Building Hybrid Test Systems, Part 1

Laying the groundwork for a successful transition

Application Note 1465-32

In the 1990s, GPIB- and VXI-based systems dominated test system architectures. With the introduction of PXI in the mid '90s and LXI in 2005, it's likely that your inventory of test equipment now consists of multiple architectures. Looking forward, one question may come to mind: When it comes time to build a new test system, which architecture will work best?

If we compare and consider the alternatives, the "best" answer is probably a hybrid of two (or maybe three) test architectures. There are at least four good reasons to consider this alternative:

- All of the functions you need are not available in a single architecture
- You can't meet every physical, wiring or performance need with a single architecture
- Certain functions are much more cost-effective in one architecture versus another
- You would like to reuse existing equipment that may not be in your preferred architecture

As an analogy, think of home audio/visual (A/V) systems. If you are setting up a home theater—or building a new entertainment room—you might choose to purchase the entire system at one time. The simplest approach is to select a set of components from one manufacturer. This maximizes the chances of all parts working together flawlessly—perhaps with one remote control—and providing the best overall "system" performance. Of course, you will need to buy cables, cabinetry and so on from other manufacturers, but the core components should work together very nicely.

Contrast this with what may be the more common situation: You have an existing A/V system and decide to add a new capability. You don't want to throw away your existing equipment, but still want everything to work well together. This may lead you to replace several components and make a number of modifications. The total cost is less than buying a new system, but operating the system may be more difficult than with new, compatible, state-of-the-art components guaranteed to work together.

Comparing approaches in test systems

Test systems follow a similar model. It can be much easier to assemble and program a system of the newest components from one architecture; however, this is not always a cost-effective path. A hybrid architecture is often chosen because either existing equipment must be used or a core capability is not available in the preferred architecture.

Building your system in a single architecture can have clear benefits:

- A simpler programming interface with consistent instrument driver types and debugging methods
- Greater simplicity in cabling and connecting of instruments, switches and test devices
- Consistency in packaging, racking and human interface
- The potential for enhanced performance

There will also be architecture-specific benefits because each one offers unique functions, capabilities and performance points.

Building a hybrid test system requires a wide range of decisions, but a few quick rules-of-thumb will simplify the process:

- Try using no more than two architectures. Using more is generally too complex and is more prone to problems. Think back to the A/V analogy: If you buy each component from a different manufacturer, you may have problems with incompatible cabling, remote controls and so on. Similar issues may arise when mixing multiple test architectures and products from multiple vendors.
- Build as much of your system as possible in a single, newer architecture. This will let you use many newer capabilities and maximize the potential benefits.
- When updating an existing system, carefully consider which parts to keep or replace—instruments, application software, instrument drivers, underlying support tools, and more. Ideally, it's best to change only one or two elements at a time. Conversely, it's important to consider the age and support status of every part of your system. We'll cover this in more detail throughout this note.

Working through the issues

To illustrate some of the complexities and considerations involved, let's start by outlining the various components or layers that make up your test system:

- System management
- Test application or program
- Input/output (I/O) library
- Instrument control
- Computing
- Connections
- Instruments
- Fixturing, cabling and routing
- Device under test (DUT)

Clearly, there are numerous choices to make within each level (Figure 1). Key factors such as your test requirements, performance needs and available resources will influence your decisions within each level—and will affect your choice of primary and secondary architectures.

Figure 1. Components of a test system

System management	**Test executive, database program**
Test application or program	**VEE Pro, MATLAB®, LV, Visual Studio®**
Input/output (I/O) library	**VISA, NI-488, SICL**
Instrument control	**SCPI, IVI-COM, IVI-C, VXI*plug&play***
Computer	**PC with Windows®, Linux or real-time OS**
Computer-to-instrument connections	**LAN, GPIB, USB, PXI, VXI**
Instruments	**DMM, scope, analyzer, source...**
Switches, fixtures, cables, routing	**Instrument-to-DUT connections**
Device under test	

The following general process can be helpful when building a hybrid test system:

1. Inventory the components you have on-hand and decide which ones to continue using.
2. Determine which measurements and requirements aren't covered by the equipment from Step 1.
3. Determine if you can make all measurements (and meet all other requirements) with just one architecture.
4. Determine how you will combine multiple architectures.
5. Resolve any software issues that result from the chosen combination of architectures.
6. Work out other issues such as computer, operating system, assembly, cabling, and shielding.
7. Order or obtain instruments, software and any other required system elements.

To ensure your success, there are a few key points to consider within Steps 1, 2, 3, 4, and 5.

Step 1. When completing the inventory of instruments, also consider your performance needs. In some cases, it is prudent to replace one or more key instruments where a newer model can significantly improve performance. The existing instrument may still be usable elsewhere.

Steps 2 and 3. These require you to select a test architecture that solves your unmet measurements and requirements. For a majority of mainstream applications, Figure 2 provides a summary of test functions available in the major architectures. The details will certainly change with time, and even an architecture with a poor score may provide the specific capabilities you need. One note: Older architectures may have more choices, but the products may not offer the same performance, features or cost-effectiveness as solutions available in newer architectures.

Figure 2. Relative functionality coverage (1 to 5) for the five major architectures

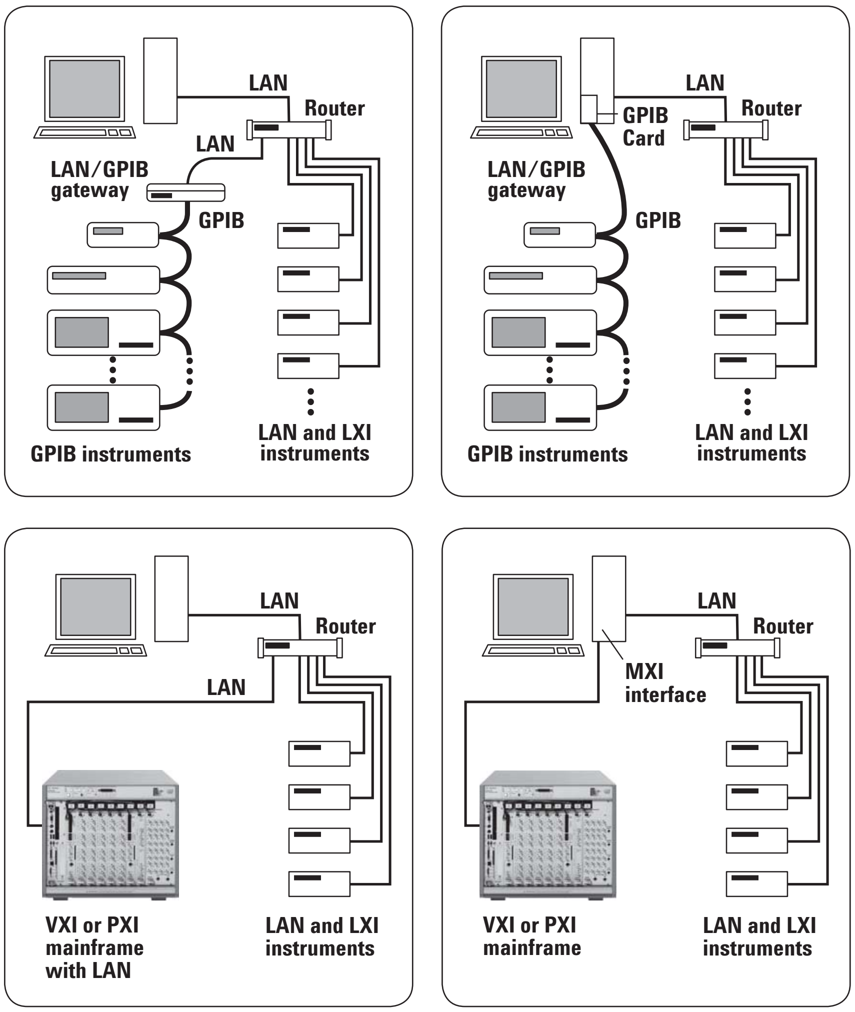
Interface function	GPIB					LXI					PXI					VXI					USB				
Power supply	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩		🟩					🟩					🟩	🟩	🟩		
DMM	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩		🟩	🟩	🟩	🟩		🟩	🟩	🟩			🟩	🟩			
Oscilloscope	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	●	🟩	🟩				🟩	🟩				🟩	🟩	🟩		
Function generator	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩			🟩	🟩	🟩			🟩	🟩	🟩			🟩	🟩			
Counter	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩			🟩	🟩	🟩			🟩	🟩	🟩			🟩	🟩			
Switching	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩			
Spectrum analyzer	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩				🟩					🟩	🟩	🟩		
RF sig generator	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩		🟩					🟩	🟩			
Digitizer	🟩	🟩	🟩			🟩					🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩			🟩	🟩	🟩		
Digital I/O	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩			🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩		🟩	🟩	🟩		
Pulse/pattern gen	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩		🟩	🟩	🟩	🟩		🟩	🟩				🟩				
Logic	🟩	🟩	🟩	🟩	🟩	🟩				●	🟩					🟩					🟩				
Digital-to-analog	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩			🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩			🟩	🟩	🟩		
Network analyzer	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩		🟩					🟩					🟩				
Data acquisition	🟩	🟩	🟩	🟩		🟩	🟩	🟩			🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	

● Additional LAN-based solutions available (not LXI-compliant)

Step 4. This has implications for several of the layers in Figure 1. For example, connectivity choices can affect your computing choice. The absolute best connectivity performance is usually obtained with dedicated I/O cards plugged into a computer, but this may limit your choice to one: a desktop computer located close to the test equipment. The LAN and USB interfaces built into today's PCs can minimize this problem, but it is also worth considering alternatives such as remote (LAN) connections and I/O gateways or converters as ways to achieve greater flexibility. Specifically, many I/O converters support programming modes that make them transparent to the system software. For instance, the Agilent E5810A LAN/GPIB gateway makes it possible to program GPIB instruments as if they were connected through a dedicated GPIB card, even though they are physically connected to a LAN.

Figure 3 shows various connection alternatives for common test architectures. Note that the modular architectures (e.g., VXI, PXI, PXI Express) have somewhat less flexibility because they use a centralized computing architecture and depend on a fast, local backplane.

Figure 3. Connection alternatives for common hybrid systems



Step 5. At first, the software issues may appear to be the most problematic part of building a hybrid test system. Within Figure 1, software elements are present in multiple layers, and each of these must be tested against the earlier decisions. Here is one suggested approach:

1. **Decide on the operating system.** Legacy systems may be on Windows® 98 or Windows NT®, but Microsoft® no longer supports either one. Windows 2000 support ends in 2010. Whether using a Windows-based OS, Linux or a real-time operating system, you may want to ensure continued support over the expected lifetime of the test system—or at least through the development phase.

2. **Check the expected support life of your application software.** If moving from an older system, you may need to upgrade to a newer revision or release. Note that your choice of application software also affects your choice of instrument driver (discussed below).

3. **Ensure I/O library support.** For new programs, we strongly recommend using VISA, which is an open, industry-standard library. This may automatically link in if you are using typical test-and-measurement application software. It will also maximize portability for future projects. Note: Many GPIB instruments are accessed via the proprietary NI-488 library. Fortunately, the NI and Agilent I/O libraries can both support mixed VISA and NI-488 calls. Be sure to use a revision of the I/O libraries that is supported on your chosen operating system.

Instrument programming

The decision to use drivers or direct programming depends on instrument architecture, age of instrument, programming language, and your personal preferences.

With PXI, PXI Express or VXI, you will need a driver to control the instruments. These architectures have a backplane connection and use register reads and writes to control their modular card-based instruments. Most modules contain limited processing power, depending on the host computer for all control and processing tasks. This requires the use of drivers that are compatible with your chosen operating system and preferred programming language.

With GPIB, LXI or USB instruments, you can generally choose between writing instructions in the native language of the instrument or using an instrument driver. SCPI is the most common native instrument language and has the advantage of being extremely portable, working in virtually every revision of every operating system. It can also provide complete, fine-grained control of the instrument. Because SCPI is the instrument's native language, it also tends to have fewer bugs than a driver, which may have been written long after the instrument was introduced. In contrast to SCPI, instrument drivers may not implement every instrument function, so this is another advantage that favors SCPI.

SCPI programming has a downside: it doesn't integrate seamlessly into your programming language. SCPI programming is executed by sending ASCII strings through the write and read functions of the programming language while drivers provide a programming interface that appears as a logical extension of the programming language.

If you chose to use a driver, you next need to determine which type to use. Early instrument drivers were proprietary, written specifically for each test-focused language. LabVIEW is the only common test language still using proprietary instrument drivers, which are required to maintain its graphical interface. If you are programming in LabVIEW, your best choice is to use these drivers although LabVIEW can also use industry standard drivers. Other languages usually call either an IVI (C or COM) or VXI*plug&play* driver.

VXI*plug&play* drivers were the original open driver used across companies in the test industry, and many older instruments have these drivers. Because they are DLL-based, all have the potential revision issues associated with shared library drivers. VXI*plug&play* drivers were updated to become IVI-C drivers. IVI-C drivers have all the same benefits and issues as VXI*plug&play* drivers. IVI-COM drivers use the Microsoft COM model, making them much more portable to today's modern languages.

There is no perfect choice for drivers. If you are using a more modern language that can use COM objects, we suggest using any IVI-COM drivers available. If there is not an IVI-COM driver available, you'll need to find an IVI-C or VXI*plug&play* driver.

Many instruments are shipped with a CD that contains instrument drivers. Both Agilent and NI also provide websites where you can download the latest drivers. Go to **www.agilent.com/find/ADN** for Agilent drivers and **www.ni.com/devzone/idnet/** for NI drivers. Other instrument manufacturers typically post drivers on their websites, too.

4. Determine your instrument-control driver or language.
Modular architectures such as VXI and PXI require an instrument driver for instrument control (e.g., to perform common measurement and control functions). In contrast, GPIB, LXI and USB instruments often have a built-in native language called Standard Commands for Programmable Instruments (SCPI), but drivers are also commonly available.

Writing directly in SCPI generally provides the best performance and finest control, but this approach may not integrate as smoothly as a driver will into your programming language. *Note:* There are multiple types of drivers, and these have been created for a number of reasons. Please refer to the "Instrument programming" sidebar for more about this topic.

At this point you're ready to assemble the rest of your system. The website **www.agilent.com/find/open** has pointers to application notes that cover the common issues in system creation.

Assessing your system priorities

In building a hybrid test system, the best choice for a given situation often depends on your priorities. Whichever way you go, there are likely to be challenges. If you understand the possible issues, however, you should be able to avoid most problems.

There are two common scenarios in system creation:

- **Minimum development time**
  You're probably combining old and new equipment and want to rapidly assemble a working system so you can start testing the DUT as soon as possible. System assembly, programming, and debugging are necessary, but you want to minimize the time spent on these activities. You want good performance, but it is not your top priority.
- **Maximum overall performance**
  You're testing a large volume of products and the per-unit test time will dominate the overall cost of test. You are willing to spend more time up front to get better system performance.

Hybrid systems are a viable approach to either scenario. The next application note in this series will cover these two scenarios and present suggested steps that will help you minimize development time and maximize overall system performance.

Conclusion

When building a new test system, the best answer may be a hybrid approach that includes two architectures that, together, can fully satisfy your test requirements. Using more than two architectures can quickly become too complex and will be prone to problems.

The two-architecture hybrid approach is viable whether you seek to minimize development time or maximize overall system performance. In an example of the former scenario, it takes just five steps to convert an all-GPIB system into a LAN-based hybrid system that uses existing GPIB instruments alongside new LXI instruments.

Related Agilent literature

The 1465 series of application notes provides a wealth of information about the creation of test systems, the successful use of LAN, WLAN and USB in those systems, and the optimization and enhancement of RF/microwave test systems.

- *Test-System Development Guide: A Comprehensive Handbook for Test Engineers*
  (pub no. 5989-5367EN)
  http://cp.literature.agilent.com/litweb/pdf/5989-5367EN.pdf

Test System Development

- *Test System Development Guide: Application Notes 1465-1 through 1465-8*
  (pub no. 5989-2178EN)
  http://cp.literature.agilent.com/litweb/pdf/5989-2178EN.pdf
- *Using LAN in Test Systems: The Basics*
  AN 1465-9 (pub no. 5989-1412EN)
  http://cp.literature.agilent.com/litweb/pdf/5989-1412EN.pdf
- *Using LAN in Test Systems: Network Configuration*
  AN 1465-10 (pub no. 5989-1413EN)
  http://cp.literature.agilent.com/litweb/pdf/5989-1413EN.pdf
- *Using LAN in Test Systems: PC Configuration*
  AN 1465-11 (pub no. 5989-1415EN)
  http://cp.literature.agilent.com/litweb/pdf/5989-1415EN.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 SCPI and Direct I/O vs. Drivers*
  AN 1465-13 (pub no. 5989-1414EN)
  http://cp.literature.agilent.com/litweb/pdf/5989-1414EN.pdf
- *Using LAN in Test Systems: Applications*
  AN 1465-14 (pub no. 5989-1416EN)
  http://cp.literature.agilent.com/litweb/pdf/5989-1416EN.pdf
- *Using LAN in Test Systems: Setting Up System I/O*
  AN 1465-15 (pub no. 5989-2409EN)
  http://cp.literature.agilent.com/litweb/pdf/5989-2409EN.pdf
- *Next-Generation Test Systems: Advancing the Vision with LXI*
  AN 1465-16 (pub no. 5989-2802EN)
  http://cp.literature.agilent.com/litweb/pdf/5989-2802EN.pdf

RF and Microwave Test Systems

- *Optimizing the Elements of an RF/Microwave Test System*
  AN 1465-17 (pub no. 5989-3321EN)
  http://cp.literature.agilent.com/litweb/pdf/5989-3321EN.pdf
- *6 Hints for Enhancing Measurement Integrity in RF/Microwave Test Systems*
  AN 1465-18 (pub no. 5989-3322EN)
  http://cp.literature.agilent.com/litweb/pdf/5989-3322EN.pdf
- *Calibrating Signal Paths in RF/Microwave Test Systems*
  AN 1465-19 (pub no. 5989-3323EN)
  http://cp.literature.agilent.com/litweb/pdf/5989-3323EN.pdf

LAN eXtensions for Instrumentation (LXI)

- *LXI: Going Beyond GPIB, PXI and VXI*
  AN 1465-20 (pub no. 5989-4371)
  http://cp.literature.agilent.com/litweb/pdf/5989-4371EN.pdf
- *10 Good Reasons to Switch to LXI*
  AN 1465-21 (pub no. 5989-4372EN)
  http://cp.literature.agilent.com/litweb/pdf/5989-4372EN.pdf
- *Transitioning from GPIB to LXI*
  AN 1465-22 (pub no. 5989-4373EN)
  http://cp.literature.agilent.com/litweb/pdf/5989-4373EN.pdf
- *Creating Hybrid Systems with PXI, VXI and LXI*
  AN 1465-23 (pub no. 5989-4374EN)
  http://cp.literature.agilent.com/litweb/pdf/5989-4374EN.pdf
- *Using Synthetic Instruments in Your Test System*
  AN 1465-24 (pub no. 5989-4375EN)
  http://cp.literature.agilent.com/litweb/pdf/5989-4375EN.pdf
- *Migrating System Software from GPIB to LAN/LXI*
  AN 1465-25 (pub no. 5989-4376EN)
  http://cp.literature.agilent.com/litweb/pdf/5989-4376EN.pdf
- *Modifying a GPIB System to Include LAN/LXI*
  AN 1465-26 (pub no. 5989-6824EN)
  http://cp.literature.agilent.com/litweb/pdf/5989-6824EN.pdf

Using Linux in Your Test Systems

Example code is available for download at **http://www.agilent.com/find/linux.**

- *Using Linux in Your Test Systems: Linux Basics*
  AN 1465-27 (pub no. 5989-6715EN)
  http://cp.literature.agilent.com/litweb/pdf/5989-6715EN.pdf
- *Using Linux to Control LXI Instruments Through VXI-11*
  AN 1465-28 (pub no. 5989-6716EN)
  http://cp.literature.agilent.com/litweb/pdf/5989-6716EN.pdf
- *Using Linux to Control LXI Instruments Through TCP*
  AN 1465-29 (pub no. 5989-6717EN)
  http://cp.literature.agilent.com/litweb/pdf/5989-6717EN.pdf
- *Using Linux to Control USB Instruments*
  AN 1465-30 (pub no. 5989-6718EN)
  http://cp.literature.agilent.com/litweb/pdf/5989-6718EN.pdf
- *Tips for Optimizing Test System Performance in Linux Soft Real-Time Applications*
  AN 1465-31 (pub no. 5989-6719EN)
  http://cp.literature.agilent.com/litweb/pdf/5989-6719EN.pdf

www.agilent.com/find/open

Agilent Email Updates

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

Agilent Direct

www.agilent.com/find/agilentdirect
Quickly choose and use your test equipment solutions with confidence.

Remove all doubt

Our repair and calibration services will get your equipment back to you, performing like new, when promised. You will get full value out of your Agilent equipment throughout its lifetime. Your equipment will be serviced by Agilent-trained technicians using the latest factory calibration procedures, automated repair diagnostics and genuine parts. You will always have the utmost confidence in your measurements.

Agilent offers a wide range of additional expert test and measurement services for your equipment, including initial start-up assistance onsite education and training, as well as design, system integration, and project management.

For more information on repair and calibration services, go to

www.agilent.com/find/removealldoubt

www.agilent.com

For more information on Agilent Technologies' products, applications or services, please contact your local Agilent office. The complete list is available at:
www.agilent.com/find/contactus

Americas	
Canada	877 894 4414
Latin America	305 269 7500
United States	800 829 4444

Asia Pacific	
Australia	1 800 629 485
China	800 810 0189
Hong Kong	800 938 693
India	1 800 112 929
Japan	81 426 56 7832
Korea	080 769 0800
Malaysia	1 800 888 848
Singapore	1 800 375 8100
Taiwan	0800 047 866
Thailand	1 800 226 008

Europe & Middle East	
Austria	0820 87 44 11
Belgium	32 (0) 2 404 93 40
Denmark	45 70 13 15 15
Finland	358 (0) 10 855 2100
France	0825 010 700* *0.125 € fixed network rates
Germany	01805 24 6333* *0.14€/minute
Ireland	1890 924 204
Israel	972 3 9288 504/544
Italy	39 02 92 60 8484
Netherlands	31 (0) 20 547 2111
Spain	34 (91) 631 3300
Sweden	0200-88 22 55
Switzerland (French)	41 (21) 8113811 (Opt 2)
Switzerland (German)	0800 80 53 53 (Opt 1)
United Kingdom	44 (0) 118 9276201

Other European Countries:
www.agilent.com/find/contactus

Revised: October 24, 2007

Agilent Technologies