Test-System Development Guide

Maximizing System Throughput and Optimizing System Deployment

Application Note 1465-7

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

Agilent Technologies

Introduction

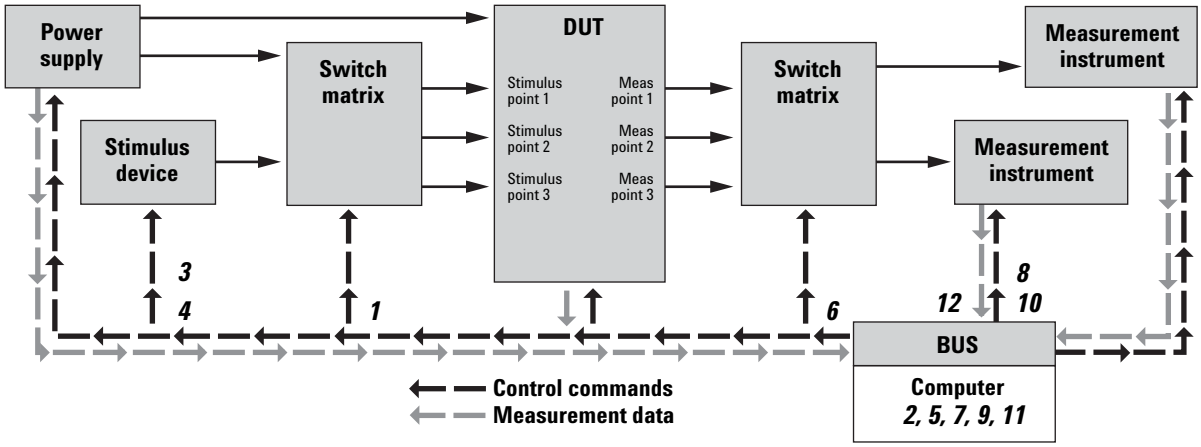
Throughput is a measure of the time it takes to test a device or product. Maximizing throughput is most critical in high-volume manufacturing, where you have thousands of products to test, and you want to test them as fast as possible. In high-volume manufacturing, you measure throughput in terms of 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:

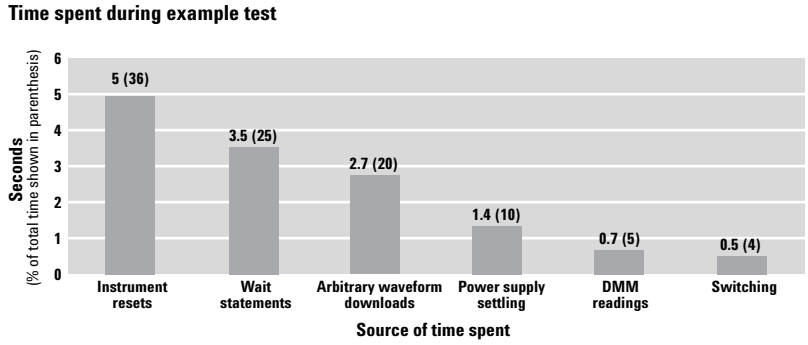
- Select instruments from vendors you trust and choose instruments with high mean time between failures (MTBF) specifications.
- Establish a good spares program: keep backup components for your system, so if an instrument fails, you can quickly swap in a replacement and restore system functionality.
- 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).

Figure 2. Hierarchy of delays in a typical test plan



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. Also, inside the digitizer the readings are analyzed by a processor that is much slower than the one in the PC, so you need to factor in this added time as well.

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 just 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.

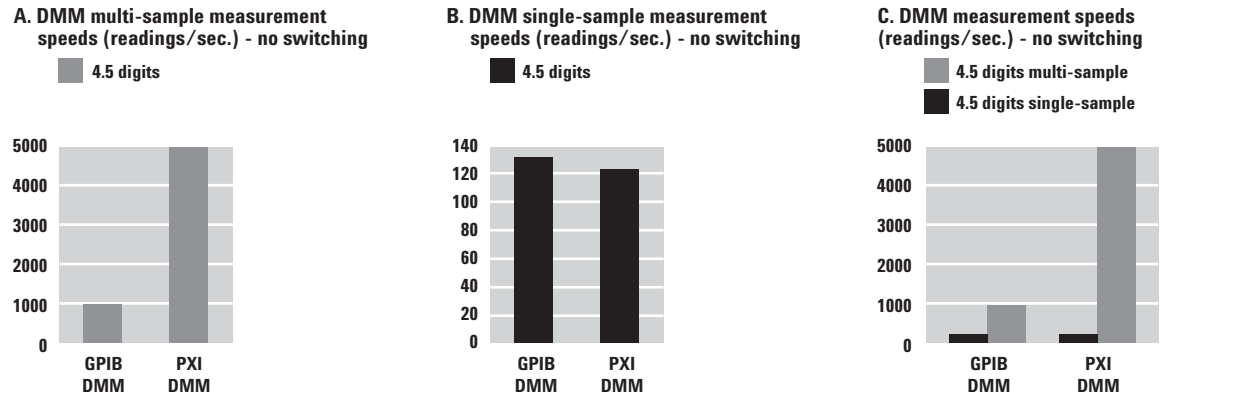


Figure 4. Testing a four-output DC-to-DC converter with a single DMM requires a complex multiplexing scheme and can involve significant delays.

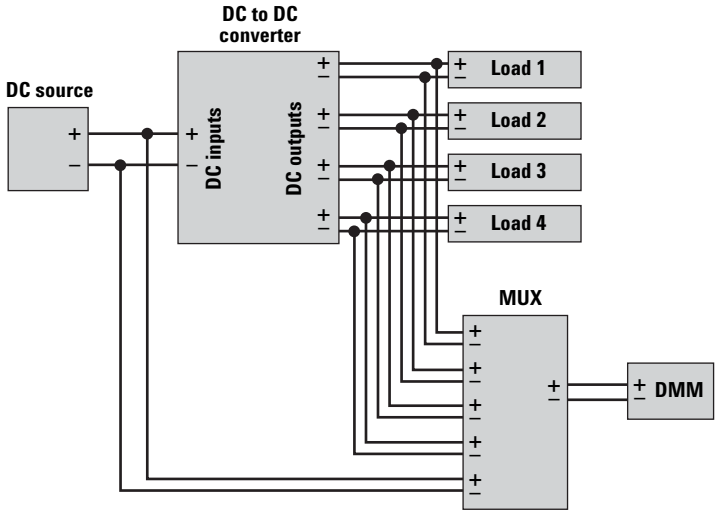
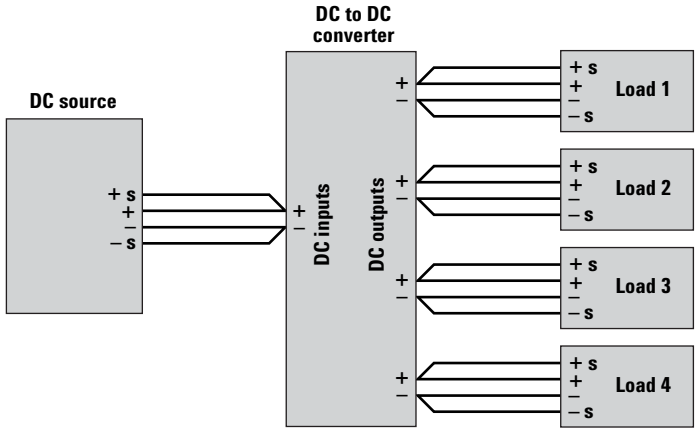


Figure 5. By using the built-in measurements in your DC power source and electronic loads, you can eliminate the DMM and MUX and significantly increase your test speed.



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.

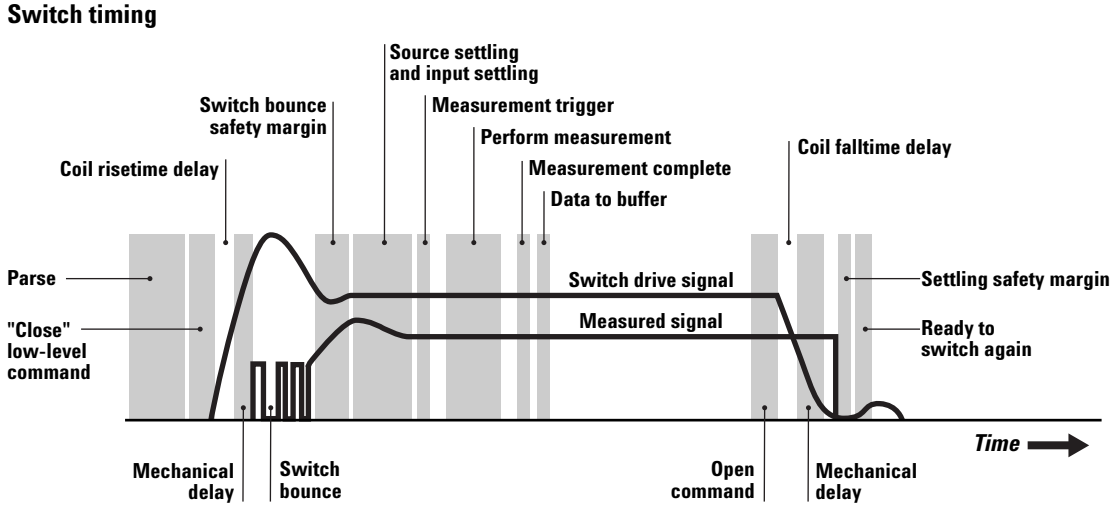
In time-critical tests, it's a good idea to watch out for downprogramming delays. Because programming up is typically faster than programming down, try to sequence multiple tests in such a way that each consecutive test is at the same or higher voltage level as the previous test. See page 9 for more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

Table 1. Relay comparison chart

	Armature relay	Reed relay	Solid-state relay
Switch speed	50/s	1000/s	1000/s
Contact resistance	Low	Very low	High
Life	1 million	10 million	>10 million
Typical failure mode	Fails open	Fails open	Fails shorted
Typical max input	250 V/2 A	100 V/100 mA	250 V/10 A

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.

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

Program step	Input conditions(stimulus to DUT)	Measurement setup (to measure signal out of DUT)	DUT measurements taken
1	Set input condition 1 (e.g., amb. temp. = 0 degrees C)		
2		Prepare measurement setup 1 (e.g., voltage)	
3			Test point 1 voltage
4	Set input condition 2 (e.g., amb. temp. = 25 degrees C)		
5			Test point 1 voltage
6	Set input condition 3 (e.g., amb. temp. = 55 degrees C)		
7		Prepare measurement setup 2 (e.g., current)	
8			Test point 1 current
9	Set input condition 1 (0 degrees C)		
10		Prepare measurement setup 1 (voltage)	
11			Test point 2 voltage
12	Set input condition 2 (25 degrees C)		
13			Test point 2 voltage
14	Set input condition 3 (55 degrees C)		
15		Prepare measurement setup 2 (current)	
16			Test point 2 current
17	Set input condition 1 (0 degrees C)		
18		Prepare measurement setup 1 (voltage)	
19			Test point 3 voltage
20	Set input condition 2 (25 degrees C)		
21			Test point 3 voltage
22	Set input condition 3 (55 degrees C)		
23		Prepare measurement setup 2 (current)	
24			Test point 3 current

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.

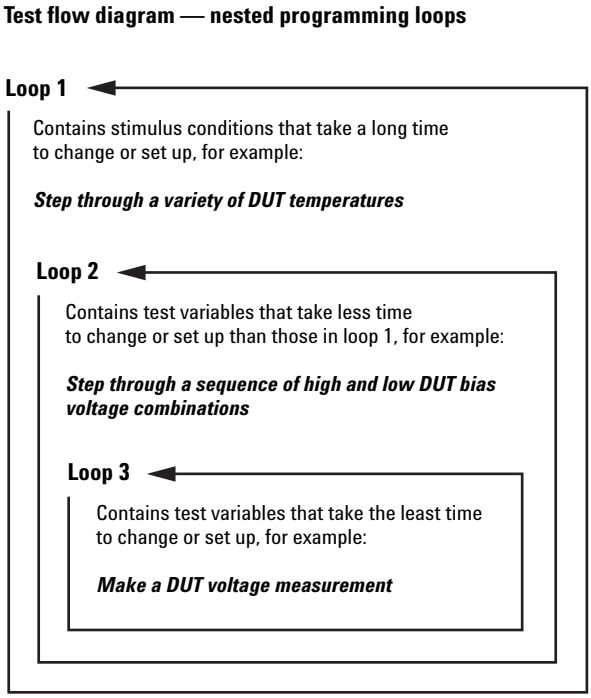


Table 3. Test sequence optimized for speed

Program step	Input conditions (stimulus to DUT)	Measurement setup (to measure signal out of DUT)	DUT measurements taken
1	Set input condition 1 (0 degrees C)		
2		Prepare measurement setup 1 (voltage)	
3			Test point 1 voltage
4			Test point 2 voltage
5			Test point 3 voltage
6	Set input condition 2 (25 degrees C)		
7			Test point 1 voltage
8			Test point 2 voltage
9			Test point 3 voltage
10	Set input condition 3 (55 degrees C)		
11		Prepare measurement setup 2 (current)	
12			Test point 1 current
13			Test point 2 current
14			Test point 3 current

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 4. Relative I/O times from a PC to an Agilent 33220A
(data taken with a 1-meter cable on an HP Kayak XU800 with an 800 MHz processor running Windows XP)

Interface	Function change	Frequency change	4K arb	64K arb
GPIB	99 ms	2 ms	20 ms	340 ms
USB 1.1	100 ms	4 ms	10 ms	185 ms
USB 2.0	99 ms	3 ms	8 ms	100 ms
LAN (socket)	100 ms	3 ms	8 ms	110 ms

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.

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. 5965-5290EN

Agilent Fault Detective Developer Application and Run-Time Engine, pub. no. 5988-4009EN

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

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: 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 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. 5968-6359E

Test System Signal Switching, (AN 1441-1), pub. no. 5988-8627EN **http://cp.literature.agilent.com/litweb/pdf/5988-8627EN.pdf**

Improving Throughput in Network Analyzer Applications, AN 1287-5, pub. no. 5966-3317E **http://cp.literature.agilent.com/litweb/pdf/5966-3317E.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."

www.agilent.com

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) (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

Printed in the USA December 9, 2004
5988-9822EN

Agilent Technologies