the same plane, and 2) it reduces wire length from the switching to the DUT. If the box you choose has cards in the rear, reverse-mount the switchbox using the rails on the rear of the rack, as shown in Fig. 4. There are two negatives to this approach: the front panel of the switching instrument is not accessible from the front of the system, and it can be harder to reach the plug-in cards for service. However, once a system is operational, it is seldom necessary to operate a switchbox from its front panel, and cards can be accessed by pulling the instrument out the back or by removing the side panel of the system.

Figure 3. Switched loads allow parametric measurements

Figure 4. Rear-mounting the switching subsystem reduces rack space and minimizes cable lengths
When you are designing the switching for your test system, you may want to build in some safety features. Particularly if you are working with high voltages or high currents, you might want to include a switch to disconnect all signals, to minimize the chance for potentially serious accidents.

Mass interconnects

A mass interconnect panel is a DUT-to-system wiring interface that allows you to use fixtures instead of wiring each connection separately. When you are designing a functional test system for a design lab, it is tempting to leave out a mass interconnect, since the product design changes so much and the extra time to rewire a fixture is not productive. It also is not as likely that you will make identical measurements on large numbers of devices. Simple clip leads may suffice, especially for small DUTs. Interface panels are relatively expensive—using one can easily double the cost of a system—but there are a couple good reasons for adding one to your design-validation, production-verification or manufacturing test system:

- A mass interconnect provides a physical location for mounting interface components such as terminal blocks, fuses, custom electronics/interfaces/conditioning, etc., between the system and the DUT. You can mount these components either to the interface frame or to a shelf attached to the frame.
- Device measurements are less likely to change due to random movements of wires.

Tips for successful switching

1. Place system switching in a box dedicated for that use, such as the Agilent 3499A/B/C switch/control mainframe or the 34970A data acquisition/switch unit. Placing all system switching in one place minimizes cost and helps to keep your system organized. Allow enough room to expand the switchbox to a larger size or to provide room for another one as your needs grow.

2. Inside the switchbox, create an instrumentation matrix. For example, create a 16 x N switch matrix, connecting instruments to the 16 “rows”, and your DUT to the “N” (column) side, allowing one matrix column per DUT pin. By making N an expandable number, in increments of, say, 16 or 32, you can handle modules that are close to your immediate needs with a way to easily expand to higher-pin-count modules in the future. When you need new instruments, simply connect them to a new set of rows. No additional wiring is needed. Since most instrumentation is low current and must be scanned across multiple points quickly, choose fast reed relays or FET switches for this architecture.

3. Also inside the switchbox, allocate a set of general-purpose relays for power supply and load connections. These relays are generally too big to allow economical creation of a high-current matrix that could programatically assign any DUT pin to any load. Therefore, bring such relay connections out directly to an interface panel where they can be connected to the appropriate pins.

When using terminal blocks on the interface makes it easy to make wiring changes as the DUT changes, allows easy connection of multiple resources to common points, and provides easy test connections for debugging the system.

For design validation, production validation and manufacturing test, mass interconnects are typically well worth the investment. They provide a fast and robust means of changing connections to different DUTs using the same system.

You can obtain more information about mass interconnects from the three major manufacturers: Virginia Panel, MAC Panel and Everett Charles Technologies/TTI Testron.

Power sources

DUT power is an integral component of a test system, whether it is a simple bias supply or an advanced system power source. Depending on your application, your DUTs can require anything from a few milliwatts to many kilowatts. There are many power supplies available for providing power to a DUT. Choosing the right one is more complicated than simply picking the right voltage and current level.

Testing your DUT will be a lot less frustrating if you choose a reliable system power source that provides a stable voltage source to power the DUT and built-in measurement capability to verify DUT performance under various operating conditions.
When you select your DUT power source, consider:

- Settling time
- Output noise
- Fast transient response
- Fast programming, especially down-programming response
- Remote sensing—compensate for voltage drop in wiring
- Built-in, accurate, voltage and DC current measurement or waveform digitization
- Small size—it’s possible to get linear performance (low noise) out of a switcher to free up rack space
- Triggering options
- Programmable output impedance
- Multiple outputs and sequencing of outputs
- Over-voltage protection
- Over-current protection
- Lead lengths
- Safety due to exposed voltages

Your choice of supply can dramatically impact system throughput, since waiting for power supplies to settle can be one of the most time-consuming elements in a typical test plan.

**DUT-specific connections (loads, serial interfaces, etc.)**

Many DUTs require components to be connected to their outputs in order to adequately stress the unit. These can take the form of resistive or reactive output loads such as resistors, light bulbs or motors, or complicated, simulated loads such as the dynamically varying current in a camera battery. In most cases, it is wise to provide a place to put such loads in a system, such as a slide-out tray on which small, discrete loads can be mounted. Some DC-programmable loads (the size and shape of a power supply) can be rack mounted. Such loads are often connected to the DUT through relays to allow the DUT to be completely disconnected from all test system resources. If you decide to use relays, locate the loads close to the switching subsystem to minimize cable lengths.

**Other architectural considerations**

**AC power distribution**—If you are designing a system that you expect to replicate and ship to areas of the world that have different power requirements, you will probably want to include a power distribution unit in your system to make it easier to convert to the appropriate scheme. Power distribution units give you a way to route power, detect power line problems, and filter the input, and they provide the potential for adding uninterruptible power supplies and an emergency off (EMO) switch input.

---

**Figure 5.** Simplified diagram showing ways you can connect loads in various configurations. A “bridge load” connects a load between two pins on the DUT, rather than between an output and ground or an output and power.
Cooling—If you do not pay attention to cooling, temperatures in a rack can easily exceed environmental conditions specified for your test instruments. When this happens, your instruments can fail prematurely and your measurement results can be jeopardized. Temperature gradients are also something to consider. If one end of the rack is ten degrees hotter than the other end, even if the overall temperature is within instrument specifications, the resulting gradient can cause some unwanted thermocouple effects or slow drift errors.

You can use fans to draw air through your system to remove heat. If you cannot create enough airflow to remove the heat with a fan, you may need to consider air conditioning your rack. There are standard NEMA enclosures that can be used for this purpose.

If you are using rack-and-stack test instruments, it is important to think through how you place the instruments in the rack. Test instruments typically pull air in on one side or through the bottom and exhaust hot air out the other side or the top. Be careful not to position an instrument’s air intake adjacent to another instrument’s exhaust vent. You will find more information about racking test instruments in the application note, *Test System Development Guide: Understanding the Effects of Racking and System Interconnections* (AN 1465-6).

Ergonomics—As you make decisions about your system architecture, keep in mind the operator’s comfort and convenience. Provide adequate work space at the correct height, depending on whether the operator will be sitting or standing. Put displays at a comfortable height and if appropriate, provide the ability to tilt the display to reduce glare and eyestrain. Make sure illumination is adequate for the tasks that need to be performed. Provide for left-handed and right-handed operators by allowing a mouse to be placed on either side of the keyboard.

Safety—If you are working with high voltages, consider using interlocks to prevent accidents. Take precautions to deal with static electricity. For moving parts that could cause bodily harm, consider using deadman switches (two switches, both of which must be engaged for the equipment to run) and EMO switches (a single switch to turn off the entire system in an emergency). Position heavy equipment low in the rack and watch how you distribute weight in the rack to prevent it from tipping over. Also consider how weight distribution would change if you were to remove an instrument for maintenance.

Future expansion—To maximize the re-usability of a functional test system, you need to design it in such a way that in the future it will be able to accommodate more instruments, more switches and bigger DUTs that require more power, without a complete re-design. To maximize your long-term flexibility, use open standards whenever possible. Make sure to allow 20 percent to 30 percent extra room in a cardcage, or 20 percent extra room in your rack to accommodate instrument additions. See Application Note 1465-1, *Test-System Development Guide: Introduction to Test-System Design*, for more ideas about planning for future expansion.

Choosing instruments for your test system

The measurement and stimulus instruments you choose for your system—whether they are rack-and-stack instruments or instruments on a card—will be driven largely by the functional and parametric tests you need to perform, and whether you are using manual, semi-automated or fully automated control for your test system.

Identify your needs

In all cases, it is wise to start by making a thorough list of the inputs and outputs of each of the devices you plan to test and the parameters you will measure. Note the accuracy and resolution you need for each measurement as well. Once the list is complete, check to make sure it does not contain redundant or unnecessary tests. Then identify possible test instruments for the required measurements and look for opportunities to use the same piece of test equipment for multiple measurements.

The types of instruments you need will vary depending on your application. However, there are several universal questions that you must answer in order to select measurement and stimulus instrumentation properly:

1. AC stimulus—How many dynamic (AC) signals do you need to apply simultaneously? This determines the number of channels of arbitrary waveform or function/signal generator you require. For applications needing more than about four channels, an instrumentation cardcage is the best solution. For applications where low cost is important, and you have few channels or isolated outputs, rack-and-stack instruments are a better solution.
2. **DC stimulus**—How many static (DC) signals to you need to apply simultaneously? This determines the number of channels of DAC (digital-to-analog converter) you will require.

3. **Measurements**—What types of measurements do you need to make, and how many simultaneously? If minimizing instrumentation costs is essential, look for ways to minimize the number of instruments you need by paying attention to the ancillary functions of instrument that might perform double duty. For example, you can perform RF power measurement with a spectrum analyzer if accuracy and speed are not critical to your application. If you only need to know the power supply voltage within 0.5 percent, you might be able to use the internal voltmeter inside your power supply, using the read-back mechanism to read voltage on terminals.

4. **Protocols**—Do you use any special serial data protocols? This determines the need for instruments to handle things like CAN, ISO-9141, J1850, and many more.

Once you have made your measurements list and answered these initial questions, you can refine your list of instrument possibilities by looking at your budget and time constraints and your requirements around measurement speed.

### Development time

When you are choosing instruments for your test system, look for instruments that will minimize your development time. You can save time by using rack-and-stack system-ready instruments that incorporate a high percentage of the measurement solution you need. For example, if you use a source with modulation capability, you don’t have to develop your own algorithm or integrate additional hardware to generate the required modulation.

If you want to minimize hardware costs, you can investigate auxiliary capabilities. However, if your goal is to minimize development time, buy instruments that are specifically designed to do the jobs you need done. Using instruments with IVI-COM drivers can save you development time. If the instrument has an IVI-COM driver, you can interchange hardware without rewriting your software, as long as you adhere to the functionality that is specific to the instrument class. See the application note, *Test-System Development Guide: Understanding Drivers and Direct I/O* (AN 1465-3), for to learn how decisions about drivers affect development time.

Test instruments with downloadable personalities also can save you development time. You download the measurement personalities for a specific application directly into the test instrument’s internal memory. Then you can simply choose from a menu of tests, and the personality’s “intelligence” automatically performs the tests, from capturing signals to displaying results. Agilent ESA-E Series spectrum analyzers, for example, have measurement personalities for testing cable TV, phase noise, cable fault, Bluetooth™, cdmaOne, GSM/GPRS, and modulation.

You typically spend a large percentage of total development time on debugging your system, particularly if you are building a new test system. You can reduce your debug time significantly by writing a diagnostic test routine that loops outputs back to inputs through a large part of the switching path. This exercise will help you quickly identify the cause of problems—whether it is a source, a measurement instrument or a switch path.

For more ideas on minimizing your development time, see the application note *Test-System Development Guide: Choosing Your Test-System Software Architecture* (AN 1465-4).
Measurement speed

If you are building a manufacturing test system (and to a lesser extent in design validation applications), the time it takes to execute each test can be critical. But figuring out how fast your system will perform measurements is harder than it appears. For example, a digitizer may be able to sample 1000 readings very fast, but if those readings are transferred to the PC over GPIB, it could take a long time. A digitizer that can have a decision-making algorithm downloaded into it could allow a simple go/no-go result to be sent back to the PC, which would make GPIB a reasonable option and may save money over a cardcage-based solution. However, it takes extra effort to create and download a decision algorithm into an instrument, which may increase development time as well as “first-run” time of the test program. Also, inside an instrument the readings will be analyzed by a much slower processor than the one in the PC, so this must be factored in as well.

Simply reading the data sheet does not tell the whole story. Maximum reading rate specifications are usually related to burst speed; that is, the speed which you can sample the signal on a single channel. But that is not the typical mode for functional test. In functional test, the system normally makes a single measurement, then changes a parameter like range or function or channel, and then makes another measurement. In this case, the burst rate is meaningless. Take for example, two multimeters—one GPIB and one PXI. Their relative speeds in a functional test mode are shown below. Although the PXI DMM is much faster than this particular GPIB DMM in burst mode, the comparable true speeds in functional test mode are nearly identical; in fact, the GPIB DMM is slightly faster (Fig. 6B).

At higher resolutions, burst rate again becomes moot, since actual reading rates are a function not only of DMM sampling times, but also of relay switching times. Since such reading times can be generally less than 10/s, these readings tend to be done only when the extra resolution is absolutely necessary.

For a discussion of how data transfer rates over different interfaces affect your system’s overall measurement speed, see pages 6-7 of the application note, Test-system Development Guide: Computer I/O Considerations (AN 1465-2). For a detailed look at ways to maximize your system throughput, see the application note Test-system Development Guide: Maximizing System Throughput and Optimizing System Deployment (AN 1465-7).

Choosing a vendor

The proper design of instrumentation requires attention to minutiae. Choose an instrument manufacturer who has been through the learning process and knows how to minimize system noise and maximize accuracy and throughput.

Simple systems are one thing, but when you put several instruments together, strange things sometimes happen. That’s when it’s nice to have local support and service. Choose a vendor who can help you with issues like repeatability, system noise, calibration and drift.

If your vendor can supply specifications that apply to a whole subsystem—like a central switch—it will save you the time and trouble of trying to add all the specifications of a multitude of vendors together to divine what the true accuracy of your system might be.

Calibration can be an expensive and time-consuming part of building a system. Make sure you don’t have to ship your system halfway across the world to get it calibrated. Calibration is especially important in the world of RF and microwave, so make sure your vendor’s support organization can handle your needs.

---

**Figure 6.** Burst speed can be misleading. 4.5-digit measurement speeds: GPIB, PXI DMM. “C” combines A & B on same scale.
Example test system

To illustrate the concepts and issues discussed in this application note, we will design a test system from scratch that can be used to test low-frequency, low/medium-pin-count, low/medium-power electronic modules. These devices are typical of the automotive and aerospace/defense industries.

Make architectural choices

Table 1 shows the architectural choices we made for this test system.

Design the system

Now, we will apply the architectural decisions to a system for testing an electronic throttle module for an automotive throttle body. According to the test specification, the following equipment is required to run the tests:

- Programmable volt/ohm/ammeter
- Programmable power supply—0-13.5 V/0-10 A
- Waveform generator capable of pulse-width modulation, 0-10 VDC, 0-3 KHz
- Low current DC voltage source—0-5VDC
- Waveform analyzer
- CAN interface
- Simulated or actual stepper motor load

Table 1. Architectural decisions for sample test system

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>Decision</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instrumentation</td>
<td>Mix card-based and rack-and-stack instrumentation speed • Use VXI for higher-speed DMM, multi-channel DACs, and digitizer • Use rack-and-stack for other test instruments</td>
<td>Most cost-effective solution; helps optimize system speed Maximize system speed; digitizer not available as rack-and-stack instrument Accuracy, ability to prototype system before writing code</td>
</tr>
<tr>
<td>Computing</td>
<td>Use an external PC, not an embedded PC</td>
<td>Lower cost, standard interfaces</td>
</tr>
<tr>
<td></td>
<td>Use only industry-standard interfaces</td>
<td>Easier support</td>
</tr>
<tr>
<td></td>
<td>Use FireWire interface to control VXI instruments</td>
<td>For speed</td>
</tr>
<tr>
<td></td>
<td>Use Microsoft® Visual Studio.NET software</td>
<td>Rapid development</td>
</tr>
<tr>
<td>Switching</td>
<td>Place switching into a separate subsystem</td>
<td>Separate cardcage-based switchbox houses low-data-rate instruments more cost effectively</td>
</tr>
<tr>
<td></td>
<td>Use a matrix switching architecture for measurement instruments and low-current stimulus</td>
<td>Ease of expandability, more flexibility in where instruments can be connected</td>
</tr>
<tr>
<td>Mass interconnect</td>
<td>Place the DUT interface panel (mass interconnect or feedthrough panels) in front of the switching subsystem</td>
<td>Minimize cable length, save rack space</td>
</tr>
<tr>
<td>Power sources</td>
<td>Use high-current power supply and allow room for more than one in the rack</td>
<td>DUT requires high current. Bigger DUTs are expected from R&amp;D in the future</td>
</tr>
<tr>
<td>DUT-specific connections</td>
<td>Connect high-current DUT pins to general-purpose relays that can be wired to power supplies and loads</td>
<td>Ability to disconnect loads from DUT to allow other measurements to be made on those pins</td>
</tr>
</tbody>
</table>

Figure 7. Functional test system
The DUT has 14 pins total on 3 connectors. Looking at various catalogs, and adopting the architecture specified earlier, we chose the instruments shown in Figure 7.

There are four GPIB instruments—the power supply, switchbox, oscilloscope, and optional second DMM (useful for debugging since it does not require use of the PC). We will use a USB/ GPIB converter for these instruments so we do not need a slot in the PC for a GPIB card. It also provides access to USB in the event the GPIB cables and instruments are eventually replaced with USB versions, thus “future-proofing” the system.

Our system uses many I/O interfaces: RS-232C, FireWire, USB, GPIB, and LAN. Using Visual Studio.NET with IVI-COM and VXIplug&play instrument drivers along with VISA I/O libraries, the control program can communicate easily with instruments on all of these interfaces. In fact, should an instrument’s I/O interface ever change (say from USB to LAN), all that will have to change in the program is the initialization string. It is also possible to specify use of an aliased name to eliminate the hard-coding of I/O addresses.

Figure 8 shows how the instruments will be connected to the switching subsystem. We are using a matrix, so any instrument can be connected to any DUT pin, and we can add new instruments easily by expanding the number of rows and columns. All connections to the DUT except for the CAN bus are switched, making it possible to measure continuity from pin to pin. We are using a star ground to avoid ground loops.

A mass interconnect is an option for this system. This particular DUT only has 14 pins, so in an R&D or design validation environment you may not require the flexibility provided by such an interface. If the number of pins is small, simply bringing them directly out of the switchbox to DUT connectors may be sufficient. In the future, if the modules you are testing have more pins, or if you need a place to put other things between the system and the DUT, you may need a commercial mass interconnect solution. Therefore, we will provide a place directly in front of the switchbox for such an interface.

<table>
<thead>
<tr>
<th>Instrument</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rack-mountable arbitrary waveform/function generator</td>
<td>Need to generate PWM signals inexpensively</td>
</tr>
<tr>
<td>Heavy-duty power supply</td>
<td>Module requires 10A of inrush current</td>
</tr>
<tr>
<td>Optional DMM</td>
<td>Debug</td>
</tr>
<tr>
<td>Oscilloscope with CAN trigger module</td>
<td>Monitors signals including CAN traffic</td>
</tr>
<tr>
<td>Dedicated switching cardcage (“switchbox”)</td>
<td>Separate cardcage-based switchbox houses low-data-rate instruments more cost effectively</td>
</tr>
<tr>
<td>4-slot VXI cage containing:</td>
<td>Provides the most channels in a reasonable form factor; space for future expansion</td>
</tr>
<tr>
<td>• Digitizer</td>
<td>For high-resolution sampling</td>
</tr>
<tr>
<td>• 16-channel DAC</td>
<td>Need a DAC for generation of a brake signal</td>
</tr>
<tr>
<td>• High-speed DMM</td>
<td>Actual measurements are fastest with this one</td>
</tr>
<tr>
<td>• An RS-232C-based CAN interface is located on a shelf behind the PC</td>
<td>Module requires CAN interface for putting module in test mode</td>
</tr>
</tbody>
</table>

**Figure 8.** Block diagram of system
We chose a 5-wire measurement bus because it allows all four leads of the DMM to be connected to different pins on the DUT, making 4-wire ohms measurements possible. We routed two matrix points to the same pin on the DUT (as shown in Fig. 9 on the Pot1 and Pot2 Gnd pins), to make the resistance measurement very accurate, since the remote sense location is made right at the DUT. If you don’t use two wires, you can still make a 4-wire ohms measurement inside the relay matrix, which in some cases may be good enough.

The fifth bus wire is connected permanently to the star ground, and so it serves as a common reference for any single-ended devices, such as the oscilloscope, or for floating devices that can be connected to ground, such as the function generator, digitizer, DAC and DMM.

When you use a matrix, you can connect multiple signal sources to the same pin. It is important not to accidentally short such sources together. Switching routines should be carefully written to either eliminate this possibility or to offer warnings when improper conditions occur.

If you need to power up and run the DUT in full-functional mode, you may need to modify the test system either with more instrument busses or with more devices connected directly to the DUT. You must carefully analyze the type of testing that is required and plan accordingly.

It is helpful to make a wiring map that shows how the DUT will connect to your system. Table 3 shows how to make one using a spreadsheet. In the future, when it becomes necessary to test a different DUT, all you need to do is to create a new spreadsheet and wire the new DUT accordingly.

Since the system has many resources available and they can be expanded without changing the basic system architecture, new DUTs are easily accommodated. The spreadsheet is constructed with DUT pin names and numbers in the rows and system resources in the columns. Since star ground is physically located outside of both the system and the DUT, it shows up in both a row and a column. Wires are connected from the DUT pin number to the relevant system resource. For example, the battery input, Vbatt (J1-1), has two wires attached to it—one to general-purpose relay 7b and one to general-purpose relay 6b, which puts remote sense of the power supply right at the DUT. In addition to DUT pins, there are other internal system connections that must be made, and they are shown in a separate section of the spreadsheet.

<table>
<thead>
<tr>
<th>DUT Pin Name</th>
<th>Pin Nr</th>
<th>System Resource Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vbatt</td>
<td>J1-1</td>
<td>7b (PS+sense), 6b (PS+)</td>
</tr>
<tr>
<td>Power Gnd</td>
<td>J1-2</td>
<td>X</td>
</tr>
<tr>
<td>Brake</td>
<td>J1-3</td>
<td>9</td>
</tr>
<tr>
<td>Accelerator</td>
<td>J1-4</td>
<td>10</td>
</tr>
<tr>
<td>CAN H</td>
<td>J1-5</td>
<td>X</td>
</tr>
<tr>
<td>CAN L</td>
<td>J1-6</td>
<td>X</td>
</tr>
<tr>
<td>Pot1 Vref</td>
<td>J2-1</td>
<td>6</td>
</tr>
<tr>
<td>Pot1 Wiper</td>
<td>J2-2</td>
<td>6</td>
</tr>
<tr>
<td>Pot1 Ground</td>
<td>J2-3</td>
<td>7,8</td>
</tr>
<tr>
<td>Pot2 Vref</td>
<td>J3-1</td>
<td>2</td>
</tr>
<tr>
<td>Pot2 Wiper</td>
<td>J3-2</td>
<td>1</td>
</tr>
<tr>
<td>Pot2 Ground</td>
<td>J3-3</td>
<td>3,4</td>
</tr>
<tr>
<td>Motor +</td>
<td>J3-4</td>
<td>12</td>
</tr>
<tr>
<td>Motor –</td>
<td>J3-5</td>
<td>11</td>
</tr>
<tr>
<td>Other connections</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PS+Sense</td>
<td></td>
<td>7a</td>
</tr>
<tr>
<td>PS+</td>
<td></td>
<td>6a</td>
</tr>
<tr>
<td>PS-Sense</td>
<td></td>
<td>5a</td>
</tr>
<tr>
<td>PS –</td>
<td></td>
<td>4a</td>
</tr>
<tr>
<td>Motor Load +</td>
<td></td>
<td>3a</td>
</tr>
<tr>
<td>Motor Load –</td>
<td></td>
<td>2a</td>
</tr>
<tr>
<td>Earth Ground</td>
<td></td>
<td>1a</td>
</tr>
<tr>
<td>Switched Earth Ground</td>
<td></td>
<td>1b X</td>
</tr>
<tr>
<td>DUT Common</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Star Ground</td>
<td></td>
<td>13,14</td>
</tr>
</tbody>
</table>

This table shows how to make a wiring map using a spreadsheet. In the future, when it becomes necessary to test a different DUT, all you need to do is to create a new spreadsheet and wire the new DUT accordingly. Wires are connected from the DUT pin number to the relevant system resource. For example, the battery input, Vbatt (J1-1), has two wires attached to it—one to general-purpose relay 7b and one to general-purpose relay 6b, which puts remote sense of the power supply right at the DUT. In addition to DUT pins, there are other internal system connections that must be made, and they are shown in a separate section of the spreadsheet.
Conclusion

Before you begin choosing test instruments for your test system, you need to make a series of high-level decisions about your system architecture. The architecture you choose for your test system will depend on whether you plan to use it for R&D, design validation, or manufacturing test and on your budget and development-time constraints, your existing expertise and your measurement throughput requirements.

Important questions to consider include:

1. Should you use a rack-and-stack or cardcage architecture?
2. If you decide on card-based instruments, should you use an embedded PC (one that fits inside an instrumentation cardcage) or an external PC?
3. Which switch topology—simple relay configurations, multiplexers or matrices—and which switch types (reel relays, FETS or armature relays) should you use?
4. Does a mass interconnect make sense for your system?
5. Which power supplies and loads should you choose?
6. Which measurement and stimulus instruments should you choose?
7. What should you do to minimize your hardware costs?
8. What should you do to minimize development time?
9. What should you do to maximize system throughput?
10. Which hardware vendor should you use?

If you answer these questions carefully, you will help you ensure that your test system produces reliable results, meets your throughput requirements, and does so within your budget.
**Glossary**

**FireWire**—a high-speed serial bus defined by the IEEE-1394 standard

**Interface**—a connection and communication media between devices and controllers, including mechanical, electrical, and protocol connections

**IVI** (interchangeable virtual instruments)—a standard instrument driver model allowing you to swap instruments without changing software. Learn more at http://www.ivifoundation.org/

**IVI-COM**—IVI-COM presents the IVI driver as a COM object.

**VISA**—virtual instrument software architecture

**Visual Studio.NET**—Microsoft’s latest version of its Visual Studio development environment

**VMEbus**—an asynchronous bus technology defined by the IEEE-1014-1987 standard. VMEbus employs a master-slave architecture and allows you to use up to 21 card slots in a single backplane.

**VXI**—a standard, open architecture for cardcage test systems. The VXIbus (VMEbus eXtensions for Instrumentation) was developed by a consortium of test-and-measurement companies to meet the needs of the modular instrument market.

**VXIplug&play**—a hardware and software standard that allows interoperability between instruments made by different manufacturers. Learn more at http://www.vxipnp.org

**Related Agilent literature**

**Data sheets**
- Agilent 3499 Switch/Control System pub. no. 5988-6103EN
- Agilent 34970A Data Acquisition/Switch Unit, pub. no. 5995-5290EN

**Application notes**

**Test-System Development Guide:**
- Understanding Drivers and Direct I/O (AN 1465-3) pub. no. 5988-0110EN http://cp.literature.agilent.com/litweb/pdf/5988-0110EN.pdf
- Understanding the Effects of Racking and System Interconnections (AN 1465-6) pub. no. 5988-9821EN http://cp.literature.agilent.com/litweb/pdf/5988-9821EN.pdf

To discover more ways to simplify system integration, accelerate system development and apply the advantages of open connectivity, please visit the Web site at www.agilent.com/find/systemcomponents. Once you’re there, you can also connect with our online community of system developers and sign up for early delivery of future application notes in this series. Just look for the link “Join your peers in simplifying test-system integration.”
By internet, phone, or fax, get assistance with all your test & measurement needs

Online assistance:
www.agilent.com/find/assist

Phone or Fax

United States:
(tel) 800 829 4444
(fax) 800 829 4433

Canada:
(tel) 877 894 4414
(fax) 800 746 4866

China:
(tel) 800 810 0189
(fax) 800 820 2816

Europe:
(tel) (31 20) 547 2111
(fax) (31 20) 547 2390

Japan:
(tel) (81) 426 56 7832
(fax) (81) 426 56 7840

Korea:
(tel) (82 2) 2004 5004
(fax) (82 2) 2004 5115

Latin America:
(tel) (650) 752 5000

Taiwan:
(tel) 0800 047 866
(fax) 0800 286 331

Other Asia Pacific Countries:
(tel) (65) 6375 8100
(fax) (65) 6836 0252
(e-mail) tm_asia@agilent.com

Microsoft is a U.S. registered trademark of Microsoft Corporation.
Bluetooth is a trademark owned by the Bluetooth SIG, Inc., U.S.A. and licensed to Agilent Technologies, Inc.

Product specifications and descriptions in this document subject to change without notice.

© Agilent Technologies, Inc. 2004
Printed in the USA December 9, 2004
5988-9820EN
This application note is part of the Test-System Development Guide series, which is designed to help you quickly design a test system that produces reliable results, meets your throughput requirements, and does so within your budget.

This application note walks you through important considerations for arranging your test equipment in a rack, including weight distribution, heat dissipation, instrument accessibility and operator ease of use. It also explores ways to minimize magnetic interference and conducted and radiated noise to maximize measurement accuracy.

See the list of additional application notes in the series on page 11.

Table of contents

Introduction 2
Choosing racks and accessories 2
Instrument layout 3
Proper weight distribution 4
Instrument accessibility and operator ease of use 5
Instrument access during development and/or troubleshooting 5
Instrument access and ease of use during testing 5
Minimizing magnetic interference 6
Power dissipation and thermal management 6
Routing power and signals 7
Multiplexing and matrix switching 7
Wiring your system 7
Minimizing noise 8
Conducted and radiated noise 8
Grounding and shielding 9
Safety and interlocks 10
Mechanical 10
Electrical 10
AC power distribution 10
Conclusion 11
Related Literature 11

Agilent Technologies
Introduction

How you arrange test-system components can affect measurement accuracy, equipment longevity and operator ease of use and safety. In this application note, we focus on the important decisions you’ll make if you are building a system from rack-and-stack test instruments, or a mixture of rack-and-stack instruments and cardcage components, and you are using a racking cabinet to hold your system components. However, many of the concepts we discuss are applicable to bench-top systems that are not racked.

Choosing racks and accessories

Before you choose your rack cabinet and accessories, you need to clearly define the quantity and size of the components your rack will house. It is also important to be aware of how users will interact with the equipment, how the equipment will be maintained and any special needs such as environmental or security considerations or the need to transport your system after it is built.

To facilitate racking, most test equipment manufacturers build test equipment according to size standards established by the Electronic Industries Alliance (EIA). The standard heights, widths and depths are illustrated in Figure 1. Instrument widths are usually specified as full module width (MW) or half or quarter MW.

When you calculate the size rack you need for your test equipment, you need to decide whether the system controller (typically a computer) and monitor also will be installed in the rack to display test procedures and results. If you are incorporating a computer and monitor, will you also need a keyboard or mouse for operator inputs? If so, be sure to add space for these items into your calculations, along with space for a work surface. If there is a work surface, consider the fact that it may prevent the user from easily accessing any instrument in the space directly below the surface.

Figure 1. Most test instruments are a whole number of standard rack units (RUs) high and either a full, half or quarter module wide. A full module is typically 482.6 cm (19 inches) wide.
You may want to consider including space for accessory drawers, as well, to provide convenient storage for manuals, spare connectors and other small accessories (see Figure 2). Slide-out shelves are useful for attaching loads and other custom equipment, and they make access easy.

To maximize the re-usability of your test system, keep your future needs in mind when you choose your rack. In the future, you may want to add more instruments and more switches and accommodate bigger devices under test (DUTs) that require more power. To maximize your long-term flexibility, allow at least 20 percent extra room in your rack to accommodate instrument additions.

**Figure 2.** Adding an accessory drawer to your rack provides convenient storage for manuals, spare connectors and other small accessories.

Other questions to consider:

- What are the physical constraints of the location where your rack will be situated? Will the floor support your system’s weight? Are doorways into the facility tall and wide enough for the rack you are considering? Is there adequate power, and does the room have adequate cooling to support the additional heat created by the system?
- Will your system need to be moved to its final destination? If so consider using multiple smaller racks and limiting total rack weight. If you need to ship the system to another location, also consider using ruggedized rack furniture with strain relief fittings and keep shipping concerns in mind (shipping company or airline size and weight requirements, etc.).
- Do you need to be able to prevent or limit access to your system? If so, consider a rack with lockable doors.
- Will you need rear access to your equipment? If the only way to gain rear access to your equipment is to move your rack, you may want to consider installing sliding shelves instead. A sliding shelf allows you to pull the instrument out of the front of the rack for easier access to the backside of equipment.

**Instrument layout**

When you plan the layout of equipment in your rack, you will attempt to achieve a number of objectives simultaneously. You will aim to:

- Ensure rack stability by carefully distributing the weight of system components in the cabinet
- Make it easy for operators to use the system and be productive
- Minimize magnetic interference
- Provide adequate power and heat dissipation
- Route power and measurement and stimulus signals to the right place as efficiently as possible
- Minimize conducted and radiated noise
- Ensure operator safety

Plan your instrument layout on paper before you start installing instruments in your rack, since you will probably change your layout multiple times before you determine the optimal layout.
Proper weight distribution

It is important to minimize the risk of your rack tipping over to prevent injury to operators and damage to expensive equipment. To achieve the greatest stability for your rack, keep the center of gravity low by placing the heaviest objects—typically power supplies and signal generators—near the bottom of the rack. You will have to balance this need with the need to make frequently adjusted equipment easily accessible to operators.

In addition to keeping the center of gravity low, make sure the weight of your system is centered (front-to-back and side-to-side) as much as possible. You may need to mount some system components in the back of the rack, rather than the front, to achieve this balance.

When you calculate your system’s center of gravity, be sure to factor in the weight of the heaviest DUT you will be testing. Your system needs to be stable with and without the DUT in place. Also consider how weight distribution would change if you were to remove an instrument from the rack for maintenance, if the operator were to lean on the work surface or place heavy manuals on it, or if heavy instruments on slide-out rails were fully extended.

To download a spreadsheet that calculates center of gravity, go to http://www.agilent.com/find/rackcenterofgravity

If you have allowed room in the rack for future expansion, you will have empty spaces in your rack. To improve weight distribution, leave some empty spaces near the top of the rack for future addition of lightweight instruments and some at the bottom to allow for future addition of heavy instruments. Use a filler panel to cover the front of the rack to keep dust out of your system and help you manage airflow. Filler panels come in the same standard heights as test instruments (see Figure 1).

Keeping the center of gravity low is especially important if you will be moving your rack to another location after it is assembled, because the risk of tipping increases when you move it. Of course, the forces acting on your system’s center of gravity will change if the system is tilted, so be sure to take this into consideration if you intend to move your system up a ramp as you move it to its final location. When you design your rack, keep in mind that ramps in industrial facilities can be angled at up to 15 degrees, so make sure the rack cannot tip over at that angle. When you push the rack up the ramp, turn the rack so the heaviest part (typically the front if your equipment is front-mounted) faces uphill, if possible.

Once your system is in its final location, you can improve its stability several ways. You can bolt it to the floor, to a wall or to another test rack. If you bolt it to another rack or to a wall, make sure you do not disturb the airflow and cooling and that you leave enough room at the back of the rack for servicing equipment. Some racks are equipped with retractable stabilization feet that you can pull out of the bottom front of the cabinet to prevent them from tipping forward.
You also can use ballast, or weights that fasten to the bottom of the rack, to improve rack stability. Most racking systems offer ballast as an option. Ballast mounted at the back of the rack cabinet helps keep the cabinet from tipping forward if you extend heavy, slide-mounted devices from the rack or if you place a heavy object on a work surface that extends from the rack.

Ballast, retractable stabilization feet and bolting rack cabinets to the wall or floor provide an extra margin of safety, but you should not rely on these measures to compensate for poor weight balance in your rack. Always make sure the center of gravity of your system is as low as possible and the weight of your system is centered as much as possible.

Instrument accessibility and operator ease of use

If your system is fully automated, you may be concerned about instrument accessibility only during system development or troubleshooting. If your system is operated manually or semi-automatically, an operator’s ability to access instruments and use them easily during testing will be an important consideration as you decide how to rack your equipment.

Instrument access during development and/or troubleshooting

When they are low on rack space, system designers sometimes “bury” instruments inside the rack behind other instruments or mount them backwards or sideways in the rack. Before you choose this tactic, determine if you will need to access the instrument during system development to verify operation or for troubleshooting, repair or calibration. If you perform periodic system self tests to verify operation, you may need access to the front panel of an instrument, making “buried” installation impractical.

In some situations, reverse-mounting (or rear-mounting) instruments in a rack makes sense. For example, if you place the DUT interface panel (mass interconnect or feedthrough panels) in front of a switching subsystem that has the plug-in cards facing the interface panel, you minimize rack space, because the switchbox and mass interconnect are in the same plane, and you reduce wire length from the switching to the DUT. If the switch box you choose has cards in the rear, you can simply reverse-mount the switchbox using the rails on the rear of the rack, as illustrated in Figure 5.

You may be able to rear mount shallow instruments behind front-mounted instruments to save rack space. This space-saving technique can be a practical way to reduce rack height if you have a problem with low doors or you need to meet airline size requirements. However, mounting instruments in both the front and back of a rack can make servicing the instruments in your rack more difficult.

Instrument access and ease of use during testing

If you are designing a manual or semi-automated system, you need to ensure that the operator can reach the necessary equipment controls and connectors/patch panels without straining. Decide whether operators will sit or stand during testing and position the work-surface height accordingly. If a test instrument has a display the operator needs to see, place it at eye level or above, and if appropriate, provide the ability to tilt the display to reduce glare and eye-strain. If the operator will interact with a computer, place the monitor where the operator can see it easily. If the operator needs to use a mouse or keyboard, avoid placing these items on the same work surface as the DUT. Provide for left-handed and right-handed operators by allowing a mouse to be placed on either side of the keyboard.

When you are planning the operator work surface, make sure operators sit or stand far enough away from the rack that they do not inadvertently hit controls with their feet.

If you plan to ship the rack to another country, consider operator height and local safety rules, and make sure adequate preparations are made for power, cooling, etc., before the rack is shipped. Obviously, local-language instructions may be necessary in some cases. Inadequate preparation can sometimes cause long delays in system deployment.
Minimizing magnetic interference

Magnetic fields generated by test-equipment transformers can interfere with a cathode ray tube (CRT) display. CRT displays are typically found in computer and oscilloscope displays. If you put a power supply directly below a scope, the magnetic field from the transformer in the power supply can cause the scope CRT to waver to the point where it may not be usable. To alleviate the problem, move the receiving instrument away from the transmitting instrument. The intensity of the magnetic field decreases as the distance from the source of the field increases; the amount by which it decreases depends upon the configuration of the source of the field and the proximity to the source, but clearly, the greater the separation between the source and the receiving instrument, the lesser the effect.

In some cases, magnetic fields also can affect performance and accuracy of instruments that don't have CRTs. For example, a voltmeter's circuitry could be susceptible to a large magnetic field produced by a transformer. If you are having measurement problems with an instrument, keep in mind that magnetic interference could be one of the causes. Try moving the affected instrument away from likely sources of magnetic fields. Power supplies, fans and high-power-consuming instruments have a higher potential for producing large magnetic fields.

If moving the instruments is not an option, try adding magnetic shielding between the different rack layers or between the instruments. High-permeability metal (Mu metal) is sold for this purpose.

Vibration, especially in the presence of a magnetic field, is a difficult problem for system designers to solve. Cables moving in a magnetic field can generate current, and charge-related noise can be caused by internal stresses in vibrating cables connected to a charge amplifier or DMM. This issue is one of the big reasons for installing a mass interconnect in the system. It minimizes the relative motion between cables, and the chance of charge movement due to pinched cables.

Power dissipation and thermal management

All test instruments produce some heat when you operate them. If you have multiple instruments producing heat in an enclosed rack, the temperature can easily exceed environmental conditions specified for your test instruments. When this happens, your instruments can fail prematurely and your measurement results may be jeopardized. Temperature gradients are also an issue. If one end of the rack is ten degrees hotter than the other end, even if the overall temperature is within instrument specifications, the resulting gradient can cause unwanted thermocouple effects or slow drift errors.

The best way to dissipate the heat inside a rack is to increase airflow. Installing extractor fans in the top of the rack, as shown in Figure 6, improves natural convection cooling by increasing the airflow in the rack. The fan moves warm air from the bottom of the rack up and out through the vented top cap, providing cooling to the entire length of the rack. It is a good idea to use a fan when internal rack temperatures are 15°C (27°F) above ambient temperature.

If you cannot create enough airflow to remove the heat with a fan, you may need to consider air conditioning your rack. There are standard NEMA enclosures that can be used for this purpose.

When you install equipment in your rack, do not block instrument fans or side air holes and be sure to follow instrument manufacturers' recommendations regarding air flow and

Figure 6. Extractor fan installed in rack
clearance around instruments. In general, place your deepest instruments at the bottom of your rack. If you place a full-depth, full-width instrument in the middle of the rack, you block airflow to the instruments below it.

Typical top-mounted extractor fans will move about 200 cfm (cubic feet per minute) of air, which is sufficient for dissipating up to 2500 W of power inside a rack. If your system uses more than 2500W, you could install additional top-mounted fans or use a 600 cfm fan in the rear rack door to increase airflow.

If your system includes high-power instruments like ac sources or electronic loads with their own fans, use ductwork to vent them directly out the back of the rack. You can make the ductwork out of sheet metal.

The amount of power an instrument dissipates typically is specified by the instrument manufacturer. If that specification is not available, you can estimate power dissipation requirements from the maximum current specification using the equation

\[
\text{Worst case power (VA)} = \text{Voltage} \times \text{Amperage}
\]

This calculation provides a conservative estimate of power dissipation requirements because power in watts, the true source of heat, is always less than or equal to power in VA. It is a good idea to use conservative figures, to safeguard against worst-case situations.

Many test instruments draw a fixed amount of current. However, a power supply draws variable current depending on how much power it is providing to the device it is powering. When you calculate heat dissipation requirements, plan around a power supply’s maximum draw.

**Routing power and signals**

Once you have resolved the weight/balance issues, calculated your airflow and power needs and planned for operator accessibility, you are ready to turn your attention to how you will get power and signals to your instruments and your DUT. Your task is to route power and measurement and stimulus signals to the right place as efficiently as possible, while keeping noise to a minimum.

**Multiplexing and matrix switching**

Switches, or relays that route power and interconnect system instrumentation and loads to your DUT, are an integral part of most automated and semi-automated test systems and some manually operated systems. Multiplexers and matrix switches make it possible to minimize the number of test instruments in your system instead of using separate instruments for each test point. Switches deliver power and stimulus signals to the DUT when they are needed and route the measurement signals back to your test instruments.

Choosing the proper switch type and topology will impact the cost, speed, safety and overall functionality of your test system. For a thorough examination of switching in test systems, see Application Note 1441-1, Test System Signal Switching.

**Wiring your system**

Power wires radiate electronic noise and both stimulus and measurement signal-carrying wires are susceptible to this noise, so to minimize interference, separate power wires from signal-carrying cables. Proper shielding and grounding techniques can help alleviate noise problems (see “Grounding and shielding” on page 9). Selecting the proper type of cable is also important. A double-shielded or triaxial cable with insulation between the two shields provides the maximum protection against noise coupling.

In some cases, you may need to separate signal measurement cables (which can be sensitive to noise) from signal stimulus cables (which can generate noise). For example, if your stimulus signal is a high-frequency square wave with rapidly changing transitions (fast edges) produced by a function generator, it will radiate more noise than a square wave with slow edges or a high-frequency sine wave, and it would be more likely to interfere with the accuracy of a low-level measurement signal. If possible, keep wires carrying high-frequency square waves and other noise-generating signals away from your measurement paths to minimize interference.

For a detailed discussion of ways to reduce noise in switch systems, see the Application Note 1441-2, Reducing Noise in Switching for Test Systems.

www.agilent.com/find/systemcomponents
**Wiring dress and termination** — Good-quality cabling is expensive, and you will get the best results if you buy the best cabling your budget will allow. Make sure the cable you select is designed for the task you have in mind and be careful not to exceed the manufacturer’s ampacity rating of the wires you choose.

It is a good idea to adopt a systematic approach to arranging and managing your system’s cables. For a large system, you may want to consider using cable harnesses or looms. For a smaller system, cable ties may be adequate for bundling cables. Be sure not to wrap power cables in the same bundle as signal cables. For all systems, decide on a consistent method for labeling cables, as it will simplify trouble-shooting, maintenance and future replacements. On the label, include either a reference to a look-up table or a full description of the cable’s signal type, connectors and purpose. It is also a good idea to document the type and supplier for each cable you use and retain copies of datasheets for all cables and connectors.

Keep your cabling as short as possible to minimize voltage drop and interference, leaving just enough slack to allow you to keep it out of the way. If your instruments are mounted on sliding shelves or rack slides, make sure you allow enough slack to allow the equipment to slide all the way out.

Wire termination devices may be already mounted on the wires you purchase, or you may build your own wire terminations. If you build your own, use gold-plated pins and match the current rating of the pins to your application. Gold-plated pins cost more, but they last longer because they do not oxidize. Ensure that the pin and the wire both can withstand the maximum current you plan to use on that signal path or power path.

For RF applications, typically you will use coaxial cable — to match the characteristic impedance of the application and to minimize radiated noise — and terminate the cables with coaxial connectors to maintain the integrity of the connection between the inside of the rack and the outside of the rack. Of course, the coaxial signal path should also be terminated with the proper characteristic impedance to minimize signal reflections.

**Strain relief** — When you wire your system, be sure to protect your investment and minimize system downtime by minimizing sources of cable stress and damage, such as vibration, extreme bending and cutting and fraying caused by sharp edges. If your cable needs to pass through the rack cabinet wall, use a gasket in the hole and support the wire adequately along its path.

If you bend a wire back and forth repeatedly, it will eventually break. For wires in your system that need to be able to move, it is important to minimize the strain on the wires. For example, fixturing wires tend to move often as you connect and disconnect your DUT. If your system is designed for high-throughput manufacturing test, you will need to replace the fixturing wires regularly and pay careful attention to strain relief. Building strain relief into your system cabling helps protect both the cables and the connectors on the test equipment. Make sure that you support cables at regular intervals inside the rack cabinet, so the connectors do not bear the full weight of the cable.

**Minimizing noise**

We have already discussed some design considerations for reducing noise, but an understanding of where noise might originate is also helpful. In systems designed for testing electronic modules, the most significant causes of noise are conductive coupling, common-impedance coupling, and electric and magnetic fields. In addition, some systems are sensitive to noise from galvanic action, thermocouple noise, electrolytic action, triboelectric effect, and conductor motion.

**Conducted and radiated noise**

One of the easiest paths for noise to couple into a circuit is a conductor leading into it, resulting in conductively coupled noise. A wire running through a noisy environment has an excellent chance of picking up unwanted noise via radiation and then conducting it directly into another circuit. The power-supply leads connected to a circuit are often the cause of conductively coupled noise. Common-impedance coupling occurs when currents from two different circuits flow through a common impedance. The ground voltage of each is affected by the other. As far as each circuit is concerned, its ground potential is modulated by the ground current flowing from the other circuit in the common ground impedance, leading to noise coupling.
Radiated magnetic and electric fields occur whenever an electric charge is moved or a potential difference exists, and can also be a cause of noise coupling. In a circuit, high-frequency interference may be unintentionally rectified and appear as a DC error. Switch-system circuitry is also susceptible to electromagnetic radiation from radio, TV, and other wireless broadcasts, and it is important to shield sensitive circuitry from these fields. If you want to make accurate measurements of low-level signals in a test-system environment, you need to pay careful attention to the details of grounding and shielding.

It is always a good idea to have a line filter and surge protector in the main power distribution unit (PDU) of the rack. Also, each instrument usually has its own line filter, to reduce conducted interference from the instrument and reduce conducted susceptibility to the instrument. But remember, there is still some residual noise that each instrument can inject into the power grid. Sometimes it becomes necessary to put an additional power filter on an individual instrument to reduce its conducted noise.

**Grounding and shielding**

Grounding and shielding are the two primary methods for reducing unwanted noise in a test system. They often work together, such as when the shielding of a cable is connected to ground. In such cases it is important to understand where to ground the cable shield in order to maximize the shield’s effectiveness. In some cases, the solution to one noise problem may reduce the effectiveness of the solution to a different noise problem, making it imperative that you thoroughly understand the noise source, method of coupling, and noise receiver so you can make the appropriate tradeoffs.

When you design a grounding system, your goal is to minimize the noise voltage generated by currents from two or more circuits flowing through a common ground impedance, and to avoid creating ground loops that are susceptible to magnetic fields and differences in ground potential.

To accomplish these goals, instrument, power and safety grounds should all be connected as close as possible to the DUT’s power ground via a “star” mechanism as shown in Figure 7. This eliminates ground loops and contributes to quiet readings.

For a detailed discussion of grounding and shielding issues, see Application Note 1441-2, Reducing Noise in Switching for Test Systems and the white paper Considerations for Instrument Grounding.

In high-frequency systems, radio frequency interference (RFI) also can cause problems. To minimize RFI, make sure your cable diameter is suitable for the signal wavelengths you are transmitting, terminate all cables in their characteristic impedances, keep cable lengths as short as possible and use only high-quality cables and connectors. For more information, see the white paper, Proper Cable Shielding Avoids RF Interference Problems in Precision Data Acquisition Systems.

---

**Figure 7.** Star ground minimizes noise and eliminates ground loops.
Safety and interlocks

It is important to protect the safety of test-system operators, as well as safeguarding your DUT and the equipment in the rack itself. You need to plan for system safety as part of your overall system design, and you need to comply with company, local, national and international safety standards and regulations that may apply.

Install a system cutoff mechanism that is activated by any action that exposes the operator to potential harm. Make sure you document safety procedures and thoroughly train operators to use them.

Mechanical safety

Fans are a potential source of danger in a test system. Make sure that any fans you use are covered with fan guards that make them inaccessible to human fingers. Positioning fans on top of rack cabinets, instead of in the cabinet wall, reduces the chance that someone’s long hair could get sucked in unintentionally.

If the rack is only waist high, be careful to consider what might happen if a liquid is spilled on top of the rack.

To safeguard against a rack tipping over, use the guidelines discussed in the “Proper weight distribution” section of this paper (see page 4).

Electrical safety

Install a system cutoff switch (often called an emergency off switch, or EMO) where operators can reach it easily. The switch should cut power to the entire system, not just the DUT. If the cutoff switch is used, make sure operating conditions are safe before you restart the system. Label all high-voltage, high-current and high-power devices in red, and make it clear they are hazardous. Devices carrying more than 42 volts AC or 60 volts DC are hazardous. After a power outage, latching relays may or may not return to a safe state. Consider what they will be controlling, and what equipment they will be connecting.

One key to electrical safety is making high voltages inaccessible to operators. If your DUT requires high voltages or high-bias current, use an interlock mechanism to cut power to the DUT when the operator is able to contact it. For example, you can use a special fixture with a see-through cover fitted with an interlock mechanism that cuts power to the device when the cover is opened. Look for a power supply with a “remote inhibit” feature that lets you remotely inhibit the output by simply making the connection between two points.

AC power distribution

In a big system with 10 to 14 instruments, you typically plug each of the instruments into terminal strips inside the rack itself. The terminal strips may get their power from a large power distribution unit (PDU), which is usually located in the bottom of the rack. The PDU typically has a single line that exits the rack and connects to a power source on the wall, floor or ceiling. When you plan your system, check the AC input current rating of individual instruments and make sure the total does not exceed the maximum current you can draw from the terminal strips or from your AC mains supply. Using maximum current figures for each instrument will help you plan for a worst-case situation and avoid tripping circuit breakers. The disadvantage of planning around maximum current draws is that you have the potential for overdesigning your system and wasting capacity.

If a single-phase power line cannot handle your needs, you will need to move to a 3-phase AC input scheme. If you do not know what power types are available at your site, get that information from your facility engineers.

If you use 3-phase equipment in your system, make sure the instruments in your rack share power evenly across all three phases. For line-to-neutral loads, you can accomplish that by designing the rack with three terminal strips, such that each strip runs off one of the three phases. Connect your test instruments so they draw current fairly equally from the three strips. To make this task easier, create a list of the instruments in the rack and the current they draw, keeping in mind which instruments consume fixed power and which draw variable current. For variable-draw instruments, use the maximum current for your calculation.
To calculate power draw for line-to-line loading in a perfectly balanced system, take the sum of the loads and divide by the square root of three to determine the current that is actually being drawn by the phase feeding the system.

If you have no 3-phase equipment in your system, you do not necessarily need to balance power evenly across all three phases. You can just size your cabling for the largest phase load. You also will want to know the actual current draw on each phase (even if they are not balanced) so you can balance correctly in your facility. You could find this number by measuring the current on each AC line with a true RMS meter.

It’s a good idea to assess the quality of the mains power before installing any system. Use a power line monitor to check for voltage spikes (surge conditions) caused by motors, RF spikes, dropouts and brownout conditions (sag conditions). This simple test can save you headaches from non-repeatable results and also save damage to the test equipment itself.

**Conclusion**

It’s one thing to connect a PC to one instrument, but when a rack might contain $100,000 or more worth of equipment, it pays to do some planning. Arranging your test equipment in a rack to maximize measurement accuracy, equipment longevity and operator ease of use and safety also takes careful planning. Whether you are using your test system for R&D, design verification or manufacturing test, you need to consider a variety of issues, including weight distribution, heat dissipation, instrument accessibility and operator ease of use, and you need to pay close attention to minimizing magnetic interference and conducted and radiated noise.

**Related Agilent literature**

Agilent 3499 Switch/Control System, pub. no. 5988-6103EN

Agilent Enclosure Solutions Product Catalog, pub. no. 5980-0450E

Instrument Racks and Accessories: Calculated System Center of Gravity http://www.agilent.com/find/rackcenterofgravity

**Application notes**

Test System Signal Switching (AN 1441-1), pub. no. 5988-8627EN

Reducing Noise in Switching for Test Systems (AN 1441-2), pub. no. 5988-8626EN


Proper Cable Shielding Avoids RF Interference Problems in Precision Data Acquisition Systems http://we.home.agilent.com/upload/cmc_upload/memo/downloads/EPSPG084828.pdf


**Test-System Development Guide:**

- Introduction to Test-System Design (AN 1465-1) pub. no. 5988-9747EN

- Computer I/O Considerations (AN 1465-2) pub. no. 5988-9818EN,

- Understanding Drivers and Direct I/O (AN 1465-3) pub. no. 5989-0101EN

- Choosing Your Test-System Software Architecture (AN 1465-4)
pub. no. 5988-9819EN

- Choosing Your Test-System Hardware Architecture and Instrumentation
(AN 1465-5) pub. no. 5988-9820EN

- Understanding the Effects of Racking and System Interconnections
(AN 1465-6) pub. no. 5988-9821EN

- Maximizing System Throughput and Optimizing Deployment
(AN 1465-7) pub. no. 5988-9822EN

- Operational Maintenance (AN 1465-8) pub. no. 5988-9823EN

- Using LAN in Test Systems: The Basics (AN 1465-9) pub. no. 5989-1412EN
• Using LAN in Test Systems: Network Configuration
  (AN 1465-10) pub no. 5989-1413EN

• Using LAN in Test Systems: PC Configuration
  (AN 1465-11) pub no. 5989-1415EN

• Using USB in the Test and Measurement Environment
  (AN 1465-12) pub no. 5989-1417EN

• Using LAN in Test Systems: Applications,
  (AN 1465-14) (available in February 2005)

To discover more ways to simplify system integration, accelerate system development and apply the advantages of open connectivity, please visit the Web site at www.agilent.com/find/systemcomponents. Once you’re there, you can also connect with our online community of system developers and sign up for early delivery of future application notes in this series. Just look for the link “Join your peers in simplifying test-system integration.”

Agilent Open Connectivity
Agilent simplifies the process of connecting and programming test systems to help engineers design, validate and manufacture electronic products. Agilent’s broad range of system-ready instruments, open industry software, PC-standard I/O and global support combine to accelerate test system development. More information is available at www.agilent.com/find/openconnect.

www.agilent.com/find/emailupdates
Get the latest information on the products and applications you select.

By internet, phone, or fax, get assistance with all your test & measurement needs

Online assistance:
www.agilent.com/find/assist
Phone or Fax

United States:
(tel) 800 829 4444
(fax) 800 829 4433

Canada:
(tel) 877 894 4414
(fax) 800 746 4866

China:
(tel) 800 810 0189
(fax) 800 820 2816

Europe:
(tel) (31 20) 547 2111
(fax) (31 20) 547 2390

Japan:
(tel) (81) 426 56 7832
(fax) (81) 426 56 7840

Korea:
(tel) (82 2) 2004 5004
(fax) (82 2) 2004 5115

Latin America:
(tel) (650) 752 5000

Taiwan:
(tel) 0800 047 866
(fax) 0800 286 331

Other Asia Pacific Countries:
(tel) (65) 6375 8100
(fax) (65) 6836 0252
(e-mail) tm_asia@agilent.com

Product specifications and descriptions in this document subject to change without notice.

© Agilent Technologies, Inc. 2004
Printed in the USA December 9, 2004
5988-9821EN
This application note is part of the Test-System Development Guide series, which is designed to help you quickly design a test system that produces reliable results, meets your throughput requirements, and does so within your budget.

This application note discusses hardware and software design decisions that affect throughput, including instrument and switch selection, as well as test-plan optimization and I/O and data transfer issues. We also discuss ways to optimize your system as you prepare to deploy it.

See the list of additional application notes in the series on page 15.

Table of contents
Introduction 2
Upfront design decisions affect throughput 3
Making hardware choices 4
   Stimulus and measurement instruments 4
   Power supplies 5
Switches 7
Controller/PC issues 8
Designing your test plan for speed 8
   Optimizing test sequencing 8
   Organizing nested loops 10
Using triggering 11
Managing wait times 11
Choosing the fastest I/O and data transfer techniques 12
Fine-tuning your system for speed 13
   Minimize software delays 13
   Minimize state changes 13
   Instrument-specific tips 14
Conclusion 14
Glossary 15
Related literature 15

Agilent Technologies
Introduction

Throughput is a measure of the time it takes to test a device or product. Maximizing throughput is most critical in high-volume manufacturing, where you have thousands of products to test, and you want to test them as fast as possible. In high-volume manufacturing, you measure throughput in terms of devices per unit time. The faster you test your devices, the lower your manufacturing costs. In design validation testing, the speed of each individual test is not as critical, but test setup time is important because you need to be able to adapt to pinouts that change often. In design validation, you measure throughput in terms of tests per unit time. The faster you can validate your designs, the faster you can get your new products to market. In R&D, throughput is seldom an issue because you are not likely to repeat tests on large numbers of devices or to perform the same test repeatedly on a single device.

Taking the time to optimize system throughput may require some additional investment up front, but later on, the payoff in lower costs and faster time to market make the investment worthwhile.

As we pointed out in Application Note 1465-5, Test-System Development Guide: Choosing Your Test-System Hardware Architecture and Instrumentation, a test system is essentially a group of subsystems that work together. The hardware you choose for these subsystems and the software you write to make these subsystems communicate and interact has a huge effect on your system throughput. So, if throughput is critical in your test application, you need to choose equipment with the performance and features required for fast testing and then configure it and program it for optimum speed. After you’ve built your system, you can tweak instrument setups and operating procedures to further optimize its speed.

In general, your system first needs to set up a test, or configure the proper stimulus and send it to your device under test (DUT). Then your system needs to actually make the measurement on the DUT and transfer the measurement data back to the computer. Figure 1 shows typical steps a computer-controlled system would take to make a measurement. (The steps do not necessarily have to be executed in the order presented.) Each of these steps takes some amount of time to execute. To optimize throughput, you need to analyze how long the steps take in your system and decide which steps you can speed up. Depending on your application and budget, you may decide to work only on the steps that have the biggest impact on your throughput, or you may decide to invest the time and money to eliminate every unnecessary millisecond in the entire process.

Figure 1. Steps involved in making measurements with a typical computer-controlled system.

Steps
1. Tell system where to connect the stimulus
2. Wait for switch to settle
3. Tell stimulus instrument what signal to send to DUT (parameter, range)
4. Tell stimulus instrument to send signal
5. Wait for stimulus to settle
6. Tell switch to send DUT signal to measurement instrument
7. Wait for switch to settle
8. Tell measurement instrument what parameter to measure and the range in which that parameter falls
9. Wait for instrument to process command and complete configuration
10. Tell instrument to make the measurement
11. Wait for instrument to process command and make the measurement
12. Transfer measurement information to computer
In a typical test system, the steps with the biggest negative impact on throughput include instrument resets, delays (wait statements) programmed into the system software and waveform downloads. Power supply settling time, voltmeter measurements and switching also play a role. Figure 2 shows the hierarchy of delays in a typical test plan.

Obviously, if your system stops functioning, your throughput drops to zero. So, in all phases of product test (R&D, design validation and manufacturing test), minimizing system downtime is critical to maximizing throughput. To minimize system downtime:

- Select instruments from vendors you trust and choose instruments with high mean time between failures (MTBF) specifications.
- Establish a good spares program: keep backup components for your system, so if an instrument fails, you can quickly swap in a replacement and restore system functionality.
- Perform regular maintenance on your system and its components. Clean fan filters regularly to avoid heat build up (high temperatures contribute to failures). For more information on this topic, see Application Note 1465-8, Test-System Development Guide: Operational Maintenance.

In this application note we focus on improving throughput for systems designed with rack-and-stack test instruments. However, most of the concepts apply to systems built with VXI card-based instruments as well. VXI systems do have features that lend themselves well to optimizing throughput. For example, VXI backplanes have a built-in triggering bus that makes it easy to implement triggering schemes that can minimize system delays (see page 11 for more information on using triggering in your system). And digitizers, which are available only as card-based instruments, can trigger and return data faster than an oscilloscope, the closest rack-and-stack equivalent. But VXI and rack-and-stack systems are similar in most other regards, and you can use many of the same techniques for optimizing measurement speeds in both types of systems.

**Upfront design decisions affect throughput**

If you are designing a new system, rather than optimizing an existing system, you will have a greater opportunity to maximize your system speed. The system hardware and software architectures, instruments, switches, and I/O interfaces you select will have a huge impact on system throughput. For a detailed discussion of system hardware and software architectures, see two of the earlier application notes in the Test-System Development Guide series: *Choosing Your Test-System Software Architecture* (Application Note 1465-4) and *Choosing Your Test-System Hardware Architecture and Instrumentation* (Application Note 1465-5).

---

![Time spent during example test](chart.png)

**Figure 2.** Hierarchy of delays in a typical test plan

<table>
<thead>
<tr>
<th>Source of time spent</th>
<th>Seconds</th>
<th>Time spent (as % of total time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instrument resets</td>
<td>5 (36)</td>
<td></td>
</tr>
<tr>
<td>Wait statements</td>
<td>3.5 (25)</td>
<td></td>
</tr>
<tr>
<td>Arbitrary waveform downloads</td>
<td>2.7 (20)</td>
<td></td>
</tr>
<tr>
<td>Power supply settling</td>
<td>1.4 (10)</td>
<td></td>
</tr>
<tr>
<td>DMM readings</td>
<td>0.7 (5)</td>
<td></td>
</tr>
<tr>
<td>Switching</td>
<td>0.5 (4)</td>
<td></td>
</tr>
</tbody>
</table>
Making hardware choices

Figuring out how fast your system will perform measurements is harder than it appears. For example, you may decide to use a digitizer instead of an oscilloscope, to take advantage of the digitizer’s higher resolution. The digitizer may be able to sample 1000 readings very fast, but if those readings are transferred to the PC over GPIB, it could take a relatively long time. If you can download a decision-making algorithm into the digitizer, you can send a simple go/no-go result back to the PC, which would make GPIB a reasonable option. However, it takes extra effort to create and download a decision algorithm into an instrument, which may increase development time as well as “first-run” time of the test program. As you can see, there are many interdependent factors that affect throughput. If you are looking for test-time reductions amounting to fractions of milliseconds, you must weigh each of these factors carefully. Even if your throughput requirements are not that exacting, the hardware choices you make can significantly affect throughput.

One important factor to consider when you are selecting your instrumentation is command processing time, or the amount of time it takes an instrument to “digest” and interpret a command. Command processing time is usually characterized on an instrument’s data sheet. If you cannot find the information, ask the instrument vendor. Command processing times can range from less than a millisecond to dozens of milliseconds. If you send a command once to an instrument, it may not have a huge impact on your overall test time. But if you are sending the command repeatedly during testing, the time it takes can have a significant impact on your throughput.

As you explore the opportunities for improving your system throughput, keep in mind that when you reduce measurement time, you may sacrifice accuracy and repeatability. If you integrate measurements over a longer period of time you will filter out random noise, and your measurements will be more accurate. Typically, you can improve measurement repeatability by averaging measurements, increasing the number of samples taken per measurement or increasing the measurement sample time, but you will sacrifice measurement speed. If you cannot compromise accuracy and repeatability, it does not mean you will not be able to improve your throughput. Measurement time per se is just one factor to consider in the overall test plan, as illustrated in Figure 1.

In design validation, you typically perform a large number of different tests, so the time you spend setting up the test system is important. To minimize development time, use rack-and-stack system-ready instruments that incorporate a high percentage of the measurement solution you need. For example, if you use a source with modulation capability, you don’t have to develop your own algorithm or integrate additional hardware to generate the required modulation. Using instruments with IVI-COM drivers can save you development time. If the instrument has an IVI-COM driver, you can interchange hardware without rewriting your software, as long as you adhere to the functionality that is specific to the instrument class. See the application note, Test-System Development Guide: Understanding Drivers and Direct I/O (AN 1465-3), to learn how decisions about drivers affect development time.

Stimulus and measurement instruments

To maximize throughput, consider creating a Pareto diagram of projected delays (see Figure 2) in the system and invest your time and money accordingly. If tests A and B are of similar duration but test A is performed much more frequently than test B, then focus your programming efforts, tricks and budget on test A.
When you are choosing instruments, it is important to pay close attention to instrument specifications. For example, the Agilent 33120A function/arbitrary waveform generator is popular for systems applications. But its successor, the 33220A function/arbitrary waveform generator, downloads arbitrary waveform files 100 times faster than the 33120A, and many of its configuration times are faster (and it also costs less than the 33120A). If you have an existing system that includes 33120A function generators, it is fairly easy to upgrade to the 33220A because the two instruments are programmed similarly, and Agilent provides documentation to help you make the switch.

When you are perusing data sheets, pay particular attention to how measurement speeds are specified. Often, measurement speed specifications are related to the speed per reading when thousands of samples are taken, which is a data-acquisition use model. In functional test, it is far more common to close some relays, take a measurement, open those relays and move on to another measurement. In this mode, the measurement instrument’s single-point reading speed is most important, and it is dramatically slower than the fastest possible multi-sample reading speeds. In most cases, you will be able to look up the single-point reading speed on the instrument’s data sheet. Figure 3 shows that two instruments may have dramatically different multi-sample measurement speeds, yet their more commonly used single-sample measurement speeds are almost identical.

Look for instruments that have built-in features that will reduce the time needed for communication overhead and post-processing. For example, some test instruments can calculate arithmetic mean, minimum, maximum, and standard deviation. When you are analyzing multiple data points, these statistical results are much more meaningful than the raw data. Using the system controller to acquire raw measurements can be very time-consuming compared to transferring a few measurement results.

**Power supplies**

Your choice of power supply can dramatically impact system throughput, since waiting for power supplies to settle is typically a time-consuming element in a test plan (see Figure 2). Check the settling time specifications of the power supplies you are considering for your system. If you can’t find a specific reference to “settling time” on the data sheet, look instead for the “programming speed,” “programming response time,” or “rise and fall time” specification. Programming speed is defined as the amount of time it takes for the instrument to reach a specified percentage of the voltage setting (typically within 0.1%), not including command processing time. Rise and fall times are typically defined as the time it takes to get from 10 percent of the final value to 90 percent of the final value for the rise time, or vice versa for the fall time. Because of the different terminology and definitions, you must be careful when comparing settling times in power supplies from different vendors.

When you are trying to boost throughput in time-critical production test systems, look for a multiple-output supply that can set multiple outputs with a single command, like the Agilent N6700 series. Otherwise, consider using multiple single-output power supplies instead of one multiple-output supply. With multiple-output power supplies, the instrument takes extra time to parse commands, because you are sending an additional parameter to indicate which of the multiple outputs it should use. Also, with most multiple-output supplies,

---

**Figure 3.** Burst speed can be misleading. This diagram compares multi-sample and single-sample 4.5-digit measurements made with a GPIB DMM and PXI DMM. “C” combines “A” and “B” on the same scale.
commands sent to the various outputs are processed sequentially, one output at a time (this can be avoided with the Agilent N6700 series). With multiple supplies, one supply can be processing a command while the next is receiving a command, so you avoid delays. For details on using this technique and other techniques, see “10 Hints for Using Your Power Supply to Decrease Test Time,” publication number 5968-6359E.

Another way to reduce test time is to choose power supplies and electronic loads that have built-in measurement features. With power supplies, these capabilities let you measure the supply’s output voltage and current. With loads, you can measure load input voltage and current.

A good example is testing a DC-to-DC converter with four outputs, where you need to measure the input voltage to the converter and each of the four outputs in order to fully test the device. If you have a single DMM to measure the voltages, you’ll need a multiplexer to sequence through the measurements (Figure 4). In addition to the complexity of this setup, your test program needs to wait for the multiplexer’s switches to move and settle for each measurement.

The DC source and loads used to test the converter have built-in functions that can take care of the measurements for you (Figure 5). They’re already connected to the DUT, and there are no switching delays, so both the setup and test phases are much faster.

Note the use of remote sensing here. Although it isn’t required, using remote sense is generally a good idea because it provides regulation and measurement at the DUT rather than at the loads or the DC source.
With no need for switching, you'll benefit from faster tests, greater reliability and simpler configurations. This same approach works well for measuring current, and it eliminates the current shunts you’d otherwise need.

Using power supplies that incorporate a feature known as downprogramming can significantly reduce test time, particularly when you need to set multiple voltage level settings. Without downprogramming, the capacitor in the supply’s output filter (or any load capacitance) can take seconds or even minutes to discharge when you reduce the output voltage level (the lighter the load, the longer it takes).

Downprogramming uses an active circuit to force the output down to the new level within a matter of milliseconds in most cases. This circuit kicks in automatically whenever the voltage level you set (either manually or programmatically) is below the present output level. The down-programming level is fixed in most supplies, but some offer programmable downprogramming.

In time-critical tests, it’s a good idea to watch out for downprogramming delays. Because programming up is typically faster than programming down, try to sequence multiple tests in such a way that each consecutive test is at the same or higher voltage level as the previous test. See page 9 for more information on test sequencing.

**Switches**

Switches, or relays that interconnect system instrumentation and loads to your DUT, are an integral part of most test systems because they allow you to use a minimum number of stimulus and measurement instruments to test multiple points on your DUT. If your test plan involves lots of switching, switch speed will have a big impact on your system’s throughput, so the type of switches and the switch topology you choose are important. For a thorough examination of switching in test systems, see Application Note 1441-1, *Test System Signal Switching*.

From a system throughput standpoint, the most important switch parameter is settling time, or the time it takes to change states from open to closed and vice versa. Figure 6 shows the different actions and the relative times required for a relay to be closed, a measurement to be performed and for the switch to reopen and be ready for the next measurement.

---

**Figure 6.** This diagram shows what happens when you tell a switch to close, take a measurement, and then reopen. The “switch drive signal” represents the actual voltage that causes the switch to change states. The resulting “measured signal” is connected from the DUT to the measurement instrument.
Electromechanical switches, such as reed and armature relays, are common in low-speed applications. They are capable of switching high voltage and current levels, but they are limited to switching rates of dozens of channels per second for armature relays to hundreds of channels per second for reed relays. Reed relays are excellent choices to connect measurement instruments and low-current stimulus to your DUT. They are relatively fast (see Table 1), although they can have a higher thermal offset voltage than armature relays. Armature relays are slower, but you can use them for higher current loads. When you use armature relays, group your tests so the relays stay connected to perform as many readings as possible at one time.

Electronic switches, such as field-effect transistor (FET) and solid-state relays, are typically used in high-speed applications. However, some FET electronic switches cannot handle high voltage or current, and they must be carefully protected from input spikes and transients. Check the electronic switch ratings carefully.

Switching topologies can be divided into three categories based on their complexity: simple relay configurations, multiplexers and matrices. The best one to use depends on the number of instruments and test points, whether connections must be simultaneous or not, cost considerations and other factors. Typically, the type of relay you choose has a bigger impact on speed than the switch topology you choose, unless you factor in the time required for reconfiguring a switching system (which, as we noted earlier, is more critical in design validation applications.) If you use a switch matrix, you will be able to quickly and easily expand and reconfigure your system as your test needs change. Expanding and reconfiguring systems that use multiplexers typically is more time consuming.

A matrix arrangement of reed relays provides an excellent way to allow any instrument to be connected to any pin on your DUT, and it permits easy expansion as you add new instruments to your system or more pins appear on your DUT. Matrices use more relays than multiplexers, so they tend to cost more. If you don’t need to connect multiple instruments to any pin, a multiplexer is a suitable solution. If you have a 1 x 20 multiplexer for example, you can take a test instrument and connect it to 20 pins, but you can’t hook anything else to those 20 pins. With those same 20 relays in a matrix, you can connect four instruments to five pins in any combination.

If you want the ultimate in throughput and your budget is not limited, you can use multiple test instruments instead of a switching scheme for making measurements on multiple test points. With multiple instruments, you can set each to the needed range and eliminate the time spent on configuring the test instrument range, as well as the time required for switches to open and close. In some cases it is worth the extra money for the test time you save.

### Controller/PC issues

Unless your PC is ancient, its processor speed is not likely to be a significant factor in your test throughput. Typically, issues associated with stimulus and measurement instruments, power supplies, switches and test software play a much bigger role in determining system speed. Your PC is not in control of data collection speed, and faster PCs don’t necessarily collect data any faster. The PC’s interface to your test system (GPIB, LAN, USB, FireWire, VXI or PXI) will certainly impact data transfer time, but that is not dependent on PC processor speed.

Processor speed is a factor only if you are relying on your PC for analyzing data and if you are using it for your software development. You want to use the fastest PC available when you are compiling programs, but you do not have to do your development work on the same computer you use to run your system.

### Designing your test plan for speed

Many test programs spend most of the time waiting. Even if you have selected the fastest-available hardware for your system, software issues can slow your test-system throughput significantly. While you can tweak your test-system programming after your system is complete (see *Fine-tuning your system for speed*, page 13), you will achieve better throughput if you design your test plan upfront to optimize test sequencing and minimize delays.

### Optimizing test sequencing

In most test systems, single-instrument measurement times have a smaller impact on overall test time than the test flow (execution sequence) you choose when you are designing your test plan.

---

**Table 1. Relay comparison chart**

<table>
<thead>
<tr>
<th></th>
<th>Armature relay</th>
<th>Reed relay</th>
<th>Solid-state relay</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch speed</td>
<td>50/s</td>
<td>1000/s</td>
<td>1000/s</td>
</tr>
<tr>
<td>Contact resistance</td>
<td>Low</td>
<td>Very low</td>
<td>High</td>
</tr>
<tr>
<td>Life</td>
<td>1 million</td>
<td>10 million</td>
<td>&gt; 10 million</td>
</tr>
<tr>
<td>Typical failure mode</td>
<td>Fails open</td>
<td>Fails open</td>
<td>Fails shorted</td>
</tr>
<tr>
<td>Typical max input</td>
<td>250 V/2 A</td>
<td>100 V/100 mA</td>
<td>250 V/10 A</td>
</tr>
</tbody>
</table>
In a production environment, first arrange your test plan so the system can find DUTs that are destined to fail as soon as possible. If a particular DUT frequently fails a certain test, move that test to the front of your test program. Ideally, of course, you should feed reports of persistent DUT failures back into R&D or production engineering so they can be resolved permanently. Agilent offers a toolset, Fault Detective Diagnostic Solutions, to help with this process. Fault Detective helps you optimize throughput by quickly diagnosing functional failures in manufacturing and by finding redundancies in your tests. This toolset also helps you maximize quality by identifying gaps in your test process.

Next, when you are ordering your tests, minimize the number of times the stimulus, DUT, and measuring instrument change states—particularly those that take a long time—by organizing the program’s execution sequence. Start by looking for tests that leave the DUT in the desired state for the next test. If the DUT needs to be turned off for the start of a test, for instance, try to sequence a preceding test that leaves it off. If a particular test requires that the DUT is warmed up, place it later in the sequence and use a system timer to guarantee the DUT has been on long enough. These techniques can yield big improvements, although they are not always feasible.

The program sequence shown in Table 2 measures voltage or current on three different DUT test points under three different sets of input conditions. In this case, the ambient temperature setting is used as an example of a stimulus to the DUT. The temperature changes for each test point, and the measurement setup must also change to make the required voltage and current measurements. Each change adds time to the test program, reducing system throughput. For example, if you are using a DMM and you change the measurement function, the DMM reconfigures the hardware and retrieves different calibration constants before making a measurement.

### Table 2. Typical test sequence

<table>
<thead>
<tr>
<th>Program step</th>
<th>Input conditions (stimulus to DUT)</th>
<th>Measurement setup (to measure signal out of DUT)</th>
<th>DUT measurements taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Set input condition 1 (e.g., amb. temp. = 0 degrees C)</td>
<td>Prepare measurement setup 1 (e.g., voltage)</td>
<td>Test point 1 voltage</td>
</tr>
<tr>
<td>2</td>
<td>Prepare measurement setup 1 (e.g., voltage)</td>
<td>Test point 1 voltage</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Set input condition 2 (e.g., amb. temp. = 25 degrees C)</td>
<td>Test point 1 voltage</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Set input condition 3 (e.g., amb. temp. = 55 degrees C)</td>
<td>Test point 1 voltage</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Prepare measurement setup 2 (e.g., current)</td>
<td>Test point 1 current</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Prepare measurement setup 2 (e.g., current)</td>
<td>Test point 1 current</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Set input condition 1 (0 degrees C)</td>
<td>Prepare measurement setup 1 (voltage)</td>
<td>Test point 2 voltage</td>
</tr>
<tr>
<td>8</td>
<td>Prepare measurement setup 1 (voltage)</td>
<td>Test point 2 voltage</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Set input condition 2 (25 degrees C)</td>
<td>Test point 2 voltage</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Set input condition 3 (55 degrees C)</td>
<td>Test point 2 voltage</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Prepare measurement setup 2 (current)</td>
<td>Test point 2 current</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Prepare measurement setup 2 (current)</td>
<td>Test point 2 current</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Set input condition 1 (0 degrees C)</td>
<td>Prepare measurement setup 1 (voltage)</td>
<td>Test point 3 voltage</td>
</tr>
<tr>
<td>14</td>
<td>Prepare measurement setup 1 (voltage)</td>
<td>Test point 3 voltage</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Set input condition 2 (25 degrees C)</td>
<td>Test point 3 voltage</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Set input condition 3 (55 degrees C)</td>
<td>Test point 3 voltage</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Prepare measurement setup 2 (current)</td>
<td>Test point 3 current</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Prepare measurement setup 2 (current)</td>
<td>Test point 3 current</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>Set input condition 1 (0 degrees C)</td>
<td>Prepare measurement setup 1 (voltage)</td>
<td>Test point 3 voltage</td>
</tr>
<tr>
<td>20</td>
<td>Prepare measurement setup 1 (voltage)</td>
<td>Test point 3 voltage</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Set input condition 2 (25 degrees C)</td>
<td>Test point 3 voltage</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Set input condition 3 (55 degrees C)</td>
<td>Test point 3 voltage</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>Prepare measurement setup 2 (current)</td>
<td>Test point 3 current</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>Prepare measurement setup 2 (current)</td>
<td>Test point 3 current</td>
<td></td>
</tr>
</tbody>
</table>
If you organize the program to minimize changes to the stimulus conditions and measurement setups, overall test time is reduced. Note that the sequence shown in Table 3 provides exactly the same number and type of DUT measurements under exactly the same set of input conditions as the previous sequence, but the overall number of programming steps has been reduced from 24 to 14. Also, the number of stimulus changes has been reduced from 8 to 2, while the measurement setup has gone from changing back and forth 5 times to changing just once.

**Organizing nested loops**
Structure the basic test flow so that slow operations like setup, DUT connections and temperature settings are in the outermost loop. Nest faster operations like one-button measurements in lower-level loops. Place your fastest operations in the lowest-level loop. You can use a test flow diagram, as shown in Figure 7, to get a better conceptual understanding of the test plan and prevent wasted time in nested loops and poor use of DUT connects and re-connects.

**Figure 7.** To minimize overall test time, structure test loops so that the most time-consuming operations are performed the fewest number of times.

**Table 3.** Test sequence optimized for speed

<table>
<thead>
<tr>
<th>Program step</th>
<th>Input conditions (stimulus to DUT)</th>
<th>Measurement setup (to measure signal out of DUT)</th>
<th>DUT measurements taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Set input condition 1 (0 degrees C)</td>
<td>Prepare measurement setup 1 (voltage)</td>
<td>Test point 1 voltage</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>Test point 2 voltage</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>Test point 3 voltage</td>
</tr>
<tr>
<td>6</td>
<td>Set input condition 2 (25 degrees C)</td>
<td></td>
<td>Test point 1 voltage</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td>Test point 2 voltage</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td>Test point 3 voltage</td>
</tr>
<tr>
<td>10</td>
<td>Set input condition 3 (55 degrees C)</td>
<td>Prepare measurement setup 2 (current)</td>
<td>Test point 1 current</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td>Test point 2 current</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td>Test point 3 current</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Programming tips for fastest throughput**

- Graphical languages are not optimized for speed, so use a textual programming language. For fastest throughput times, write your test program in Visual C++ or C#.

- For fastest test execution, use direct I/O instead of drivers to send commands to your test instruments. However, if test development time is more critical in your application than test execution time, use drivers to minimize development time.

- Avoid the indiscriminate use of the reset command (*RST) to return test instruments to a known state after a measurement. It is best to place resets at the beginning of a test program to initialize the hardware the first time the program is run, then to manage the instrument states carefully so that they are in a benign state (equivalent to the reset state) at the end of the program.

- Use binary data format when transferring large amounts of measurement data.

- Do not use SLEEP statements for instrument-specific timing (consider the operation complete command, *OPC?, the wait command *WAI, and READ statements instead).

**Using triggering**

In typical test routines, it is common to apply a stimulus to a DUT, insert a delay (wait statement) in the system software to give the stimulus instrument and DUT time to stabilize and then instruct a test instrument to take a measurement on the DUT. However, the length of the required delay is typically a guess. Instead of adding delays to a test routine to assure that enough time has elapsed for the stimulus and DUT to stabilize, use triggering from the stimulus instrumentation (and sometimes from the DUT itself) to initiate a reading as soon as possible, especially if wait time delays comprise a significant proportion of your test time. Also, once a triggered sequence has been started, it is possible to make other measurements while waiting for the triggered measurement to finish.

You can use triggering built into a VXI or PXI backplane or with point-to-point wiring in a rack-and-stack system. In a rack-and-stack system, you need the right cables, the right connectors and a strategy for what is going to trigger what. In a VXI or PXI system, triggering is easier to implement because you don’t have to do any special wiring.

**Managing wait times**

When you are writing your test-system software, you can minimize delays by overlapping wait periods within specific tests. Here’s a typical sequence:

- Apply a load to the DUT or set up its programmed state and wait for DUT output to settle
- Connect relays to engage measurement equipment and wait for relays to close
- Set up measurement instrument and wait for setup to complete
- Initiate measurement and wait for measurement to complete
- Disconnect relays
- Turn off power source
- Wait for DUT output to settle

Each step usually involves a wait while the action completes. In addition, most DUTs need time to stabilize after power is applied or a load condition has changed. By separating the programming and wait stages, you can rearrange the test to program one instrument while waiting for another:

- Apply load to the DUT
- Connect relays to engage measurement equipment
- Set up measurement instrument
- Wait for the longest of all previous actions to complete:
  - Relays to close
  - Measurement instrument to settle
  - DUT output to settle
- Initiate measurement
- Wait for measurement to complete
- Disconnect relays
- Turn off power source
- Wait for DUT output to settle

Overlapping the wait periods minimizes overall delays. While the DUT is settling, the test program is busy programming the relays and setting up the measurement instrument.

To implement an overlapped wait, use a common or global timer. Each programming routine that sets up an instrument or DUT can tell a global timer how long each action will take; this identifies which action requires the longest wait. Then, when a measurement or other test requires that the previous commands be completed, a call to a single wait function will wait until the global timer expires before continuing.
• Apply load to the DUT
• Connect relays to engage measurement equipment
• Set up measurement instrument
• Wait for global timer
• Initiate measurement
• Wait for global timer
• Disconnect relays
• Turn off power source

With this approach, the test does not have to wait any more than is absolutely necessary for instrument setup, and the programming is simpler, too.

Other techniques for reducing software delays are discussed in the Fine-tuning your system for speed section on page 13.

Choosing the fastest I/O and data transfer techniques

In some test systems, I/O speed is not a major determining factor in overall throughput. This is especially true in RF systems, where the network analyzer or spectrum analyzer may take some time to complete a measurement. However, in systems that rely on unprocessed data, or where real-time control is important, your choice of I/O for your connection between your computer and your test system hardware can have a big impact on the overall test time. Dedicated-private LANs sometimes yield the best results. However, tests indicate that the extra overhead of all the layers of LAN and VISA actually make LAN slower than or comparable to GPIB unless you are transferring a lot of data. For the typical transactional model of electronic functional test, LAN may not be the best choice for measurement instruments. Still, even if you are not transferring large quantities of data, LAN can perform nicely for tasks like controlling stimulus devices or power supplies. LAN tends to run at its fastest if you make a direct socket connection.

USB is about three times faster than GPIB. FireWire is about four times faster than 100 Mbit LAN, so it is the best choice for a connection to a VXI system. MXI is faster yet, but requires a proprietary interface card in the PC. Table 4 shows the relative speeds for various operations for a stimulus instrument having GPIB, USB and LAN interfaces. The instrument’s internal speed clearly dominates setup changes, making I/O choices seem moot, but download speeds get much better with LAN and USB when large amounts of data are involved.

Dramatic improvements in LAN speeds are on the horizon. When LAN speeds accelerate into the gigabit range, using LAN will be a much faster way to transfer data than GPIB, USB, VXI or PXI. For more information about I/O and its effect on system throughput, see Application Note 1465-2, Test-System Development Guide: Computer I/O Connectivity Considerations and Application Note 1475-1, Modern Connectivity – Using USB and LAN Converters.

Keep in mind that if your instrument’s throughput is slow, you are not going to get greater throughput by changing to a faster I/O interface. You can improve your throughput by minimizing the number of GPIB transactions you send. When possible, send multiple GPIB commands at one time. This reduces bus turnaround times and allows the instrument, in some cases, to operate on the commands as quickly as possible.

The character format you use to transfer data can also affect the data transfer rate. You can choose from a variety of general formats, including character string, ASCII, or binary. Binary code is handled as bit streams, typically in block-length message units. These message units are more compact than those made up of string and ASCII characters, and therefore they can be transferred more quickly.

For example, when you are downloading a data file for an arbitrary waveform to a function generator, downloading floating-point values (a character string) is slower than downloading binary values, but using floating-point values is more convenient when creating the arbitrary waveform. Here, you need to decide which is a higher priority, faster data transfer (binary), or ease of use (floating-point values in the form of a character string).

<table>
<thead>
<tr>
<th>Interface</th>
<th>Function change</th>
<th>Frequency change</th>
<th>4K arb</th>
<th>64K arb</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIB</td>
<td>99 ms</td>
<td>2 ms</td>
<td>20 ms</td>
<td>340 ms</td>
</tr>
<tr>
<td>USB 1.1</td>
<td>100 ms</td>
<td>4 ms</td>
<td>10 ms</td>
<td>185 ms</td>
</tr>
<tr>
<td>USB 2.0</td>
<td>99 ms</td>
<td>3 ms</td>
<td>8 ms</td>
<td>100 ms</td>
</tr>
<tr>
<td>LAN (socket)</td>
<td>100 ms</td>
<td>3 ms</td>
<td>8 ms</td>
<td>110 ms</td>
</tr>
</tbody>
</table>
Fine-tuning your system for speed

Whether you are turning on a new system or fine-tuning an existing system, there are a number of techniques you can use to improve throughput. Relatively small adjustments to system software, instrument setups and operating procedures can help you optimize your system speed.

Minimize delays

As we noted in Figure 2, delays (wait statements) programmed into system software typically cause systems to run at suboptimal speeds. When you run a test program there are some operations — such as measuring a complex signal or moving data to an array — that take additional time to complete before the next command can be executed. If these operations do not complete before the next command in your program is executed, errors can occur and the program may halt. When debugging test routines, programmers frequently “fix” the problem by programming in a delay after the operation and before the next command. This is fine as a temporary fix for correcting an error, but it is important to remove the delays, or at least to make them as short as possible, once you find the real cause of the measurement problem. Leaving unnecessary delays in a program slows down the overall system throughput.

An alternative to using a delay is to use system-level control commands such as *OPC? (operation complete) to inform the control software that an operation is complete, which is especially useful for variable-length operations. Many instruments are IEEE-command compliant which means they are able to use the *OPC and *OPC? commands. Using *OPC? at the end of a command tells the instrument to return a +1 in response to the query as soon as the instrument command has finished executing. The next command in the program sequence can execute without any unnecessary delay.

You also can use SRQs (GPIB service requests) and IRQs (Windows interrupt requests) to minimize delays in your test software. The interrupt structure eliminates the necessity to conduct a poll or a loop waiting for something to happen. Such loops are time-consuming to write and slow to execute. With an SRQ or an IRQ, the hardware tells the control software when it is ready to have its data read (similar to a trigger).

Minimize state changes

We discussed ordering tests to minimize state changes in the “Designing your test plan for speed” section on page 8. If you optimized the order of your tests during the design phase, you may not need to tweak it after your system is up and running. If you are fine-tuning existing system software that was not written with speed in mind, you may find many opportunities to improve your throughput by reordering tests. Range and function changes are relatively slow and can interfere with fast tests. To compensate, arrange your tests such that tests involving different parameters or different ranges are grouped rather than intermixed. It is also helpful to pick a range that gives the needed resolution for most measurements and then keep it there. If you need to test multiple ranges or multiple parameters and your budget allows, you can use multiple test instruments and set each to a specific range or parameter.

Instrument-specific tips

To maximize throughput, make sure your test instruments are configured for speed. The following suggestions apply to many of today’s instruments:

• Make sure you are using the latest version of the instrument’s firmware. Firmware upgrades sometimes include significant measurement-speed enhancements.

• Turn off the display. Updating the display slows the reading time.

• Turn off all math/processing, unless using it allows the instrument to send a single pass/fail result instead of a stream of data.

• Set autozero to “once” or “off,” as this feature can double measurement time. However, do this only if the temperature drift in the system is minimal. Otherwise, an autozero should be performed periodically.

• Use the lowest-level commands you can. Instead of using “measure?,” use “config” “init” and “fetch?.” You do have to pay attention to where and how your readings are stored when you use these commands. For example, the Agilent 34401A multimeter treats “read?” and “init” followed by “fetch?” exactly the same except for where it stores the readings. INIT/FETCH buffers the readings, whereas READ places them immediately to the output buffer. By omitting this extra buffering step, you can get your reading to your computer faster.

• Use the fewest digits of resolution needed for the required accuracy.

• Avoid using “auto-range.” Define the expected value of a measurement so the instrument spends less time searching for the proper range. Bear in mind, though, that a malfunctioning DUT could result in a reading outside of the selected range. Your program must be able to react to overload readings correctly.

• Whenever possible, use preset states that can be used to recall instrument state setups.

www.agilent.com/find/systemcomponents 13
In addition to the general techniques listed above, there are specific techniques you can try with different types of test instruments:

**Function generators**
- Configure your setups in advance and store them into memory locations. Instead of sending down multiple commands to configure the instrument, you can recall the instrument state with a single command.
- When downloading arbitrary waveform data, send it in binary format rather than ASCII. Download the smallest number of arbitrary waveform points you can.
- Consider using modulation to respond to your system (AM, FM, PWM, PM, FSK). If you need the generator to respond to something else in your system, rather than reading a value and reconfiguring the function generator, see if you can use a control signal or even a conditioned signal as an external modulation signal.

**Counters**
- Use ASCII format for fastest throughput (note: this is different from other instruments)
- Select the trigger level instead of using auto level
- Use the auto arming mode
- Disable printing operation
- Define the trigger command so the fetch command does not need to be sent for every measurement
- For some measurements, a counter may produce readings in which the last few digits are not stable. This can slow a test if a human operator needs to discern the difference in readings. Truncating the last digits will produce a more understandable display, but some tests require that extra resolution. Have the counter calculate the arithmetic mean if you require high resolution and a stable reading, or use a limit-testing mode.

**Digital multimeters**
- When using a scanning meter like the Agilent 34970A, wire adjacent channels so that the DMM doesn’t have to switch modes or ranges
- Select the shortest channel delay (zero)
- Turn off scaling
- Turn off alarms
- Use the fast filter (PLC 0.02)
- Turn off T/C (thermocouple) check. Some scanning meters will check for the existence of a thermocouple by looking for a short circuit before attempting to read the thermocouple voltage.
- Shield the measurement setup to reduce noise pick-up from the operating environment. Shielding may allow you to make measurements with shorter measurement times (aperture) or with less filtering and still achieve sufficient noise rejections to obtain the required accuracy.
- Try to make all readings with the DMM “LO” terminal connected to circuit low. DMMs have fairly large values of capacitance between “LO” and earth which must be charged (increases settling time) when you make “floating” measurements.

**Scopes and digitizers**
- If you are importing raw data, use binary transfer mode. Specifically, use byte or word formats. Word format is more accurate but requires twice as much data to be sent over the bus. Some scopes produce more than 8-bit resolution, but many acquisition modes produce only 8-bit data. In these cases, transferring word versus byte data will take twice as long and not provide any additional resolution. It is important to know how and when the instrument produces extra resolution.
- Capture only as much data as you need to analyze.
- Turn off special features like mask test, jitter analysis and FFT functions.
- Make sure you have an adequate trigger rate, and use the fastest sweep speed (timebase scale) that is consistent with your application. Long acquisition times and/or slow trigger rates can gate your throughput if your analysis program is very fast.

**Power supplies**
- If your power supply has list mode, use it to store complete instrument setup states and recall them with a single command, rather than sending a long series of configuration steps.
- Use the built-in measurement capabilities
- Use power supplies with downprogramming capability

**Conclusion**
If you want to maximize system throughput, you need to choose the right equipment and program it for optimum speed. The system hardware and software architectures, instruments, switches, and I/O interfaces you select have a huge impact on system throughput. If you carefully evaluate the complex interplay of the hardware and software elements of your test system, you will find many opportunities for improving the speed with which your system performs measurements. After you’ve built your system, you can tweak instrument setups and operating procedures to optimize speed. The time you spend doing so will help lower your costs and accelerate your time to market.
Glossary

C#—(pronounced “C sharp”)—a new C++-like, component-oriented language that was built to run on the .NET Framework

FireWire—a high-speed serial bus defined by the IEEE 1394 standard

Interface—a connection and communication media between devices and controllers, including mechanical, electrical, and protocol connections

IVI—interchangeable virtual instruments—a standard instrument driver model allowing you to swap instruments without changing software. Learn more at http://www.ivifoundation.org/

IVI-COM—IVI-COM presents the IVI driver as a COM object in Visual Basic.

VISA—virtual instrument software architecture

VXI—VXI is a standard, open architecture for cardcage test systems. The VXIbus (VMEbus eXtensions for Instrumentation) was developed by a consortium of test-and-measurement companies to meet the needs of the modular instrument market.

Related Agilent literature

Data sheets

Agilent 3499 Switch/Control System, pub. no. 5988-6103EN
Agilent 34970A Data Acquisition/Switch Unit, pub. no. 5985-5290EN
Agilent Fault Detective Developer Application and Run-Time Engine, pub. no. 5988-4009EN
Agilent 33220A 20 MHz Function/Arbitrary Waveform Generator, pub. no. 5988-8944EN
Agilent N6700 Low-Profile Modular Power System, pub. no. 5989-0489EN

Application notes

Test-System Development Guide:

• Introduction to Test-System Design (AN 1465-1) pub. no. 5988-9747EN http://cp.literature.agilent.com/litweb/pdf/5988-9747EN.pdf
• Computer I/O Considerations (AN 1465-2) pub. no. 5988-9818EN. http://cp.literature.agilent.com/litweb/pdf/5988-9818EN.pdf
• Understanding Drivers and Direct I/O (AN 1465-3) pub. no. 5989-0110EN http://cp.literature.agilent.com/litweb/pdf/5989-0110EN.pdf
• Choosing Your Test-System Software Architecture (AN 1465-4) pub. no. 5988-9819EN http://cp.literature.agilent.com/litweb/pdf/5988-9819EN.pdf
• Choosing Your Test-System Hardware Architecture and Instrumentation (AN 1465-5) pub. no. 5988-9820EN http://cp.literature.agilent.com/litweb/pdf/5988-9820EN.pdf
• Understanding the Effects of Racking and System Interconnections (AN 1465-6) pub. no. 5988-9821EN http://cp.literature.agilent.com/litweb/pdf/5988-9821EN.pdf
• Maximizing System Throughput and Optimizing Deployment (AN 1465-7) pub. no. 5988-9822EN http://cp.literature.agilent.com/litweb/pdf/5988-9822EN.pdf
• Operational Maintenance (AN 1465-8) pub. no. 5988-9823EN http://cp.literature.agilent.com/litweb/pdf/5988-9823EN.pdf
• Using LAN in Test Systems: Applications, (AN 1465-14) (available in February 2005) Modern Connectivity—Using USB and LAN Converters. (AN 1475-1) pub no. 5989-0123EN
  10 Hints for Using Your Power Supply to Decrease Test Time, pub. no. 5988-6359E


Application notes

To discover more ways to simplify system integration, accelerate system development and apply the advantages of open connectivity, please visit the Web site at www.agilent.com/find/systemcomponents. Once you’re there, you can also connect with our online community of system developers and sign up for early delivery of future application notes in this series. Just look for the link “Join your peers in simplifying test-system integration.”
By internet, phone, or fax, get assistance with all your test & measurement needs

Online assistance: www.agilent.com/find/assist

Phone or Fax

United States:
(tel) 800 829 4444
(fax) 800 829 4433

Canada:
(tel) 877 894 4414
(fax) 800 746 4866

China:
(tel) 800 810 0189
(fax) 800 829 2816

Europe:
(tel) (31 20) 547 2111
(fax) (31 20) 547 2390

Japan:
(tel) (81) 426 56 7832
(fax) (81) 426 56 7840

Korea:
(tel) (82 2) 2004 5004
(fax) (82 2) 2004 5115

Latin America:
(tel) (650) 752 5000

Taiwan:
(tel) 0800 047 866
(fax) 0800 286 331

Other Asia Pacific Countries:
(tel) (65) 6375 8100
(fax) (65) 6836 8252
(e-mail) tm_asia@agilent.com

Microsoft and Windows are U.S. registered trademarks of Microsoft Corporation.

Product specifications and descriptions in this document subject to change without notice.

© Agilent Technologies, Inc. 2004
Printed in the USA December 9, 2004
5988-9822EN
This application note is part of the Test-System Development Guide series, which is designed to help you design a test system that produces reliable results, meets your throughput requirements, and does so within your budget.

This application note examines what to do as your system is put to use. It covers issues related to worldwide deployment, calibration, diagnostics and repair, cleaning, upgrades and expansion.

See the list of additional application notes in the series on page 7.

Table of contents

Introduction 2
Worldwide deployment considerations 2
  Power 2
  Cooling 3
  Line frequency 3
  Logistics and ergonomics 3
Calibration 4
  In-house calibration 4
  Contracted calibration 5
  Swap 5
Diagnostics and repair 5
Cleaning 6
Upgrades and expansion 6
Conclusion 7
Related Literature 7
Introduction

Once you’ve created and debugged your test system, you will be putting it to use. But even the best-designed system requires routine calibration and maintenance, and will occasionally fail. Planning for such eventualities will help to reduce the system’s downtime.

The issues most often encountered are:

- Worldwide deployment considerations
- Calibration
- Diagnostics and repair
- Cleaning
- Upgrades and expansion

Worldwide considerations

Sometimes, systems are shipped from country to country as needs change or manufacturing lines are moved. If you are building a system that might be transported elsewhere, you need to account for the difference in line voltage and line frequency, both from the standpoint of equipment power input and changes to cooling fans that may be required. In addition, there are ergonomic considerations you should think about because of differences in culture or physical characteristics of the operators who will use the system.

Power

Your system is composed of instruments, power supplies and computing equipment that could all be required to run on different line voltages and frequencies. If your system will travel from country to country, you must plan for the changes in voltage or it will be a tedious job changing the equipment’s fuses and input switches. Some older equipment must be removed from the system and have its top covers removed in order to reach the internal switches. If possible, choose equipment that runs from 90-252V (to handle Japan’s 100V lines at low-line and Europe’s 240V lines at high-line) without requiring changes to switch settings or fuses. Information on the most common line voltages, power plug styles and other useful data for various parts of the world is available from the U.S. Department of Commerce at http://www.ita.doc.gov/media/Publications/pdf/current2002FINAL.pdf.

Another useful item to consider when shipping systems from country to country is a power distribution unit (PDU). These devices can convert 3-phase inputs into line-to-neutral or line-to-line voltages, and they also can detect low- or high-line conditions. They sometimes can be connected to uninterruptible power supplies, too. A good PDU will also have an emergency off (EMO) switch input, allowing the operator to shut off all or some of the power in an emergency. Figure 1 shows typical wiring for a PDU that is used in many Agilent systems.

Figure 1. A typical AC power distribution unit.
Cooling

Fans are another problem area when line voltage varies. A 240V fan may work when operated at 120V, but at a much lower speed. Thus, the airflow may no longer be sufficient to cool the system. Conversely, a 120V fan may burn up when connected to 240V. It can be a nuisance to replace the fans every time the system is shipped from country to country. But fans that can be operated from any line voltage are produced in smaller quantities and are thus much more expensive than single-voltage AC fans.

DC fans, though, can be an excellent choice for systems that must be moved often. A small, fixed 12 V or 24 V DC power supply with universal AC input (i.e., 100-240 VAC) can be installed in the system and connected to the DC fan(s). Other advantages of DC fans are:

- More control over airflow and noise. The speed of the fan is directly related to the input voltage. A 24-volt DC fan can typically be operated between 12 and 28 volts DC. At 12 volts DC, the fan will operate at half speed, producing less air and less noise.

- The life expectancy of a DC fan is higher than that of a comparable AC fan, since DC fans are many times more efficient. The correspondingly low heat dissipation reduces the thermal load on the bearings, thereby increasing lifetime.

In non-air-conditioned factories, temperatures sometimes may exceed the ability of simple fans to keep the instruments operating within their specifications. In this case, consider a dedicated air conditioner for the system. NEMA enclosures are available for a wide variety of rack sizes. These completely enclose the system, and provide a way to attach air conditioner intake and exhaust. Appropriate ductwork must also be added to the factory. See http://www.nema.org

Line frequency

The frequency of AC line voltage varies in different parts of the world. In the U.S., 60 Hz is standard. In many other countries, it is 50 Hz. While this won’t affect most modern power supplies, it can certainly affect signal measurements. It is common to take low-noise DMM readings with a “1-line-cycle” integration time. At 60 Hz, this is 16.667 ms. At 50 Hz, it is 20 ms. Some DMMs, such as Agilent’s 34401A, automatically adjust their integration time based on internally measured line frequency. Others must have this information programmed into them. It is important to set your DMM correctly based on line frequency.

At lower frequencies, the magnetizing current of transformers and motors can go up, even to the point of saturating the core. This can cause nonlinear magnetic fields and overheating of the core, especially at 47 Hz, creating a situation where products designed in a 60 Hz environment can cause problems in other parts of the world.

Logistics and ergonomics

The doorways in many European countries are short, since many buildings are old and were built when people were shorter. Thus, a 2-meter rack may not fit through the doors. Taller racks also require larger aircraft to transport them. If you build systems this tall, your shipping costs may be significantly higher over the life of the system if it is moved or shipped frequently.

In some Asian countries where real estate is in scarce supply and space is at a premium, facility aisles and hallways are extremely narrow. It may be difficult or impossible to move a deeper- or wider-than-normal system to its intended location. Once positioned, it could be difficult to open front or rear doors.

The average population height varies country-to-country, too. Use care to place keyboards and monitors at an elevation that is not too high for shorter operators. It is also a good idea to provide keyboard/mouse trays with adjustable heights and provisions for left or right-handed operators.

Your safety department can provide you with modern guidelines for ergonomic standards.

www.agilent.com/find/systemcomponents
Calibration

Most electronic instruments require periodic calibration that is traceable to a government standards agency such as NIST (National Institute of Standards and Technology) in the U.S. This requirement guarantees that measurements meet their published accuracy specifications. Calibration is not the same thing as diagnostics, which are simple tests to verify that the instrument is operating and taking measurements that are at least close to what they should be. Diagnostic tests and fixtures are discussed in the next section.

It may seem logical to build calibration fixtures that would allow your system to be automatically calibrated without having to remove equipment. Unfortunately, such fixtures would be prohibitively expensive. Calibration requires use of components that meet stringent specifications under closely controlled conditions of temperature and humidity. Oil-baths containing “standard” resistors at controlled temperatures, frequency-measuring equipment that connects to the NIST cesium-beam frequency standard and the like are not easily contained in a removable fixture.

There are three ways of assuring that a test system is calibrated:

• Have an in-house calibration lab perform calibration either in the system or by removing instruments, calibrating them and returning them to the system.

• Hire a firm that provides calibration services at the location of your system.

• Swap instruments with calibrated spares, then send the replaced units out for calibration.

Whichever plan you use, it is essential to track the date of each instrument’s last calibration, and to set up a method for notifying appropriate personnel when the next calibration due date arrives. You could simply place a dated sticker in a conspicuous place on the instrument whenever it is calibrated, and have someone check dates periodically, or you could program the system with “due” software that notifies appropriate personnel automatically.

In addition to regular calibration, keeping a log is a good practice. It’s good to be able to correlate manufacturing anomalies to the particular operator, time of day, calibration period, run number and to many other manufacturing variables.

Before you build that test system in Germany for shipment to Thailand, for example, try to answer these questions: Do I have the same calibration system in both places? If not, can I guarantee the measurements made by my test system here will be the same after I ship the test system overseas? Can I get the accuracy I need in both places, and are the calibration services adequate?

In-house calibration lab

If you do not already have an in-house calibration department, you might consider setting one up, although the cost and time to do so can be considerable. If you intend to offer calibration services to others outside your company, your customers may require you to have international accreditation. A good place to start is the International Laboratory Accreditation Cooperation (www.ilac.org). Members of ILAC, such as the American Association for Laboratory Accreditation (A2LA—www.a2la.net), will certify your lab after you have met their requirements, a process that can take from four to nine months once the lab is fully operational. If you desire to have your lab accredited, the international standard ISO 17025 will apply. It is not necessary to become accredited, but at the least, you may wish to become ISO 9000 certified.
Fluke Corporation has a great deal of support for creation of calibration facilities. There are several companies that provide support software for Fluke’s Met/Track and Met/Cal software. More information is available from http://calibration.fluke.com.

Contract services
For a broad range of calibration services covering many types of instruments, professional instrument calibration services are available from Agilent. See www.agilent.com/find/calibration for details. Non-Agilent equipment is included. Contracted services can be arranged in various levels, from single instruments on an as-needed basis to scheduled volume on-site calibration (VOSCAL).

Swap and return
The third method of calibrating equipment is to simply replace units when they need calibration with others that are still within their calibration period. This requires keeping one or two spares on-hand, which can be expensive. However, it is a good idea to keep some spares handy anyway if system uptime is critical. We’ll talk about this more in the next section.

There is one caveat in swapping instruments. A replacement may be completely within its calibration specifications, but if it is operating at the opposite end of its calibration range from the original instrument and the production device being tested is already near its limit, a statistical variation could result that is large enough to cause a yield problem. The solution is to run a statistical analysis on the results. This analysis is called a “Gage R&R” study, and it is covered in the next section.

Diagnostics & Repair
Perhaps the hardest thing to do once you have a test system finished is to spend some extra time designing a diagnostics test program that can help locate the source of problems when they arise. But it is time well spent. Here’s what to do:

• Execute a self-test on every instrument that has this capability.
• Measure the output of every stimulus device with an appropriate measurement device to verify that all instruments are working and taking readings that are nominally correct. This is not sufficient to guarantee that they are in calibration, but it is good enough for a diagnostic tool.
• Feed a small DC voltage from a stimulus device (digital-to-analog converter, power supply, etc.) successively through all internally available switching paths and back to a DMM. This verifies the switching subsystem.
• Create a special fixture that loops signals that cannot be automatically connected internally back into the system. This is called a diagnostic fixture. Use the same procedure previously described to measure continuity of these paths.
• Read switch cycle count information from any switch box that has this capability. This data can give you early warning of relays that are nearing the end of their specified lives.

• Some instruments can do limited internal automatic calibration (sometimes called “auto-adjustment”). This automatic procedure should be done periodically, but not necessarily every time diagnostic test programs are run. Keep a programmatic calendar to remind the operator to run such programs when the due date occurs (usually about every 30 days).
• Attach a known good device under test (DUT) to the system and run a full suite of tests on it. This technique is not foolproof, since characteristics of such a “golden DUT” can change over time as components age. A useful way to counter this effect is to periodically run a “Gage Reliability and Reproducibility” (Gage R&R) test on the system. There are two sources of variation in any system: the variation of the product and the variation of the measurement system. The purpose of conducting the Gage R&R is to be able to distinguish between the two so as to reduce the measurement system variation if it is excessive. This means running a large quantity of known good boards on the system periodically to obtain a statistical sampling that can be compared to reference data to see if there is any long-term drift in the measurements. Such a study can also be used initially to study the measurement statistical parameters, which can be used to set acceptable upper and lower limits on each test. Look for “statistical process control” (SPC) and “statistical quality control” (SQC) software tools that can help you create such data.
In a production environment, diagnostics can be run daily or at the beginning of a shift. In a design validation or R&D environment, running the test once a week or less may be adequate. Once a problem is identified, the next step is to fix it. There are several things you can do to ensure fast repair:

- Make it easy to replace instruments. Make sure that mounting screws are not hidden, that cables are easily removed from the instrument (and labeled so they are replaced correctly), and that instruments are not buried, necessitating removal of other instruments to get to them.
- Although PCI slots in a rack-mounted computer are tempting spots to put instruments (since they do not take up additional rack space), remember that removing the computer from the rack to get to them is tedious and time-consuming.
- Use a limited set of custom cables and keep spares on-hand in case they need to be replaced. Use standard, easily available cables whenever possible.
- Fixture connectors can wear out over time. Have a good stock of replacement connectors available.
- Computers are a constant source of problems. Hard disks go bad, monitors quit, and keyboards and mice get dirty. Have spares available. Most importantly, keep important files somewhere else or back up the computer regularly to guard against loss of data.
- Maintain an inventory of spare instruments — this can be expensive, but so is a down production line. Remember, too, that the cost of many plug-in cards for PXI and VXI is greater than an equivalent rack-and-stack instrument because rack-and-stack instruments typically are produced in higher volumes.

Thus, it is less expensive to inventory spares of box instruments, and they can double as debug tools when not in use inside a system.

- Place more than one of a key instrument in your system when you design it. For example, an inexpensive DMM could be integrated into the system for use during manual debug, but pressed into service should the main, high-speed DMM require service. With IVI drivers, such interchangeability should not require a change to the software.
- Heat and thermal gradients are enemies of any test system. Provide adequate airflow to minimize heat rise, and avoid a situation where you are continually changing the thermal environment of the test equipment.

Cleaning

Maintaining good airflow through your system is essential, because it keeps the temperature under control, assuring that instruments are operated within their temperature specifications. Many instruments have removable air filters, so be sure to inspect these regularly and clean or replace them when necessary. Some racks are also available with air filters. These should also be inspected regularly. Keep cables away from the filters. If cables must be moved in order to reach the filters, the flexing can make the cables eventually break, causing reliability problems unrelated to dirty air.

If many operators will be using the system, it is a good idea to periodically clean the keyboard, mouse, barcode reader and touchscreen, as applicable. You generally can use simple household cleansers. Disease can be spread easily from one person to the next via these devices. Trained operators may be hard to find, so keep them healthy!

Upgrades and expansion

If you’ve designed your system well, using the concepts highlighted in earlier papers of this System Developers’ Guide, it will be able to handle new instruments easily. You’ve left extra space in the rack for additional or bigger instruments, and you’ve allowed expansion room in your switching or instrumentation cardcage if present. You’ve also designed the switching system in such a way as to allow instruments to be added to the system by simply plugging the new inputs and outputs into a place you’ve reserved for future instruments (such as the unused rows of a switching matrix, as described in Application Note1465-5, Choosing Your Test-System Hardware Architecture and Instrumentation). You’ve got room in your fixturing system for more pins, and you’ve developed a small set of reusable cables to connect those into your instruments and switches.

In the software realm, you’ve planned for upgrades by doing regression testing every time a major piece of software is changed. This means allowing time to re-run the Gage R&R, diagnostic test plan and/or known-good DUT when the operating system, test executive, drivers or other support routines are modified. You’ve also documented the software and allowed for code changes to be easily tracked. You’ve written the software in an environment standard to the PC industry so anyone familiar with languages like Visual Basic or C can take over the system software and make necessary changes as the years go by.
Conclusion

Test systems have made the task of repetitive testing both faster and more reliable, but there’s a much to consider to keep them running. You must factor in worldwide power issues, calibration, diagnostics, repair, cleaning, upgrades and expansion. At Agilent, we appreciate the talent and effort required to design, build and implement exceptional test systems. We hope this series of application notes in the System Developer’s Guide has helped make your life a little easier.

If you are creating a test system or need help with one you already use, you can find more help at Agilent Developer Network, www.agilent.com/find/adn, or search our systems information at www.agilent.com/find/systemcomponents.

Related Agilent literature

Application notes

Test-System Development Guide:
- Introduction to Test-System Design
  (AN 1465-1) pub. no. 5988-9747EN
- Computer I/O Considerations
  (AN 1465-2) pub. no. 5988-9818EN,
- Understanding Drivers and Direct I/O
  (AN 1465-3) pub. no. 5989-0110EN
- Choosing Your Test-System Software Architecture (AN 1465-4)
  pub. no. 5988-9819EN
- Choosing Your Test-System Hardware Architecture and Instrumentation
  (AN 1465-5) pub. no. 5988-9820EN
- Understanding the Effects of Racking and System Interconnections
  (AN 1465-6) pub. no. 5988-9821EN
- Maximizing System Throughput and Optimizing Deployment
  (AN 1465-7) pub. no. 5988-9822EN
- Operational Maintenance
  (AN 1465-8) pub. no. 5988-9823EN
- Using LAN in Test Systems: The Basics
  (AN 1465-9) pub. no. 5989-1412EN
- Using LAN in Test Systems: Network Configuration
  (AN 1465-10) pub. no. 5989-1413EN
- Using LAN in Test Systems: PC Configuration
  (AN 1465-11) pub. no. 5989-1415EN
- Using USB in the Test and Measurement Environment
  (AN 1465-12) pub. no. 5989-1417EN
- Using LAN in Test Systems: Applications,
  (AN 1465-14) (available in February 2005)

To discover more ways to simplify system integration, accelerate system development and apply the advantages of open connectivity, please visit the Web site at www.agilent.com/find/systemcomponents. Once you’re there, you can also connect with our online community of system developers and sign up for early delivery of future application notes in this series. Just look for the link “Join your peers in simplifying test system integration.”
By internet, phone, or fax, get assistance with all your test & measurement needs

Online assistance:
www.agilent.com/find/assist

Phone or Fax

United States:
(tel) 800 829 4444
(fax) 800 829 4433

Canada:
(tel) 877 894 4414
(fax) 800 746 4866

China:
(tel) 800 810 0189
(fax) 800 829 2816

Europe:
(tel) (31 20) 547 2111
(fax) (31 20) 547 2390

Japan:
(tel) (81) 426 56 7832
(fax) (81) 426 56 7840

Korea:
(tel) (82 2) 2004 5004
(fax) (82 2) 2004 5115

Latin America:
(tel) (650) 752 5000

Taiwan:
(tel) 0800 047 866
(fax) 0800 286 331

Other Asia Pacific Countries:
(tel) (65) 6375 8100
(fax) (65) 6836 2552
(e-mail) tm_asia@agilent.com

Product specifications and descriptions in this document subject to change without notice.

© Agilent Technologies, Inc. 2004
Printed in the USA December 9, 2004
5988-9823EN
By internet, phone, or fax, get assistance with all your test & measurement needs

Online assistance:
www.agilent.com/find/assist

Phone or Fax

United States:
(tel) 800 829 4444
(fax) 800 829 4433

Canada:
(tel) 877 894 4414
(fax) 800 746 4866

China:
(tel) 800 810 0189
(fax) 800 829 2816

Europe:
(tel) (31 20) 547 2111
(fax) (31 20) 547 2390

Japan:
(tel) (81) 426 56 7832
(fax) (81) 426 56 7840

Korea:
(tel) (82 2) 2004 5004
(fax) (82 2) 2004 5115

Latin America:
(tel) (650) 752 5000

Taiwan:
(tel) 0800 047 866
(fax) 0800 286 331

Other Asia Pacific Countries:
(tel) (65) 6375 8100
(fax) (65) 6838 0252
(e-mail) tm_asia@agilent.com

Product specifications and descriptions in this document subject to change without notice.

© Agilent Technologies, Inc. 2004
Printed in the USA December 28, 2004
5989-2178EN