SCPI programming: Strengths and weaknesses

Larry Desjardin - November 24, 2013

The vast majority of programmable bench instruments, whether GP-IB or LXI, use a command language known as SCPI, Standard Commands for Programmable Instrumentation. To understand its purpose, a little history is in order.

Set the wayback machine to the early 1970s when Hewlett-Packard invented “HP-IB”, later named GPIB for General Purpose Interface Bus by the industry, and adopted as IEEE-488 by the IEEE. This revolutionized automated testing, as instruments from multiple vendors could be combined and automated from a test system controller.

In those days, very simple ASCII characters were sent to an instrument from a test system controller to control it. For example, “F1” could represent the DC Volt function for a DMM (digital multimeter), and “R2” could represent one of the voltage ranges. Results were typically sent back to the controller in ASCII form as well.

Those commands weren’t standardized. That is, each vendor was free to choose which commands initiated which actions for any given product model. Buy a similar product from another vendor, and it would use a completely different set of commands, so complete reprogramming would be needed. Indeed, different models from the same vendor would differ. Besides interoperability, it was difficult to read the commands when debugging or inspecting the code. There was no intuitive understanding of these cryptic commands. Some issues seemed fundamental. If one DMM offered ranges that were .3, 3, 30, and 300 volts, how could any programming language offer compatibility to another that had ranges of .1, 1, 10, 100, and 1000 volts?

These issues were largely solved by SCPI in 1990. Proposed by Hewlett Packard as TMSL, Test and Measurement System Language, it was adopted by others and managed by the SCPI Consortium. It is now managed by the IVI Foundation, IVI being an abbreviation for Interchangeable Virtual Instrument. The secret to its technical success was to define a signal-oriented language that would be interpreted by an instrument in real time.

In the SCPI world, **MEAS:VOLT:DC? 10.0,0.001** commands a DMM to measure the value of a DC voltage that may be as high as 10 volts to at least 1 millivolt of resolution. Note that how a DMM does this, and what ranges it has, are irrelevant. The DMM will make the measurement, and does so regardless of vendor or model. The instrument doesn’t even have to be a DMM, as long as it can make the measurement. Additionally, anyone reading the command string will know the purpose of the command. SCPI commands are very close to a natural language.

SCPI is widely deployed in the industry today by dozens of vendors. It offers interoperability and ease of use. Instruments may be programmed from any operating system, since the interpretation of
the ASCII commands is performed by the instruments, not instrument drivers.

That also is its weakness. Interpretation of ASCII natural language commands, and the subsequent handling of ASCII data, takes time. This time is often much longer than the measurement itself. Readers will probably recognize this as the traditional interpreted language versus compiled language trade-off. What if you could compile these commands ahead of time? Modular systems such as PXI or AXIe, combined with IVI drivers, offer this. Essentially all of the error checking, parsing, and ASCII conversion is done at compile time, while high-speed register access is performed at run time through the PCI Express memory map of the PC test system controller directly to the instrument.

For the above DMM example, an IVI function call to the instrument would be `dmm.DCVoltage.Measure (10.0,0.001)`.

Think the time between this and the interpreted SCPI equivalent is inconsequential? Not so. Agilent showed a reduction of round-trip measurement time from 3.3ms for their traditional box DMM to 66us for their PXI DMM, though the actual internal measurement speed was essentially the same. Instruments and test system controllers can both be upgraded to faster processors. So, while both times will decrease with technology, the 50 to one speed advantage cannot be easily mitigated. It is inherent in the two system architectures.

Perhaps it is unfair to say that SCPI also means Slow but Compatible Programmable Instruments, but the speed improvements demonstrated above by memory-mapped modular systems have been repeated time and time again by multiple vendors. If you feel the need for speed, this is the obvious solution, and one chief reason modular systems are gaining in popularity.

Switching to IVI drivers does not loose compatibility, however. The IVI Foundation, when creating the IVI driver architecture, leveraged heavily from SCPI. Indeed, the above IVI-C driver call for a DMM looks remarkably like a SCPI command string. The key thing is, it is parsed at compile time by a powerful test system controller, versus interpreted at run time by the instrument processor.

SCPI programming does retain one key advantage, however. The IVI function calls assume a Windows-based test system controller. This is acceptable by the 90% of test systems that deploy Windows, but compatibility is lost when switching a non-Windows system, such as Linux. Module vendors offer drivers on a case by case basis for other operating systems, but typically these are vendor specific drivers and vendor specific I/O kernels.

Let’s wrap it up: If you aren’t concerned about speed, SCPI is fine. Need to optimize speed- look at IVI drivers combined with a PCIe-based modular standard such as PXI or AXIe. Need to program outside of Windows? SCPI is your safest bet. Need speed, but not on Windows? That’s a subject for another time. Caveat emptor.

See also:

- Integrating traditional and modular instruments
- PCI Express: Espresso for test systems