KEITHLEY

Application Note Series

Number 2889

Optimizing Switched Measurements with the Series 3700 System Switch/Multimeter and Series 2600 System SourceMeter® Instruments Through the Use of TSP

Introduction

Keithley Instruments introduced test and measurement instruments equipped with on-board Test Script Processors (TSP™) in 2005 with the introduction of the first Series 2600 System SourceMeter® instruments. TSP allows message-based programming, much like SCPI, with enhanced capabilities for controlling test sequencing/flow, decision making, and instrument autonomy with the use of user-defined test scripts.

With Keithley's introduction of the Series 3700 System Switch/Multimeter, the latest line of TSP-enabled instruments, it's more important than ever for test system developers to have a thorough understanding of the advantages of this technology and how to apply it to their system configurations. This application note will address how switching to TSP-based instruments and using TSP programming best practices can produce up to a four-fold improvement in test system throughput in some circumstances. TSP is a very flexible hardware/software architecture that allows many different implementation choices. The primary tradeoffs are programming complexity versus throughput. Relatively simple programming techniques will allow noticeable throughput improvements. Slightly more complex programming can produce significantly higher throughput gains. This application note will present a variety of programming/throughput scenarios to allow users to select the most appropriate approaches for their applications.

Test Script Processor Overview

The use of an on-board Test Script Processor has made it possible to create "smart" instruments, i.e., instruments with built-in decision-making capabilities, which reduces the need to communicate so frequently with an external controller over the bus. This approach to test system design allows smart instrument systems to be much more efficient than those that rely on standard SCPI-based programming. As the number of TSP-based instruments grows, test system developers will have greater flexibility to build test systems with far higher throughput without compromising measurement integrity.

Keithley's TSP-based instruments incorporate a new technology that allows creating multi-channel I-V test systems economically. TSP-Link™ is a high speed trigger synchronization/inter-unit communication bus, which test system builders can use to connect multiple instruments in a master/slave configuration. Once connected, all the TSP-Link equipped instruments in a system can be programmed and operated under the control of the master unit or units, just as if they were housed in the same chassis. The test system can have multiple master/slave groups, which can be used, for example, to handle multi-die or multi-device testing in parallel. By serving as an external backplane without the need for a chassis/mainframe, TSP-Link provides virtually unlimited flexibility to scale a test system's channel count up or down as the application requires, while ensuring seamless integration. Combining TSP-Link with a very flexible programmable trigger model ensures unmatched speed and accuracy.

Master/slave operation (TSP-Link) produces much faster test times. Through Keithley's TSP-Link, multiple instruments are connected together and can used like they are part of the same physical unit for simultaneous multi-channel testing, as opposed to sequentially accessing multiple instruments.

Test System Description

NOTE – All times are based on the inclusion of a source and measurement function with every channel. Switching times would be faster if no source or measurement is included.

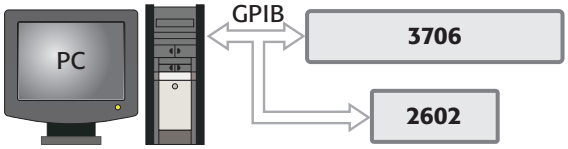
In the application examples illustrated here, a computer is used to control both the instruments, using a GPIB bus or TSP-Link to send each command. The devices under test (DUTs) are 10 LEDs. Each LED is sourced and measured by the Model 2602 Dual-Channel System SourceMeter® instrument. The Model 3706 System Switch/Multimeter, equipped with a Model 3723 switch card, is used as the switch controller.

This note examines the different ways these instruments can be used together, as well as ways to optimize their setup for best overall speed.

Demonstrated Examples:

- Setup 1 is somewhat similar to a traditional test system setup. Both the Model 2602 and Model 3706 are treated as masters. This is the slowest method of those illustrated, but will help demonstrate how the TSP approach improves upon old methods.
- Setup 2 makes greater use of the TSP. This setup uses the Model 2602 as the master and the Model 3706 as the slave. The script used demonstrates how TSP-Link and flow control decrease test time.
- Setup 3 uses the trigger model, along with TSP-Link, to deliver the fastest standard test times.
- Setup 4 goes one step further by customizing a switch card with software (TSP). This allows parallel testing, virtually doubling throughput.

Setup 1



Setup 2

Setups 3 and 4

Traditional GPIB control

The GPIB interface is a general-purpose bus; its primary use is to connect one or more GPIB-compatible instruments to a PC. It allows data to be exchanged one byte at a time among several different devices at speeds of 100k/second to 10MB/second. When combined with an effective handshaking protocol, GPIB is the preferred method of communication in test and measurement systems. Adding TSP programs to the already-effective GPIB bus makes for a much faster and more powerful solution.

The following test setup is used to simulate a traditional GPIB configuration. With this configuration, each piece of hardware is considered a master. It exploits none of the advantages that TSP-Link offers. TSP commands are used much like standard SCPI commands in this example.

An application created with LabVIEW 8.2 with NI VISA is used as the control software. As is true in SCPI programming, each TSP command can be sent individually or grouped with others. When TSP commands are grouped together and sent to an instrument at one time, they are called a script.

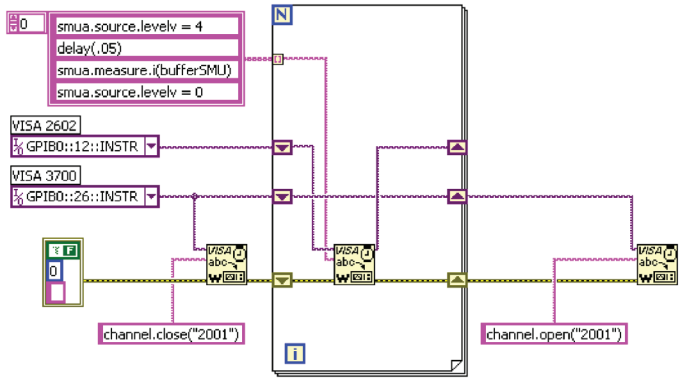


Figure 1. LabVIEW Code Sample

Test Results

Using traditional GPIB control led to varying results, depending on how and when certain commands were sent. Results ranged from 20 channels/second to 250 channels/second (when the TSP command `loadandrunscript` was used). The `loadandrunscript` command will be discussed in further detail later in this document, but essentially it allows the user to preload a significant number of TSP commands to be called later, a method that reduces test time significantly.

Test Summary

This example uses the Model 2602 to source and measure during each loop iteration. A for loop was used in LabVIEW and commands such as `channel.open` and `channel.close` were used, along with various source and measure commands.

As mentioned previously, this method is not the fastest method; the top speeds it can achieve are less than half of those possible with the other methods described later in this document.

TSP-Link—The Basics of TSP Control

Test Setup

For this test, the instruments are connected through TSP-Link, with the Model 2602 acting as the master. For this test, no trigger model is used, just a script that will perform the tests doing a single line execution. In a TSP-Link enabled system, one instrument (the unit that is connected to a communication bus such as

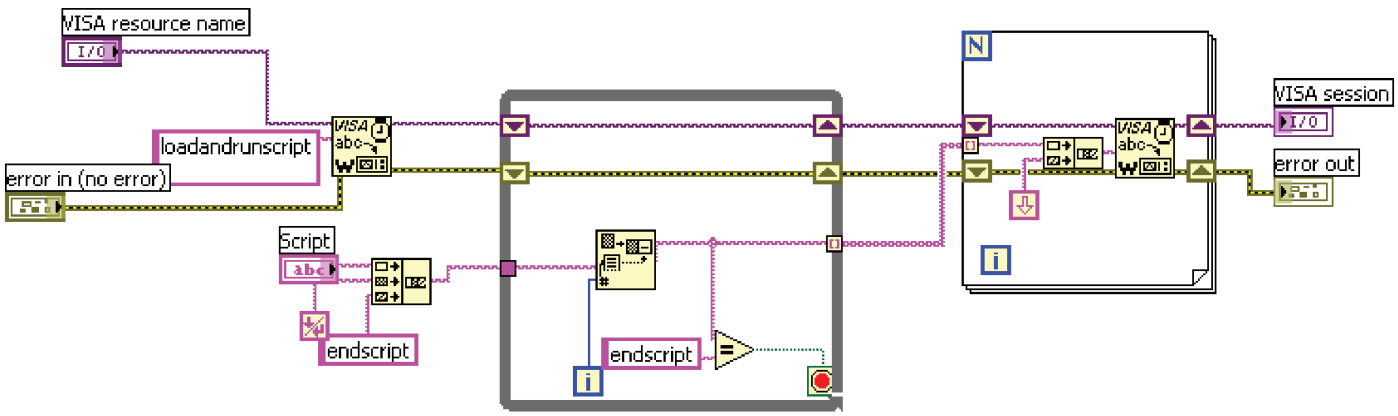


Figure 2. `loadandrunscript`

```
for i = 1, 10 do
        slaveNode37xx.channel.exclusiveclose(mem_pattern[i])
        smu.source.levelv = sourceLevelV
        delay(sourceSettleTime)
        smu.measure.i(bufferSMU)
        smu.source.levelv = 0
end
```

Figure 3. TSP Loop Sample for mem_pattern

GPIB or Ethernet) must be considered the master. The TSP script runs on the master instrument and controls the slave unit, so the same script program controls both instruments.

The concept of a "table" is introduced in this method of testing. In TSP, this is nothing more than a variable. However, this variable is an array of data, in this case, a list of channels.

```
mem_pattern = {"2001", "2002", "2003", "2004",
"2005", "2006", "2007", "2008", "2009", "2010"}
```

The TSP script will use a loop to step through each channel in the array. Various TSP commands for switching, such as `channel.open` and `channel.exclusiveclose` were used, along with various source and measure commands, such as `smu.source.rangev`, `smu.measure.i`, and `smu.source.levelv`.

Test Results

Test speeds of 360 channels/second were achieved with a source and measurement function.

Test Summary

A 1.5× throughput improvement was made possible by implementing the basics of TSP. Capabilities such as TSP-Link, for loops, and buffers eliminate GPIB bus traffic, unleashing the speed of TSP.

TSP-Link: Beyond the Basics (DIO as a Trigger Bus)

Test Setup

This test uses exactly the same script as in the previous test, only this time the TSP trigger model is included. The trigger model uses the Model 2602's and the Model 3706's highly flexible and extremely powerful digital I/O control.

The following commands improve throughput by using the trigger model:

```
scan.scancount
digio.trigger[].mode = digio.TRIG_FALLING
trigger.channel.stimulus = digio.trigger[].EVENT_ID
digio.trigger[].assert()
```

Test Results

Depending on the DUT, this method can switch up to 850 cycles/second with a measurement. Switch speed was so fast when using this method that a delay of 450μs was added prior to measurement to allow sufficient source settling time (DUT dependant). Even with the addition of settling time, it was possible to switch at greater than 670 cycles/second.

Test Summary

A 3× throughput improvement was achieved by implementing the basics of TSP plus the trigger model. Capabilities such as TSP-Link, for loops, buffers, and the trigger model eliminate GPIB bus traffic, unleashing the speed of TSP.

TSP-Link – Advanced Methods (Parallel Test)

Test Setup

For this configuration, the number of channel poles was changed from two to four.

```
channel.setpole("2001:2010", channel.POLES_FOUR)
```

What this actually does is close channels on the Model 3723 1×30 Reed Multiplexer card in groups of two. Following the example, if we close channel 2001, channel 2031 will close with it. This does not sound like particularly significantly until one considers the implications for the use of the Model 2602.

The Model 2602 is a two-channel SMU, which means that while SMUA is testing the DUT on channel 2001, SMUB can test the DUT on channel 2032. Simply by changing a few command lines, we have introduced parallel testing without changing hardware.

```
smua.source.levelv = sourceLevelV   --Set source value to turn LED on
smub.source.levelv = sourceLevelV
delay(sourceSettleTime)
smua.measure.i(bufferSMUa)
smub.measure.i(bufferSMUb)
smua.source.levelv = 0
smub.source.levelv = 0
digio.trigger[triggerLine].assert()
```

Figure 4. Parallel Test Method

As shown in ***Figure 4***, both SMUs measure using the same script, and the trigger is still used to cycle through the channel scan.

Test Results and Summary

Through the use of TSP and TSP-Link, switching throughput has been improved from 250 channels/second to 1100 channels/second, which represents a throughput improvement of greater than 4×.

TSP-Link—Summary

This note can only suggest the potential for throughput improvement possible through the use of TSP and TSP-Link. By using high performance hardware such as the Model 2602 and Model 3706, coupled with some creative thinking, it's possible to create custom configurations optimized for high speed test and measurement.

Table 1. Test Method Summary

Method	Switching Time	Comments
Traditional GPIB	20 channels/ second to 250 channels/second	Test times vary based on programming styles. This method does not take advantage of the increased throughput potential of TSP.
Basic TSP Programming	360 channels/ second	Improve test time by using the basics of TSP. Novice TSP programmer can expect these results. Some C background is desirable.
Basic TSP with Digital Triggering	850 channels/ second	By adding digital I/O triggering to TSP, users can expect a 2× improvement. Novice TSP programmer or C programmer with some instrument programming background can expect these results.
Advanced TSP Programming with Digital Triggering	1100 channels/ second	Experienced TSP programmers or C programmers willing to learn TSP commands can easily attain these results. Some instrument experience is desirable.

Consult a Keithley representative or the Keithley applications group for advice on the equipment and resources available to create smarter, more cost-effective test solutions based on TSP and TSP-Link technologies.

Appendix A – TSP Commands

NOTE: The following list provides the TSP commands used throughout this document. They may not be used in every specific case and will not be used in all cases.

Configuration Commands

Series 3700:

`channel.connectrule`

- Indicates the connection rule for closing and opening channels in the system.
- `channel.BREAK _ BEFORE _ MAKE` or 1 to have BBM connections for relays in system relays
- When the connection rule is set to `channel.OFF`, the command being processed will use the most optimal connection rule at that time. Therefore, the same command may be connected using BBM and then later use MBB. Use this setting when you don't care which rule is used and don't need a specific rule each time a command is processed. Otherwise, use one of the other two settings to indicate your desired connection rule.
- This attribute is not applicable for analog output, digital I/O, totalizer, and DAC cards. It does apply to switch cards like EMR, read and solid-state relay cards.
- Default setting and `channel.reset` rule is `channel.BREAK _ BEFORE _ MAKE`

`scan.reset()`

- Resets the scanning aspects of the system to factory defaults.
- This command will only reset the scan aspects of the system to factory defaults. Settings effects are:

  1. Trigger model gets reset to factory defaults.
  2. List of channels and/or channel patterns to scan get cleared along with their associated DMM configurations if overriding ones set by `dmm.setconfig` or using multiple ones.
  3. Creates an empty scan.

- The rest of the settings are unaffected. To reset the entire system to factory defaults, use the reset command.

`scan.create(ch _ list, dmm _ config)`

- Creates a list of channels and/or channel patterns to scan.
- Use this function to replace an existing list of channels and/or channel patterns to scan. The existing scan list is lost after this command. These items purge the old list and start a new scan list. The items in `ch _ list` will be scanned in the order specified in the parameter list.
- If the optional `dmm _ config` parameter is not specified then, the configuration (`dmm.setconfig`, `dmm.getconfig`) associated with that channel or channel pattern will be used. However, if a `dmm _ config` is specified then, that configuration is used for each channel or channel pattern specified in a temporary override mode. It does not modify the assigned configuration of a channel or channel list.

`channel.setpole(ch _ list, value)`

- Specifies the pole setting for a list of channels.
- The parameter string may contain "slotX", where X equals 1 to 6, or "allslots".
- Command processing will stop as soon as an error is detected and no pole settings will be modified. Only with no errors will the specified channels have their poles setting changed.
- Recall channel patterns do not have a pole setting associated with them. The user manipulates the analog backplane relays for the desired pole setting by using the `channel.setbackplane` function for channels. However, for channel patterns, the analog backplane relays must be specified when creating the pattern (see `channel.pattern.set` and `channel.pattern.snapshot`).
- For channels, as the pole setting changes, their analog backplane relays, specified by `channel.setbackplane`, get cleared. Therefore, after a pole-setting change, the user needs to add the desired analog backplane relays for desired pole setting by using `channel.setbackplane`. The analog backplane relays, specified by `dmm.setbackplane`, don't get updated based on pole setting. They get manipulated based on the DMM configuration assigned to a channel. When clearing the backplane channels, this may involve clearing the paired channel whether pairing or unpairing channels. For example, on a 40-channel card, channels 1 and 21 are paired when the poles for channel 1 is set to 4. Therefore, when setting the poles setting on channel 1 to four will clear the backplane channels for channels 1 and 21.

Likewise, they both will be cleared when the poles setting is set back to two on channel 1.

scan.scancount

- Set or query the scan count value.
- This attribute sets the scan count in the trigger model. During a scan, the Series 3700 instrument will iterate through the arm layer of the trigger model this many times. After performing this count iterations, the 3700 instrument will return to idle.
- If this count is set to zero, the 3700 instrument will stay in the trigger model indefinitely until aborted.
- The reset value for this attribute is 1.

scan.trigger.channel.stimulus

- Channel event detector stimulus selection.
- This attribute selects which event(s) will cause the channel event detector to enter the detected state. Set this attribute to nil to bypass waiting for an event.
- Trigger event IDs
  1. digio.trigger[line].EVENT _ ID
  2. display.trigger.EVENT _ ID
  3. trigger.EVENT _ ID
  4. trigger.blender[N].EVENT _ ID
  5. trigger.timer[N].EVENT _ ID
  6. tsplink.trigger[N].EVENT _ ID
  7. lan.trigger[N].EVENT _ ID
  8. scan.trigger.EVENT _ SCAN _ READY
  9. scan.trigger.EVENT _ SCAN _ START
  10. scan.trigger.EVENT _ CHANNEL _ READY
  11. scan.trigger.EVENT _ IDLE
  12. scan.trigger.EVENT _ MEASURE _ COMP
  13. scan.trigger.EVENT _ SEQUENCE _ COMP
  14. scan.trigger.EVENT _ SCAN _ COMP

Series 2600:

smuX.makebuffer(buffersize)

- buffersize = Number of readings that can be stored.
- RAM reading buffers can be allocated dynamically. These are created and allocated with the smuX.makebuffer(buffer) function, where buffersize is the number of readings the buffer can store.
- Dynamically allocated reading buffers can be used interchangeably with the smuX.nvbufferY buffers.
- A RAM buffer can be deleted using nil. The following command deletes mybuffer:

buffer.clear()

- This function clears all readings from the indicated buffer.

buffer.appendmode

- Append mode for the reading buffer.
- Assigning to this attribute enables or disables the buffer append mode.
- With append mode on, the first new measurement will be stored at rb[n+1], where n is the number of readings stored in the buffer.
- The default is buffer append mode off.

smuX.source.autorangeY

- Source auto range control (on/off).
- This attribute indicates the source auto range state. Its value will be smuX.AUTORANGE _ OFF when the SMU source circuit is on a fixed range and smuX.AUTORANGE _ ON when it is in auto range mode.
- Setting this attribute to smuX.AUTORANGE _ OFF puts the SMU on a fixed source range. The fixed range used will be the range the SMU source circuit was currently using.
- Setting this attribute to smuX.AUTORANGE _ ON puts the SMU source circuit into auto range mode. If the source output is on, the SMU will immediately change range to the range most appropriate for the value being sourced if that range is different from the SMU range.
- Auto range will disable if the source level is edited from the front panel.

smuX.measure.autorangeY

- Measure auto range setting.
- This attribute indicates the measurement auto range state. Its value will be smuX.AUTORANGE _ OFF when the SMU measure circuit is on a fixed range and smuX.AUTORANGE _ ON when it is in auto range mode.
- Setting this attribute to smuX.AUTORANGE _ OFF puts the SMU on a fixed range. The fixed range used will be the range the SMU measure circuit was currently using.
- Setting this attribute to smuX.AUTORANGE _ ON puts the SMU measure circuit into auto range mode. It will remain on its present measure range until the next measurement is requested.

smuX.source.rangeY

- Source range.
- Reading this attribute returns the positive full-scale value of the source range the SMU is currently using. Assigning to this attribute sets the SMU on a fixed range large enough to source the given value. The instrument will select the best range for sourcing a value of rangeval.
- smuX.source.rangeX is primarily intended to eliminate the time required by the automatic range selection performed by a sourcing instrument. Because selecting a fixed range will prevent auto-ranging, an over-range condition can occur, for example, sourcing 10.0V on the 6.0V range.

smuX.source.levelY

- Source levels.
- This attribute configures the source level of the voltage or current source.
- If the source is configured as a voltage source and the output is on, the new `smuX.source.levelv` setting will be sourced immediately. If the output is off or if the source is configured as a current source, the voltage level will be sourced when the source is configured as a voltage source and the output is turned on.
- If the source is configured as a current source and the output is on, the new `smuX.source.leveli` setting will be sourced immediately. If the output is off or if the source is configured as a voltage source, the current level will be sourced when the source is configured as a current source and the output is turned on.
- The sign of `sourceval` dictates the polarity of the source. Positive values generate positive voltage or current from the high terminal of the source relative to the low terminal. Negative values generate negative voltage or current from the high terminal of the source relative to the low terminal.
- The reset function sets the source levels to 0V and 0A.

`smuX.measure.nplc`

- Integration aperture for measurements.
- The integration aperture is based on the number of power line cycles (NPLC), where 1PLC for 60Hz is 16.67ms (1/60) and 1 PLC for 50Hz is 20ms (1/50).
- The reset function sets the aperture to 1.0.

`smuX.source.output`

- Source output control (on/off).
- Reading this attribute gives the output state of the source. Setting this attribute will turn the output of the source on or off. The default for the source is off. When the output is turned on, the SMU will source either voltage or current as dictated by the `smuX.source.func` setting.

`loadandrunscript`

- This command loads and runs a script or in this case several lines of code; programming techniques such as a loop, timers, waits, and functions can also be used.

KEITHLEY

A GREATER MEASURE OF CONFIDENCE

KEITHLEY INSTRUMENTS, INC. ■ 28775 AURORA ROAD ■ CLEVELAND, OHIO 44139-1891 ■ 440-248-0400 ■ Fax: 440-248-6168 ■ 1-888-KEITHLEY ■ www.keithley.com

BELGIUM
Sint-Pieters-Leeuw
Ph: 02-363 00 40
Fax: 02-363 00 64
www.keithley.nl

CHINA
Beijing
Ph: 8610-82255010
Fax: 8610-82255018
www.keithley.com.cn

FINLAND
Espoo
Ph: 09-88171661
Fax: 09-88171662
www.keithley.com

FRANCE
Saint-Aubin
Ph: 01-64 53 20 20
Fax: 01-60-11-77-26
www.keithley.fr

GERMANY
Germering
Ph: 089-84 93 07-40
Fax: 089-84 93 07-34
www.keithley.de

INDIA
Bangalore
Ph: 080-26771071-73
Fax: 080-26771076
www.keithley.com

ITALY
Milano
Ph: 02-553842.1
Fax: 02-55384228
www.keithley.it

JAPAN
Tokyo
Ph: 81-3-5733-7555
Fax: 81-3-5733-7556
www.keithley.jp

KOREA
Seoul
Ph: 82-2-574-7778
Fax: 82-2-574-7838
www.keithley.co.kr

MALAYSIA
Kuala Lumpur
Ph: 60-3-4041-0899
Fax: 60-3-4042-0899
www.keithley.com

NETHERLANDS
Gorinchem
Ph: 0183-63 53 33
Fax: 0183-63 08 21
www.keithley.nl

SINGAPORE
Singapore
Ph: 65-6747-9077
Fax: 65-6747-2991
www.keithley.com.sg

SWEDEN
Solna
Ph: 08-50 90 46 00
Fax: 08-655 26 10
www.keithley.com

SWITZERLAND
Zürich
Ph: 044-821 94 44
Fax: 41-44-820 30 81
www.keithley.ch

TAIWAN
Hsinchu
Ph: 886-3-572-9077
Fax: 886-3-572-9031
www.keithley.com.tw

UNITED KINGDOM
Theale
Ph: 0118-929 75 00
Fax: 0118-929 75 19
www.keithley.co.uk

Printed in the U.S.A. No. 2889 Oct. 07 3KGP