Maximizing Performance of LXI-based Test Systems

May 10, 2014 Edition

Table of Contents

1 Introduction

LXI Devices (LAN eXtensions for Instrumentation) are LAN-equipped instruments that conform to a comprehensive set of rules detailed in the LXI Standard (www.lxistandard.org). Major Electronic Test Equipment Manufacturers and Technical Consultants formed the LXI Consortium in 2004, and over 40 companies currently participate and maintain that standard. LXI provides the basis by which test systems built from multiple vendors provide a common interface and experience.

LXI brings LAN into the test system and provides a wide range of flexibility to the test system engineer. In particular, LXI Devices benefit from these major LAN features:

- The ubiquitous nature of LAN
- Its high performance data transfers
- Low cost, readily available infrastructure
- Flexibility for wired or wireless communication
- Local and Remote (synchronized) access
- Abundance of multiple protocols for varied functionality
- Ability to embed Web servers within each instrument

The focus of this paper is on understanding how to achieve better test system performance, easier integration, and faster development using LXI Devices. Key assertions of this paper:

- I/O latency has very little impact in most test systems
- Distributed intelligence allows overlap processing that can greatly improve performance
- There are a handful of best practices providing better performance for any test system
- LXI provides additional features to improve test system development and integration.

Test system performance consists of multiple factors, and achieving the best performance requires understanding those factors. You will discover that although the fastest I/O interface provides clear benefits for some applications, the dominating performance factors for test systems include physics - such as signal and power supply settling times, measurement times, interconnection switching times, and synchronization between instruments.

This paper describes the common components of test systems and walks you through the factors just listed. It includes a case study of typical test system performance, and it describes the performance of a cross-section of LXI Devices, as provided through the testing of devices by LXI Consortium members. Best practices for developing any test system include key insights from test engineers. It will wrap up with the some key benefits LXI Devices provide to achieve better test system development and integration.

1.1 Other sources of information

The LXI Consortium has written other documents to help test system developers: ***LXI Primer***, ***LXI Getting Started Guide***, ***Building LXI-based Test Systems***, and ***Introducing LXI to Your Network Administrator***, located on the LXI Consortium Website at GuidesForUsingLXI.

Please refer to these documents for detailed information about LXI, how LXI Devices behave on the LAN, how to connect multiple LXI Devices into a Test System, and the importance of working with your Network Administrator to meet Test System and company LAN requirements.

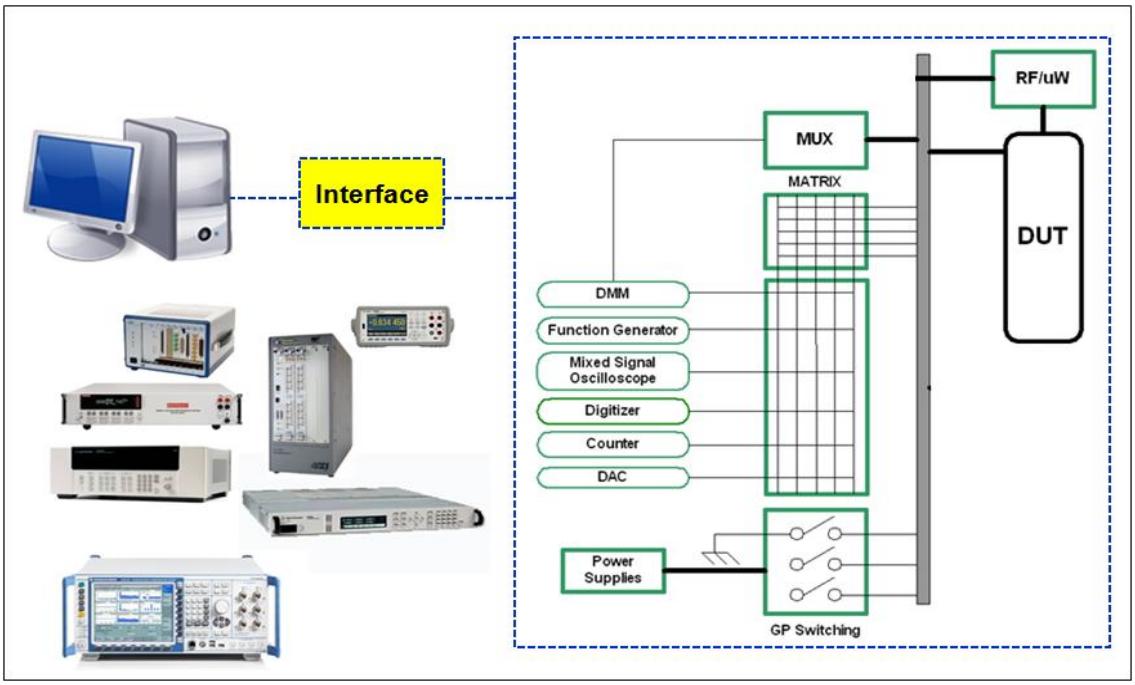
2 Test System Components and Architecture

The architecture of a test system involves the interconnection of computer, measurement, and stimulus devices, and it includes the interconnection of signals between these devices. Test system integration of varied instrumentation requires early planning to achieve desired measurement and speed performance. This section will identify the varied instrumentation solutions often used in test systems, which forms the basis for understanding the performance factors of a test system related to architecture.

2.1 Typical Test System Configuration

A typical test system configuration provides matrix, general purpose, multiplexing, and even RF and microwave switching to connect between the instruments and the Device-Under-Test (DUT). Such a system may use instruments mounted in a rack, each with their own communication interface, display, power supply. The system may also be composed of VXI, PXI, AXIe, or proprietary mainframes, with instruments and switches residing on modules plugged into one or more mainframe slots and a single communication interface to the computer.

It is not practical to put all devices into these modular formats. For example, power supplies and supply loads can be bulky, requiring high power and therefore designed as stand-alone solutions with their own power supplies and communication interfaces such as GPIB, LAN, or USB. In addition, RF and microwave switching can be bulky and often requires specialized mounting in custom cases with drive control provided by other programmable devices.



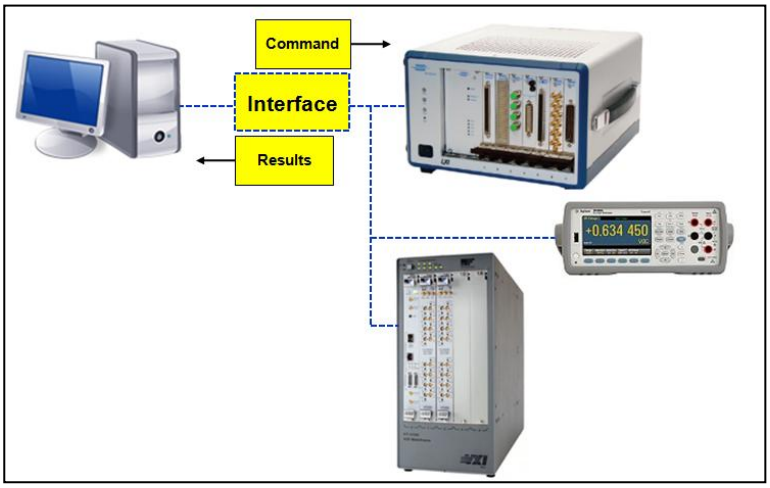
General Purpose Test System

Due to the varying needs for power supplies, loads, instrument displays to show signal measurements, etc., it is very common to have hybrid test systems consisting of mainframe products with modules and stand-alone instruments. This means you are likely to need multiple communication interfaces to control all of these devices. Multiple communication interfaces require careful consideration of synchronizing the control stimulus and response signals. The Section on ***Best Practices for General Test System Development*** covers device synchronization for improving performance.

3 Smart and Simple Devices

A typical test system may contain "Smart" devices, which can perform operations on their own with little interaction from the controlling computer. These could be instrument modules in VXI, PXI, AXIe, or proprietary mainframes, stand-alone instruments, or a controller in a mainframe that accepts commands for all other modules in the mainframe.

LXI Devices, like GPIB, are inherently "Smart" because they can think on their own, without the test system computer controlling every operation. They can do this because they have an embedded controller that controls all of the hardware operations. The three instruments below illustrate various implementations of LXI Devices: PXI and VXI mainframes with embedded controllers, and a stand-alone DMM.



Computer Talks to Embedded Controller

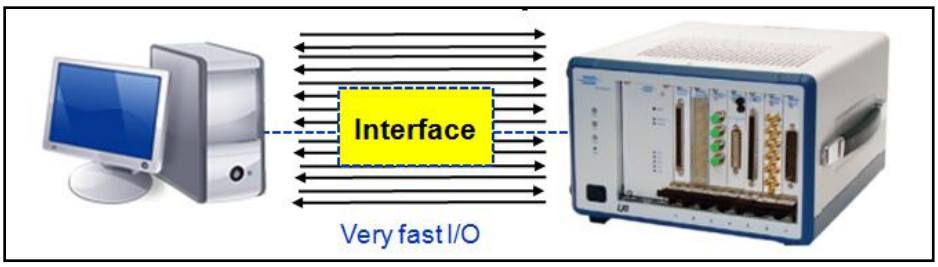
An LXI Device receives an ASCII language command or short sequence of configuration bytes that tells it to perform some operation, which it then performs without any further interaction with the computer. If the command requests results from the device, those results return to the computer in the desired format, which can be ASCII or binary data.

Internally, the embedded controller interacts with the modules or hardware using an embedded driver. In the case of the mainframe products, the VXI, PXI, or AXIe bus is the internal high-speed interface used to control the hardware. For the DMM, it may be something similar or proprietary.

Each LXI Device can serve up a Web page to assist the test system developer in configuring the device, and most of the mainframe versions of LXI actually provide a Web page for manually interacting with and monitoring each module in the mainframe.

The modules in the mainframe products may offer different degrees of intelligence to offload the embedded controller operations, but this is not visible to the test system computer since it communicates only with the embedded controller.

If you remove the embedded controller module from the VXI, PXI, or AXIe mainframe and replace it with a high-speed interface back to the computer, then the computer executes the computer-installed driver that controls the hardware over the high-speed interface. Every register read and write and every block transfer of data involves the computer, as seen the following figure.



Computer Control of Modules Directly

When the computer controls every register read and write of a module over the high-speed interface, it makes a module appear "Simple". "Simple" means the computer must configure the module completely.

Many modules have some sort of intelligence, which can come in the form of embedded processor capabilities in an FPGA (Field Programmable Gate Arrays), or it could be a simple parallel-to-serial-to-parallel circuit controlling multiple switches on a high-density module. In either case, the computer may or may not have to wait for the operation to complete before configuring the next device.

This concept of telling a device what to do and not waiting around for it to complete is a key method of improving system performance. Distributing intelligence among the various instruments or modules allows the test system developer the option of using ***Overlap*** operations. Overlap operations can contribute the greatest performance enhancement in a test system.

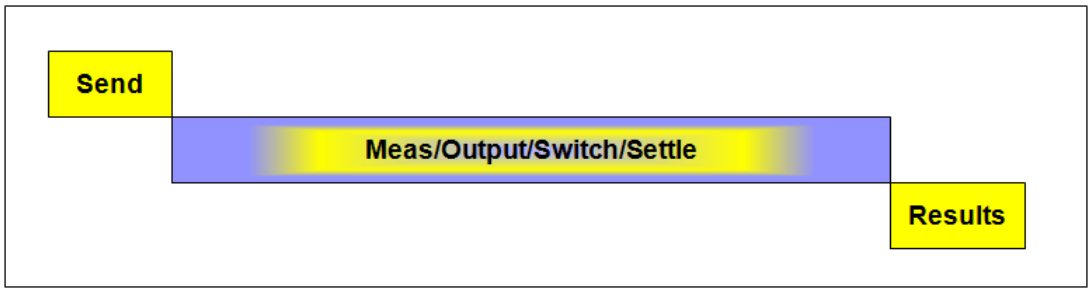
4 Identifying Key Performance Factors

This section establishes the foundation for the Case Study in ***Section 5*** and the performance measurements in ***Section 6***. It identifies the most significant areas affecting test system performance.

A test program consists of sequences of configuring instruments, performing measurements, and returning measurement results. As described in ***Section 2***, a test system usually consists of many instruments and/or modules to perform these operations. The following describes these typical test system operations.

4.1 Transaction Model

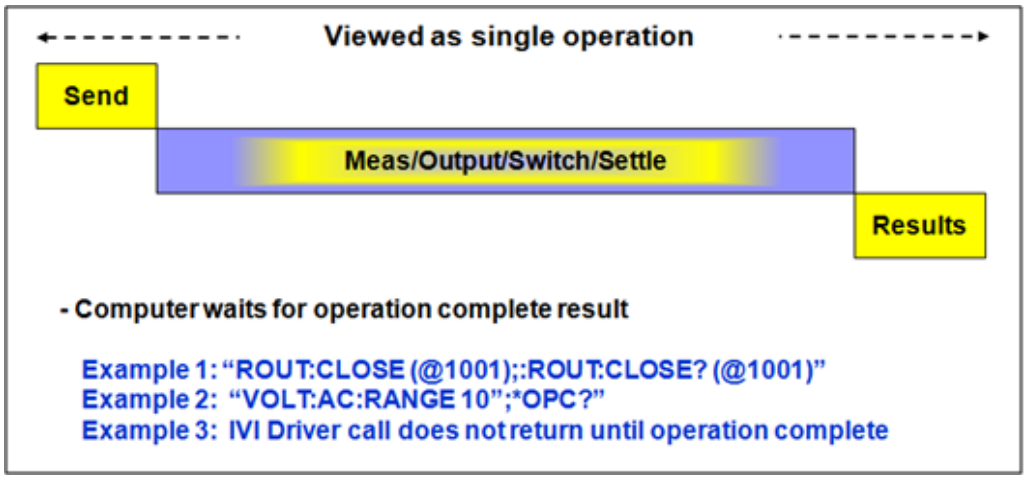
The "**Send – Meas/Output/Switch/Settle – Results"** illustrated in the figure below represents a single transaction between the computer and the device. The **Send** represents sending the command or register operations before the hardware starts to change. The **Results** can be a measurement or simply a status the operation succeeded. Overall, it does represent an interaction with the computer where the computer is waiting for the **Meas/Output/Switch/Settle** to complete before it moves on to the next transaction.



Typical Transaction between Computer and Device

4.2 Serialized Operations

The Transaction Model usually leads to serialized operations in most test systems, due to the driver call, programming method, or by choice of the programmer. The driver call usually performs all three operations before returning, so the overall operation completes before moving onto the next step in the sequence. That is, the driver call operates as a single operation between the computer and the device.



Transaction Model represents Single Operation

The first two examples in the figure illustrate a very common command language called SCPI (Standard Commands for Programmable Instruments). **Example #1** represents a compound SCPI statement, where multiple SCPI statements occur in a single write operation, thus reducing I/O overhead. The "`ROUT:CLOSE`" operation is performed first before the "`ROUT:CLOSE?`" query executes. The computer waits for the return value, which synchronizes the test program before moving on to the next sequence. **Example #2** is similar, but the result this time is the SCPI Operation Complete query. **Example #3** represents a driver call, and most driver calls fully complete their operations before returning to the calling program. A driver might issue ASCII-type commands to an embedded controller, like an LXI Device, or it may be controlling every operation of a module over a high-speed interface such as PCIe.

4.3 Distributed Operation

LXI Devices, like GPIB, have their own embedded controller and can execute the operation while the test system computer is executing some other part of the sequence, as long as the particular step in the sequence does not need the results of that transaction immediately. This means the computer can break up the transactions into parts, permitting it to send command operations and return later to check on the hardware progress with queries. For example, the computer sends the SCPI "`ROUT:CLOSE`" command and similar commands to other instruments. It then returns later with the "`ROUT:CLOSE?`" or "`*OPC?`" to verify the operation completed.

Some modules in VXI, PXI, or AXIe can also perform a similar operation. ***Section 3*** referred to modules with some intelligence that can complete certain operations without the computer involvement. In those cases, the driver call may provide a *no-wait* or *overlap* option, which permits the computer to come back later to check on the completion of the operation using a different type of driver call. These are general terms, and support of such functionality is dependent upon hardware and driver implementation.

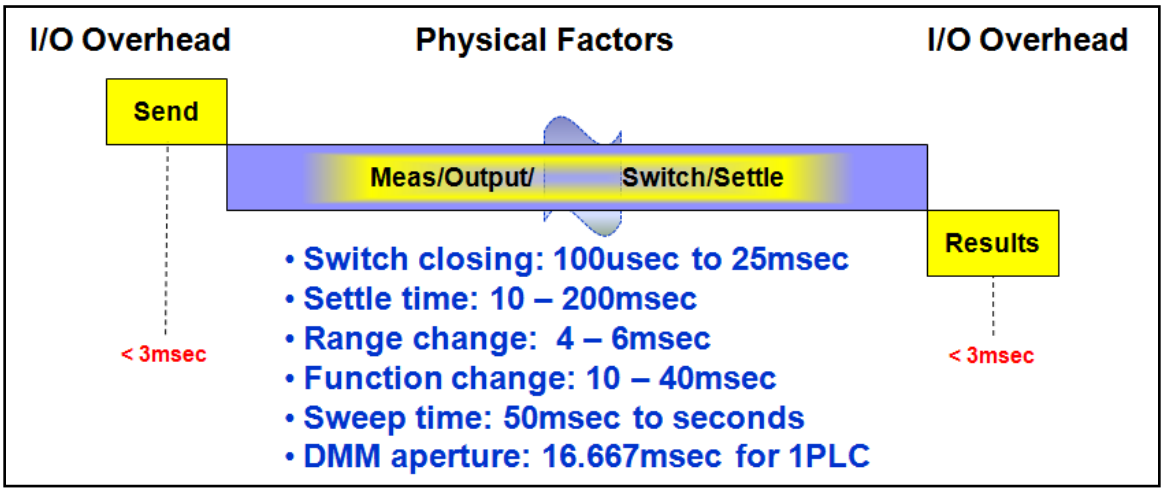
Referred to in this document as an ***Overlap*** operation, hardware control distributes across the computer and devices. This separation of the **Send** from waiting for the hardware to complete and the **Results** or operation complete is a major key to improving test system performance.

4.4 Physics Factors Affecting Performance

In most test systems, the **Meas/Output/Switch/Settle** dominates over the **Send** and **Results**. Input and output signal settling, mechanical switching times, Range changes, Function changes, Measurement apertures, Sweep times, etc. vary widely from microseconds to seconds. This is due to signal frequencies, the presence of noise, power supply settling times, etc.

For example, trying to reject the presence of 60Hz noise requires either heavy external filtering on each input channel – which slows the response time of switching, or you must measure over the period of the noise to average it out. One power line cycle (PLC) is 16.67msec, which would dominate over I/O time.

Solid-state switches provide very fast switching (10 to100usec), but they are limited by applications requiring high resolution/accuracy or operating in the presence of high voltages. RF and general purpose switches may require up to 25msec for mechanical parts to settle. Switching a signal to a DUT may be fast, but a long signal path (usually at least 6 feet in most test systems) into a high impedance load with capacitance (typically 300-800pF) takes time for the output to change from one level to another. See the next figure for typical physical factor effects.



Physical Factors affecting Test System Performance

Notice the wide variation of time spent for the **Meas/Output/Switch/Settle** portion. It is very common to make long measurements to reject noise, measure jitter of a signal, capture the frequency content of a waveform, waiting for large voltage or current swings on supplies to settle, etc.

This figure also introduces a performance number for the **Send** and **Results** operation times. ***Section 6*** illustrates typical LXI Device performance numbers from various LXI Consortium member companies. For the Case Study in the next section, we will assume the slowest LXI Device illustrated in ***Section 6***, which took up to 3msec for a single transaction over LAN.

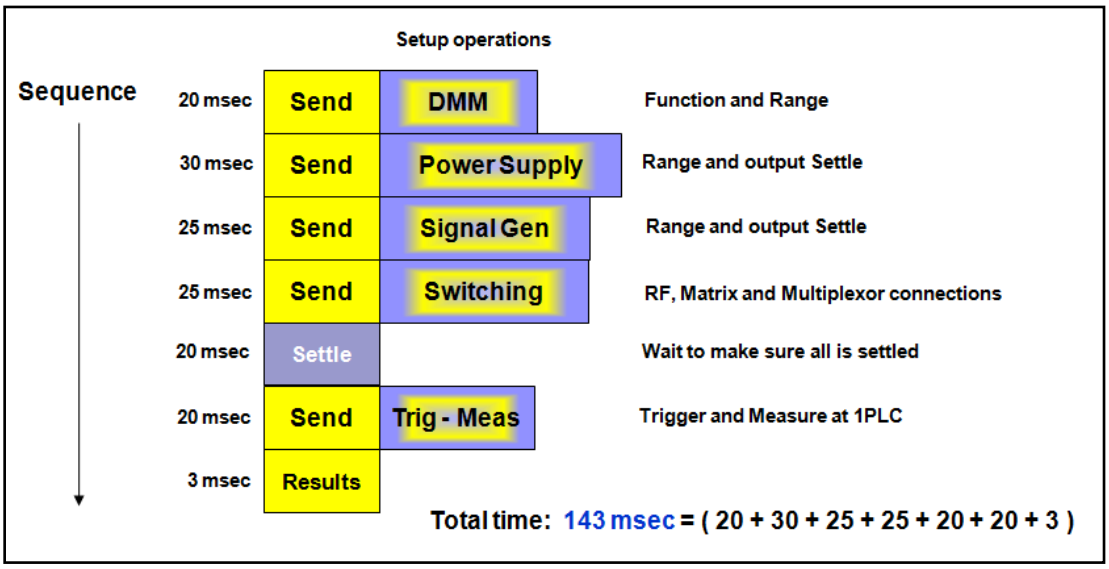
The next section explores the use of these performance factors.

5 Enhancing Performance – A Case Study

LXI Devices permit leveraging their inherent distributed intelligence to enhance performance. First, lets us assume no overlapping of operations takes place, so every transaction completes before the test sequence moves on to the next transaction. Next, we assume each device takes around 3msec per **Send-Results** operation, assuming compound SCPI statements or something equivalent for a similar device. This results in a worst-case scenario for the test sequence, which is actually how most test programs actually operate.

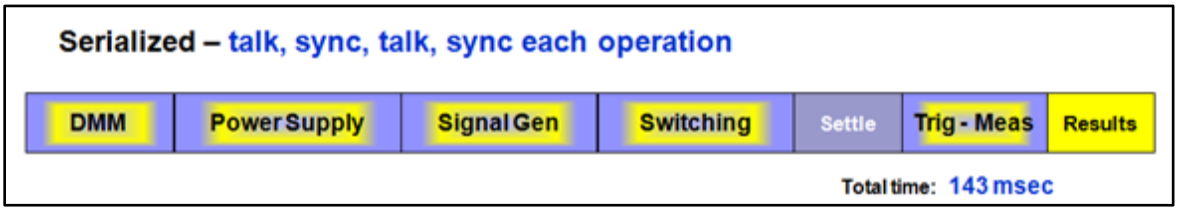
5.1 Baseline Performance

A sub-test sequence involves a DMM, Power Supply, Signal Generator, some switching, and a DMM. The figure below illustrates the sequence of programming. Note the I/O time is included in each operation, and a SCPI compound state or similar operation is used to minimize I/O time.



Serialized Operations

The **143msec** represents the time it should take if you wait for every operation to complete. This is the baseline. You can think of the entire sequence as one operation following the other in time:



Baseline Results

In many cases, the situation described for Serialized Operations is much worse. Many programmers send a **Reset** to each instrument before setting it up for the desired operations. A **Reset** can take many, many milliseconds. It is also very common for developers to use **Wait** statements until arriving at desired results. **Reset** and **Wait** statements have proved to be large contributors to poor test system performance. Synchronizing devices can eliminate the need for **Wait** Statements, as discussed in ***Section 7*** on ***Best Practices for General Test System Development***.

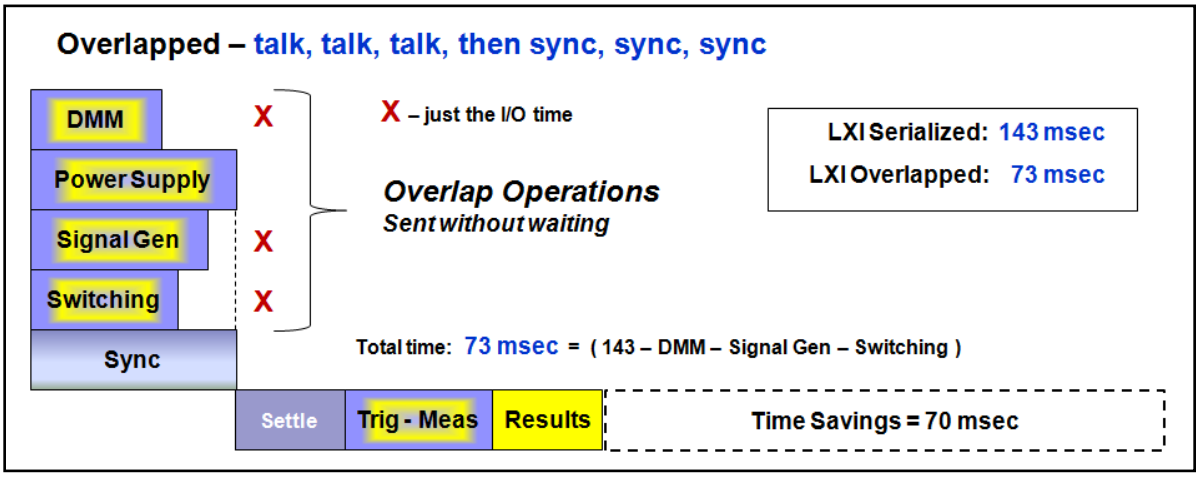
5.2 Using Distributed Intelligence

For all LXI Devices, you can send the setup commands to each instrument as fast as you can (just the I/O time involved), using compound SCPI statements, for example. Then you can return later to synchronize to the longest expected operation. The longest "pole in the tent" dictates the operation to single out. All of the others will complete during this longest operation, and all their setup times then subtract from the overall test sequence time.

In this example, configuring the Power Supply takes the longest time. The configuration time for the DMM, Signal Generator, and switching is masked. You need only understand which instrument takes the longest to configure, and make that the limiting factor. In some cases, you may be wrong, so it is wise to synchronize the remaining instruments after testing the longest to make sure they all complete. In each case, these will be very fast operations and would add only another 9msec (3 x 3msec), assuming our worst-case I/O performance number.

Dependent upon the test application, you may have to settle all of the Switching first before applying signals, but this only forces you to serialize the switching before certain operations. Others can still be overlapped.

The figure below illustrates the results of this overlapping sequence. Note overlapping of operations saves 70msec of time, an almost 50% reduction.

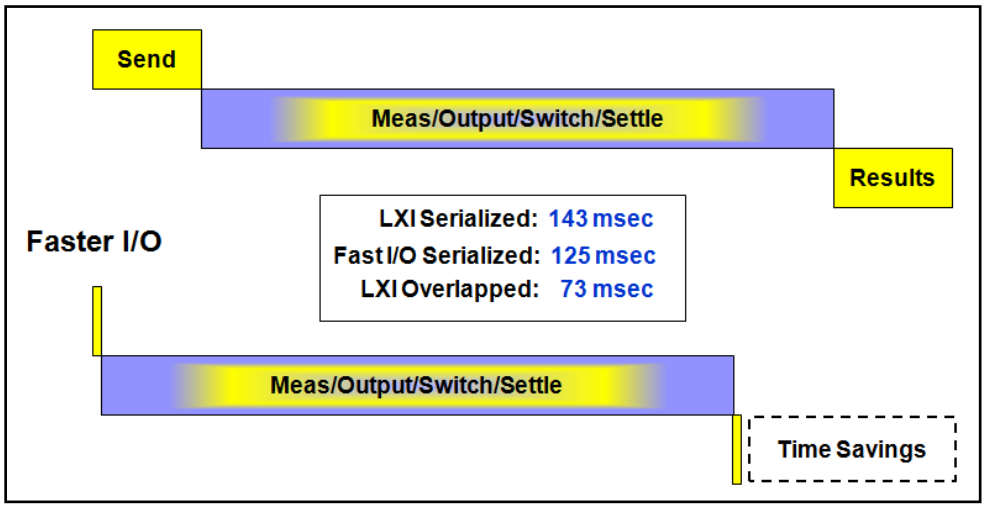


Using Overlapped Operations

5.3 Using Faster I/O

Returning to the "**Send – Meas/Output/Switch/Settle – Results"** Transaction Model, the **Send** and **Results** blocks represent the I/O time between the computer and the device. In the Case Study, the baseline calculations assume LXI Devices with 3msec of I/O overhead due to the LAN driver overhead (on both the computer and device), the speed of LAN, and the parsing of the commands or queries sent to each device.

If you replace the **Send** and **Results** time with a much faster interface, such as PCIe, and assume the device programming operates in 10's of microseconds, the **Send** and **Results** essentially become zero, and you are only waiting for the hardware to complete the operation. The computer may or may not be involved with various aspects of the **Meas/Output/Switch/Settle** portion. For analysis purposes, we can ignore that aspect for now and only focus on the **Send** and **Results**.



Using Faster I/O

This particular figure visually indicates significant savings in time. When compared to a **Meas/Output/Switch/Settle** time of 10msec, the removal of 3-6msec of time would be huge. However, the subtraction of 3msec from each operation in the baseline calculation of the case study results in an overall savings of only 18msec (6 x 3msec) out of 143msec, a savings of only 12.5%.

Section 6 illustrates that most LXI Devices operate much faster over LAN than the conservative 3msec used in this case study, so the savings of time using Faster I/O becomes even less significant.

Faster I/O vs. Overlap Operations vs. Measurement Physics

It appears that in many, if not most cases, I/O time, often referred to as I/O Latency, matters little compared to the typical **Meas/Output/Switch/Settle** times. Another conclusion is that overlapping of instrument configurations provides the greatest opportunity for performance enhancement.

6 Example Performance Numbers for LXI Devices

This section gives a cross-section summary of LXI Device I/O performance. There are currently over 2000 LXI devices, as of 2014. LXI devices address a wide range of measurement and control, from low-cost Digital Multimeters to expensive Spectrum Analyzers and Oscilloscopes. These latter devices often have powerful embedded computers, on par with or faster than desktop computers.

The LXI Consortium began its baseline testing by choosing a fast desktop computer, on par with the embedded computer found in a high-end instrument like a spectrum analyzer. Using the computer to emulate an LXI Device without measurement hardware, consortium members measured the performance of simple queries and block transfers over a 1Gbit LAN interface to another fast desktop computer. This would provide a reference point to measure performance of actual LXI devices.

Desktop Computer Emulating LXI Device, Gigabit LAN:

Simple Query:	~250usec
Block Transfer:	> 100Mbyte/second

In both of these cases, the measurements resulted from using the LAN protocol TCP/IP Sockets, as implemented in the HiSLIP protocol. The block transfer approached the LAN Gigabit theoretical limit when transferring 1 million bytes.

Actual LXI Devices are likely to be somewhat slower since their embedded processors are handling displays, measurement firmware and hardware, etc. in the background while being queried.

Before getting into LXI Device performance, it is important to realize there are three protocols for communicating with LXI Devices over LAN, of which LXI devices support one, two, or all three.

6.1 Multiple LAN Protocols for LXI Devices

Many LXI Devices include LAN, GPIB, and USB interfaces. GPIB has been the standard for instrumentation control for decades, and a great deal of existing test system software controls GPIB devices. With both LXI and GPIB devices representing stand-alone instruments, one of LXI's driving goals is to replace the expensive and slow GPIB interface in test systems. In order to make migration of existing software written for GPIB instrumentation to LAN, new communication protocols emerged.

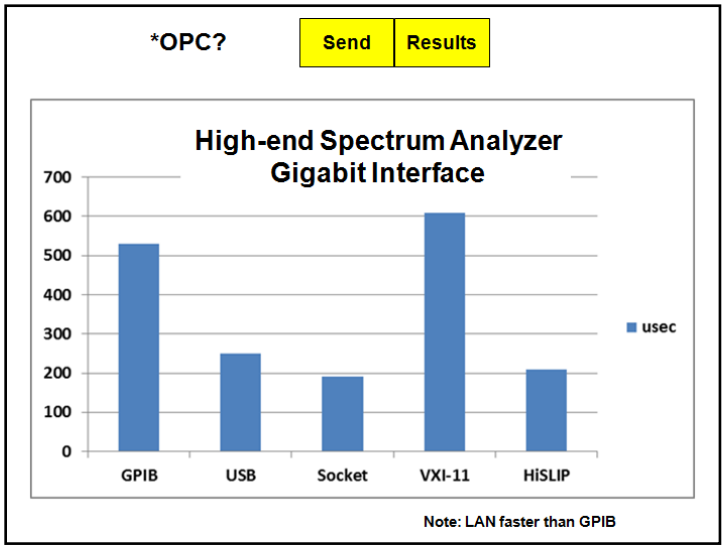
The initial protocol, named VXI-11, provides equivalent GPIB operations like Device Clear, Service Request, Group Execute Trigger, etc. VXI-11 has been in use for years, but its implementation using the Remote Procedure Call (RPC) protocol requires additional overhead in communication that affects performance. The IVI Foundation developed HiSLIP (High Speed LAN Instrument Protocol), based upon TCP/IP Sockets, to achieve higher performance. Both protocols benefit from the high-speed block transfer achievable with LAN, many times the capability of GPIB.

The third protocol is raw TCP/IP Sockets, which is closer in function to a serial interface operation like RS-232. That is, you can transfer high speed ASCII or binary to and from the device, but the connection does not have the equivalent GPIB operations like Device Clear, Service Request, etc., present in VXI-11 and HiSLIP. Refer to ***Appendix A*** for more information on interaction between the computer and LXI Devices.

VISA I/O Libraries (Virtual Instrument Software Architecture) provide access to the VXI-11, HiSLIP, and raw TCP/IP Socket protocols. VISA controls instruments in virtually all test systems. The introduction of VXI-11 and HiSLIP protocols for LAN devices essentially makes migrating test system software from GPIB to LAN as easy as changing the VISA address from using "`GPIB0::22::INSTR`" to "`TCPIP::156.140.95.14::inst0::INSTR`". Support for test system software migration from GPIB to LAN continues to be a driving force and goal for the LXI Consortium and IVI Foundation. .

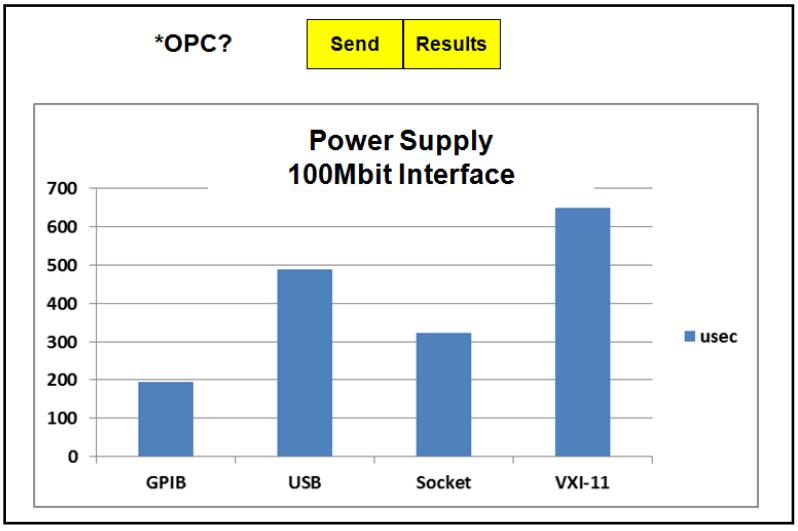
6.2 Transactional Performance

The next three figures are examples of LXI Device performance from cooperating members of the LXI Consortium. Tests were performed on a number of LXI devices all using the same performance test tool. In these cases, the SCPI "`*OPC?`" query was used for benchmarking. The I/O time only includes the time to send the "`*OPC?`", parse the "`*OPC?`", and return the results. No hardware operations are involved in the query, so these benchmarks are measuring just the **Send-Results** portions of the Transactional Model.

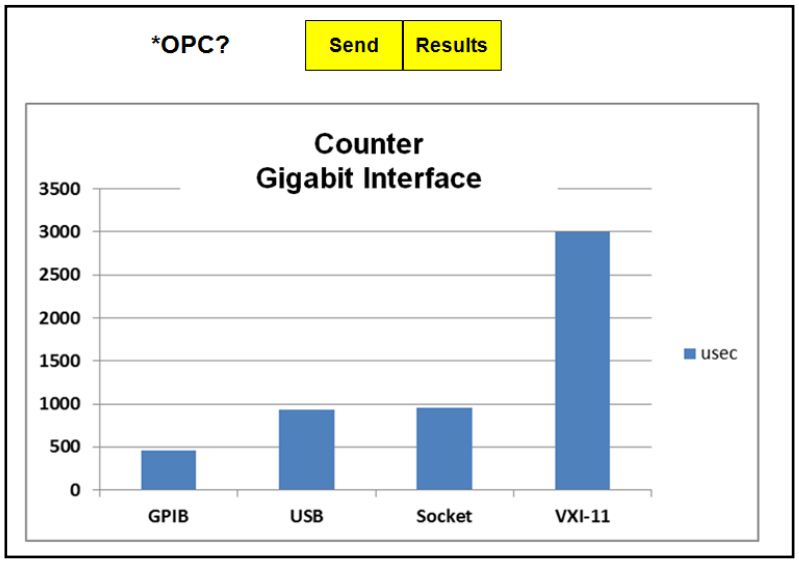


As you can see from the data results for this Spectrum Analyzer, LAN TCP/IP Sockets are actually faster than its GPIB implementation. Notice that rather than 3msec for the **Send-Results**, the time is about 200usec for this device. VXI-11 is slower, but still only 600usec.

The next figure illustrates a much less expensive power supply with a slower processor and only a 100Mbit LAN interface. As you can see, the I/O performance is close to the Spectrum Analyzer. The engineers of this product spent a lot of time optimizing their LAN driver to achieve results that even show comparable VXI-11 performance with the Spectrum Analyzer. This particular product shipped before the HiSLIP protocol became available. If implemented, the HiSLIP performance should be comparable to the TCP/IP Socket results.



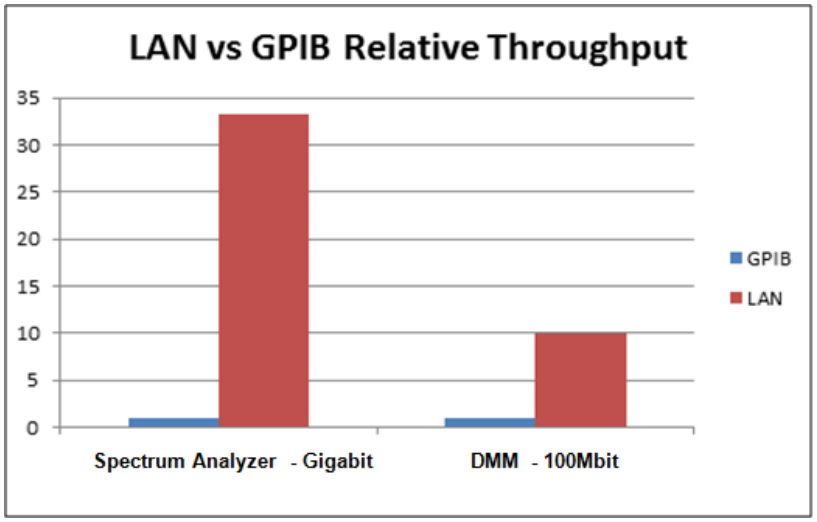
The following is a product with a slower implementation of VXI-11 but one with TCP/IP Socket speed still below 1msec. Note this product also shipped before the HiSLIP implementation was available. This performance benchmark is what drove the 3msec time of the **Send-Results** used in the Case Study**.**



6.3 LAN Block Transfers

Since one of LXI's goals is to provide a better migration path for systems using GPIB devices, LXI Device block transfers illustrate the huge difference and benefit of LAN over GPIB. The following figure illustrates a relatively low-cost DMM using a 100Mbit interface and a Spectrum Analyzer with a Gigabit interface, where results are in megabytes/second. For the DMM, its GPIB performance is not capable of reaching its own 50K reading/second rate. LAN easily handles a continuous flow 50K readings/second with the LAN interface.

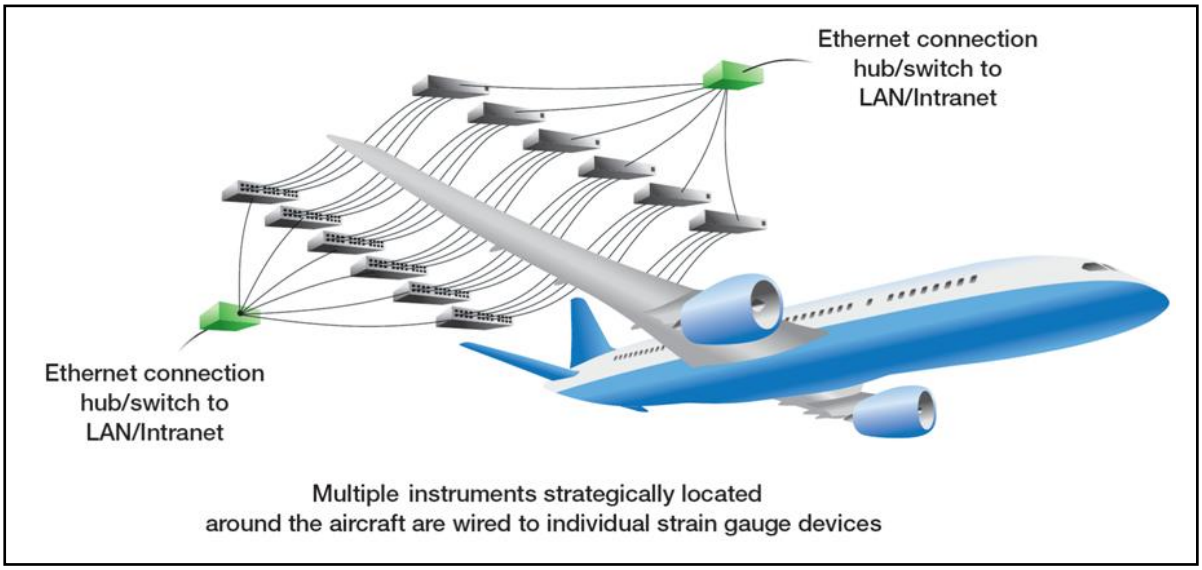
Performance varies considerably between LXI devices because a product usually only optimizes its LAN performance to meet the demands of its specifications. As in the case of the DMM, its embedded processor can easily handle LAN speed of 10 million bytes per second and is more than adequate to maintain 50K readings/sec or to fetch those readings from memory at even greater speed. There was no need to push its architecture or add cost to achieve greater performance.



Block Transfers in Data Acquisition

Data Acquisition test systems collect data – lots of it, and they need to collect it fast. Often that data has to travel relatively long distances back to the collection computer. LAN is the ideal medium for such cases.

The application below involves testing an airframe under stress. The test system computer is in an observation/control room and connected to all LXI data acquisition devices with a single Gigabit LAN cable via commercial grade LAN switches. The test system computer configures each LXI Device to make strain, temperature, etc. measurements, which they each perform on their own after initiated. The computer initiates the LXI devices and begins mechanical flexing of the wings. The computer then begins streaming data from the LXI devices over the Gigabit LAN.



Air Frame Stress Testing

7 Best Practices for General Test System Development

The following are other factors you should consider when developing test programs: device-to-device synchronization, understanding the triggering model of instruments, and generally avoiding the use of **Reset** and **Wait** Statements.

In addition, some LXI Devices provide additional features you may not be aware of – such as LXI LAN Event Messaging and the LXI Clock Synchronization, using the IEEE 1588 Timing Protocol, covered in ***Section 8***, ***Exploring Other Benefits of LXI***.

Consult the vendor documentation for your specific products to understand how the following insights apply in your test system application.

7.1 Device Synchronization

The overlap of device setup in the Case Study of the previous section used device synchronization. There are techniques available for both "Smart" and "Simple" devices. For the device controlled directly by the computer over an interface like PCIe, the driver would have to support a *no-wait* type option for the following operations to work. LXI devices typically provide all of the following synchronize operations.

Polling of status

This is the most common method used to test if a device is ready to proceed. Its status subsystem typically has multiple status bits to describe the existing status of the device: operation complete, device triggered, waiting for trigger, idle, sweeping, etc. You can poll the device or multiple devices in a loop until they are ready for the next operation.

Service Request

GPIB devices pioneered this capability of the device to send a service request byte across the interface to signal any of the preceding bit conditions. LXI Devices that support the VXI-11 or HiSLIP protocols automatically inherit this capability.

Wait for Operation Complete

As noted previously, the SCPI commands "`ROUT:CLOS? (@1101)`" and "`*OPC?`" provide a method to specifically wait on a query to determine when this particular operations completes.

Wait Statements

This is more common than it should be. A typical developer will program devices and make measurements, finding the results reflect devices are not ready. He may empirically determine the amount of time to wait for all signals to settle, or he may just put in a "long enough" **Wait** statement to achieve the desired results. If it works and is fast enough, the engineer can move on to the next test sequence.

The problem with this is indeterminism. The programmer often wastes time in most cases, and could possibly introduce race conditions that change with signal frequencies, applied currents, impedances, etc. What works today may not work reliably tomorrow, especially if devices change, a redesigned DUT, temperatures change, etc.

Use **Wait** statements only when you cannot actually query the device you are waiting for. When waiting for physical elements to settle, you could connect a voltmeter to the particular signal and make a fast, low-resolution measurement to determine when the signal has settled.

7.2 Device Triggering and Waiting for Trigger

Some of the same techniques used in device synchronization apply when triggering devices. Triggering is a type of synchronization because you want the measurement to occur at the right time, when all devices have settled and are ready to make measurements.

Auto-triggering devices operate on their own internal metrics and do not necessarily synchronize well with the input signal changes. Auto-triggering works well if you have another lever to adjust, such as a signal level. Oscilloscopes, for example, provide this. Triggering at zero volts may be problematic for noisy signals, but triggering the same signal at 100mV might provide a very good result.

Although it seems like a waste of time, it is sometimes better to perform an initial measurement on an unknown repetitive signal in order to trigger it properly on the second pass. For example, suppose you are measuring the transient response of a power supply by cycling between resistive loads. The first pass measurement of the repetitive signal might be to capture the highest peak. The trigger level is then set to 80% of that peak in order to avoid triggering on indeterminate smaller noise spikes.

Most measurement devices have external trigger inputs, and some have trigger outputs. Switch devices usually have a Channel Advance input signal and a Channel Closed output signal. A DMM and Switch can actually trigger each other without intervention from the test computer. If you tie the DMM's Trigger Input to the Switch's Channel Closed, and its Trigger Output to the Switch's Channel Advance, the computer need only close the first channel in a list of channels, and the DMM-Switch will trigger each other to make measurements on every channel until the list ends. As long as you set up the list of channels for the same DMM function, you can eliminate all I/O overhead associated with triggering. The Switch or the DMM can also control the reading rate with trigger or channel settle delays.

A number of applications, articles, and papers on LXI Device Triggering and Synchronization reside on the LXI Consortium Website at: http://www.lxistandard.org/Resources/Default.aspx

Other recommended papers on LXI Triggering and Synchronization:

http://www.wheelwrightenterprises.com/LxiSyncTrigger.pdf

7.3 Setting up Devices

It is very common to see **Reset** statements prevalent in test programs. Some devices are very complex, and it is often easier to reset the device than to make an incremental change to its state. This is especially true for Oscilloscopes, Spectrum and Network Analyzers. It makes sense to use a **Reset** initially at the beginning of a test sequence involving many tests, but you should try to avoid a **Reset** before every test.

Changing functions, ranges, and other parameters provides a wealth of opportunity for performance enhancements. Many devices have macro and micro changes for adjusting for output and input operations. For example, a Digital Multimeter may have a macro driver call or command that completely sets up the function. If the device is already on that function and you simply want to make a larger signal measurement, it is better to change the range rather than completely re-configure the device. Changing range is a micro operation, whereas changing the function is a macro operation, involving default settings, ranges, etc.

Autoscale on the bench is an easy way for interactive use of the device. Autoscale can take a lot of time, and it is inefficient when programming. A test program should know what to expect from the device and set up the measurement devices accordingly. Another similar operation is Autorange. In the case of a voltmeter or counter, Autorange picks the range to maximize the signal for measurement. Autorange takes

time because it is making input signal measurements, so it is better to pick the range, since you know what to expect from the device.

Noise rejection is another area that can slow down measurements in many devices. Noise rejecting for an Oscilloscope can produce a large amount of digital processing by the device, and subsequent slower measurements. If possible, provide filtering of the signal at the DUT. A Digital Multimeter has similar capabilities with changing its aperture window to reject noise. Use only as much noise-rejection as you need. For example, the default (or **Reset**) value might be 10PLC (Power Line Cycles), which is 166.67msec. A value of 1 PLC, or 16.67msec may be perfectly adequate and results in a 10x improvement in measurement speed.

7.4 Combining SCPI Statements into Compound Statements

Section 4 introduced the concept of the SCPI compound statement, where multiple SCPI commands combine into a single I/O statement to improve I/O performance. The SCPI-1999 Standard (found on the IVI Foundation Website - http://www.ivifoundation.org/specifications/default.aspx) supports commands and queries in compound statements. All you need do is properly terminate each command or query with a semicolon (;) and to make sure the syntax tree follows the SCPI parser rules.

Here are three examples of compound statements

- Command Only: "`VOLT:AC:RANGE 10;:TRIG:COUNT 10;:INIT`"
- Command/Query: "`VOLT:AC:RANGE 10;*OPC?`"
- Query Only: "`VOLT:AC:RANGE?;:TRIG:COUNT?`"

The first line returns no results. The Second line is the method described in ***Section 4*** where you can synchronize the Operation Complete after performing the operation(s). The little-known Third method often eludes programmers, where multiple comma-separated results return for each query:

```
Results: 10,1
```

Notice the use of the colon (:) before the next SCPI statement in the sequence. This forces the SCPI parser to restart at the beginning of the SCPI language tree. This colon is not necessary when the next command is a 488.2 Common command, preceded by the star (*) character.

The VISA I/O Library has query functions to read a list of returned items. Modern languages, such as C#, provide easy parsing of lists into Real, Integer, or String variable arrays. The benefit of using compound statements is particularly useful for LAN, since a single output statement can contain 100's of bytes and traverses the LAN connection at millions of bytes per second.

8 Exploring Other Benefits of LXI

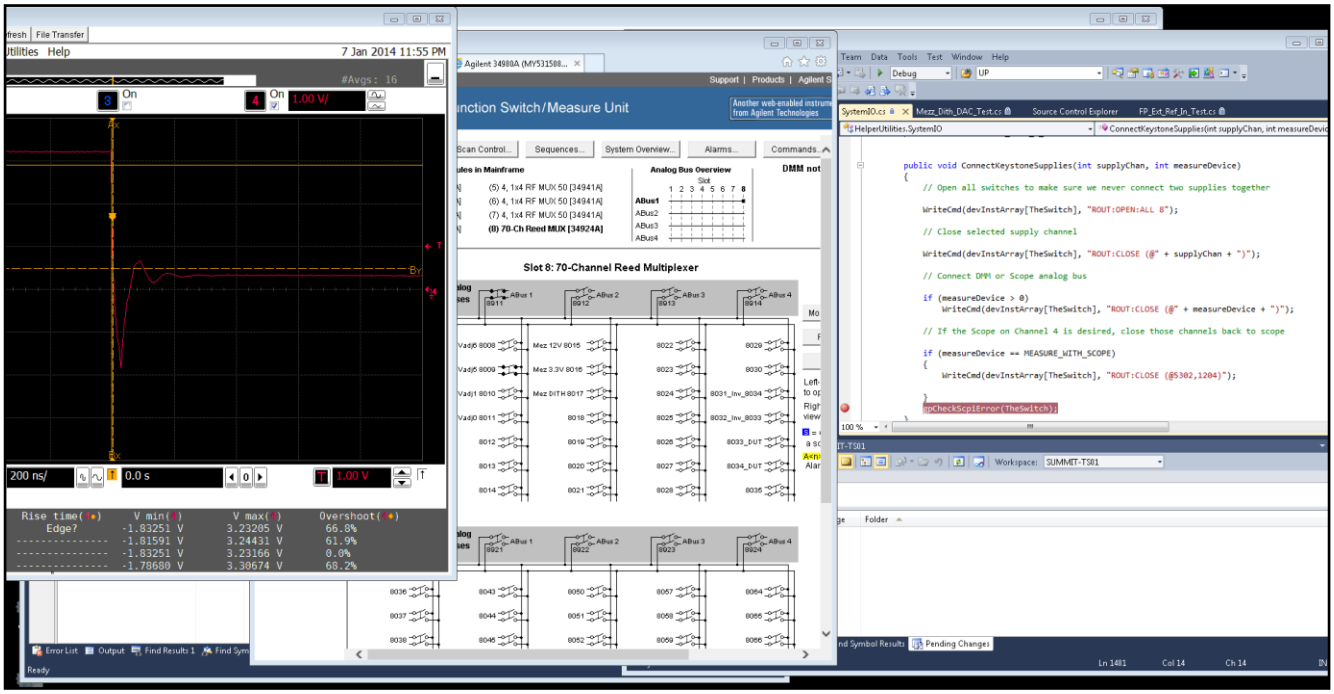
LXI devices have a baseline or core set of features that all must support to be conformant to the standard. One important feature is the built-in Web Server. Most LXI devices support Web pages for a "window" into the state of the device, both for monitor and control. LXI also supports Extended Functions in LXI, which are optional for devices. This section will summarize some of these key benefits.

8.1 Built-in Web Server

Test system developers who have used LXI devices rave about the Web pages provided by these devices. The Web page gives the programmer a window into the state of the instrument, or it may be a virtual front panel for the instrument. Either way, it is very useful to "Pause" program execution and then verify your test code is configuring instruments properly through its Web page. Here is an example from an engineer:

- Test system is almost 100 yards from my desk
- It is located in a climate-controlled Test Lab
- It is noisy and somewhat chilly in the Test Lab
- The test station setup is not suited for test code development

Unless there is a need to move cables around or to install another DUT for testing, virtually all test development takes place in the Test Engineer's cubicle. The following figure illustrates a typical screen shot of this engineer's computer:



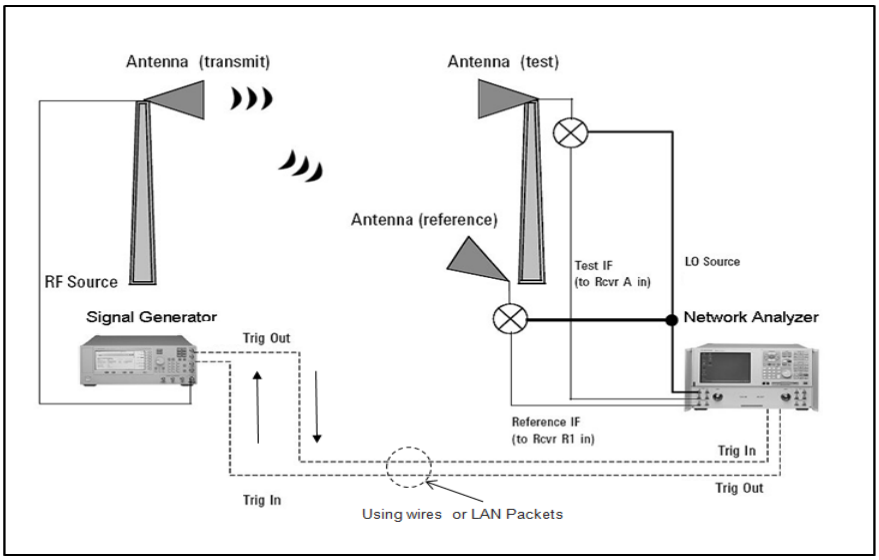
Example Screen Shot Using LXI Device Web Pages

The Test Engineer has a Remote-Login to the test system computer. The above figure shows using Visual Studio with a breakpoint set in the test program. There are two LXI Device Web pages open – one showing the current screen of the Oscilloscope, and the other shows switch positions to verify proper connection to the DUT. The Test Engineer estimates this saves him days of development time, because he does not have to include queries in his test program to verify the configuration of the test system, and he does not have to work in the uncomfortable environment of the Test Lab.

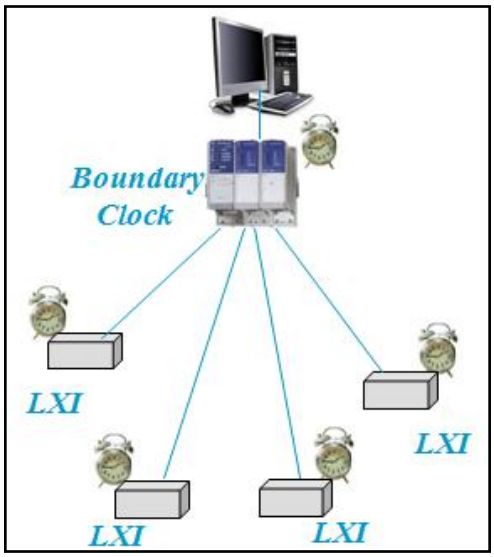
8.2 LXI LAN Events

Some LXI Devices that support LXI Messaging and Events can send out a LAN packet indicating it has transitioned to some state. The LAN packet can be specifically addressing another LXI Device or the computer, or it could be a broadcast packet for multiple LXI devices.

LXI Devices supporting LXI Events and Messaging can signal each other over LAN rather than using hardware signals. Far-Field Antenna testing often requires a long distance between the Signal Generator and Spectrum or Network Analyzer, sometimes as far as line-of-sight permits. Some "Smart" instruments can sequence through a downloaded list of frequencies to test the antenna and only need trigger each other when sequencing through that list. When the two instruments are very far apart, connecting wires between the Trigger In and Trigger Out is difficult. However, LXI Device Signal Generators and Spectrum or Network Analyzers may support LAN packet triggering (LXI Event Messaging), so you need only provide a site-to-site LAN connection, which could be a virtual connection through the Internet.



Far-Field Antenna Testing



LXI Clock Synchronization

8.3 LXI Clock Synchronization

The LXI Specification adopted the IEEE 1588 protocol to synchronize LXI Devices to a stable system clock over LAN, rather than using trigger cables. Using this technique, devices can provide stimulus and triggering based upon the time of day.

Not all LXI Devices provide this useful capability, but those who do provide the test developer with the ability to configure measurements on time instead of command. Device synchronization at the microsecond level or less is achievable. LAN affords distances between devices up to 100 meters on a single Switch or Router, and the IEEE 1588 algorithms adapt to devices at different distances from the Master.

This finds many uses in remote applications where devices are synchronizing measurements in various locations. IEEE 1588 Masters or Boundary Clocks connect to a subnet to keep LXI devices synchronized, and these Boundary Clocks typically connect to a worldwide standard.

In addition, measurements taken by various devices are locally time-stamped by LXI devices for post-measurement analysis, thus providing a means to determine event occurrences between locations.

9 Conclusions

This paper walked through the various performance factors in test system development and identified key areas where LXI devices can provide clear benefits due to their inherent distributed intelligence and basic LXI features, such as built-in Web pages. Let us take each of the assertions made in the Introduction and assign key areas of learning derived from the paper.

- I/O latency has very little impact in most test systems - Measurement technology and physics drive the performance in most test systems.
- Distributed intelligence allows overlap processing that can greatly improve performance – LXI devices operating in parallel with each other during configuration saves the most time.
- There are a handful of best practices providing better performance for any test system - proper device synchronization between DUT measurements and instruments reduces wasted time.
- LXI provides additional functions to improve test system development and integration – LXI Web pages give you a "window" into the state of the device during program development. LXI Extended Functions provide remote control capability of devices through the infrastructure of LAN and other standards such as IEEE 1588.

The LXI Consortium has written other documents to help test system developers: ***LXI Primer***, ***LXI Getting Started Guide***, ***Building LXI-based Test Systems***, and ***Introducing LXI to Your Network Administrator***, located on the LXI Consortium Website at GuidesForUsingLXI.

Please refer to these documents for detailed information about LXI, how LXI Devices behave on the LAN, how to connect multiple LXI Devices into a test system, and the importance of working with your Network Administrator to meet Test System and company LAN requirements.

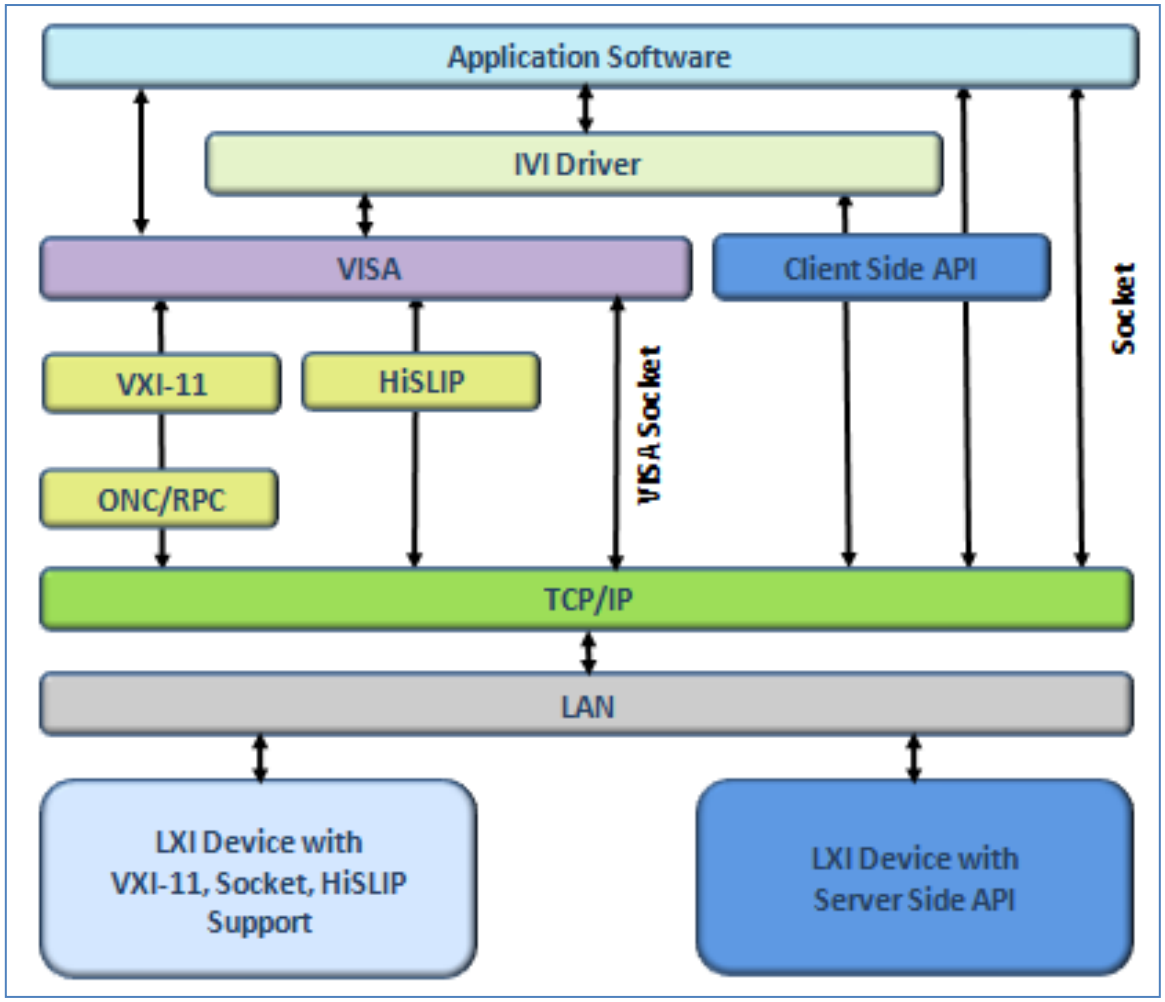
Other links to applications, videos, articles, and papers you may find useful:

- IVI Driver and SCPI-1999 Specifications – http://www.ivifoundation.org/specifications/default.aspx
- LXI Consortium Resources - http://www.lxistandard.org/Resources/Default.aspx
- LXI Synchronization - http://www.wheelwrightenterprises.com/LxiSyncTrigger.pdf
- LXI – An Integrator's View - http://ieeexplore.ieee.org/Xplore/home.jsp

Appendix A – Communication Protocols

The following figure represents the various software layers for virtually all communication with LXI Devices. This illustration gives insight into performance, since traversing more software layers likely slows communication. However, with modern computers, the overhead of traversing software layers is not significant. In the case of VXI-11, there is an additional protocol over LAN that results in twice as many packet interactions. Since I/O time is almost always much less than the time related to the physics involved in switching, measuring, and settling of signals, using any of the protocols illustrated is practical. Pick the one that is easiest to use and provides the best overall function, and you can focus on your test program rather than the interface.

The Client Side API talking to the LXI Server Side is typically a proprietary interaction between the computer and the LXI Device. Whereas interfaces like VXI-11 and HiSLIP typically transfer ASCII-type commands, the Client-Server software can be virtually any protocol and data interaction riding on top of TCP/IP.



LXI Device Communication protocols