Series 2600A
System SourceMeter® Instruments

Semiconductor Device Test
Applications Guide

Contains Programming Examples

KEITHLEY

A GREATER MEASURE OF CONFIDENCE

Although this Guide was originally developed as an applications resource for Series 2600 System SourceMeter® instruments, the application information and sample test scripts provided are equally suitable for use with Keithley's newest SMU line, the Series 2600A. To implement any of these applications with the new models, simply substitute the equivalent new model number for the original, that is, Model 2602A to replace Model 2602, Model 2612A to replace Model 2612, etc.

Table of Contents

List of Illustrations

Section 1
General Information

1.1 Introduction

The following paragraphs discuss the overall hardware and software configurations of the system necessary to run the example application programs in this guide.

1.2 Hardware Configuration

1.2.1 System Configuration

Figure 1-1 shows the overall hardware configuration of a typical test system. The various components in the system perform a number of functions:

Series 2600 System SourceMeter Instruments: System SourceMeter instruments are specialized test instruments capable of sourcing current and simultaneously measuring voltage, or sourcing current and simultaneously measuring voltage. A single Source-Measure Unit (SMU) channel is required when testing two-terminal devices such as resistors or capacitors. Three- and four-terminal devices, such as BJTs and FETs, may require two or more SMU channels. Dual-channel System SourceMeter instruments, such as the Models 2602, 2612, and 2636, provide two SMUs in a half-rack instrument. Their ease of programming, flexible expansion, and wide coverage of source/measure signal levels make them ideal for testing a wide array of discrete components. Before starting, make sure the instrument you are using has the source and measurement ranges that will fit your testing specifications.

Test fixture: A test fixture can be used for an external test circuit. The test fixture can be a metal or nonmetallic enclosure, and is typically equipped with a lid. The test circuit is mounted inside the test fixture. When hazardous voltages (>30Vrms, 42Vpeak) will be present, the test fixture must have the following safety requirements:

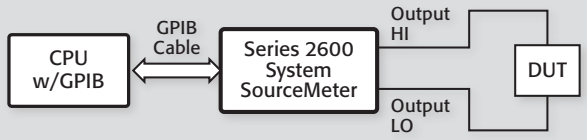

Figure 1-1. Typical system configuration for applications

WARNING

To provide protection from shock hazards, an enclosure should be provided that surrounds all live parts. Nonmetallic enclosures must be constructed of materials suitably rated for flammability and the voltage and temperature requirements of the test circuit. For metallic enclosures, the test fixture chassis must be properly connected to safety earth ground. A grounding wire (#18 AWG or larger) must be attached securely to the test fixture at a screw terminal designed for safety grounding. The other end of the ground wire must be attached to a known safety earth ground.

Construction Material: A metal test fixture must be connected to a known safety earth ground as described in the WARNING above.

WARNING

A nonmetallic test fixture must be constructed of materials that are suitable for flammability, voltage, and temperature conditions that may exist in the test circuit. The construction requirements for a nonmetallic enclosure are also described in the WARNING above.

Test Circuit Isolation: With the lid closed, the test fixture must completely surround the test circuit. A metal test fixture must be electrically isolated from the test circuit. Input/output connectors mounted on a metal test fixture must also be isolated from the test fixture. Internally, Teflon® standoffs are typically used to insulate the internal pc-board or guard plate for the test circuit from a metal test fixture.

Interlock Switch: The test fixture must have a normally open interlock switch. The interlock switch must be installed so that, when the lid of the test fixture is opened, the switch will open, and when the lid is closed, the switch will close.

WARNING

When an interlock is required for safety, a separate circuit should be provided that meets the requirements of the application to protect the operator reliably from exposed voltages. **The output enable pin**

on the *digital I/O port* on the Models 2601 and 2602 System SourceMeter instruments is not suitable for control of safety circuits and should not be used to control a safety interlock. The Interlock pin on the *digital I/O port* for the Models 2611, 2612, 2635, and 2636 can be used to control a safety interlock.

Computer: The test programs in this document require a PC with IEEE-488 (GPIB) communications and cabling.

Software: Series 2600 System SourceMeter instruments each use a powerful on-board test sequencer known as the Test Script Processor (TSP™). The TSP is accessed through the instrument communications port, most often, the GPIB. The test program, or script, is simply a text file that contains commands that instruct the instrument to perform certain actions. Scripts can be written in many different styles as well as utilizing different programming environments. This guide discusses script creation and management using Keithley Test Script Builder (TSB), an easy-to-use program that allows you to create, edit, and manage test scripts. For more information on TSB and scripting, see *Section 2: Using Test Script Builder* of the Series 2600 Reference Manual.

Connections and Cabling: High quality cabling, such as the Keithley Model 2600-BAN or Model 7078-TRX-3 triaxial cables, should be used whenever possible.

1.2.2 Remote/Local Sensing Considerations

In order to simplify the test connections, most applications in this guide use local sensing for the SMUs. Local sensing requires connecting only two cables between the SMUs and the test fixture (OUTPUT HI and OUTPUT LO).

When sourcing and/or measuring voltage in a low impedance test circuit, there can be errors associated with IR drops in the test leads. Using four-wire remote sense connections optimizes voltage source and measure accuracy. When sourcing voltage, four-wire remote sensing ensures that the programmed voltage is delivered to the DUT. When measuring voltage, only the voltage drop across the DUT is measured. Use four-wire remote sensing for the following source-measure conditions:

- Sourcing and/or measuring voltage in low impedance (<1kΩ) test circuits.
- Enforcing voltage compliance limit directly at the DUT.

1.3 Graphing

All of the programs in this guide print the data to the TSB Instrument Console. In some cases, graphing the data can help you visualize the characteristics of the DUT. One method of graphing is to copy and paste the data from the TSB Instrument Console and place it in a spreadsheet program such as Microsoft Excel.

After the script has run, and the data has been returned to the Instrument Console, you can highlight it by using the PC's mouse: depress the Control and c (commonly written as Ctrl+c) keys on the keyboard simultaneously, switch to an open Excel worksheet, and depress Control and v simultaneously (Ctrl+v). The data should now be placed in the open worksheet columns so you can use the normal graphing tools available in your spreadsheet program to graph the data as needed.

This Applications Guide is designed for Series 2600 instrument users who want to create their own scripts using the Test Script Builder software. Other options include LabTracer® 2 software, the Automated Characterization Suite (ACS), and a LabVIEW driver.

Section 2
Two-terminal Device Tests

2.1 Introduction

Two-terminal device tests discussed in this section include voltage coefficient tests on resistors, leakage tests on capacitors, and diode characterization.

2.2 Instrument Connections

Figure 2-1 shows the instrument connections for two-terminal device tests. Note that only one channel of a Source-Measure Unit (SMU) is required for these applications. Be aware that multi-channel models, such as the Model 2602, can be used, but are not required to run the test program.

WARNING

Lethal voltages may be present. To avoid a possible shock hazard, the test system should be equipped with protective shielding and a safety interlock circuit. For more information on interlock techniques, see Section 10 of the Series 2600 Reference manual.

Turn off all power before connecting or disconnecting wires or cables.

NOTES

1. Remote sensing connections are recommended for optimum accuracy. See paragraph 1.2.2 for details.
2. If measurement noise is a problem, or for critical, low level applications, use shielded cable for all signal connections.

2.3 Voltage Coefficient Tests of Resistors

Resistors often show a change in resistance with applied voltage with high megohm resistors (>$10^9\Omega$) showing the most pronounced effects. This change in resistance can be characterized as the voltage coefficient. The following paragraphs discuss voltage coefficient tests using a single-channel Model 2601 System SourceMeter instrument. The testing can be performed using any of the Series 2600 System SourceMeter instruments.

2.3.1 Test Configuration

The test configuration for voltage coefficient measurements is shown in **Figure 2-2**. One SMU sources the voltage across the resistor under test and measures the resulting current through the resistor.

2.3.2 Voltage Coefficient Calculations

Two different current readings at two different voltage values are required to calculate the voltage coefficient. Two resistance read-

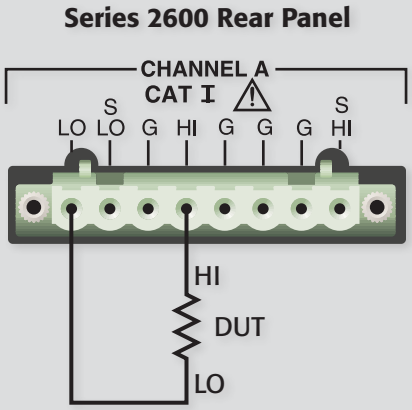

Figure 2-1. Series 2600 two-wire connections (local sensing)

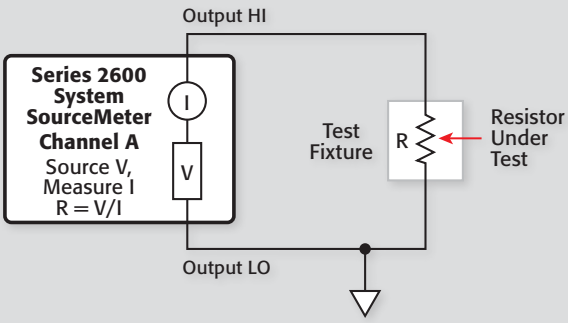

Figure 2-2. Voltage coefficient test configuration

ings, R_1 and R_2, are then obtained, and the voltage coefficient in %/V can then be calculated as follows:

$$\text{Voltage Coefficient (\%/V)} = \frac{100\ (R_2 - R_1)}{R_1\ (V_2 - V_1)}$$

where: R_1 = resistance calculated with first applied voltage (V_1).

R_2 = resistance calculated with second applied voltage (V_2).

For example, assume that the following values are obtained:

$R_1 = 1.01 \times 10^{10}\Omega$

$R_2 = 1 \times 10^{10}\Omega$

$(V_2 - V_1) = 10V$

The voltage coefficient is:

$$\text{Voltage Coefficient (\%/V)} = \frac{100\ (1\times10^3)}{1\times10^{10}\ (10)} = 0.1\%/V$$

2.3.3 Measurement Considerations

A couple of points should be noted when using this procedure to determine the voltage coefficient of high megohm resistors. Keep in mind that any leakage resistance in the test system will degrade the accuracy of your measurements. To avoid such problems, use only high quality test fixtures that have insulation resistances greater than the resistances being measured. Using isolation resistances 10× greater than the measured resistance is a good rule of thumb. Also, make certain that the test fixture sockets are kept clean and free of contamination as oils and dirt can lower the resistance of the fixture and cause error in the measurement.

There is an upper limit on the resistance value that can be measured using this test configuration. For one thing, even a well-designed test fixture has a finite (although very high) path isolation value. Secondly, the maximum resistance is determined by the test voltage and current-measurement resolution of the test instrument. Finally, the instrument has a typical output impedance of $10^{15}\Omega$. To maximize measurement accuracy with a given resistor, use the highest test voltages possible.

2.3.4 Example Program 1: Voltage Coefficient Test

Program 1 demonstrates programming techniques for voltage coefficient tests. Follow the steps that follow to use the test program. To reiterate, this test requires a single Source-Measure channel. For this example, we will refer to the single-channel Model 2601 System SourceMeter instrument. The test program can be used with the multi-channel members of the Series 2600 family with no modification.

1. With the power off, connect the Model 2601 System SourceMeter instrument to the computer's IEEE-488 interface.
2. Connect the test fixture to the instrument using appropriate cables (see **Figure 2-1**).
3. Turn on the instrument, and allow the unit to warm up for two hours for rated accuracy.
4. Turn on the computer and start Test Script Builder (TSB). Once the program has started, open a session by connecting to the instrument. For details on how to use TSB, see the Series 2600 Reference Manual.
5. You can simply copy and paste the code from Appendix A in this guide into the TSB script editing window (Program 1: Voltage Coefficient), manually enter the code from the appendix, or import the TSP file *'Volt_Co.tsp'* after downloading it to your PC.

   If your computer is currently connected to the Internet, you can click on this link to begin downloading: http://www.keithley.com/data?asset=50914.
6. Install the resistor being tested in the test fixture. The first step in the operation requires us first to send the code to the instrument. The simplest method is to right-click in the open script window of TSB, and select 'Run as TSP file'. This will compile the code and place it in the volatile run-time memory of the instrument. To store the program in non-volatile memory, see the "TSP Programming Fundamentals" section of the Series 2600 Reference Manual.
7. Once the code has been placed in the instrument run-time memory, we can run it simply by calling the function '`Volt_Co()`'. This can be done by typing the text '`Volt_Co()`' after the active prompt in the Instrument Console line of TSB.
8. In the program *'Volt_Co.tsp'*, the function `Volt_Co(v1src, v2src)` is created. The variables `v1src` and `v2src` represent the two test voltage values applied to the device-under-test (DUT). If they are left blank, the function will use the default values given to these variables, but you can specify what voltages are applied by simply sending voltages that are in-range in the function call. As an example, if you wanted to source 2V followed by 10V, simply send `Volt_Co(2, 10)` to the instrument.
9. The instrument will then source the programmed voltages and measure the respective currents through the resistor. The calculated voltage coefficient and two resistance values will then be displayed in the Instrument Console window of TSB.

2.3.5 Typical Program 1 Results

The actual voltage coefficient you obtain using the program will, of course, depend on the resistor being tested. The typical voltage coefficient obtained for a 10GΩ resistor (Keithley part number R-319-10G) was about 8ppm/V (0.008%/V).

2.3.6 Program 1 Description

At the start of the program, the instrument is reset to default conditions, and the error queue and data storage buffers are cleared. The following configuration is then applied before the data collection begins:

- Source V, DC mode
- Local sense
- 100mA compliance, autorange measure
- 1NPLC line cycle integration
- `v1src:` 100V
- `v2src:` 200V

The instrument then sources `v1src`, checks the source for compliance in the function named `Check _ Comp()`, and performs a measurement of the current if compliance is false. The source then applies `v2src` and performs a second current measurement.

The function `Calc _ Val()` then performs the calculation of the voltage coefficient based on the programmed source values and the measured current values as described in Section 2.3.2, Voltage Coefficient Calculations.

The instrument output is then turned off and the function `Print _ Data()` is run to print the data to the TSB window.

Note: If the compliance is true, the instrument will abort the program and print a warning to the TSB window. Check the DUT and cabling to make sure everything is connected correctly and re-run the test.

2.4 Capacitor Leakage Test

One important parameter associated with capacitors is leakage current. Once the leakage current is known, the insulation resistance can be easily calculated. The amount of leakage current in a capacitor depends both on the type of dielectric as well as the applied voltage. With a test voltage of 100V, for example, ceramic dielectric capacitors have typical leakage currents in the nanoamp to picoamp range, while polystyrene and polyester dielectric capacitors exhibit a much lower leakage current—typically in the femtoamp (10^{-15}A) range

2.4.1 Test Configuration

Figure 2-3 shows the test configuration for the capacitor leakage test. The instrument sources the test voltage across the capacitor, and it measures the resulting leakage current through the device. The resistor, R, is included for current limiting, and it also helps to reduce noise. A typical value for R is 1MΩ, although that value can be decreased for larger capacitor values. Note, however, that values less than 10kΩ are not recommended.

2.4.2 Leakage Resistance Calculations

Once the leakage current is known, the leakage resistance can easily be calculated from the applied voltage and leakage current value as follows:

$R = V/I$

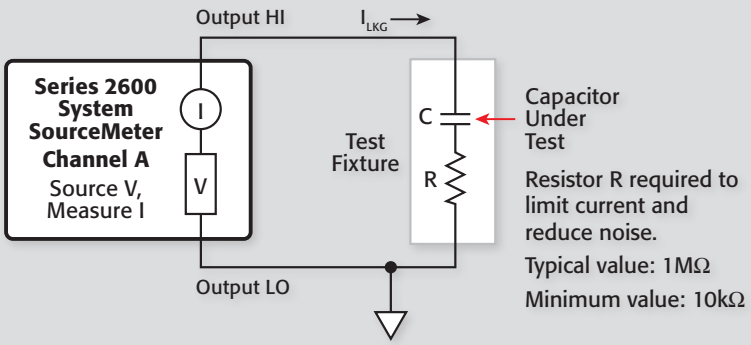

Figure 2-3. Test configuration for capacitor leakage test

For example, assume that you measured a leakage current of 25nA with a test voltage of 100V. The leakage resistance is simply:

R =100/25nA = 4GΩ ($4 \times 10^9\Omega$)

2.4.3 Measurement Considerations

After the voltage is applied to the capacitor, the device must be allowed to charge fully before the current measurement can be made. Otherwise, an erroneous current, with a much higher value, will be measured. The time period during which the capacitor charges is often termed the "soak" time. A typical soak time is seven time constants, or 7RC, which would allow settling to less than 0.1% of final value. For example, if R is 1MΩ, and C is 1μF, the recommended soak time is seven seconds. With small leakage currents (<1nA), it may be necessary to use a fixed measurement range instead of auto ranging.

2.4.4 Example Program 2: Capacitor Leakage Test

Program 2 performs the capacitor leakage test described above. Follow the steps that follow to run the test using this program.

WARNING

Hazardous voltage may be present on the capacitor leads after running this test. Discharge the capacitor before removing it from the test fixture.

1. With the power off, connect the instrument to the computer's IEEE-488 interface.
2. Connect the test fixture to the instrument using appropriate cables.
3. Turn on the instrument, and allow the unit to warm up for two hours for rated accuracy.
4. Turn on the computer and start Test Script Builder (TSB). Once the program has started, open a session by connecting to the instrument. For details on how to use TSB, see the Series 2600 Reference Manual.
5. You can simply copy and paste the code from Appendix A in this guide into the TSB script editing window (Program 2), manually enter the code from the appendix, or import the TSP file '*Cap_Leak.tsp*' after downloading it to your PC.

   If your computer is currently connected to the Internet, you can click on this link to begin downloading: http://www.keithley.com/data?asset=50927.
6. Discharge and install the capacitor being tested, along with the series resistor, in the appropriate axial component sockets of the test fixture.

WARNING

Care should be taken when discharging the capacitor, as the voltage present may represent a shock hazard!

7. Now, we must send the code to the instrument. The simplest method is to right-click in the open script window of TSB, and select '*Run as TSP file*'. This will compile the code and place it in the volatile run-time memory of the instrument. To store the program in non-volatile memory, see the "TSP Programming Fundamentals" section of the Series 2600 Reference Manual.
8. Once the code has been placed in the instrument run-time memory, we can run it at any time simply by calling the function '`Cap _ Leak()`'. This can be done by typing the text '`Cap _ Leak()`' after the active prompt in the Instrument Console line of TSB.
9. In the program '*Cap_Leak.tsp*', the function `Cap _ Leak(vsrc)` is created. The variable `vsrc` represents the test voltage value applied to the device-under-test (DUT). If it is left blank, the function will use the default value given to the variable, but you can specify what voltage is applied by simply sending a voltage that is in-range in the function call. As an example, if you wanted to source 100V, simply send `Cap _ Leak(100)` to the instrument.
10. The instrument will then source the programmed voltage and measure the respective current through the capacitor. The measured current leakage and calculated resistance value will then be displayed in the Instrument Console window of TSB.

NOTE

The capacitor should be fully discharged before running the test. This can be accomplished by sourcing 0V on the device for the soak time or by shorting the leads together. Care should be taken because some capacitors can hold a charge for a significant period of time and could pose an electrocution risk.

The soak time, denoted in the code as the variable `l _ soak`, has a default value of 10s. When entering the soak time, choose a value of at least 7RC to allow settling to within 0.1% of final value. At very low currents (<500fA), a longer settling time may be required to compensate for dielectric absorption, especially at high voltages.

2.4.5 Typical Program 2 Results

As pointed out earlier, the exact value of leakage current will depend on the capacitor value as well as the dielectric. A typical value obtained for 1μF aluminum electrolytic capacitor was about 80nA at 25V.

2.4.6 Program 2 Description

At the start of the program, the instrument is reset to default conditions, the error queue, and data storage buffers are cleared. The following configuration is then applied before the data collection begins:

- Source V, DC mode
- Local sense
- 10mA compliance, autorange measure
- 1 NPLC Line cycle integration
- `vsrc`: 40V

The instrument then sources `vsrc`, checks the source for compliance in the function named `Check _ Comp()`, and performs a measurement of the current if compliance is false.

The function `Calc _ Val()` then performs the calculation of the leakage resistance based on the programmed source value and the measured current value as described in paragraph 2.4.2, Leakage Resistance Calculations.

The instrument output is then turned off and the function `Print _ Data()` is run to print the data to the TSB window.

Note: If the compliance is true, the instrument will abort the program and print a warning to the TSB window. Check the DUT and cabling to make sure everything is connected correctly and re-run the test.

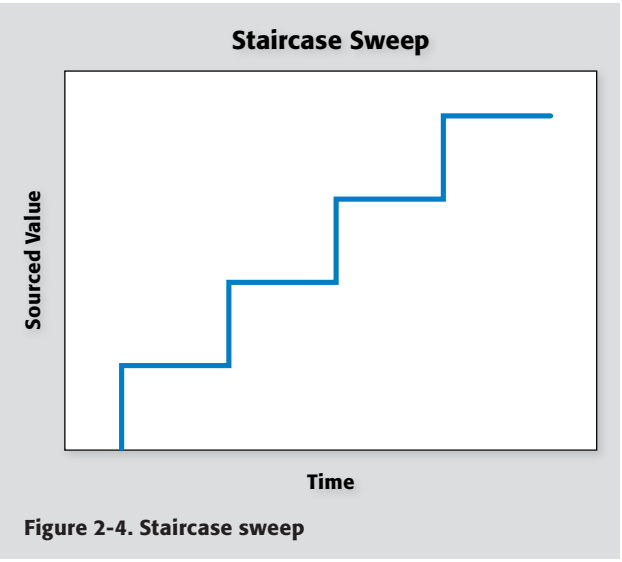

Figure 2-4. Staircase sweep

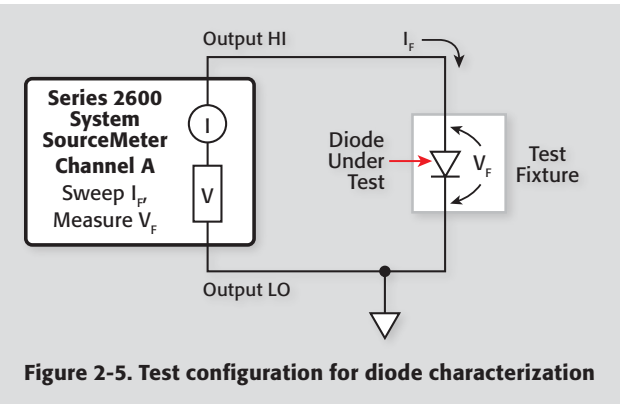

Figure 2-5. Test configuration for diode characterization

2.5 Diode Characterization

The System SourceMeter instrument is ideal for characterizing diodes because it can source a current through the device, and measure the resulting forward voltage drop (V_F) across the device. A standard technique for diode characterization is to perform a staircase sweep (**Figure 2-4**) of the source current from a starting value to an end value while measuring the voltage at each current step. The following paragraphs discuss the test configuration and give a sample test program for such tests.

2.5.1 Test Configuration

Figure 2-5 shows the test configuration for the diode characterization test. The System SourceMeter instrument is used to source the forward current (I_F) through the diode under test, and it also measures the forward voltage (V_F) across the device. I_F is swept across the desired range of values, and V_F is measured at each current. Note that the same general configuration could be used to measure leakage current by reversing the diode, sourcing voltage, and measuring the leakage current.

2.5.2 Measurement Considerations

Because the voltages being measured will be fairly small (≈0.6V), remote sensing can be used to minimize the effects of voltage drops across the test connections and in the test fixture. Remote sensing requires the use of the Sense connections on the System SourceMeter channel being used, as well as changing the code to reflect remote sensing. For more information on remote sensing, see the Series 2600 Reference Manual.

2.5.3 Example Program 3: Diode Characterization

Program 3 demonstrates the basic programming techniques for running the diode characterization test. Follow these steps to use this program:

1. With the power off, connect the instrument to the computer's IEEE-488 interface.
2. Connect the test fixture to the instrument using appropriate cables.
3. Turn on the instrument, and allow the unit to warm up for two hours for rated accuracy.
4. Turn on the computer and start Test Script Builder (TSB). Once the program has started, open a session by connecting to the instrument. For details on how to use TSB, see the Series 2600 Reference Manual.
5. You can simply copy and paste the code from Appendix A in this guide into the TSB script editing window (Program 3A, Diode Forward Characterization), manually enter the code from the appendix, or import the TSP file 'Diode_Fwd_Char.tsp' after downloading it to your PC.

   If your computer is currently connected to the Internet, you can click on this link to begin downloading: http://www.keithley.com/data?asset=50924.
6. Install a small-signal silicon diode such as a 1N914 or 1N4148 in the appropriate axial socket of the test fixture.
7. Now, we must send the code to the instrument. One method is simply to right-click in the open script window of TSB, and select 'Run as TSP file'. This will compile the code and place it in the volatile run-time memory of the instrument. To store the program in non-volatile memory, see the "TSP Programming Fundamentals" section of the Series 2600 Reference Manual.
8. Once the code has been placed in the instrument run-time memory, we can run it at any time simply by calling the function 'Diode_Fwd_Char()'. This can be done by typing the text '`Diode_Fwd_Char()`' after the active prompt in the Instrument Console line of TSB.
9. In the program 'Diode_Fwd_Char.tsp', the function `Diode_Fwd_Char(ilevel, start, stop, steps)` is created. The variable `ilevel` represents the current value applied to the device-under-test (DUT) both before and after the staircase sweep has been applied. The `start` variable represents the starting current value for the sweep, stop represents the end current value, and steps represents the number of steps in the sweep. If any values are left blank, the function will use the default value given to that variable, but you can specify what voltage is applied by simply sending a voltage that is in-range in the function call.
10. As an example, if you wanted to configure a test that would source 0mA before and after the sweep, with a sweep start value of 1mA, stop value of 10mA, and 10 steps, you would simply send `Diode_Fwd_Char(0, 0.001, 0.01, 10)` to the instrument.
11. The instrument will then source the programmed current staircase sweep and measure the respective voltage at each step. The measured and sourced values are then printed to the screen (if using TSB). To graph the results, simply copy and paste the data into a spreadsheet such as Microsoft Excel and chart.

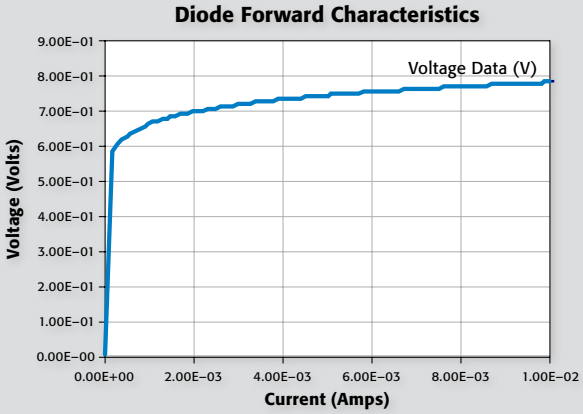

Figure 2-6. Program 3 results: Diode forward characteristics

2.5.4 Typical Program 3 Results

Figure 2-6 shows typical results obtained using Example Program 3. These results are for a 1N914 silicon diode.

2.5.5 Program 3 Description

At the start of the program, the instrument is reset to default conditions, the error queue, and data storage buffers are cleared. The following configuration is then applied before the data collection begins:

- Source I
- Local sense
- 10V compliance, autorange measure
- `Ilevel`: 0A
- `start`: 0.001A
- `stop`: 0.01A
- `steps`: 10

The instrument then sources `ilevel`, dwells `l_delay` seconds, and begins the staircase sweep from `start` to `stop` in steps. At each current step, both the current and voltage are measured.

The instrument output is then turned off and the function `Print _ Data()` is run to print the data to the TSB window. To graph the results, simply copy and paste the data into a spreadsheet such as Microsoft Excel and chart.

2.5.6 Using Log Sweeps

With some devices, it may be desirable to use a log sweep because of the wide range of currents necessary to perform the test. Program 3B performs a log sweep of the diode current.

If your computer is currently connected to the Internet, you can click on this link to begin downloading 'Diode_Fwd_Char_Log.tsp': http://www.keithley.com/data?asset=50923.

Note that the start and stop currents are programmed just as before, although with a much wider range than would be practical with a linear sweep. With log sweep, however, the `points` parameter, which defines the number of points per decade, replaces the steps parameter that is used with the linear sweep.

To run the Log sweep, we must send the code to the instrument. One method is simply to right-click in the open script window of TSB, and select 'Run as TSP file'. This will compile the code and place it in the volatile run-time memory of the instrument. To store the program in non-volatile memory, see the "TSP Programming Fundamentals" section of the Series 2600 Reference Manual.

Once the code has been placed in the instrument run-time memory, we can run it at any time simply by calling the function 'Diode_Fwd_Char_Log()'. This can be done by typing the text '`Diode _ Fwd _ Char _ Log()`' after the active prompt in the Instrument Console line of TSB.

2.5.7 Using Pulsed Sweeps

In some cases, it may be desirable to use a pulsed sweep to avoid device self-heating that could affect the test results. Program 3C performs a staircase pulse sweep. In this program, there are two additional variables ton and toff, where ton is the source on duration and toff is the source off time for the pulse. During the toff portions of the sweep, the source value is returned to the ilevel bias value.

If your computer is currently connected to the Internet, you can click on this link to begin downloading 'Diode_Fwd_Char_Pulse.tsp': http://www.keithley.com/data?asset=50922.

Section 3
Bipolar Transistor Tests

3.1 Introduction

Bipolar transistor tests discussed in this section include: tests to generate common-emitter characteristic curves, Gummel plot, current gain, and transistor leakage tests.

3.2 Instrument Connections

Figure 3-1 shows the instrument connections for the bipolar transistor tests outlined in this section. Two Source-Measure channels are required for the tests (except for the leakage current test, which requires only one Source-Measure channel).

Keithley Model 2600-BAN cables or Model 7078-TRX-3 low noise triaxial cables are recommended to make instrument-to-test fixture connections. In addition, the safety interlock connecting cables must be connected to the instrument and fixture if using instrumentation capable of producing greater than 42V.

WARNING

Lethal voltages may be exposed when working with test fixtures. To avoid a possible shock hazard, the fixture must be equipped with a working safety interlock circuit. For more information on the interlock of the Series 2600, please see the Series 2600 Reference Manual.

NOTES

Remote sensing connections are recommended for optimum accuracy. See paragraph 1.2.2 for details.

If measurement noise is a problem, or for critical, low level applications, use shielded cable for all signal connections.

3.3 Common-Emitter Characteristics

Common-emitter characteristics are probably the most familiar type of curves generated for bipolar transistors. Test data used to generate these curves is obtained by sweeping the base current (I_B) across the desired range of values at specific increments. At each be current value, the collector-emitter voltage (V_{CE}) is swept across the desired range, again at specific increments. At each V_{CE} value, the collector current (I_C) is measured.

Once the data is collected, it is conveniently printed (if using TSB). You can then use the copy-and-paste method to place the data into a spreadsheet program such as Microsoft Excel. Common

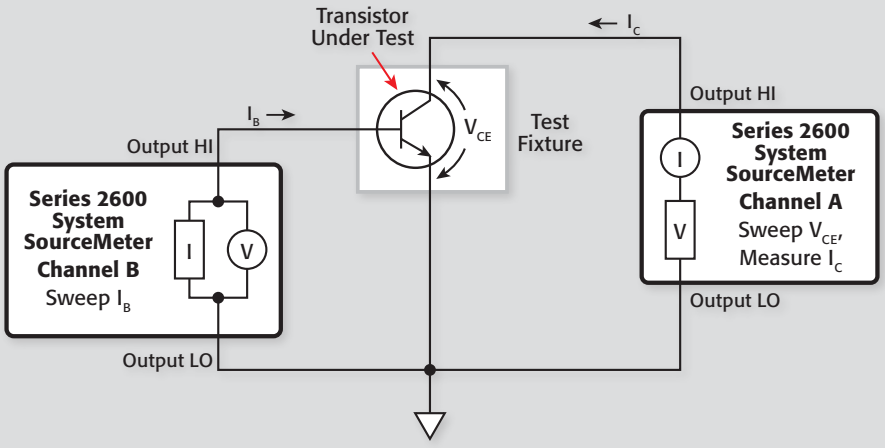

Figure 3-1. Test configuration for common-emitter tests

plotting styles include graphing I_C vs. V_{CE} for each value of I_B. The result is a family of curves that shows how I_C varies with V_{CE} at specific I_B values.

3.3.1 Test Configuration

Figure 3-1 shows the test configuration for the common-emitter characteristic tests. Many of the transistor tests performed require two Source-Measure Units (SMUs). The Series 2600 System SourceMeter instruments have dual-channel members such as the Model 2602, 2612, and 2636. This offers a convenient transistor test system all in one box. The tests can be run using two single-channel instruments, but the code will have to be modified to do so.

In this test, SMUB sweeps I_B across the desired range, and SMUA sweeps V_{CE} and measures I_C. Note that an NPN transistor is shown as part of the test configuration. A small-signal NPN transistor with an approximate current gain of 500 (such as a 2N5089) is recommended for use with the test program below. Other similar transistors such as a 2N3904 may also be used, but the program may require modification.

3.3.2 Measurement Considerations

A fixed delay period of 100ms, which is included in the program, may not be sufficient for testing some devices. Also, it maybe necessary to change the programmed current values to optimize the tests for a particular device.

3.3.3 Example Program 4: Common-Emitter Characteristics

Program 4 can be used to run common-emitter characteristic tests on small-signal NPN transistors. In order to run the program, follow these steps:

1. With the power off, connect a dual-channel System SourceMeter instrument to the computer's IEEE-488 interface.
2. Connect the test fixture to both units using appropriate cables (see **Figure 3-1**).
3. Turn on the instrument and allow the unit to warm up for two hours for rated accuracy.
4. Turn on the computer and start Test Script Builder (TSB). Once the program has started, open a session by connecting to the instrument. For details on how to use TSB, see the Series 2600 Reference Manual.
5. You can simply copy and paste the code from Appendix A in this guide into the TSB script editing window (Program 4), manually enter the code from the appendix, or import the TSP file '*BJT_Comm_Emit.tsp*' after downloading it to your PC.

   If your computer is currently connected to the Internet, you can click on this link to begin downloading: http://www.keithley.com/data?asset=50930.
6. Install an NPN transistor such as a 2N5089 in the appropriate transistor socket of the test fixture.
7. Now, we must send the code to the instrument. The simplest method is to right-click in the open script window of TSB, and select '*Run as TSP file*'. This will compile the code and place it in the volatile run-time memory of the instrument. To store the program in non-volatile memory, see the "TSP Programming Fundamentals" section of the Series 2600 Reference Manual.
8. Once the code has been placed in the instrument run-time memory, we can run it at any time simply by calling the function '`BJT _ Comm _ Emit()`'. This can be done by typing the text '`BJT _ Comm _ Emit()`' after the active prompt in the Instrument Console line of TSB.
9. In the program '*BJT_Comm_Emit.tsp*', the function `BJT _ Comm _ Emit(istart, istop, isteps, vstart, vstop, vsteps)` is created.
   - `istart` represents the sweep start current value on the base of the transistor
   - `istop` represents the sweep stop value
   - `isteps` is the number of steps in the base current sweep
   - `vstart` represents the sweep start voltage value on the collector-emitter of the transistor
   - `vstop` represents the sweep stop voltage value
   - `vsteps` is the number of steps in the base current sweep

   If these values are left blank, the function will use the default values given to the variables, but you can specify each variable value by simply sending a number that is in-range in the function call. As an example, if you wanted to have the base current swept from 1μA to 100μA in 10 steps, and the collector-emitter voltage (V_{CEO}) to be swept from 0 to 10V in 1V steps, you would send `BJT _ Comm _ Emit(1E-6, 100E-6, 10, 0, 10, 10)` to the instrument.
10. The instrument will then source the programmed start current on the base, sweep the voltage on the collector-emitter, and measure the respective current through the collector-emitter. The base current will be incremented and the collector-emitter sweep will take place again. After the final base source value and associated collector-emitter sweep, the collector-emitter voltage (V_{CE}), measured collector-emitter current (I_{CE}), and base current (I_B) values will then be displayed in the Instrument Console window of TSB.

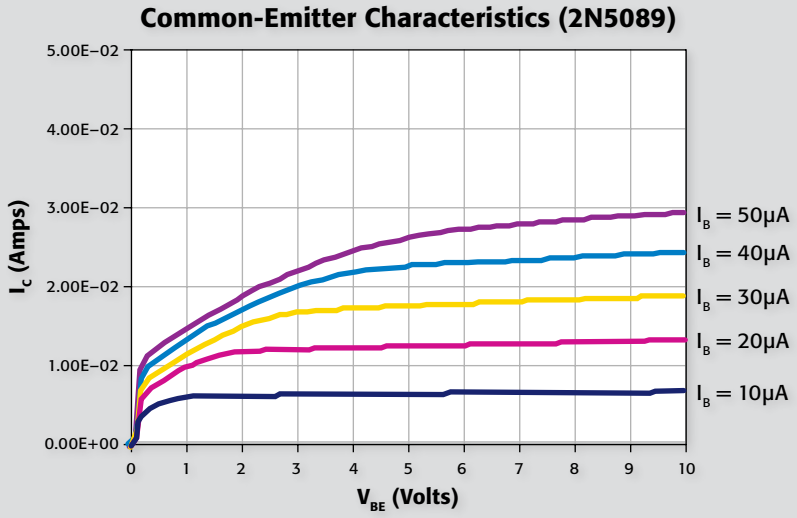

Figure 3-2. Program 4 results: Common-emitter characteristics

3.3.4 Typical Program 4 Results

Figure 3-2 shows typical results generated by Example Program 4. A 2N5089 NPN transistor was used to generate these test results.

3.3.5 Program 4 Description

For the following program description, refer to the program listing below.

- Source I
- IV compliance, 1.1V range
- Local sense
- `istart` current: 10M
- `istop` current: 50µA
- `isteps`: 5

Following SMUB setup, SMUA, which sweeps VCE and measures IC, is programmed as follows:

- Source V
- Local sensing
- 100mA compliance, autorange measure
- 1 NPLC Line cycle integration (to reduce noise)
- `vstart`: 0V
- `vstop`: 10V
- `vsteps`: 100

Once the two units are configured, the SMUB sources `istart`, SMUA sources `vstart`, and the voltage (V_{CE}) and current (I_{CE}) for SMUA are measured. The source value for SMUA is then incremented by `l _ vstep`, and the sweep is continued until the source value reaches `vstop`. Then, SMUB is incremented by `l _ istep` and SMUA begins another sweep from `vstart` to `vstop` in `vsteps`. This nested sweeping process continues until SMUB reaches `istop`.

The instrument output is then turned off and the function `Print _ Data()` is run to print the data to the TSB window. To graph the results, simply copy and paste the data into a spreadsheet such as Microsoft Excel and chart.

3.4 Gummel Plot

A Gummel plot is often used to determine current gain variations of a transistor. Data for a Gummel plot is obtained by sweeping the base-emitter voltage (V_{BE}) across the desired range of values at specific increments. At each V_{BE} value, both the base current (I_B) and collector current (I_C) are measured.

Once the data are taken, the data for I_B, I_C, and V_{BE} is returned to the screen. If using TSB, a plot can be generated using the "copy-and-paste" method in a spreadsheet program such as Microsoft Excel. Because of the large differences in magnitude between I_B and I_C, the Y axis is usually plotted logarithmically.

3.4.1 Test Configuration

Figure 3-3 shows the test configuration for Gummel plot tests. SMUB is used to sweep V_{BE} across the desired range, and it also

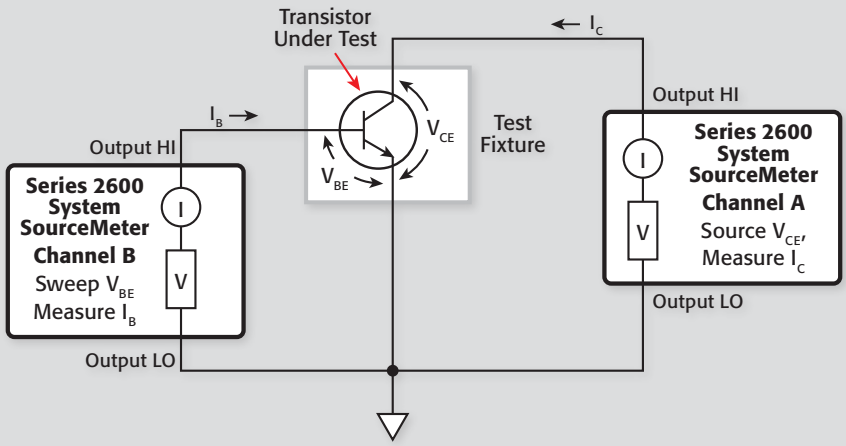

Figure 3-3. Gummel plot test configuration

measures I_B. SMUA sets V_{CE} to the desired fixed value, and it also measures I_C.

Due to the low current measurements associated with this type of testing, the Keithley Model 2636 System SourceMeter instrument is recommended. Its low level current measurement capabilities and dual-channel configuration are ideal for producing high quality Gummel plots of transistors.

3.4.2 Measurement Considerations

As written, the range of V_{BE} test values is from 0V to 0.7V in 0.01V increments. It may be necessary, however, to change these limits for best results with your particular device. Low currents will be measured so take the usual low current precautions.

3.4.3 Example Program 5: Gummel Plot

Program 5 demonstrates the basic programming techniques for generating a Gummel plot. Follow these steps to run this program:

1. With the power off, connect a dual-channel System SourceMeter instrument to the computer's IEEE-488 interface.
2. Connect the test fixture to both units using appropriate cables.
3. Turn on the instrument and allow the unit to warm up for two hours for rated accuracy.
4. Turn on the computer and start Test Script Builder (TSB). Once the program has started, open a session by connecting to the instrument. For details on how to use TSB, see the Series 2600 Reference Manual.
5. You can simply copy and paste the code from Appendix A in this guide into the TSB script editing window (Program 5), manually enter the code from the appendix, or import the TSP file 'Gummel.tsp' after downloading it to your PC.

   If your computer is currently connected to the Internet, you can click on this link to begin downloading: http://www.keithley.com/data?asset=50918
6. Install an NPN transistor such as a 2N5089 in the appropriate transistor socket of the test fixture.
7. Now, we must send the code to the instrument. The simplest method is to right-click in the open script window of TSB, and select 'Run as TSP file'. This will compile the code and place it in the volatile run-time memory of the instrument. To store the program in non-volatile memory, see the "TSP Programming Fundamentals" section of the Series 2600 Reference Manual.
8. Once the code has been placed in the instrument run-time memory, we can run it at any time simply by calling the function 'Gummel()'. This can be done by typing the text '`Gummel()`' after the active prompt in the Instrument Console line of TSB.
9. In the program 'Gummel.tsp', the function `Gummel (vbestart, vbestop, vbesteps, vcebias)` is created.
   - `vbestart` represents the sweep start voltage value on the base of the transistor
   - `vbestop` represents the sweep stop value
   - `vbesteps` is the number of steps in the base voltage sweep

- `vcebias` represents the voltage bias value on the collector-emitter of the transistor

If these values are left blank, the function will use the default values given to the variables, but you can specify each variable value by simply sending a number that is in-range in the function call. As an example, if you wanted to have the base voltage swept from 0.1V to 1V in 10 steps, and the collector-emitter voltage (V_{CE}) to be biased 5V, you would send `Gummel(0.1, 1, 10, 5)` to the instrument.

10. The base-emitter voltage will be swept between 0V and 0.7V in 0.01V increments, and both I_B and I_C will be measured at each V_{BE} value. Note that a fixed collector-emitter voltage of 10V is used for the tests.
11. Once the sweep has been completed, the data (I_B, I_C, and V_{BE}) will be presented in the Instrument Console window of TSB.

3.4.4 Typical Program 5 Results

Figure 3-4 displays a typical Gummel plot as generated by Example Program 5. Again, the transistor used for this example was a 2N5089 NPN silicon transistor.

3.4.5 Program 5 Description

SMUB, which sweeps V_{BE} and measures I_B, is set up as follows:

- Source V
- 1mA compliance, autorange measure
- Local sensing
- 1 NPLC Line cycle integration
- `vbestart`: 0V
- `vbestop`: 0.7V
- `vbesteps`: 70

SMUA, which sources V_{CE} and measures I_C, is programmed in the following manner:

- Source V
- Local sensing
- 100mA compliance, autorange measure
- 1 NPLC Line cycle integration
- Constant sweep (number of points programmed to 71), V_{CE} = 10V
- `vcebias`: 10V

Following unit setup, both unit triggers are armed, and the instruments are placed into the operate mode (lines 320 and 330).

Once triggered, SMUB sets V_{BE} to the required value, and SMUA then sets V_{CE} and measures I_C at I_B. At the end of its measurement, SMUB increments V_{BE} and the cycle repeats until V_{BE} reaches the value set for `vbestop`.

During the test, V_{BE}, I_B, and I_C are measured. Once the test has completed, the data is written to the Instrument Console of TSB and can be graphed in a spreadsheet program using the "copy-and-paste" method of data transfer.

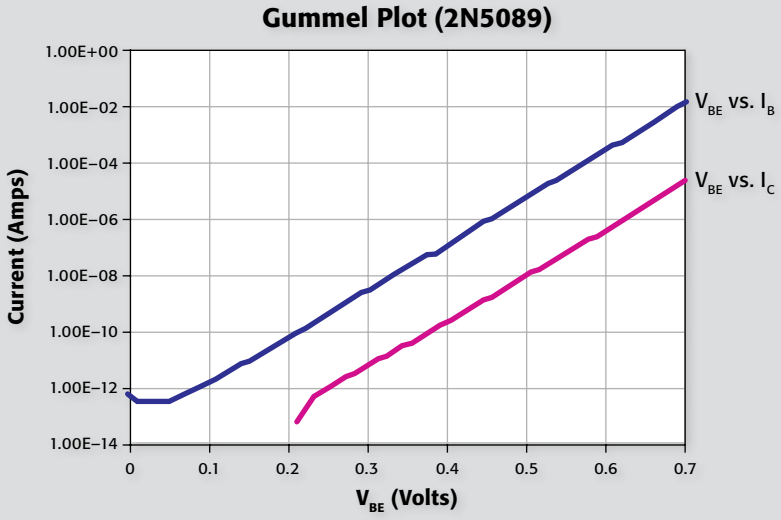

Figure 3-4. Program 5 results: Gummel plot

3.5 Current Gain

The following paragraphs discuss two methods for determining DC current gain, as well as ways to measure AC current gain.

3.5.1 Gain Calculations

The common-emitter DC current gain of a bipolar transistor is simply the ratio of the DC collector current to the DC base current of the device. The DC current gain is calculated as follows:

$$ß = \frac{I_C}{I_B}$$

where: ß = current gain

I_C = DC collector current

I_B = DC base current

Often, the differential or AC current gain is used instead of the DC value because it more closely approximates the performance of the transistor under small-signal AC conditions. In order to determine the differential current gain, two values of collector current (I_{C1} and I_{C2}) at two different base currents (I_{B1} and I_{B2}) are measured. The current gain is then calculated as follows:

$$ß_{ac} = \frac{\Delta I_C}{\Delta I_B}$$

where: ßa = AC current gain

$\Delta I_C = I_{C2} - I_{C1}$

$\Delta I_B = I_{B2} - I_{B1}$

Tests for both DC and AC current gain are generally done at one specific value of V_{CE}. AC current gain tests should be performed with as small a ΔI_B as possible so that the device remains in the linear region of the curve.

3.5.2 Test Configuration for Search Method

Figure 3-5 shows the test configuration for the search method of DC current gain tests and AC gain tests. A dual-channel System SourceMeter instrument is required for the test. SMUB is used to supply I_{B1} and I_{B2}. SMUA sources V_{CE}, and it also measures the collector currents I_{C1} and I_{C2}.

3.5.3 Measurement Considerations

When entering the test base currents, take care not to enter values that will saturate the device. The approximate base current value can be determined by dividing the desired collector current value by the typical current gain for the transistor being tested.

3.5.4 Example Program 6A: DC Current Gain Using Search Method

Use Program 6A to perform DC current gain tests on bipolar transistors. Proceed as follows:

1. With the power off, connect a dual-channel System SourceMeter instrument to the computer's IEEE-488 interface.
2. Connect the test fixture to both units using appropriate cables.
3. Turn on the System SourceMeter instrument and allow the unit to warm up for two hours for rated accuracy.

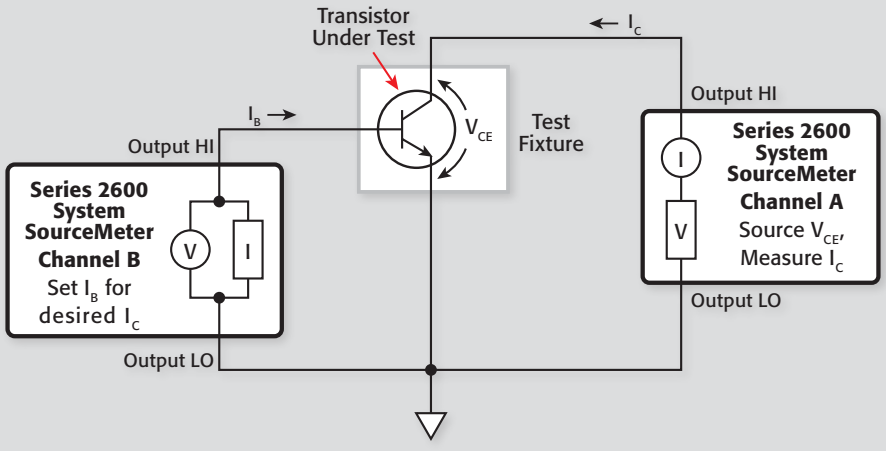

Figure 3-5. Test configuration for current gain tests using search method

4. Turn on the computer and start Test Script Builder (TSB). Once the program has started, open a session by connecting to the instrument. For details on how to use TSB, see the Series 2600 Reference Manual.
5. You can simply copy and paste the code from Appendix A in this guide into the TSB script editing window (Program 6A), manually enter the code from the appendix, or import the TSP file '*DC_Gain_Search.tsp*' after downloading it to your PC.

   If your computer is currently connected to the Internet, you can click on this link to begin downloading: http://www.keithley.com/data?asset=50925
6. Install an NPN transistor such as a 2N5089 in the appropriate transistor socket of the test fixture.
7. Now, we must send the code to the instrument. The simplest method is to right-click in the open script window of TSB, and select '*Run as TSP file*'. This will compile the code and place it in the volatile run-time memory of the instrument. To store the program in non-volatile memory, see the "TSP Programming Fundamentals" section of the Series 2600 Reference Manual.
8. Once the code has been placed in the instrument run-time memory, we can run it at any time simply by calling the function 'DC_Gain_Search()'. This can be done by typing the text '`DC _ Gain _ Search()`' after the active prompt in the Instrument Console line of TSB.
9. In the program '*DC_Gain_Search.tsp*', the function `DC _ Gain _ Search(vcesource, lowib, highib, targetic)` is created.
   - `vcesource` represents the voltage value on the collector-emitter of the transistor
   - `lowib` represents the base current low limit for the search algorithm
   - `highib` represents the base current high limit for the search algorithm
   - `targetic` represents the target collector current for the search algorithm
10. If these values are left blank, the function will use the default values given to the variables, but you can specify each variable value by simply sending a number that is in-range in the function call. As an example, if you wanted the collector-emitter voltage (V_{CE}) to be 2.5V, the base current low value at 10nA, the base current high value at 100nA, and the target collector current to be 10µA, you would send `DC _ Gain _ Search(2.5,10E-9, 100E-9, 10E-6)` to the instrument.
11. The sources will be enabled, and the collector current of the device will be measured. The program will perform an iterative search to determine the closest match to the target I_C (within ±5%). The DC current gain of the device at specific I_B and I_C values will then be displayed on the computer CRT. If the search is unsuccessful, the program will print "Iteration Level Reached". This is an error indicating that the search reached its limit. Recheck the connections, DUT, and variable values to make sure they are appropriate for the device.
12. Once the sweep has been completed, the data (I_B, I_C, and ß) will be presented in the Instrument Console window of TSB.

3.5.5 Typical Program 6A Results

A typical current gain for a 2N5089 would be about 500. Note, however, that the current gain of the device could be as low as 300 or as high as 800.

3.5.6 Program 6A Description

Initially, the iteration variables are defined and the instrument is returned to default conditions. SMUB, which sources I_B, is set up as follows:

- Source I
- IV compliance, 1.1V range
- Local sense

SMUA, which sources V_{CE} and measures I_C, is configured as follows:

- Source V
- Local sense
- 100mA compliance, autorange measure

Once the SMU channels have been configured, the sources values are programmed to 0 and the outputs are enabled. The base current (I_B) is sourced and the program enters into the binary search algorithm for the target I_C by varying the V_{CE} value, measuring the I_C, comparing it to the target I_C, and adjusting the V_{CE} value, if necessary. The iteration counter is incremented each cycle through the algorithm. If the number of iterations has been exceeded, a message to that effect is displayed, and the program halts.

Assuming that the number of iterations has not been exceeded, the DC current gain is calculated and displayed in the Instrument Console window of the TSB.

3.5.7 Modifying Program 6A

For demonstration purposes, the I_C target match tolerance is set to ±5%. You can, of course, change this tolerance as required. Similarly, the iteration limit is set to 20. Again, this value can be adjusted for greater or fewer iterations as necessary. Note that it

may be necessary to increase the number of iterations if the target range is reduced.

3.5.8 Configuration for Fast Current Gain Tests

Figure 3-6 shows the test configuration for an alternate method of current gain tests—one that is much faster than the search method discussed previously. SMUB is used to supply V_{CE}, and it also measures I_B. SMUA sources the emitter current (I_E) rather than the collector current (I_C). Because we are sourcing emitter current instead of collector current, the current gain calculations must be modified as follows:

$$ß = \frac{I_E - I_B}{I_B}$$

WARNING

When a System SourceMeter instrument is programmed for remote sensing, hazardous voltage may be present on the SENSE and OUTPUT terminals when the unit is in operation regardless of the programmed voltage or current. To avoid a possible shock hazard, always turn off all power before connecting or disconnecting cables to the Source-Measure Unit or the associated test fixture.

NOTE

Because of the connection convention used, I_E and V_{CE} must be programmed for opposite polarity than normal. With an NPN transistor, for example, both V_{CE} and I_E must be negative.

3.5.9 Example Program 6B: DC Current Gain Using Fast Method

Use Program 6B in Appendix A to demonstrate the fast method of measuring current gain of bipolar transistors. Proceed as follows:

1. With the power off, connect a dual-channel System SourceMeter instrument to the computer's IEEE-488 interface.
2. Connect the test fixture to both units using appropriate cables. Note that OUTPUT HI of SMUB is connected to the base of the DUT, and SENSE HI of SMUB is connected to the emitter.
3. Turn on the System SourceMeter instrument and allow the unit to warm up for two hours for rated accuracy.
4. Turn on the computer and start Test Script Builder (TSB). Once the program has started, open a session by connecting to the instrument. For details on how to use TSB, see the Series 2600 Reference Manual.
5. You can simply copy and paste the code from Appendix A in this guide into the TSB script editing window (Program 6B), manually enter the code from the appendix, or import the TSP file '*DC_Gain_Fast.tsp*' after downloading it to your PC.

   If your computer is currently connected to the Internet, you can click on this link to begin downloading: http://www.keithley.com/data?asset=50926
6. Install an NPN transistor such as a 2N5089 in the appropriate transistor socket of the test fixture.
7. Now, we must send the code to the instrument. The simplest method is to right-click in the open script window of TSB, and select '*Run as TSP file*'. This will compile the code and place it in the volatile run-time memory of the instrument.

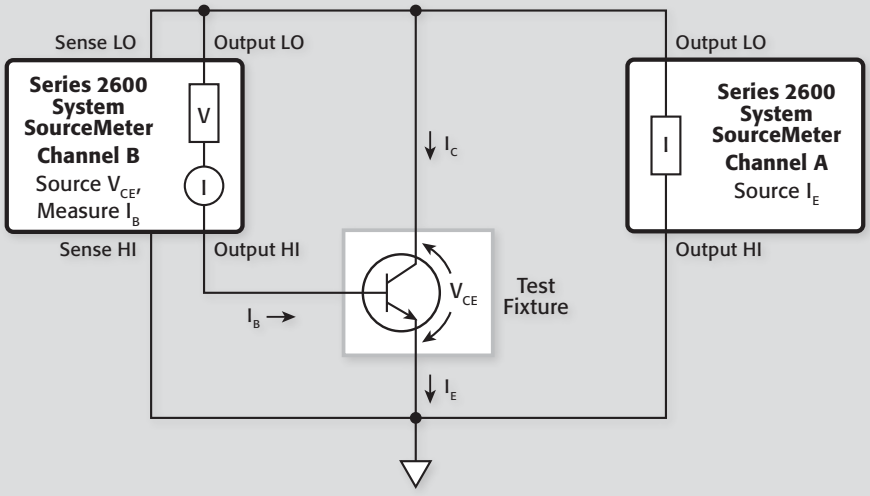

Figure 3-6. Test configuration for fast current gain tests

To store the program in non-volatile memory, see the "TSP Programming Fundamentals" section of the Series 2600 Reference Manual.

8. Once the code has been placed in the instrument run-time memory, we can run it at any time simply by calling the function 'DC_Gain_Search_Fast()'. This can be done by typing the text '`DC _ Gain _ Search _ Fast()`' after the active prompt in the Instrument Console line of TSB.
9. In the program '*DC_Gain_Search_Fast.tsp*', the function `DC _ Gain _ Search _ Fast(vcesource, istart, istop, isteps)` is created.
   - `vcesource` represents the voltage value on the collector-emitter of the transistor
   - `istart` represents the start value for the base current sweep
   - `istop` represents the stop value for the base current sweep
   - `isteps` represents the number of steps in the base current sweep

   If these values are left blank, the function will use the default values given to the variables, but you can specify each variable value by simply sending a number that is in-range in the function call. As an example, if you wanted to have the collector-emitter voltage (V_{CE}) be 2.5V, the base current sweep start value at 10nA, the base current sweep stop value at 100nA, and the number of steps to be 10, you would send `DC _ Gain _ Search _ Fast(2.5,10E-9, 100E-9, 10)` to the instrument.
10. The sources will be enabled, and the collector current of the device will be measured.
11. Once the sweep has been completed, the data (I_B, I_C, and ß) will be presented in the Instrument Console window of TSB. Note that the program reverses the polarity of the emitter currents in order to display true polarity.

3.5.10 Program 6B Description

Initially, both units are returned to default conditions. SMUB, which sources V_{CE} and measures I_B, is set up as follows:

- Source V
- 1mA compliance, autorange measure
- Remote sense
- `vcesource`: –10V

SMUA, which sources I_E, is configured as follows:

- Source I
- Local sense
- 11V compliance, autorange
- `istart`: –1mA
- `istop`: –10mA
- `isteps`: 10
- 10ms delay
- Staircase sweep mode

Both SMU outputs are then zeroed and enabled. Next, SMUB sources V_{CE} and SMUA begins the current sweep on the emitter current (I_E) from istart to istop in isteps. At each point in the sweep, SMUB measures the base current (I_B). Upon completion of the sweep, the current gain (ß) is calculated and the data (I_B, I_C, and ß) is printed to the Instrument Console of the TSB.

3.5.11 Example Program 7: AC Current Gain

NOTE

For the sake of simplicity, this program does not include the iterative search algorithm included in Program 6A. To test at a specific IC value, first use Program 6A to determine the base current at that target value, and enter I_B values slightly higher and lower when prompted to do so in Program 7.

1. With the power off, connect a dual-channel System SourceMeter instrument to the computer's IEEE-488 interface.
2. Connect the test fixture to both units using appropriate cables.
3. Turn on the instrument and allow the unit to warm up for two hours for rated accuracy.
4. Turn on the computer and start Test Script Builder (TSB). Once the program has started, open a session by connecting to the instrument. For details on how to use TSB, see the Series 2600 Reference Manual.
5. You can simply copy and paste the code from Appendix A in this guide into the TSB script editing window (Program 7), manually enter the code from the appendix, or import the TSP file '*AC_Gain_.tsp*' after downloading it to your PC.

   If your computer is currently connected to the Internet, you can click on this link to begin downloading: http://www.keithley.com/data?asset=50931.
6. Install a small-signal NPN silicon transistor such as a 2N5089 in the appropriate transistor socket of the test fixture.
7. Now, we must send the code to the instrument. The simplest method is to right-click in the open script window of TSB, and select '*Run as TSP file*'. This will compile the code and place it in the volatile run-time memory of the instrument.

To store the program in non-volatile memory, see the "TSP Programming Fundamentals" section of the Series 2600 Reference Manual.

8. Once the code has been placed in the instrument run-time memory, we can run it at any time simply by calling the function '`AC _ Gain()`'. This can be done by typing the text '`AC _ Gain()`' after the active prompt in the Instrument Console line of TSB.
9. In the program '*AC_Gain.tsp*', the function `AC _ Gain (vcesource, ib1, ib2)` is created.
   - `vcesource` represents the voltage value on the collector-emitter of the transistor
   - `ib1` represents the first value for the base current
   - `ib2` represents the second value for the base current

   If these values are left blank, the function will use the default values given to the variables, but you can specify each variable value by simply sending a number that is in-range in the function call. As an example, if you wanted to have the collector-emitter voltage (V_{CE}) be 2.5V, the base current initial value at 100nA, and the base current second value at 200nA you would send `AC _ Gain(2.5,100E-9, 200E-9)` to the instrument.

   Keep the two values as close together as possible so that the device remains in its linear operating region. A change in I_B of about 20% from one value to another would be a good starting point.
10. The sources will be zeroed and then enabled. The program will execute a two-point source and measure process.
11. Once the measurements have completed, the data (I_{B1}, I_{C1}, I_{B2}, I_{C2}, and ß) will be presented in the Instrument Console window of TSB.

3.5.13 Typical Program 7 Results

The differential current gain obtained for a given sample of a 2N5089 NPN transistor would typically be about the same as the DC current gain—about 500. Again, values could range from a low of 300 to a high of 800 or so.

3.5.14 Program 7 Description

After both units are returned to default conditions, SMUB is set up as follows:

- Source I
- IV compliance, 1.1V range
- Local sense

SMUA is configured as follows:

- Source V
- Local sense
- 100mA compliance

The collector-emitter voltage (V_{CE}) will then be set. Then, the base current will be set to the I_{B1} value and the collector current (I_{C1}) will be measured. Next, the base current will be set to the I_{B2} value and I_{C2} will be measured. The AC current gain of the device will then be calculated and printed to the Instrument Console window of TSB.

3.5.15 Modifying Program 7

As with the DC current gain, AC current gain is often tested at specific values of I_C. Again, a search algorithm similar to the one in Program 6A could be added to the program. Such an algorithm would allow you to enter the desired collector current values, and it would then perform an iterative search to determine automatically the two correct base current values that would result in the desired collector currents.

3.6 Transistor Leakage Current

Leakage currents, such as I_{CEO} (collector-base, emitter open) and I_{CEO} (collector-emitter, base open) can be tested using a single-channel System SourceMeter instrument. The following paragraphs discuss I_{CEO} tests and also include an example program for making such tests.

3.6.1 Test Configuration

Figure 3-7 shows the basic test configuration for performing I_{CEO} tests. The SMU sources the collector-emitter voltage (V_{CEO}) and the instrument also measures I_{CEO}. Often, V_{CEO} is swept across the desired range of values, and the resulting I_{CEO} values can be plotted against V_{CEO}, as is the case with the example program included in this section.

The base of the transistor should be left open. The same general circuit configuration can be used to measure I_{CEO}; connect the SMU between the collector and base, and leave the emitter open instead.

Breakdown tests can also be performed using the same I_{CEO} circuit setup. In this case, the SMU is used to source I and measured the breakdown voltage (V) in order to control device power at breakdown better.

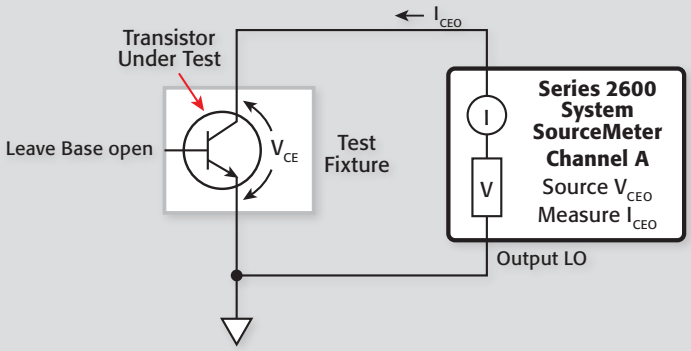

Figure 3-7. Configuration for I_{CEO} tests

3.6.2 Example Program 8: I_{CEO} Test

Use Program 8 to run I_{CEO} tests on bipolar transistors. Follow these steps to run the program:

1. With the power off, connect a dual-channel System SourceMeter instrument to the computer's IEEE-488 interface.
2. Connect the test fixture to both units using appropriate cables.
3. Turn on the instrument and allow the unit to warm up for two hours for rated accuracy.
4. Turn on the computer and start Test Script Builder (TSB). Once the program has started, open a session by connecting to the instrument. For details on how to use TSB, see the Series 2600 Reference Manual.
5. You can simply copy and paste the code from Appendix A in this guide into the TSB script editing window (Program 8), manually enter the code from the appendix, or import the TSP file 'Iceo.tsp' after downloading it to your PC.

   If your computer is currently connected to the Internet, you can click on this link to begin downloading: http://www.keithley.com/data?asset=50917.
6. Install a small-signal NPN silicon transistor such as a 2N3904 in the appropriate transistor socket of the test fixture.
7. Now, we must send the code to the instrument. The simplest method is to right-click in the open script window of TSB, and select '*Run as TSP file*'. This will compile the code and place it in the volatile run-time memory of the instrument. To store the program in non-volatile memory, see the "TSP Programming Fundamentals" section of the Series 2600 Reference Manual.
8. Once the code has been placed in the instrument run-time memory, we can run it at any time simply by calling the function '`Iceo()`'. This can be done by typing the text '`Iceo()`' after the active prompt in the Instrument Console line of TSB.
9. In the program '*Iceo.tsp*', the function `Iceo(vstart, vstop, vsteps)` is created.
   - vstart represents the initial voltage value in the V_{CE} sweep
   - vstop represents the final voltage value in the V_{CE} sweep
   - vsteps represents the number of steps in the sweep

   If these values are left blank, the function will use the default values given to the variables, but you can specify each variable value by simply sending a number that is in-range in the function call. As an example, if you wanted to have the start voltage be 1V, the stop value be 11V, and the number of steps be 20, you would send `Iceo(1, 11, 20)` to the instrument.
10. The sources will be zeroed and then enabled. The program will execute a voltage sweep on the collector-emitter and measure the collector-emitter current (I_{CEO}) at each point.
11. Once the measurements have completed, the data (V_{CE} and I_{CE}) will be presented in the Instrument Console window of TSB.

3.6.3 Typical Program 8 Results

Figure 3-8 shows an example I_{CEO} vs. V_{CEO} plot generated by Program 8. The device used for this example was a 2N3904 NPN transistor.

3.6.4 Program 8 Description

The instrument is returned to default conditions. SMUA, which sweeps V_{CEO} and measures I_{CEO}, is set up as follows:

- Source V
- Local sense
- 10mA compliance, autorange measure
- 1 NPLC Line cycle integration

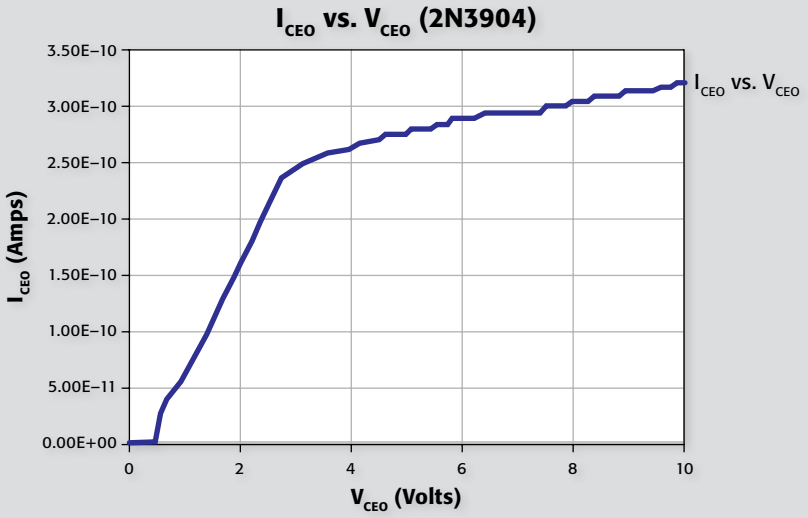

Figure 3-8. Program 8 results: I_{CEO} vs. V_{CEO}

- `vstart`: 0V
- `vstop`: 10V
- `vsteps`: 100

After setup, the output is zeroed and enabled. A linear voltage sweep from the start to the stop value is performed. At each step, the collector-emitter current (I_{CEO}) is measured.

Upon sweep completion, the output is disabled and the data is written to the Instrument Console window of TSB.

3.6.5 Modifying Program 8

For different sweep values, simply modify the `vstart`, `vstop`, and `vstep` values and source range parameter as appropriate.

In order to speed up the test procedure, you may wish to use a faster integration period. Simply change the `l _ nplc` value. Note, however, that changing this parameter may result in unacceptable reading noise.

Section 4
FET Tests

4.1 Introduction

FET tests discussed in this section include tests to generate common-source characteristic curves, and transconductance tests. Example programs for each of these applications are also included.

4.2 Instrument Connections

Two SMU channels are required for the tests and a dual-channel instrument from the Series 2600 System SourceMeter line is recommended. A test fixture with safety interlock is recommended for connections to the FET under test.

For general-purpose measurements with most of the Series 2600 instruments, Model 2600-BAN cables are recommended. For low current tests (<1mA) or when using a low current instrument like the Model 2636, Model 7078-TRX-3 triax cables are recommended to make instrument-to-test fixture connections.

WARNING

Lethal voltages may be exposed when the test fixture lid is open. To avoid a possible shock hazard, a safety interlock circuit must be connected before use. Connect the fixture screw to safety earth ground using #18 AWG minimum wire before use. Turn off all power before connecting or disconnecting wires or cables

NOTES

Remote sensing connections are recommended for optimum accuracy. See paragraph 1.2.2 for details.

If measurement noise is a problem, or for critical, low level applications, use shielded cable for all signal connections.

4.3 Common-Source Characteristics

One of the more common FET tests involving family of curves is common-source characteristics. Such tests are very similar to the common-emitter characteristic tests outlined earlier except, of course, for the fact that an FET rather than a bipolar transistor is involved.

Test data for common-source characteristics are obtained by sweeping the gate-source voltage (V_{GS}) across the desired range of values at specific increments. At each V_{GS} value, the drain-source voltage (V_{DS}) is swept through the required range, once again at the desired increments. At each V_{DS} value, the drain current (I_D) is measured. Plots can then be made from this data to show I_D vs. V_{DS} with one curve for each value of V_{GS}.

4.3.1 Test Configuration

Figure 4-1 shows the test configuration for the common-source tests. SMUB sweeps V_{GS}, while SMUA sweeps V_{DS}, and the instrument also measures I_D. For this programming example, a small-signal, N-channel FET such as a SD210 is recommended.

4.3.2 Example Program 9: Common-Source Characteristics

Program 9 outlines general programming techniques for measuring common-source characteristics. Follow these steps to use this program:

1. With the power off, connect a dual-channel System SourceMeter instrument to the computer's IEEE-488 interface.
2. Connect the test fixture to both units using appropriate cables.
3. Turn on the instrument and allow the unit to warm up for two hours for rated accuracy.
4. Turn on the computer and start Test Script Builder (TSB). Once the program has started, open a session by connecting to the instrument. For details on how to use TSB, see the Series 2600 Reference Manual.
5. You can simply copy and paste the code from Appendix A in this guide into the TSB script editing window (Program 9), manually enter the code from the appendix, or import the TSP file '*FET_Comm_Source.tsp*' after downloading it to your PC.

   If your computer is currently connected to the Internet, you can click on this link to begin downloading: http://www.keithley.com/data?asset=50921.

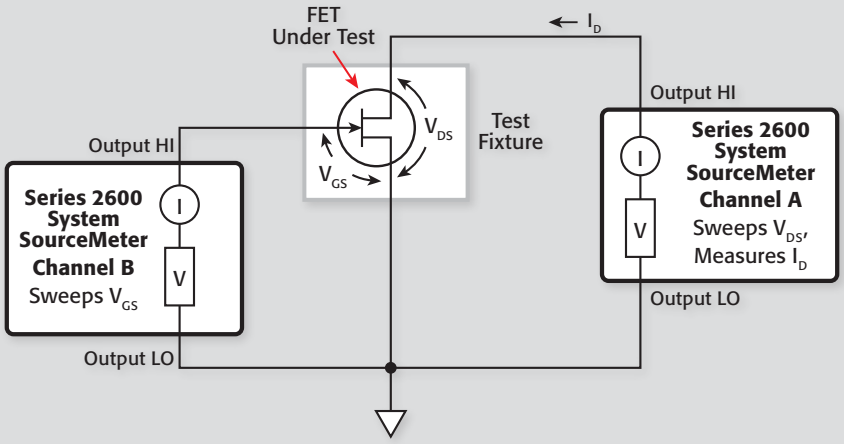

Figure 4-1. Test configuration for common-source tests

6. Install an N-channel FET such as an SD210 in the appropriate transistor socket of the test fixture.
7. Now, we must send the code to the instrument. The simplest method is to right-click in the open script window of TSB, and select '*Run as TSP file*'. This will compile the code and place it in the volatile run-time memory of the instrument. To store the program in non-volatile memory, see the "TSP Programming Fundamentals" section of the Series 2600 Reference Manual.
8. Once the code has been placed in the instrument run-time memory, we can run it at any time simply by calling the function '`FET _ Comm _ Source()`'. This can be done by typing the text '`FET _ Comm _ Source()`' after the active prompt in the Instrument Console line of TSB.
9. In the program '*FET_Comm_Source.tsp*', the function `FET _ Comm _ Source(vgsstart, vgsstop, vgssteps, vdsstart, vdsstop, vdssteps)` is created.
   - `vgsstart` represents the initial voltage value in the gate-source V_{GS} sweep
   - `vgsstop` represents the final voltage value in the gate-source V_{GS} sweep
   - `vgssteps` represents the number of steps in the sweep
   - `vdsstart` represents the initial voltage value in the drain-source V_{DS} sweep
   - `vdsstop` represents the final voltage value in the drain-source V_{DS} sweep
   - `vdssteps` represents the number of steps in the sweep

   If these values are left blank, the function will use the default values given to the variables, but you can specify each variable value by simply sending a number that is in-range in the function call. As an example, if you wanted to have the start voltages for V_{GS} and V_{DS} sweeps be 1V, the stop value be 11V, and the number of steps be 20, you would send `FET _ Comm _ Source(1, 11, 20, 1, 11, 20)` to the instrument.
10. The sources will be zeroed and then enabled. The program will execute a sweep of V_{GS} values between 0V and 10V using 2V steps. At each V_{GS} step, V_{DS} will be stepped between 0V and 10V at 0.1V increments. At each increment, I_D will be measured.
11. Once the measurements have been completed, the data (V_{GS}, V_{DS}, and I_{DS}) will be presented in the Instrument Console window of TSB.

4.3.3 Typical Program 9 Results

Figure 4-2 shows a typical plot generated by example Program 9. A 2N4392 N-channel JFET was used to generate these curves.

4.3.4 Program 9 Description

The unit is returned to default conditions. Next, SMUB, which sweeps V_{GS}, is programmed as follows:

- Source V
- 1mA compliance, 1mA range
- Local sense
- `vgsstart`: 0V
- `vgsstop`: 10V
- `vgssteps`: 5

SMUA, which sweeps V_{DS} and measures I_D, is configured as follows:

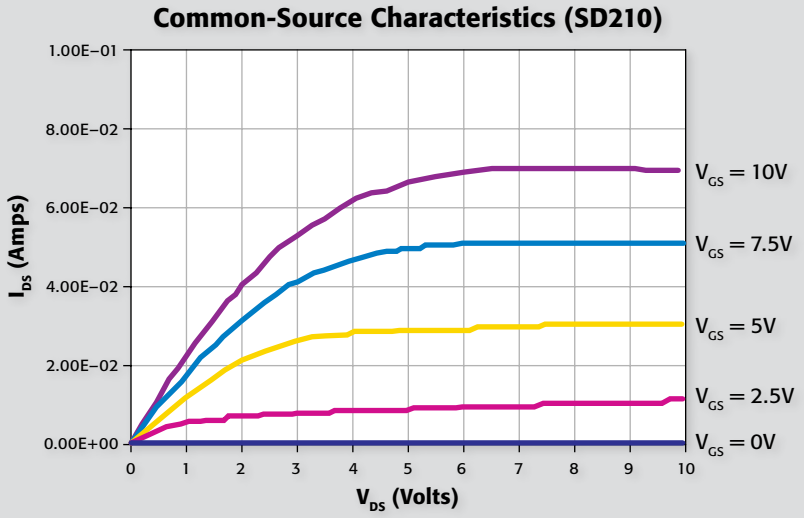

Figure 4-2. Program 9 results: Common-source characteristics

- Source V
- Local sensing
- 100mA compliance, autorange measure
- `vdsstart`: 0V
- `vdsstop`: 10V
- `vdssteps`: 100
- 1 NPLC Line cycle integration

Following setup of both units, the outputs are zeroed and enabled. The first gate-source bias (V_{GS}) source value is applied and the drain-source voltage (V_{DS}) sweep is started. At each point in the V_{DS} sweep, the drain current (I_D) is measured. When the final V_{DS} value is reached, the drain-source voltage is returned to 0V, the gate-source voltage (V_{GS}) is incremented, and the V_{DS} sweep begins again.

Upon reaching the final V_{DS} value, the outputs are zeroed, disabled, and the data (V_{GS}, V_{DS}, and I_D) is printed to the Instrument Console Window of TSB, where it can be copied and pasted to a spreadsheet for graphing.

4.3.5 Modifying Program 9

For other V_{GS} values, simply modify the `vgsstart`, `vgsstop`, and `vgssteps` variables as required.

Similarly, V_{DS} can be swept over a different range by changing the vdsstart, vdsstop, and vdsstep variables to the desired values.

4.4 Transconductance Tests

The forward transconductance (g_{fs}) of an FET is usually measured at a specific frequency (for example, 1kHz). Such a test can be simulated with DC values by using as small an incremental change in DC parameters as possible. For example, assume that we source two gate-source voltages, V_{GS1} and V_{GS2}, and measure two resulting drain currents, I_{D1} and I_{D2}. The forward transconductance can then be approximated as follows:

$$g_{fs} = \frac{\Delta_{ID}}{\Delta V_{GS}}$$

where: g_{fs} = forward transconductance (S)

$\Delta_{ID} = I_{D2} - I_{D1}$

$\Delta V_{GS} = V_{GS2} - V_{GS1}$

Two common plots involving g_{fs} include g_{fs} vs. V_{GS} and g_{fs} vs. I_D. The programming examples included in this section demonstrate how to generate g_{fs} vs. V_{GS} and g_{fs} vs. I_D plots.

4.4.1 Test Configuration

Figure 4-3 shows the general test configuration for transconductance tests. SMUB sweeps V_{GS}, while SMUA sources V_{DS} and also measures I_D. g_{fs} values are computed from incremental changes in I_D and V_{DS}. Note that an N-channel FET such as a SD210 is recommended for use with the example programs that follow.

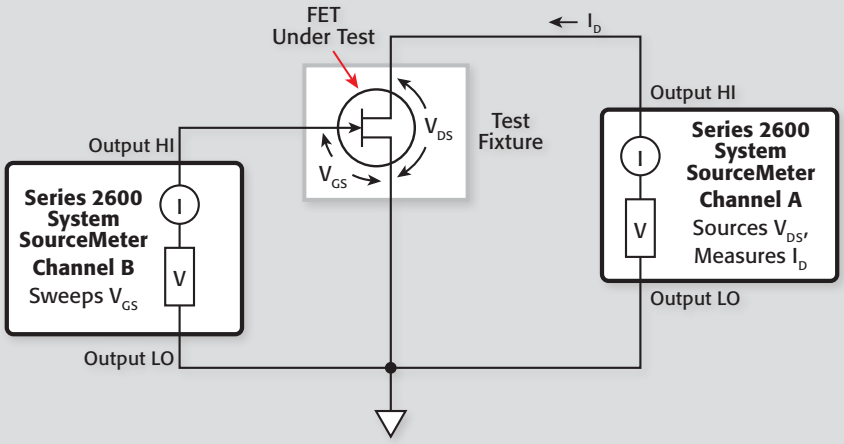

Figure 4-3. Configuration for transductance tests

4.4.2 Example Program 10: Transconductance vs. Gate Voltage Test

Use Program 10 to generate a typical g_{fs} vs. V_{GS} plot as well as a g_{fs} vs. I_D.

1. With the power off, connect a dual-channel System SourceMeter instrument to the computer's IEEE-488 interface.
2. Connect the test fixture to both units using appropriate cables.
3. Turn on the instrument and allow the unit to warm up for two hours for rated accuracy.
4. Turn on the computer and start Test Script Builder (TSB). Once the program has started, open a session by connecting to the instrument. For details on how to use TSB, see the Series 2600 Reference Manual.
5. You can simply copy and paste the code from Appendix A in this guide into the TSB script editing window (Program 10), manually enter the code from the appendix, or import the TSP file '*Transconductance.tsp*' after downloading it to your PC.

   If your computer is currently connected to the Internet, you can click on this link to begin downloading: http://www.keithley.com/data?asset=50916.
6. Install an N-channel FET such as an SD210 in the appropriate transistor socket of the test fixture.
7. Now, we must send the code to the instrument. The simplest method is to right-click in the open script window of TSB, and select '*Run as TSP file*'. This will compile the code and place it in the volatile run-time memory of the instrument. To store the program in non-volatile memory, see the "TSP Programming Fundamentals" section of the Series 2600 Reference Manual.
8. Once the code has been placed in the instrument run-time memory, we can run it at any time simply by calling the function '`Transconductance()`'. This can be done by typing the text '`Transconductance()`' after the active prompt in the Instrument Console line of TSB.
9. In the program '*Transconductance.tsp*', the function `Transconductance(vgsstart, vgsstop, vgssteps, vdsbias)` is created.
   - `vgsstart` represents the initial voltage value in the gate-source V_{GS} sweep
   - `vgsstop` represents the final voltage value in the gate-source V_{GS} sweep
   - `vgssteps` represents the number of steps in the sweep
   - `vdsbias` represents the voltage value applied to the drain-source terminal of the FET

   If these values are left blank, the function will use the default values given to the variables, but you can specify each variable value by simply sending a number that is in-range in the function call. As an example, if you wanted to have the start voltages for V_{GS} sweeps be 1V, the stop value be 11V, the number of steps be 20, and the V_{DS} value as 5V, you would send `Transconductance(1, 11, 20, 5)` to the instrument.
10. The sources will be zeroed and then enabled. The instrument will apply V_{DS} and execute a sweep of V_{GS} values between 0V and 5V using 100 steps. At each increment, I_D will be measured.

11. Once the measurements have completed, the data (V_{GS}, V_{DS}, I_D, and g_{fs}) will be presented in the Instrument Console window of TSB.

4.4.3 Typical Program 10 Results

Figure 4-4 shows a typical g_{fs} vs. V_{GS} plot as generated by the example program. Again, an SD210 N-channel FET was used for the example plot.

Figure 4-5 shows a typical g_{fs} vs. I_D plot generated by the example program.

4.4.4 Program 10 Description

The instrument is returned to default conditions. SMUB, which sweeps V_{GS}, is programmed as follows:

- Source V
- 1mA compliance, autorange
- Local sense
- `vgsstart`: 0V
- `vgsstop`: 5V
- `vgssteps`: 100

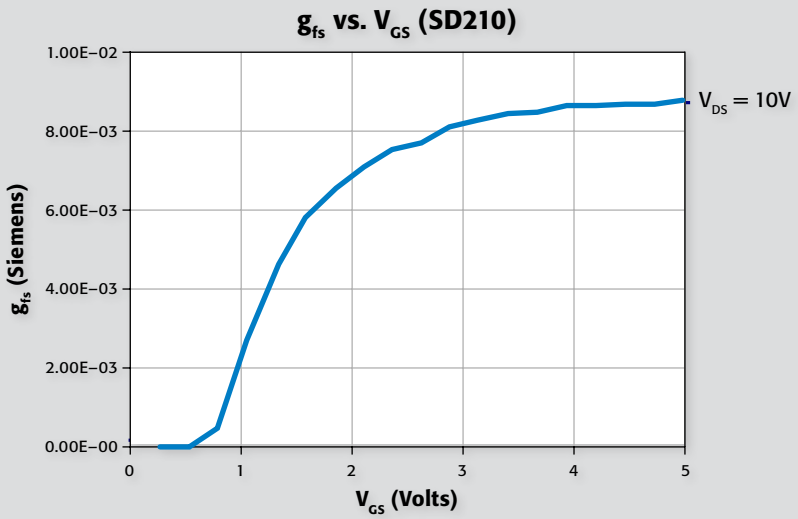

Figure 4-4. Program 10 results: Transconductance vs. V_{GS}

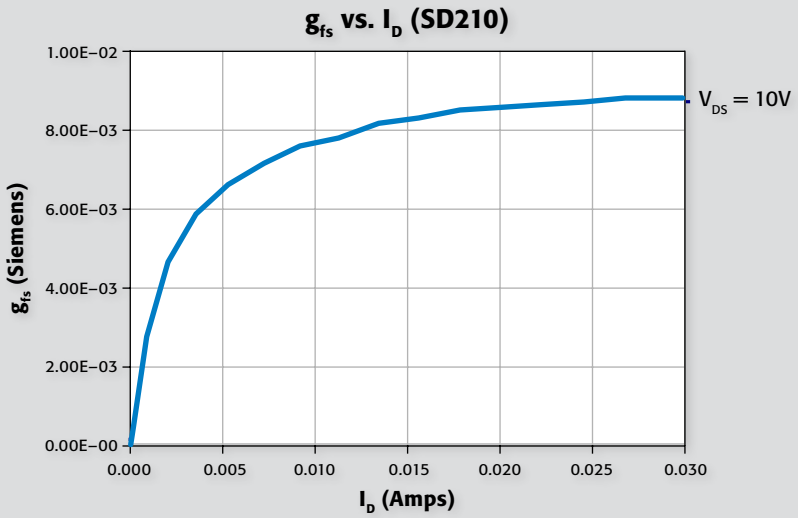

Figure 4-5. Program 10 results: Transconductance vs. I_D

SMUA, which sources V_{DS} and measures I_D, is then configured in the following manner:

- Source V
- Local sense
- 100mA compliance, autorange measure
- 1 NPLC Line cycle integration
- `vdsbias`:10V

Following setup of both units, the outputs are zeroed and enabled. SMUA applies the V_{DS} bias, and SMUB begins the V_{GS} voltage sweep. At each step in the V_{GS} sweep, SMUA measured the drain current (I_D). The process repeats until all points in the sweep have been taken.

Next, we encounter the part of the program where the transconductance values are calculated. Each transconductance value is computed from ΔI_D and ΔV_{GS}. Finally, the data (V_{GS}, I_D, and g_{fs}) is printed to the Instrument Console of TSB. You can then copy and paste the data to a spreadsheet to graph g_{fs} vs. V_{GS} and g_{fs} vs. I_D.

4.5 Threshold Tests

The threshold voltage (V_T) is a critical parameter for FET characterization, as well as process control. Basically, there are a number of methods for determining V_T, including several transconductance methods, the two-point extrapolated V_T method, as well as the V_T @ I_D search method. In this paragraph, we will discuss the I_D search method for finding V_T, along with a self-biasing method that takes advantage of the special capabilities of the Series 2600 System SourceMeter instruments.

4.5.1 Search Method Test Configuration

Figure 4-6 shows the general test configuration for the search method threshold voltage tests. SMUB sources V_{GS}, while SMUA sources V_{DS} and also measures I_D. An iterative search process is included in the program to allow you to enter a target I_D value.

4.5.2 Example Program 11A: Threshold Voltage Tests Using Search Method

Use Program 11A to perform the V_T test using the search for target I_D method.

1. With the power off, connect a dual-channel System SourceMeter instrument to the computer's IEEE-488 interface.
2. Connect the test fixture to both units using appropriate cables.
3. Turn on the instrument and allow the unit to warm up for two hours for rated accuracy.
4. Turn on the computer and start Test Script Builder (TSB). Once the program has started, open a session by connecting to the instrument. For details on how to use TSB, see the Series 2600 Reference Manual.
5. You can simply copy and paste the code from Appendix A in this guide into the TSB script editing window (Program 11A), manually enter the code from the appendix, or import the TSP file 'FET_Thres_Search.tsp' after downloading it to your PC.

   If your computer is currently connected to the Internet, you can click on this link to begin downloading: http://www.keithley.com/data?asset=50919.
6. Install an N-hannel FET such as an SD210 in the appropriate transistor socket of the test fixture.

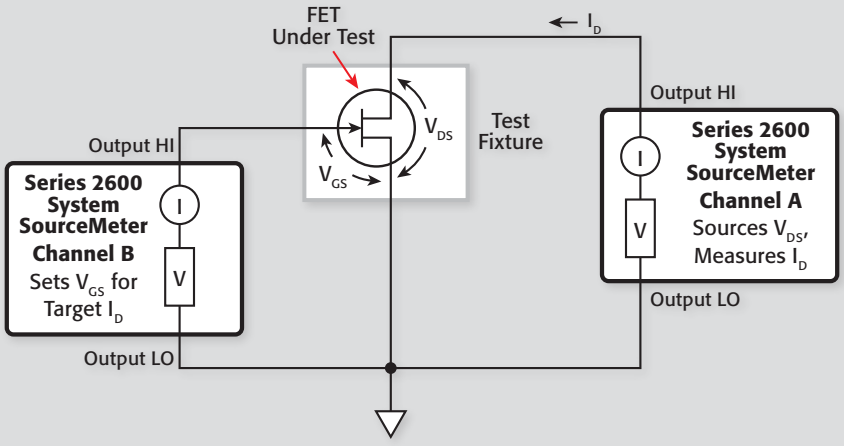

Figure 4-6. Configuration for search method threshold tests

7. Now, we must send the code to the instrument. The simplest method is to right-click in the open script window of TSB, and select '*Run as TSP file*'. This will compile the code and place it in the volatile run-time memory of the instrument. To store the program in non-volatile memory, see the "TSP Programming Fundamentals" section of the Series 2600 Reference Manual.
8. Once the code has been placed in the instrument run-time memory, we can run it at any time simply by calling the function '`FET _ Thres _ Search()`'. This can be done by typing the text '`FET _ Thres _ Search()`' after the active prompt in the Instrument Console line of TSB.
9. In the program '*FET_Thres_Search.tsp*', the function `FET _ Thres _ Search(vdssource, lowvgs, highvgs, targetid)` is created.
   - `vdssource` represents the voltage value on the drain-source of the transistor
   - `lowvgs` represents the gate-source voltage low limit for the search algorithm
   - `highvgs` represents the gate-source voltage high limit for the search algorithm
   - `targetid` represents the target drain current for the search algorithm

   If these values are left blank, the function will use the default values given to the variables, but you can specify each variable value by simply sending a number that is in-range in the function call. As an example, if you wanted to have the drain-source voltage (V_{DS}) be 2.5V, the gate-source voltage low value at 0.7V, the gate-source voltage high value at 1.5V, and the target drain current at 2μA, you would send `FET _ Thres _ Search(2.5, 0.7, 1.5, 2E-6)` to the instrument.
10. The sources will be enabled, and the collector current of the device will be measured. The program will perform an iterative search to determine the closest match to the target I_D (within ±5%). If the search is unsuccessful, the program will print "Iteration Level Reached". This is an error indicating that the search reached its limit. Recheck the connections, DUT, and variable values to make sure they are appropriate for the device.
11. Once the sweep has been completed, the data (I_D, V_{GS}, and V_{DS}) will be presented in the Instrument Console window of TSB.

4.5.3 Program 11A Description

Initially, the instrument is returned to default conditions. SMUB, which sources V_{GS}, is programmed as follows:

- Source V
- 1mA compliance, autorange
- Local sense

SMUA, which sources V_{DS} and measures I_D, is then configured in the following manner:

- Source V
- Local sense
- 100mA compliance, autorange measure
- 1 NPLC Line cycle integration

Once the SMU channels have been configured, the sources values are programmed to 0 and the outputs are enabled. The drain-source voltage (V_{DS}) is sourced, compliance is checked with the function `Check _ Comp()`, and the program enters into the binary search algorithm for the target drain current (I_D) by varying the gate-source voltage (V_{GS}) value, measuring the I_D, comparing it to the target I_D, and adjusting the V_{GS} value, if necessary. The iteration counter is incremented each cycle through the algorithm. If the number of iterations has been exceeded, a message to that effect is displayed, and the program halts.

Assuming that the number of iterations has not been exceeded, the data is displayed in the Instrument Console window of the TSB.

4.5.4 Modifying Program 11A

As written, the program sets the number of iterations to search for target I_D to 20. You can change this by adjusting the `l _ k _ max` variable to perform the iterative search as many times as is necessary. Similarly, the allowed range for the I_D target search is ±5%. Again, you can make this tolerance range as tight as necessary by modifying the limits in line 155. Note that reducing the target range will probably require an increase in the number of iterations as well.

4.5.5 Self-bias Threshold Test Configuration

Figure 4-7 shows the general test configuration for the self-bias method of threshold voltage tests. SMUB sources the drain current (assumed to be the same as the source current), and it also measures the threshold voltage, V_T. SMUA sources V_{DS}. This arrangement allows very rapid threshold voltage measurement (milliseconds per reading) at very low currents, and it can be used with both enhancement-mode and depletion-mode FETs. Note that the high impedance sensing circuits and the floating capabilities of the Series 2600 System SourceMeter instruments are key characteristics that allow this special configuration to be used.

WARNING

When a System SourceMeter instrument is programmed for remote sensing, hazardous voltage may be present on the SENSE and OUTPUT terminals when the unit is in operate regardless of the programmed voltage or current. To avoid a possible shock hazard, always turn off power before connecting or disconnecting cables to the SourceMeasure Unit or the associated test fixture.

NOTE

Entered values for both V_{DS} and I_D are adjusted to the reverse polarity because of the connection configuration used. For example, for an N-channel FET, both V_{DS} and I_D must be negative.

As an example, entering a V_{DS} of 5V will result in –5V actually being applied at the output.

These values will result in proper biasing of the DUT. Also, the sign of the measured V_T value will be reversed.

4.5.6 Example Program 11B: Self-bias Threshold Voltage Tests

Use Program 11B to perform the self-bias threshold voltage test.

1. With the power off, connect a dual-channel System SourceMeter instrument to the computer's IEEE-488 interface.
2. Connect the test fixture to both units using appropriate cables. Note that OUTPUT HI of SMUA is connected to the OUTPUT LO of SMUB, while SENSE HI of SMUA is connected to the OUTPUT HI of SMUB.
3. Turn on the instrument and allow the unit to warm up for two hours for rated accuracy.
4. Turn on the computer and start Test Script Builder (TSB). Once the program has started, open a session by connecting to the instrument. For details on how to use TSB, see the Series 2600 Reference Manual.
5. You can simply copy and paste the code from Appendix A in this guide into the TSB script editing window (Program 11B), manually enter the code from the appendix, or import the TSP file '*FET_Thres_Fast.tsp*' after downloading it to your PC.

   If your computer is currently connected to the Internet, you can click on this link to begin downloading from http://www.keithley.com/data?asset=50920.
6. Install an NPN FET such as a SD210 in the appropriate transistor socket of the test fixture.
7. Now, we must send the code to the instrument. The simplest method is to right-click in the open script window of TSB, and select '*Run as TSP file*'. This will compile the code and place it in the volatile run-time memory of the instrument. To store the program in non-volatile memory, see the "TSP Programming Fundamentals" section of the Series 2600 Reference Manual.

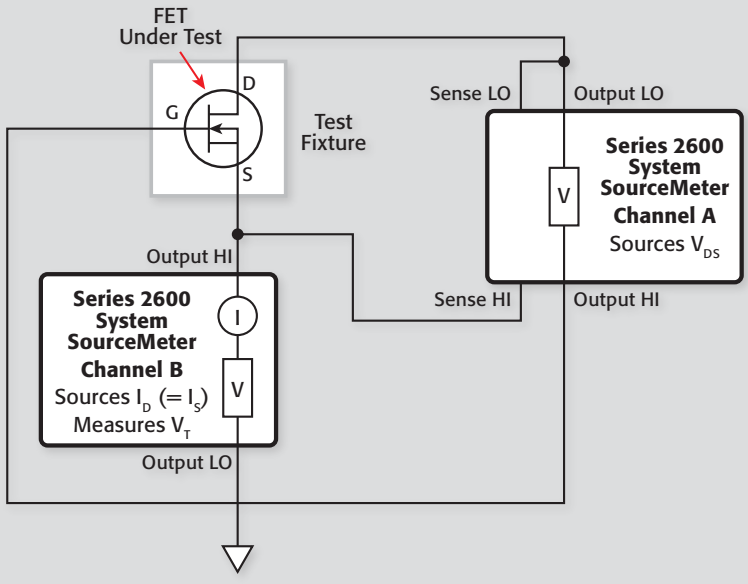

Figure 4-7. Configuration for self-bias threshold tests

8. Once the code has been placed in the instrument run-time memory, we can run it at any time simply by calling the function 'FET_Thres_Fast()'. This can be done by typing the text '`FET _ Thres _ Fast()`' after the active prompt in the Instrument Console line of TSB.
9. In the program '*FET_Thres_Fast().tsp*', the function `FET _ Thres _ Fast(vdssource, istart, istop, isteps)` is created.
   - `vdssource` represents the voltage value on the drain-source of the transistor
   - `istart` represents the start value for the drain current sweep
   - `istop` represents the stop value for the drain current sweep
   - `isteps` represents the number of steps in the current sweep

   If these values are left blank, the function will use the default values given to the variables, but you can specify each variable value by simply sending a number that is in-range in the function call. As an example, if you wanted to have the drain-source voltage (V_{DS}) be 0.25V, the drain current sweep start value at 0.20μA, the drain current sweep stop value at 2μA, and the number of steps be 15, you would send `FET _ Thres _ Fast(0.25, 200E-9, 2E-6, 15)` to the instrument.
10. The sources will be enabled, and the collector current of the device will be measured.
11. Once the sweep has been completed, the data (V_{DS}, V_T, and I_D) will be presented in the Instrument Console window of TSB. Note that the program reverses the polarity of the emitter currents in order to display true polarity.

4.5.7 Program 11B Description

Initially, the instrument is returned to default conditions. Next, SMUB, which sources I_D and measures V_T, is programmed as follows:

- Source I
- 11V compliance, autorange
- Local sense
- 1 NPLC integration rate
- `istart`: 0.5μA
- `istop`: 1μA
- `isteps`: 10

Next, SMUA, which sources V_{DS}, is configured in the following manner:

- Source V
- Remote sensing
- 100mA compliance, autorange
- `vdssource`: 0.5V

Once the SMU channels have been configured, the sources values are programmed to 0 and the outputs are enabled. The drain-source voltage (V_{DS}) is sourced and the drain current (I_D) is swept. At each point in the sweep, the threshold voltage (V_T) is measured.

The data is displayed in the Instrument Console window of the TSB.

Note that both I_D and V_T values are corrected for proper polarity.

4.5.8 Modifying Program 11B

As written, the program tests for threshold voltages at 10 values of I_D between 0.5μA and 1μA in 10 increments. These values can be changed to the required values simply by modifying the corresponding variables in the program.

Section 5
Using Substrate Bias

5.1 Introduction

To this point in this guide, we have focused on performing tests on devices that do not require substrate bias. Because many devices, especially those in complex packages, do require some form of substrate bias, our discussion would not be complete without discussing methods for applying and programming substrate bias.

In the following paragraphs, we will discuss applying substrate bias by adding another Series 2600 System SourceMeter instrument.

5.2 Substrate Bias Instrument Connections

WARNING

Interlock circuits must be connected before use. Connect the fixture ground to safety earth ground using #18 AWG minimum wire before use. Turn off all power before connecting or disconnecting wires or cables.

5.2.1 Source-Measure Unit Substrate Bias Connections and Setup

Figure 5-1 shows test connections when using two Series 2600 System SourceMeter instruments because the tests outlined in the following sections require three SMUs. Two SMUs supply the same functions as outlined earlier in this guide, and a third SMU is used to apply substrate bias. In the past, this would have required connecting and coordinating three separate instruments, each with only one SMU.

To simplify hardware integration, the Keithley Series 2600 System SourceMeter instruments are equipped with a few features that make the task of multi-channel testing much easier. For example, we can use a dual-channel instrument such as the Keithley Model 2602, 2612, or 2636 and a single-channel Instrument such as the Model 2601, 2611, or 2635. Therefore, we need only two instruments to perform the test. All of the following programs will also work using two dual-channel instruments with no modification.

For instrument-to-instrument communication, Keithley's Series 2600 System SourceMeter instruments employ an expansion interface known as TSP-Link™ interface. TSP-Link allows expanding test systems to include up to 16 TSP-Link enabled instruments.

In a TSP-Link-enabled system, one of the nodes (instruments) is the master, which is generally denoted as Node 1, while the other nodes in the system are slaves. One GPIB connection is required to link the controlling PC and the master instrument. All other master/slave connections require a simple TSP-Link connection using a crossover Ethernet cable. Additional instruments can be connected as slaves by simply connecting each slave to one another serially using additional crossover Ethernet cables and configuring each instrument for use as a TSP-Link node.

More information on TSP-Link features can be found in the Series 2600 System SourceMeter Reference Manual.

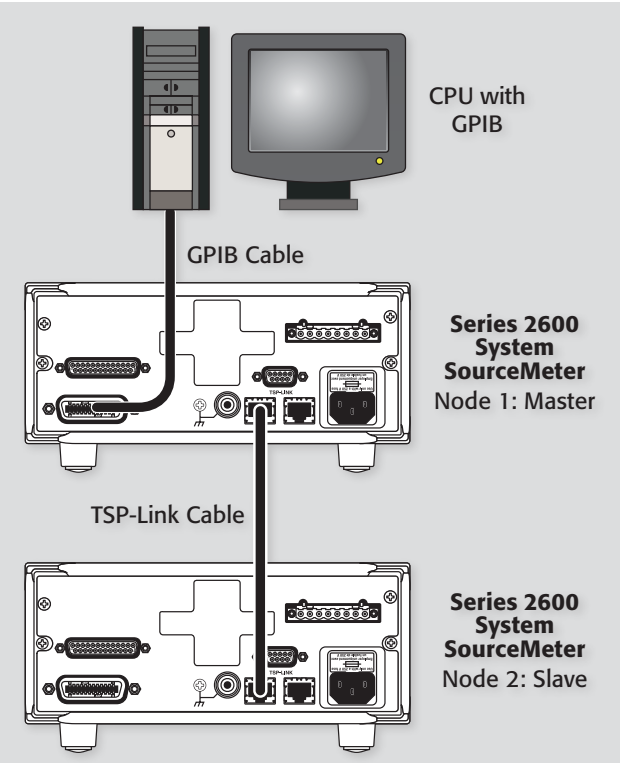

Figure 5-1. TSP-Link connections for two instruments

A test fixture with appropriate shielding and safety interlock mechanisms is recommended for test connections, along with Model 7078-TRX-3 triax cables for low current measurements. Note that the connecting cables to the second instrument, assume that local sensing will be used even though that may not be the situation in many cases.

5.2.2 Voltage Source Substrate Bias Connections

Figure 5-2 shows bias connections using a single-channel Model 2635 Low Current System SourceMeter instrument for substrate bias connections. Two additional SMU channels are added using a dual-channel Model 2602 System SourceMeter instrument. Note that remote sensing is not used in this application; remote sensing could be added by connecting the sense terminals of the Model 2635 to the sense connections on the test fixture and adding additional remote sense commands to the program.

NOTES

Remote sensing connections are recommended for optimum accuracy. See paragraph 1.2.2 for details.

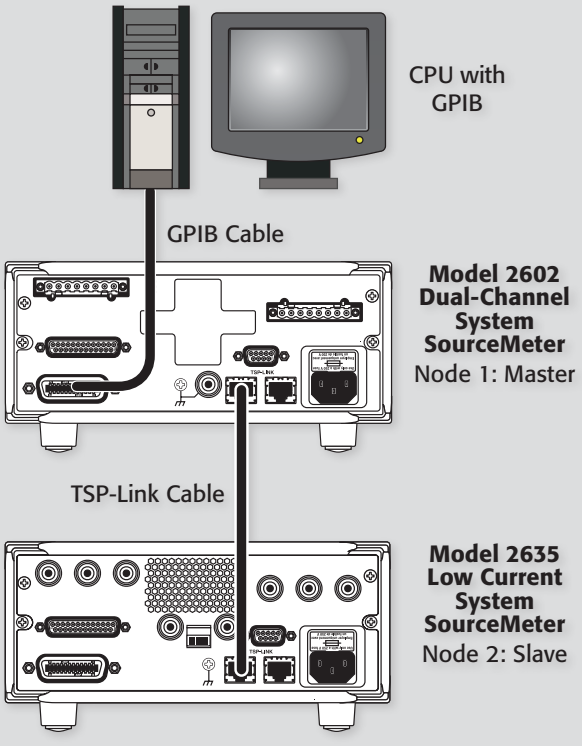

Figure 5-2. TSP-Link instrument connections

If measurement noise is a problem or for critical, low level applications, use shielded cable for all signal connections.

5.3 Source-Measure Unit Substrate Biasing

The following paragraphs discuss using three SMU channels to provide substrate biasing: a dual-channel instrument, such as a Model 2602 or 2636, and a single-channel instrument, such as a 2601 or 2635. All of the example programs will work with two dual-channel instruments with no modification.

In the first example, the substrate current (I_{SB}) is measured as the gate-source voltage (V_{GS}) is swept across the desired range. The program generates a plot of I_{SB} vs. V_{GS}. In the second example, the third SMU channel provides substrate bias for common-source characteristic tests.

5.3.1 Program 12 Test Configuration

Figure 5-3 shows the test configuration for Program 12. SMUB of Node 1 is used to sweep V_{GS}, while SMUA of Node 1 sources V_{DS}. SMUA of Node 2 applies a user-defined substrate bias (V_{SB}) to the device under test: it also measures the substrate current (I_{SB}).

5.3.2 Example Program 12: Substrate Current vs. Gate-Source Voltage

Program 12 demonstrates methods to generate an I_{SB} vs. V_{GS} plot. Follow these steps to use this program.

1. With the power off, connect the dual-channel Instrument to the computer's IEEE-488 interface. Connect the single-channel Instrument to the dual-channel master using a crossover Ethernet cable.
2. Connect the test fixture to both units using appropriate cables.
3. Turn on the instruments and allow the units to warm up for two hours for rated accuracy.
4. Configure the TSP-Link communications for each instrument.

   Slave: A single-channel instrument such as the Model 2601, 2611, or 2635.

   1. Press the MENU key to access MAIN MENU.
   2. Select the COMMUNICATION menu. (Skip this step if the Series 2600 instruments used have firmware Revision 1.4.0 or later installed.)

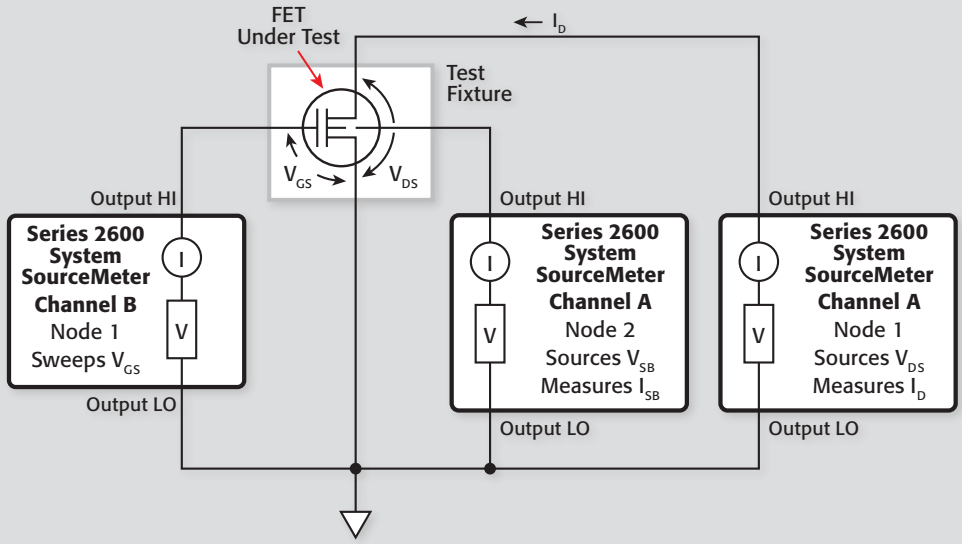

Figure 5-3. Program 12 test configuration

3. Select the TSPLINK_CFG menu. (If the Series 2600 instruments used have firmware Revision 1.4.0 or later installed, the menu name should be TSPLINK.)
4. Select the NODE menu.
5. Set the NODE number to 2 and press ENTER.

Master: A dual-channel instrument such as the Model 2602, 2612, or 2636.

1. Press the MENU key to access MAIN MENU.
2. Select the COMMUNICATION menu. (Skip this step if the Series 2600 instruments used have firmware Revision 1.4.0 or later installed.)
3. Select the TSPLINK_CFG menu. (If the Series 2600 instruments used have firmware Revision 1.4.0 or later installed, the menu name should be TSPLINK.)
4. Select the NODE menu.
5. Set the NODE number to 1 for the master and press ENTER.
6. Select the TSPLINK_CFG menu. (If the Series 2600 instruments used have firmware Revision 1.4.0 or later installed, the menu name should be TSPLINK.)
7. Select the RESET to initialize the TSP-Link.

5. Turn on the computer and start Test Script Builder (TSB). Once the program has started, open a session by connecting to the master instrument. For details on how to use TSB, see the Series 2600 Reference Manual.
6. You can simply copy and paste the code from Appendix A in this guide into the TSB script editing window (Program 12), manually enter the code from the appendix, or import the TSP file '*FET_Isb_Vgs.tsp*' after downloading it to your PC.

   If your computer is currently connected to the Internet, you can click on the following link to begin downloading: http://www.keithley.com/data?asset=50964.
7. Install an NPN FET such as a SD210 in the appropriate transistor socket of the test fixture.
8. Now, we must send the code to the instrument. The simplest method is to right-click in the open script window of TSB and select '*Run as TSP file*'. This will compile the code and place it in the volatile run-time memory of the instrument. To store the program in non-volatile memory, see the "TSP Programming Fundamentals" section of the Series 2600 Reference Manual.
9. Once the code has been placed in the instrument run-time memory, we can run it at any time simply by calling the function 'FET_Isb_Vgs()'. This can be done by typing the text '`FET _ Isb _ Vgs()`' after the active prompt in the Instrument Console line of TSB.
10. In the program '*FET_Isb_Vgs().tsp*', the function `FET _ Isb _ Vgs(vdssource, vsbsource,vgsstart,vgsstop, vgssteps)` is created.
    - `vdssource` represents the voltage value on the drain-source of the transistor
    - `vsbsource` represents the voltage value on the substrate-source of the transistor

- `vgsstart` represents the start value for the gate-source voltage sweep
- `vgsstop` represents the stop value for the gate-source voltage sweep
- `vgssteps` represents the number of steps in the sweep

If these values are left blank, the function will use the default values given to the variables, but you can specify each variable value by simply sending a number that is in range in the function call. As an example, if you wanted the drain-source voltage (V_{DS}) to be 2V, substrate-source (V_{SB}) to be –2V, the gate-source (V_{GS}) voltage sweep start value at 1V, the gate-source sweep stop value at 12V, and the number of steps to be 15, you would send `FET _ Isb _ Vgs(2, -2, 1, 12, 15)` to the instrument.

11. The sources will be enabled, and the gate-source voltage sweep will be executed.
12. Once the sweep has been completed, the data (I_D, V_{GS}, and I_{SB}) will be presented in the Instrument Console window of TSB.

5.3.3 Typical Program 12 Results

Figure 5-4 shows a typical plot generated by example Program 12 using an SD210 MOSFET.

5.3.4 Program 12 Description

After the SMUs are returned to default conditions, Node 1 SMUB, which sweeps V_{GS}, is configured as follows:

- Source V
- 1μA compliance, autorange
- Local sense
- `vgsstart`: 0V
- `vgsstop`: 10V
- `vgssteps`: 10

Next, Node 1 SMUA, which sources V_{DS}, is set up to operate in the following manner:

- Source V
- Local sensing
- 100mA compliance, autorange
- `vdssource`: 1V

Finally, Node 2 SMUA, which sources V_{SB} and measures I_{SB}, is programmed as follows:

- Source V
- Local sensing
- 1 compliance, autorange measure
- 1 NPLC Line cycle integration

After both instruments are set up, the outputs are zeroed and enabled. The bias values V_{SB} and V_{DS} are applied, then the V_{GS} sweep begins. At each point in the sweep, the drain current (I_D) and substrate leakage (I_{SB}) are measured.

After the sweep is complete, the data (I_D, V_{GS}, and I_{SB}) is printed to the Instrument Console of TSB.

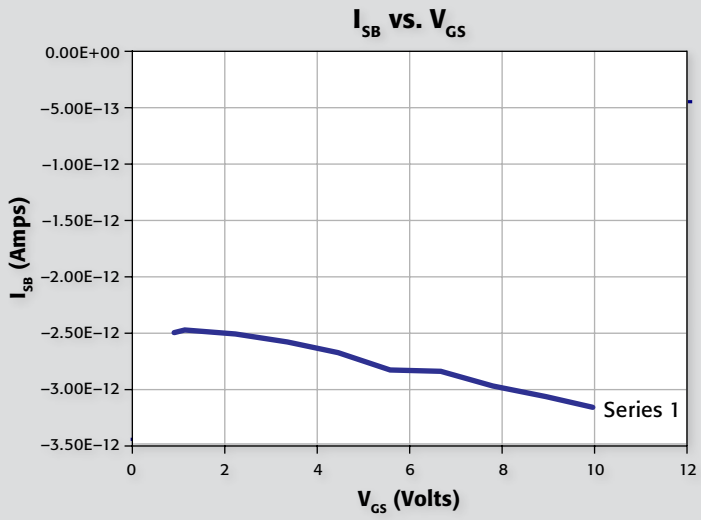

Figure 5-4. Program 12 typical results: I_{SB} vs. V_{GS}

5.3.5 Modifying Program 12

For different sweeps, the variables for V_{GS} start, V_{GS} stop, and V_{GS} step values can be changed as required. For different sweep lengths, array size and loop counter values must be adjusted accordingly. You can also change the V_{DS} value, if desired, by modifying that parameter accordingly.

5.3.6 Program 13 Test Configuration

Figure 5-5 shows the test configuration for Program 13. Unit #1 is used to sweep V_{GS}; Unit #2 sweeps V_{DS} and measures I_D. Unit #3 applies a user-defined substrate bias to the device under test. Common source characteristics are generated by data taken when the program is run.

5.3.7 Example Program 13: Common-Source Characteristics with Source-Measure Unit Substrate Bias

Program 13 demonstrates common-source characteristic test programming with substrate bias. Follow these steps to use this program.

1. With the power off, connect the dual-channel SourceMeter instrument to the IEEE-488 interface of the computer. Connect the single-channel SourceMeter instrument to the dual-channel master using a crossover Ethernet cable.
2. Connect the test fixture to both units using appropriate cables.
3. Turn on the instruments and allow the units to warm up for two hours for rated accuracy.
4. Configure the TSP-Link communications for each instrument.

   Slave: A single-channel instrument such as the Model 2601, 2611, or 2635.

   1. Press the MENU key to access MAIN MENU.
   2. Select the COMMUNICATION menu. (Skip this step if the Series 2600 instruments used have firmware Revision 1.4.0 or later installed.)
   3. Select the TSPLINK_CFG menu. (If the Series 2600 instruments used have firmware Revision 1.4.0 or later installed, the menu name should be TSPLINK.)
   4. Select the NODE menu.
   5. Set the NODE number to 2 and press ENTER.

   Master: A dual-channel instrument such as the Model 2602, 2612, or 2636.

   1. Press the MENU key to access MAIN MENU.
   2. Select the COMMUNICATION menu. (Skip this step if the Series 2600 instruments used have firmware Revision 1.4.0 or later installed.)
   3. Select the TSPLINK_CFG menu. (If the Series 2600 instruments used have firmware Revision 1.4.0 or later installed, the menu name should be TSPLINK.)
   4. Select the NODE menu.
   5. Set the NODE number to 1 for the master and press ENTER.

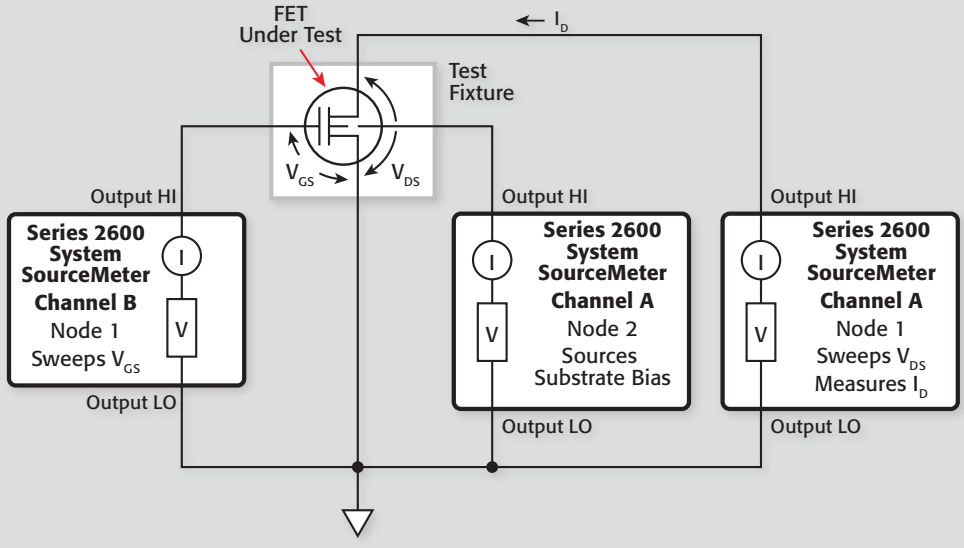

Figure 5-5. Program 13 test configuration

6. Select the TSPLINK_CFG menu. (If the Series 2600 instruments used have firmware Revision 1.4.0 or later installed, the menu name should be TSPLINK.)
7. Select the RESET to initialize the TSP-Link.

5. Turn on the computer and start Test Script Builder (TSB). Once the program has started, open a session by connecting to the master instrument. For details on how to use TSB, see the Series 2600 Reference Manual.

   You can simply copy and paste the code from Appendix A in this guide into the TSB script editing window (Program 13), manually enter the code from the appendix, or import the TSP file '*FET_Comm_Source_Vsb.tsp*' after downloading it to your PC.

   If your computer is currently connected to the Internet, click on the following link to begin downloading: http://www.keithley.com/data?asset=50921.

6. Install an NPN FET such as an SD210 in the appropriate transistor socket of the test fixture.
7. Now, we must send the code to the instrument. The simplest method is to right-click in the open script window of TSB and select '*Run as TSP file*'. This will compile the code and place it in the volatile run-time memory of the instrument. To store the program in non-volatile memory, see the "TSP Programming Fundamentals" section of the Series 2600 Reference Manual.
8. Once the code has been placed in the instrument run-time memory, we can run it at any time simply by calling the function 'FET_Comm_Source_Vsb()'. This can be done by typing the text '`FET _ Comm _ Source _ Vsb()`' after the active prompt in the Instrument Console line of TSB.
9. In the program '*FET_Comm_Source_Vsb().tsp*', the function `FET _ Comm _ Source _ Vsb(vgsstart, vgsstop, vgssteps, vdsstart, vdsstop, vdssteps, vsbsource)` is created.
   - `vgsstart` represents the start value for the gate-source voltage sweep
   - `vgsstop` represents the stop value for the gate-source voltage sweep
   - `vgssteps` represents the number of steps in the sweep
   - `vdsstart` represents the start value for the drain-source voltage sweep
   - `vdsstop` represents the stop value for the drain-source voltage sweep
   - `vdssteps` represents the number of steps in the sweep
   - `vsbsource` represents the substrate bias voltage

   If these values are left blank, the function will use the default values given to the variables, but you can specify each variable value by simply sending a number that is in-range in the function call. As an example, if you wanted to have the gate-source (V_{GS}) voltage sweep start value at 1V, the gate-source sweep stop value at 12V and the number of steps to be 10, the drain-source (V_{DS}) voltage sweep start value at 1V, the drain-source sweep stop value at 12V and the number of steps to be 80, and the substrate bias to be –2V, you would send

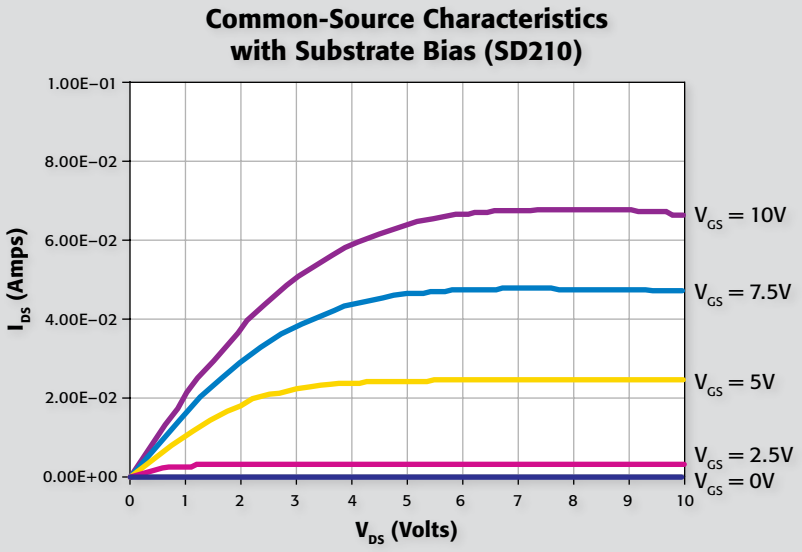

Figure 5-6. Program 13 typical results: Common-source characteristics with substrate bias

`FET _ Comm _ Source _ Vsb(1, 12, 10, 1, 12, 80, -2)` to the instrument.

10. The sources will be enabled, and the substrate bias is applied, the gate-source voltage value is applied, and the drain-source sweep is executed. The gate-source voltage value is then incremented and the drain-source sweep is re-run.
11. Once the gate-source sweep has been completed, the data (V_{SB}, V_{GS}, V_{DS}, and I_D) will be presented in the Instrument Console window of TSB.

5.3.8 Typical Program 13 Results

Figure 5-6 shows a typical plot generated by Example Program 13.

5.3.9 Program 13 Description

Both instruments are returned to default conditions. Node 1 SMUB, which sweeps V_{GS}, is configured as follows:

- Source V
- 1mA compliance, autorange
- Local sense
- `vgsstart`: 0V
- `vgsstop`: 10V
- `vgssteps`: 5

Next, Node 1 SMUA, which sweeps V_{DS} and measures I_D, is set up to operate in the following manner:

- Source V
- Local sensing
- 100mA compliance, autorange measure
- 1 NPLC Line cycle integration
- `vdsstart`: 0V
- `vdsstop`: 10V
- `vdssteps`: 100

Finally, Node 2 SMUA, which provides substrate bias, is programmed as follows:

- Source V
- Local sensing
- 10mA compliance, autorange measure

Both instruments are returned to default conditions; the sources are zeroed and enabled. The substrate bias (V_{SB}) and gate-source (V_{GS}) are applied and the program enters the main program loop to perform five I_D vs. V_{DS} sweeps, one for each of five V_{GS} values. Node 1 SMUA then cycles through its sweep list, setting V_{DS} to the required values, and measuring I_D at each step along the way. The program then loops back for the next sweep until all five sweeps have been performed.

Next, all three SMU outputs are zeroed and disabled. Finally, the data is written to the Instrument Console of the TSB.

5.3.10 Modifying Program 13

For different sweeps, the V_{GS} start, V_{GS} stop, V_{GS} steps, V_{DS} start, V_{DS} stop, and V_{DS} steps values can be changed as required. For different sweep lengths, array size and loop counter values must be adjusted accordingly.

5.4 BJT Substrate Biasing

The following paragraphs discuss using one dual-channel and one single-channel Series 2600 System SourceMeter instrument to perform tests on a four-terminal device, such as a BJT, with substrate bias. The example shown in this section is a modified version of the common-emitter BJT test presented previously in the guide.

5.4.1 Program 14 Test Configuration

Figure 5-7 shows the test configuration for Program 14. Node 1 SMUB is used to sweep I_B, while Node 1 SMUA sweeps V_{CE} and measures I_C. Node 2 SMUA applies the substrate bias (V_{SB}) to the device under test.

5.4.2 Example Program 14: Common-Emitter Characteristics with a Substrate Bias

Program 14 demonstrates common-emitter characteristic test programming with substrate bias. Proceed as follows:

1. With the power off, connect the dual-channel System SourceMeter instrument to the computer's IEEE-488 interface. Connect the single-channel System SourceMeter instrument to the dual-channel master using a crossover Ethernet cable.
2. Connect the test fixture to both units using appropriate cables.
3. Turn on the instruments and allow the units to warm up for two hours for rated accuracy.
4. Configure the TSP-Link communications for each instrument.

   Slave: A single-channel instrument such as the Model 2601, 2611, or 2635.

   1. Press the MENU key to access MAIN MENU.

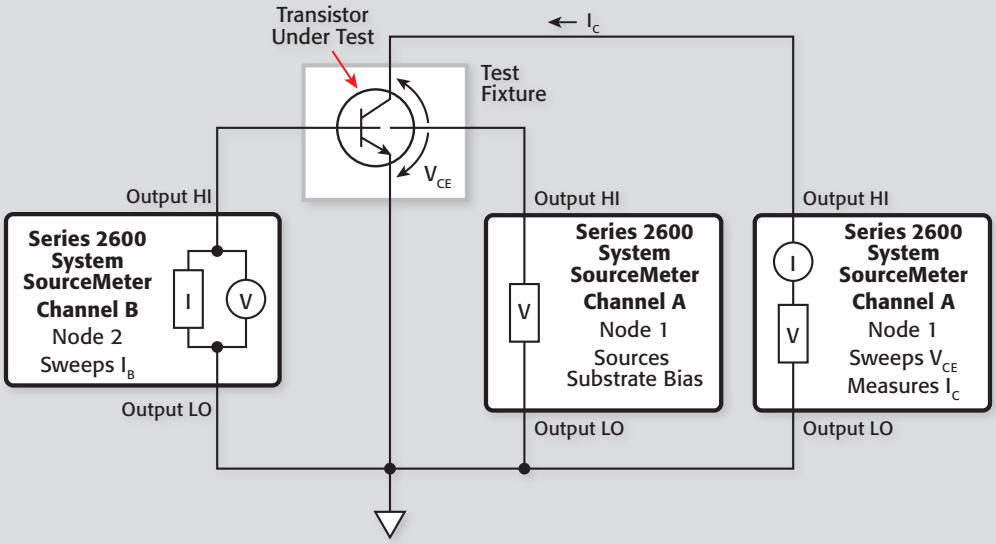

Figure 5-7. Program 14 test configuration

2. Select the COMMUNICATION menu. (Skip this step if the Series 2600 instruments used have firmware Revision 1.4.0 or later installed.)
3. Select the TSPLINK_CFG menu. (If the Series 2600 instruments used have firmware Revision 1.4.0 or later installed, the menu name should be TSPLINK.)
4. Select the NODE menu.
5. Set the NODE number to 2 and press ENTER.

Master: A dual-channel instrument such as the Model 2602, 2612, or 2636.

1. Press the MENU key to access MAIN MENU.
2. Select the COMMUNICATION menu. (Skip this step if the Series 2600 instruments used have firmware Revision 1.4.0 or later installed.)
3. Select the TSPLINK_CFG menu. (If the Series 2600 instruments used have firmware Revision 1.4.0 or later installed, the menu name should be TSPLINK.)
4. Select the NODE menu.
5. Set the NODE number to 1 for the master and press ENTER.
6. Select the TSPLINK_CFG menu. (If the Series 2600 instruments used have firmware Revision 1.4.0 or later installed, the menu name should be TSPLINK.)
7. Select the RESET to initialize the TSP-Link.

5. Turn on the computer and start Test Script Builder (TSB). Once the program has started, open a session by connecting to the master instrument. For details on how to use TSB, see the Series 2600 Reference Manual.

   You can simply copy and paste the code from Appendix A in this guide into the TSB script editing window (Program 14), manually enter the code from the appendix, or import the TSP file 'BJT_Comm_Emit_Vsb.tsp' after downloading it to your PC.

   If your computer is currently connected to the Internet, you can click on this link to begin downloading: http://www.keithley.com/data?asset=50928.

6. Install a BJT with substrate connections in appropriate transistor socket of the test fixture. The test is optimized for BJTs with source requirements similar to a 2N3904.
7. Now, we must send the code to the instrument. The simplest method is to right-click in the open script window of TSB, and select '*Run as TSP file*'. This will compile the code and place it in the volatile run-time memory of the instrument. To store the program in non-volatile memory, see the "TSP Programming Fundamentals" section of the Series 2600 Reference Manual.
8. Once the code has been placed in the instrument run-time memory, we can run it at any time simply by calling the function 'BJT_Comm_Emit_Vsb()'. This can be done by typing the text '`FET _ Comm _ Source _ Vsb()`' after the active prompt in the Instrument Console line of TSB.
9. In the program '*BJT_Comm_Emit_Vsb().tsp*', the function `BJT _ Comm _ Emit _ Vsb(istart,`

`istop, isteps, vstart, vstop, vsteps, vsbsource)` is created.

- `istart` represents the start value for the base current sweep
- `istop` represents the stop value for the base current sweep
- `isteps` represents the number of steps in the sweep
- `vstart` represents the start value for the collector-emitter voltage sweep
- `vstop` represents the stop value for the collector-emitter voltage sweep
- `vsteps` represents the number of steps in the sweep
- `vsbsource` represents the substrate bias voltage

If these values are left blank, the function will use the default values given to the variables, but you can specify each variable value by simply sending a number that is in-range in the function call. As an example, if you wanted to have the base current (I_B) current sweep start value at 20μA, the base current sweep stop value at 200μA and the number of steps to be 10, the collector-emitter (V_{CE}) voltage sweep start value at 1V, the collector-emitter sweep stop value at 12V and the number of steps to be 80, and the substrate bias to be –2V, you would send `BJT _ Comm _ Em t _ Vsb(20E-6, 200E-6, 10, 1, 12, 80, -2)` to the instrument.

10. The sources will be enabled, and the substrate bias is applied, the base current value is applied, and the collector-emitter voltage sweep is executed. The base current value is then incremented and the collector-emitter sweep is re-run.
11. Once the gate-source sweep has been completed, the data (I_B, V_{SB}, V_{CE}, and I_C) will be presented in the Instrument Console window of TSB.

5.4.3 Typical Program 14 Results

Figure 5-8 shows a typical plot generated by example Program 14.

5.4.4 Program 14 Description

After both instruments are returned to default conditions, Node 1 SMUB, which sweeps IB, is configured as follows:

- Source I
- IV compliance, 1.1V range
- Local sense
- `istart`: 10μA
- `istop`: 50μA
- `isteps`: 5

Next, Node 1 SMUA, which sweeps V_{CE} and measures I_C, is set up to operate in the following manner:

- Source V
- Local sensing
- 100mA compliance, autorange measure
- 1 NPLC Line cycle integration

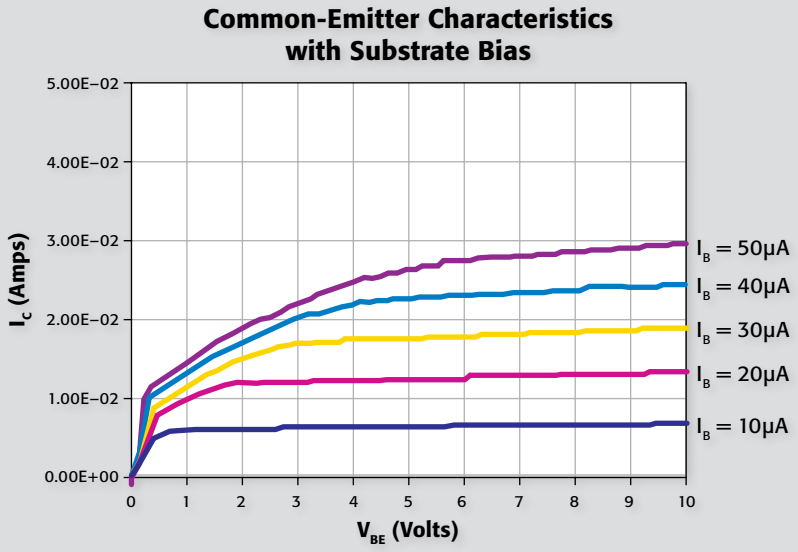

Figure 5-8. Program 14 typical results: Common-emitter characteristics with substrate bias

- `vstart`: 0V
- `vstop`: 10V
- `vsteps`: 100

Finally, Node 2 SMUA, which provides substrate bias, is programmed:

- Source V
- Local sensing
- 100mA compliance, autorange measure
- `vsbsource`: 1V

After the instruments have been set up, the outputs are zeroed and enabled. The substrate bias (V_{SB}) and first base current (I_B) values are applied. Then, the collector-emitter voltage sweep begins. At each point in the sweep, the collector current is measured. The program enters the main program loop to perform five I_C vs. V_{CE} sweeps, one for each of five I_B values.

Upon completion of the base current sweep, all outputs are zeroed and disabled. The data is written to the Instrument Console of TSB.

5.4.5 Modifying Program 14

For different sweeps, the base current start, stop, step, and the collector-emitter voltage start, stop, and step values can be changed as required. For different sweep lengths, loop counter values must be adjusted accordingly.

Section 6
High Power Tests

6.1 Introduction

Many devices, such as LED arrays and power FETs, require large current or voltage values for operation or characterization, which can create issues when testing. While System SourceMeter instruments are extremely flexible, they do have power limitations. For example, a single SMU channel of a Model 2602 can deliver up to 40W of power. That translates to sourcing 1A at 40V or 40V at 1A. What do we do if our device requires 2A at 40V?

Luckily, the answer is straightforward if we take certain precautions.

The following examples illustrate how to configure a dual-channel instrument, such as a Model 2602, 2612, or 2636, to deliver higher current or voltage values.

6.1.1 Program 15 Test Configuration

Figure 6-1 shows the test configuration for Program 15. SMUA and SMUB outputs are wired in parallel: SMUA Output HI to SMUB Output HI and SMUA Output LO to SMUB output LO. This effectively doubles the maximum current output and can deliver a total of 2A at 40V.

In this example, local sense is being used to measure voltage, but you can use remote sensing from one of the SMU channels if high accuracy voltage measurements are required. See paragraph 1.2.2 for more information on remote sensing.

6.1.2 Example Program 15: High Current Source and Voltage Measure

Program 15 demonstrates how to deliver higher current sourcing values using a dual-channel System SourceMeter instrument. Follow these steps to use this program.

1. With the power off, connect the dual-channel Instrument to the computer's IEEE-488 interface.
2. Connect the test fixture to both units using appropriate cables.
3. Turn on the instrument and allow the unit to warm up for two hours for rated accuracy.
4. Turn on the computer and start Test Script Builder (TSB). Once the program has started, open a session by connecting to the instrument. For details on how to use TSB, see the Series 2600 Reference Manual.
5. You can simply copy and paste the code from Appendix A in this guide into the TSB script editing window (Program 15), manually enter the code from the appendix, or import the TSP file '*KI2602Example_High_Current.tsp*' after downloading it to your PC.

   If your computer is currently connected to the Internet, you can click on this link to begin downloading: http://www.keithley.com/data?asset=50965.

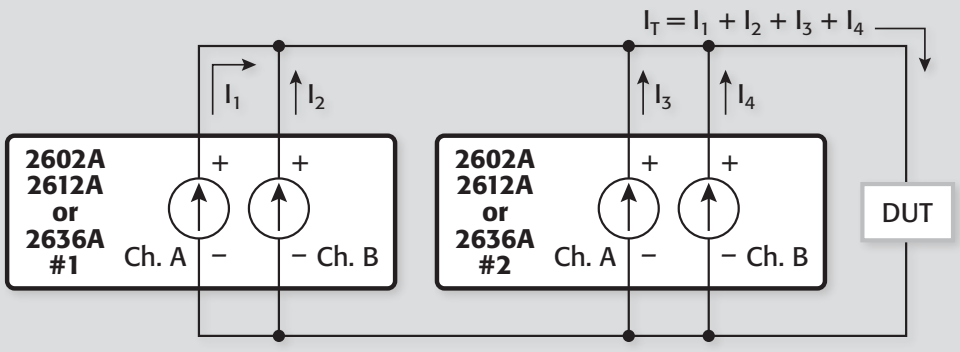

Figure 6-1. High current (SMUs in parallel)

6. Install a device (Power FET, LED array, etc.) in the appropriate transistor socket of the test fixture.
7. Now, we must send the code to the instrument. The simplest method is to right-click in the open script window of TSB, and select '*Run as TSP file*'. This will compile the code and place it in the volatile run-time memory of the instrument. To store the program in non-volatile memory, see the "TSP Programming Fundamentals" section of the Series 2600 Reference Manual.
8. Once the code has been placed in the instrument run-time memory, we can run it at any time simply by calling the function 'RunHighCurrent(sourcei,points)', where `sourcei` is the desired current value and `points` is the number of voltage measurements.
9. In the program '*KI2602Example_High_Current.tsp*', the function `RunHighCurrent(sourcei,points)` is created.
   - `sourcei` represents the current value delivered to the DUT. Note that the programmed current value for each SMU is half the isource value.
   - `points` represents the number of voltage measurements acquired

   If you wanted to source 2A total to the DUT and collect 100 voltage measurements, you would send `RunHighCurrent(2, 100)` to the instrument.
10. The sources will be enabled, and the current source and voltage measurements will be executed.
11. Once the measurements have been completed, the data will be presented in the Instrument Console window of TSB.

6.1.3 Program 15 Description

After the SMUs are returned to default conditions, SMUA is configured as follows:

- Source I
- 40V compliance, autorange
- Local sense
- 1 NPLC integration rate
- `sourcei`: Desired DUT current
- `points`: Number of points to measure

Next, SMUB is set up to operate in the following manner:

- Source I
- Local sensing
- 40V, autorange
- `sourcei`: Desired DUT current

After the instrument is set up, the outputs are zeroed and enabled. Each SMU performs a DC current source and SMUA begins to measure the voltage. When the data collection has reached the desired number of points, the outputs are disabled and the voltage data is printed to the Instrument Console of TSB.

6.2 Instrument Connections

WARNING

If either SMU reaches a compliance state, the instrument, device, or both could be damaged. To avoid this, set the compliance value to the maximum for your instrument and avoid open or other high resistance states for the SMUs when in Current Source mode.

6.2.1 Program 16 Test Configuration

Figure 6-2 shows the test configuration for Program 16: SMUA and SMUB outputs are wired in series, SMUA Lo to SMUB Hi, SMUA Hi to DUT, SMUB Lo to DUT. This effectively doubles the maximum voltage output and can deliver a total of 80V at 1A using a Model 2602 System SourceMeter instrument.

6.2.2 Example Program 16: High Voltage Source and Current Measure

Program 16 demonstrates how to deliver higher voltage sourcing values using a dual-channel System SourceMeter instrument. Follow these steps to use this program.

1. With the power off, connect the dual-channel Instrument to the computer's IEEE-488 interface.

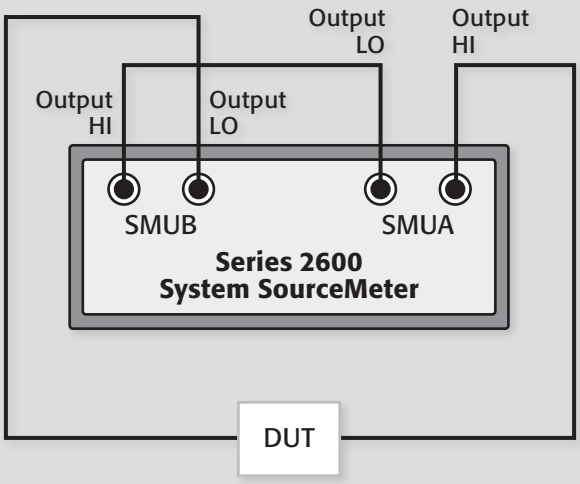

Figure 6-2. High voltage (SMUs in series)

2. Connect the test fixture to both units using appropriate cables.
3. Turn on the Instrument and allow the unit to warm up for two hours for rated accuracy.
4. Turn on the computer and start Test Script Builder (TSB). Once the program has started, open a session by connecting to the instrument. For details on how to use TSB, see the Series 2600 Reference Manual.
5. You can simply copy and paste the code from Appendix A in this guide into the TSB script editing window (Program 16), manually enter the code from the appendix, or import the TSP file 'KI2602Example_High_Voltage.tsp' after downloading it to your PC.

   If your computer is currently connected to the Internet, you can click on this link to begin downloading: http://www.keithley.com/data?asset=50966.
6. Install a device (Power FET, LED array, etc.) in the appropriate transistor socket of the test fixture.
7. Now, we must send the code to the instrument. The simplest method is to right-click in the open script window of TSB, and select '*Run as TSP file*'. This will compile the code and place it in the volatile run-time memory of the instrument. To store the program in non-volatile memory, see the "TSP Programming Fundamentals" section of the Series 2600 Reference Manual.
8. Once the code has been placed in the instrument run-time memory, we can run it at any time simply by calling the function '`RunHighVoltage(sourcev, points)`', where `sourcei` is the desired voltage value and `points` is the number of voltage measurements.
9. In the program '*KI2602Example_High_Voltage.tsp*', the function `RunHighVoltage(sourcev,points)` is created.
   - `sourcev` represents the voltage value delivered to the DUT Note that the actual voltage value programmed for each SMU is half the sourcev value.
   - `points` represents the number of voltage measurements acquired

   If you wanted to source 80V total to the DUT and collect 100 voltage measurements, you would send `RunHighVoltage(80, 100)` to the instrument.
10. The sources will be enabled, and the voltage source and current measurements will be executed.
11. Once the measurements have been completed, the data will be presented in the Instrument Console window of TSB.

6.2.3 Program 16 Description

After the SMUs are returned to default conditions, SMUA is configured as follows:

- Source V
- 1A compliance, autorange
- 1 NPLC integration rate
- `sourcev`: DUT voltage
- `points`: Number of points to measure

Next, SMUB is set up to operate in the following manner:

- Source I
- 1A, autorange
- `sourcev`: DUT voltage

After the instrument is set up, the outputs are zeroed and enabled. Each SMU performs a DC voltage source and SMUA begins to measure the current. When the data collection has reached the desired number of points, the outputs are disabled and the current data is printed to the Instrument Console of TSB.

Warning:
If either SMU reaches a compliance state, the instrument, device, or both could be damaged. To avoid this, set the compliance value to the maximum for your instrument and avoid shorting the SMUs when in Voltage Source mode.

Appendix A
Scripts

Section 2. Two-Terminal Devices

Program 1. Voltage Coefficient of Resistors

```
--[[
Volt_Co():

This program performs a voltage coefficient measurement on a 10GΩ part.

Required equipment:

(1) Single-channel Keithley Series 2600 System SourceMeter® instrument
(1) 10GΩ resistor

Running this script creates functions that can be used to measure the voltage coefficient
of resistances.

      The functions created are:
            1. Volt_Co(v1src, v2src) --Default values v1src = 100V, v2src = 200V
            2. Check_Comp()
            3. Calc_Val(v1src, v2src, i1meas, i2meas)
            4. Print_Data(voltco,res1,res2)

      See detailed information listed in individual functions.

To run:

1) From Test Script Builder
    - Right-click in the program window, select "Run as TSP"
       - At the TSP> prompt in the Instrument Control Panel, type Volt_Co()

3) From an external program
      - Send the entire program text as a string using standard GPIB Write calls.

Rev1: JAC 5.9.2007
]]--


function Volt_Co(v1src, v2src) --Configure instrument to supply two user-defined
                               --voltages and measure current.

      --Instrument variables.
```

```
local l_srcdelay = 0 --Source delay before measurement
local l_icmpl = 1E-3 --Source compliance
local l_nplc = 1 --Measurement Integration Rate

local l_v1src = v1src --First voltage source value
local l_v2src = v2src --Second voltage source value

--Define measured and calculated variables
local l_i1meas = 0 --Initialize first current measurement
local l_res1 = 0 --Initialize first resistance measurement

local l_i2meas = 0 --Initialize second current measurement
local l_res2 = 0 --Initialize second resistance measurement

local l_voltco = 0 --Initialize voltage coefficient calculation

local l_comp_val = false --Initialize compliance variable

--Default values and level check
if (l_v1src == nil) then --Use default value
      l_v1src = 100
end --if

if (l_v1src > 100) then --Coerce source value within range
      l_v1src = 100
      print("Maximum voltage value is 202V!!")
end   --if

if (l_v2src == nil) then --Use default value
      l_v2src = 200
end --if

if (l_v2src > 200) then  --Coerce source value within range
      l_v2src = 200
      print("Maximum voltage value is 202V!!")
end   --if

--Configure source and measure settings
smua.reset() --Reset SMU
errorqueue.clear() --Clear the error queue

smua.source.func = smua.OUTPUT_DCVOLTS --Output Voltage

smua.source.levelv = 0 --Source 0 before enabling output
smua.measure.nplc = l_nplc --Set integration rate

smua.source.autorangev = smua.AUTORANGE_ON --Enable source autorange
smua.source.limiti = l_icmpl

smua.measure.autorangei = smua.AUTORANGE_ON --Enable measurement autorange
```

```
	smua.source.output = smua.OUTPUT_ON --Enable output
	smua.source.levelv = l_v1src --Source programmed value

	l_comp_val = Check_Comp() --check compliance

	if l_comp_val == true then

	smua.source.output = smua.OUTPUT_OFF --Disable output
	smua.source.levelv = 0 --Return source to 0
	else
		delay(l_srcdelay) --wait before making measurement

		l_i1meas = smua.measure.i() --measure current

		smua.source.levelv = l_v2src --Source programmed value
		delay(l_srcdelay)   --wait before making measurement

		l_i2meas = smua.measure.i() --Measure current

	smua.source.output = smua.OUTPUT_OFF --Disable output
	l_voltco, l_res1, l_res2 = Calc_Val(l_v1src, l_v2src, l_i1meas,l_i2meas)
--calculate
	Print_Data(l_voltco, l_res1, l_res2) --print
   end --if

end --function Volt_Co()


function Check_Comp() --Function checks state of compliance, if true, prints warning and
returns
                      --to run_test()

	local l_comp_val = false --Initialize local variable

	l_comp_val = smua.source.compliance --Check compliance

	if l_comp_val == true then
	print("")
	print("SMU Source in Compliance!")
	print("Ensure proper connections, stable device, and settings are correct")
	print("Rerun Test")
	print("")
	end --if

	return l_comp_val

end --function Check_Comp()
```

```
function Calc_Val(v1src, v2src, i1meas, i2meas)      --function calculates resistance and
voltage coefficient
      local l_res1 = v1src/i1meas --Return quotient = resistance calculation
      local l_res2 = v2src/i2meas --Return quotient = resistance calculation
      local l_voltco = 100*(l_res2-l_res1)/(l_res1*(v2src-v1src)) --Return quotient =
voltage coefficient

      return l_voltco, l_res1, l_res2 --Return values

end --function Calc_Val()

function Print_Data(voltco,res1,res2)
      local l_voltco = voltco
      local l_res1 = res1
      local l_res2 = res2
      print("")
      print("**** Data ****")
      print("")
      print("Voltage Coefficient: ", voltco, "%/V") --Print Voltage Coefficient
      print("")
      print("Resistance R1: ", res1, "Ohms") --Print resistance value
      print("Resistance R2: ", res2, "Ohms") --Print resistance value
end --function Print_Data()

--Volt_Co() --Call Volt_Co()
```

Program 2. Capacitor Leakage Test

```
--[[
Cap_Leak():

This program performs capacitor leakage measurement.

Required equipment:

(1) Single-channel Keithley Series 2600 System SourceMeter instrument
(1) 1µF aluminum electrolytic capacitor

Running this script creates functions that can be used to test capacitors.

      The functions created are:
            1. Cap_Leak(vsrc,soak) --Default value vsrc = 40V
            2. Check_Comp()
            3. Calc_Val(vsrc, leaki)
            4. Print_Data(leaki, leakres)

      See detailed information listed in individual functions.

To run:

1) From Test Script Builder
      - At the TSP> prompt in the Instrument Control Panel, type Cap_Leak()

2) From an external program
      - Send the entire program text as a string using standard GPIB Write calls.

Rev1: JAC 5.22.2007
]]--



function Cap_Leak(vsrc, soak) --Configure instrument to source user-defined voltage
                              --and measure current.

      --Instrument variables.
      local l_soak = soak --Source delay before measurement (Recommended 7RC)
      local l_icmpl = 1E-2 --Source compliance
      local l_nplc = 1 --Measurement Integration Rate
      local l_vsrc = vsrc--Voltage source value

      --Define measured and calculated variables
      local l_leaki = 0 --Initialize leakage current measurement
      local l_leakres = 0 --Initialize leakage resistance measurement
      local l_comp_val = false --Initialize compliance variable

      --Default setting and level check
```

```
	if (l_vsrc == nil) then --Use default value
		l_vsrc = 40
	end --if

	if (l_vsrc > 100) then --Coerce source value within range
		l_vsrc = 100
		print("Maximum voltage value is 100V!!")
	end	--if

	if (l_soak == nil) then --Use default value
		l_soak = 10
	end --if

	--Configure source and measure settings
	smua.reset() --Reset SMU
	errorqueue.clear() --Clear the error queue

	smua.source.func = smua.OUTPUT_DCVOLTS --Output Voltage

	smua.source.levelv = 0 --Source 0 before enabling output
	smua.measure.nplc = l_nplc --Set integration rate

	smua.source.autorangev = smua.AUTORANGE_ON --Enable source autorange
	smua.source.limiti = l_icmpl

	smua.measure.autorangei = smua.AUTORANGE_ON --Enable measurement autorange

--Begin test
smua.source.output = smua.OUTPUT_ON --Enable output
	smua.source.levelv = l_vsrc --Source programmed value
	delay(soak) --wait before making measurement

	l_comp_val = Check_Comp() --check compliance


	if l_comp_val == true then

		smua.source.output = smua.OUTPUT_OFF --Disable output

	else

			l_leaki = smua.measure.i() --measure current
		smua.source.output = smua.OUTPUT_OFF --Disable output
		l_leakres = Calc_Val(l_vsrc, l_leaki) --calculate
		Print_Data(l_leaki, l_leakres) --print

	end --if

end --function Cap_Leak()
```

```
function Check_Comp() --Function checks state of compliance, if true, prints warning and
returns
                      --to run_test()

       local l_comp_val = false --Initialize local variable

       l_comp_val = smua.source.compliance --Check compliance

       if l_comp_val == true then
              print("")
              print("SMU Source in Compliance!")
              print("Ensure proper connections, stable device, and settings are correct")
              print("Rerun Test")
              print("")
       end --if

       return l_comp_val

end --function Check_Comp()

function Calc_Val(vsrc, leaki)   --function calculates resistance and voltage coefficient
       local l_vsrc = vsrc --Pass global source variable to local
       local l_leaki = leaki --Pass global current variable to local
       local l_leakres = 0 --Initialize leakage resistance local

       l_leakres = vsrc/leaki --Return quotient = resistance calculation

       return l_leakres

end --function Calc_Val()

function Print_Data(leaki, leakres)
       local l_leaki = leaki
       local l_leakres = leakres
       print("")
       print("**** Data ****")
       print("")
       print("Leakage Current: ", l_leaki, "A") --Print Leakage Current
       print("")
       print("Leakage Resistance: ", l_leakres, "Ohms") --Print resistance value

end --function Print_Data()

--Cap_Leak() --Call Cap_Leak() function
```

Program 3. Diode Characterization

Program 3A. Diode Characterization Linear Sweep

```
--[[
Diode_Fwd_Char(): USES TABLES

This program performs a forward characterization test on a diode and prints data.

Required equipment:

(1) Single-channel Keithley Series 2600 System SourceMeter instrument
(1) Silicon diode or equivalent

Running this script creates functions that can be used to measure the IV characteristics
of diodes.

       The functions created are:
              1. Diode_Fwd_Char(ilevel, start, stop, steps) --Default values ilevel = 0s,
start = 1ma, stop = 10ma

--steps = 10
              2. Print_Data(steps,volt,curr)


       See detailed information listed in individual functions.

To run:

1) From Test Script Builder
       - Right-click in the program window, select "Run as TSP"
       - At the TSP> prompt in the Instrument Control Panel, type Diode_Fwd_Char()

2) From an external program
       - Send the entire program text as a string using standard GPIB Write calls.

Rev1: JAC 5.22.2007

]]--


------------------ Keithley TSP Function ------------------

function Diode_Fwd_Char(ilevel, start, stop, steps) --Configure instrument to source a bias
current
--and perform a current sweep from start to stop in a user-defined number of steps. Returns
measured
--voltage and current values.

       --Global variables
```

```
local l_irange = 100E-2 --Current source range
local l_ilevel = ilevel --Initial source value
local l_vcmpl = 6 --Source compliance

--Shared local variables
local l_nplc = 1 --Integration rate of measurement

--Local sweep variables
local l_start = start --Sweep start current
local l_stop =  stop --Sweep stop current
local l_steps = steps --Number of steps in sweep
local l_delay = 0.001 --Source delay

--Default values and level check
if (l_ilevel == nil) then --Use default value
      l_ilevel = 0
end --if

if (l_ilevel > 0.1) then --Coerce value
      l_ilevel = 0.1
end --if

if (l_start == nil) then --Use default value
      l_start = 1E-4
end --if

if (l_start > 1) then --Coerce value
      l_start = 1
end --if

if (l_stop == nil) then --Use default value
      l_stop = 1E-2
end --if

if (l_stop > 1) then --Coerce value
      l_stop = 1
end --if

if (l_steps == nil) then --Use default value
      l_steps = 100
end --if

if (l_steps > 1E3) then --Coerce value
      l_steps = 1E3
end --if

local l_step = (l_stop - l_start)/ (l_steps - 1) --Current step size
local l_source_val = l_start --Source value during sweep
local l_i = 1 --Iteration variable
```

```
	--Data tables
	local l_curr = {} --Create data table for sourced current
	local l_volt = {} --Create data table for measured voltage

	smua.reset() --Reset SMU
	errorqueue.clear() --Clear the error queue

	--Configure SMUA source and measure settings
	smua.source.func = smua.OUTPUT_DCAMPS
smua.source.autorangei = smua.AUTORANGE_ON --Enable source autorange
smua.source.leveli = l_ilevel --Source
smua.source.limitv = l_vcmpl
smua.measure.autorangev = smua.AUTORANGE_ON --Enable measure autorange

smua.measure.nplc = l_nplc --Measurement integration rate

smua.source.output = smua.OUTPUT_ON --Enable Output

	--Execute sweep
	for l_i = 1, l_steps  do
		--smua.source.leveli = l_source_val
		delay(l_delay) --Wait before measurement
		l_volt[l_i] = smua.measure.v() --Measure voltage
		l_curr[l_i] = smua.measure.i() --Measure current
		l_source_val = l_source_val + l_step --Calculate new source value
		smua.source.leveli = l_source_val --Increment source
	end--for

	smua.source.output = smua.OUTPUT_OFF --Disable output
	smua.source.leveli = l_ilevel --Return source to bias level

	Print_Data(l_steps, l_volt, l_curr)

end--function Diode_Fwd_Chr()

function Print_Data(steps,volt,curr)
--Print Data to output queue

	--Local Variables
	local l_steps = steps
	local l_volt = volt
	local l_curr = curr

	print("Voltage Data (V):")

	for l_i = 1, l_steps  do
		print(l_volt[l_i])
	end

	print("")
```

```
	print("Source Current Data (A):")

	for l_i = 1, l_steps  do
		print(l_curr[l_i])
	end

end --function Print_Data()

--Diode_Fwd_Chr()
```

Program 3B. Diode Characterization Log Sweep

```
--[[
Diode_Fwd_Char_Log(): USES TABLES

This program performs a log sweep forward characterization test on a diode and prints
data.

Required equipment:

(1) Single-channel Keithley Series 2600 System SourceMeter instrument
(1) Silicon diode or equivalent

Running this script creates functions that can be used to measure the IV characteristics
of diodes.

	The functions created are:
		1. Diode_Fwd_Char_Log(ilevel, start, stop, points) --Default values ilevel =
0s,
                                                --start = 1ua, stop = 10ma
                                                        --points = 10
		2. Print_Data(steps,volt,curr)

	See detailed information listed in individual functions

To run:

1) From Test Script Builder
	- Right-click in the program window, select "Run as TSP"
	- At the TSP> prompt in the Instrument Control Panel, type Diode_Fwd_Char_Log()

2) From an external program
	- Send the entire program text as a string using standard GPIB Write calls.

Rev1: JAC 10.12.2007

]]--
```

```
------------------ Keithley TSP Function ------------------

function Diode_Fwd_Char_Log(ilevel, start, stop, points) --CConfigure instrument to source
a bias
 --current, and perform a logarithmic current sweep from start to stop in a user-defined
number of points per decade.
 --Returns measured voltage and current values.

       --Global variables
       local l_irange = 100E-2 --Current source range
       local l_ilevel = ilevel --Initial source value
       local l_vcmpl = 6 --Source compliance

       --Shared local variables
       local l_nplc = 1 --Integration rate of measurement

       --Local sweep variables
       local l_start = start --Sweep start current
       local l_stop =  stop --Sweep stop current
       local l_points = points --Number of steps in sweep
       local l_delay = 0.01 --Source delay

       --Default values and level check
       if (l_ilevel == nil) then --Use default value
             l_ilevel = 0
       end --if

       if (l_ilevel > 0.1) then --Coerce value
             l_ilevel = 0.1
       end --if

       if (l_start == nil) then --Use default value
             l_start = 1E-6
       end --if

       if (l_start > 1) then --Coerce value
             l_start = 1
       end --if

       if (l_stop == nil) then --Use default value
             l_stop = 1E-2
       end --if

       if (l_stop > 1) then --Coerce value
             l_stop = 1
       end --if

       if (l_points == nil) then --Use default value
             l_points = 10
```

```
	end --if

	if (l_points > 1E3) then --Coerce value
		l_points = 1E3
	end --if

	local  l_step = (math.log10(l_stop) - math.log10(l_start))/(l_points - 1)
--Current step size

	local l_source_val = math.log10(l_start) --Source value during sweep

	local l_i = 1 --Iteration variable

	--Data tables
	local l_curr = {} --Create data table for sourced current
	local l_volt = {} --Create data table for measured voltage

	smua.reset() --Reset SMU
	errorqueue.clear() --Clear the error queue

	--Configure SMUA source and measure settings
	smua.source.func = smua.OUTPUT_DCAMPS
smua.source.autorangei = smua.AUTORANGE_ON --Enable source autorange
smua.source.leveli = l_ilevel --Source bias
smua.source.limitv = l_vcmpl
smua.measure.autorangev = smua.AUTORANGE_ON --Enable measure autorange

smua.measure.nplc = l_nplc --Measurement integration rate

	smua.source.output = smua.OUTPUT_ON --Enable Output

	--Execute sweep
	for l_i = 1, l_points  do

		smua.source.leveli = math.pow(10, l_source_val)  -- Program source to sweep
level.
		delay(l_delay) --Wait before measurement
		l_volt[l_i] = smua.measure.v() --Measure voltage
		l_curr[l_i] = smua.measure.i() --Measure current
			l_source_val = l_source_val + l_step --Increment source value
	end--for

	smua.source.output = smua.OUTPUT_OFF --Disable output
	smua.source.leveli = l_ilevel --Return source to bias level

	Print_Data(l_points, l_volt, l_curr)

end--function Diode_Fwd_Chr()

function Print_Data(points,volt,curr)
```

```
--Print Data to output queue

       --Local Variables
       local l_points = points
       local l_volt = volt
       local l_curr = curr

       print("Voltage Data (V):")

       for l_i = 1, l_points  do
             print(l_volt[l_i])
       end

print("")
       print("Source Current Data (A):")

       for l_i = 1, l_points  do
             print(l_curr[l_i])
end

end --function Print_Data()

--Diode_Fwd_Chr_Log()
```

Program 3C. Diode Characterization Pulsed Sweep

```
--[[
Diode_Fwd_Char_Pulse(): USES TABLES

This program performs a forward characterization test on a diode using a pulsed source and
prints data. The default is a 50% duty cycle (i.e., ton = toff)

Required equipment:

(1) Single-channel Keithley Series 2600 System SourceMeter instrument
(1) Silicon diode or equivalent

Running this script creates functions that can be used to measure the IV characteristics
of diodes.

       The functions created are:
1. Diode_Fwd_Char_Pulse(ilevel, start, stop, ton, toff, steps) --Default values ilevel =
0s, start =
                                                                --1ma, stop = 10ma, ton =
10ms,
   --toff = 10ms, steps = 10
       2. Print_Data(steps,volt,curr)

       See detailed information listed in individual functions
```

```
To run:

1) From Test Script Builder
      - Right-click in the program window, select "Run as TSP"
      - At the TSP> prompt in the Instrument Control Panel, type Diode_Fwd_Char()

2) From an external program
      - Send the entire program text as a string using standard GPIB Write calls.

Rev1: JAC 5.22.2007

]]--


------------------ Keithley TSP Function ------------------

function Diode_Fwd_Char_Pulse(ilevel, start, stop, ton, toff, steps) --Configure instrument
to source a
--bias current,and perform a pulsed current sweep from start to stop in a user-defined
number of steps.
--Each pulse is on ton (s) and off toff (s) and returns to the bias level during the toff
time.
--Returns measured voltage and current values.

      --Global variables
      local l_irange = 100E-2 --Current source range
      local l_ilevel = ilevel --Initial source value
      local l_vcmpl = 6 --Source compliance

      --Shared local variables
      local l_nplc = 0.1 --Integration rate of measurement

      --Local sweep variables
      local l_start = start --Sweep start current
      local l_stop =  stop --Sweep stop current
      local l_steps = steps --Number of steps in sweep
      local l_delay = 0.001 --Source delay
      local l_ton = ton --Pulse on duration
      local l_toff = toff --Pulse off duration
      local l_tonwm --Adjusted Pulse on duration to accomodate Measurement Duration

      --Default values and level check
      if (l_ilevel == nil) then --Use default value
            l_ilevel = 0
      end --if

      if (l_ilevel > 1E-1) then --Coerce value
            l_ilevel = 1E-1
      end --if
```

```
if (l_start == nil) then --Use default value
      l_start = 1E-3
end --if

if (l_start > 0.1) then --Coerce value
      l_start = 0.1
end --if

if (l_stop == nil) then --Use default value
      l_stop = 1E-2
end --if

if (l_stop > 0.1) then --Coerce value
      l_stop = 0.1
end --if

if (l_ton == nil) then --Use default value
      l_ton = 10E-3
end --if

if (l_ton > 1E-1) then --Coerce value
      l_ton = 1E-1
end --if

if (l_toff == nil) then --Use default value
      l_toff = 10E-3
end --if

if (l_toff > 1E-1) then --Coerce value
      l_toff = 1E-1
end --if

if (l_steps == nil) then --Use default value
      l_steps = 100
end --if

if (l_steps > 1E3) then --Coerce value
      l_steps = 1E3
end --if

local l_step = (l_stop - l_start)/ (l_steps - 1) --Current step size
local l_source_val = l_start --Source value during sweep
local l_i = 1 --Iteration variable

--Data tables
local l_curr = {} --Create data table for sourced current
local l_volt = {} --Create data table for measured voltage
```

```
      l_tonwm = l_ton - (2*smua.measure.nplc/localnode.linefreq) - 250E-6 --Adjust pulse
duration by
--accounting for measurement time

      smua.reset() --Reset SMU
      errorqueue.clear() --Clear the error queue

      --Configure SMUA source and measure settings
      smua.source.func = smua.OUTPUT_DCAMPS
smua.source.autorangei = smua.AUTORANGE_ON --Enable source autorange
       smua.source.leveli = l_ilevel --Source
smua.source.limitv = l_vcmpl
smua.measure.autorangev = smua.AUTORANGE_ON --Enable measure autorange

smua.measure.nplc = l_nplc --Measurement integration rate

      smua.source.output = smua.OUTPUT_ON --Enable Output

      --Execute sweep
      for l_i = 1, l_steps  do
            smua.source.leveli = l_source_val
            delay(l_tonwm)            -- Wait pulse time - measurement & overhead time.
            l_volt[l_i] = smua.measure.v() --Measure voltage
            l_curr[l_i] = smua.measure.i() --Measure current
             smua.source.leveli = l_ilevel  -- Return source to bias level.
             delay(l_toff)               -- Wait pulse off time.
            l_source_val = l_source_val + l_step --Calculate new source value
            smua.source.leveli = l_source_val --Increment source --]]
      end--for

      smua.source.output = smua.OUTPUT_OFF --Disable output
      smua.source.leveli = l_ilevel --Return source to bias level

      Print_Data(l_steps, l_volt, l_curr)

end--function Diode_Fwd_Chr()

function Print_Data(steps,volt,curr)
--Print Data to output queue

      --Local Variables
      local l_steps = steps
      local l_volt = volt
      local l_curr = curr

      print("Voltage Data (V):")

      for l_i = 1, l_steps  do
            print(l_volt[l_i])
      end
```

```
        print("")
        print("Source Current Data (A):")

        for l_i = 1, l_steps  do
              print(l_curr[l_i])
        end

end --function Print_Data()

--Diode_Fwd_Chr_Pulse()
```

Section 3. Bipolar Transistor Tests

Program 4. Common-Emitter Characteristics

```
--[[
BJT_Comm_Emit(): USES TABLES

This program applies a bias to the base of a BJT (Ib) and sweeps voltage on the collector/
emitter (VCE). The VCE, IB, and IC are then printed.

Required equipment:

(1) Dual-channel Series 2600 System SourceMeter instrument
(1) 2N5089 NPN Transistor

Running this script creates functions that can be used to measure the common emitter
characteristics of transistors. The default values are for an NPN transistor type 2N5089.

       The functions created are:
       1. BJT_Comm_Emit(istart, istop, isteps, vstart, vstop, vsteps)
                 --Default values istart = 10uA, istop = 50uA, isteps = 5, vstart = 0V,
vstop = 10V, vsteps = 100
       2. Print_Data(isteps,vsteps, ce_volt,ce_curr, base_curr)

       See detailed information listed in individual functions.

1) From Test Script Builder
- At the TSP> prompt in the Instrument Control Panel, type BJT_Comm_Emit()

2) From an external program
       - Send the entire program text as a string using standard GPIB Write calls.


Rev1: JAC 5.22.2007

]]--


------------------ Keithley TSP Function ------------------

function BJT_Comm_Emit(istart, istop, isteps, vstart, vstop, vsteps) --Configure SMUB to
source a bias
--current on the base and SMUA performs a voltage sweep on the Collector-Emitter from
start to stop in a
--user-defined number of steps.
--SMUB then increments to next bias value and continues to stop value.
--Returns measured voltage and current values.

       --Global variables
       local l_irange = 100E-6 --Base current source range
```

```
local l_vcmpl = 1 --Base source compliance

local l_vrange = 40 --Collector-emitter voltage source range
local l_icmpl = 100E-3 --Collector-emitter source compliance

--Shared local variables
local l_nplc = 1 --Integration rate of measurement

--Local sweep variables
local l_istart = istart --Base sweep start current
local l_istop =  istop --Base sweep stop current
local l_isteps = isteps --Number of steps in sweep

local l_vstart = vstart --Collector-emitter sweep start voltage
local l_vstop =  vstop --Collector-emitter sweep stop voltage
local l_vsteps = vsteps --Number of steps in sweep

--Default values and level check
if (l_istart == nil) then --Use default value
      l_istart = 10E-6
end --if

if (l_istart > 100E-6) then --Coerce value
      l_istart = 100E-6
end --if

if (l_istop == nil) then --Use default value
      l_istop = 50E-6
end --if

if (l_istop > 500E-6) then --Coerce value
      l_istop = 500E-6
end --if

if (l_isteps == nil) then --Use default value
      l_isteps = 5
end --if

if (l_isteps > 100) then --Coerce value
      l_isteps = 100
end --if

local l_istep = (l_istop - l_istart)/ (l_isteps - 1) --Current step size
local l_isource_val = l_istart --Source value during sweep
local l_i = 1 --Iteration variable

if (l_vstart == nil) then --Use default value
      l_vstart = 0
end --if
```

```
	if (l_vstart > 100E-3) then --Coerce value
		l_vstart = 100E-3
	end --if

	if (l_vstop == nil) then --Use default value
		l_vstop = 10
	end --if

	if (l_vstop > 40) then --Coerce value
		l_vstop = 40
	end --if

	if (l_vsteps == nil) then --Use default value
		l_vsteps = 100
	end --if

	if (l_vsteps > 2E+2) then --Coerce value
		l_vsteps = 2E+2
	end --if

	local l_vstep = (l_vstop - l_vstart)/ (l_vsteps - 1) --Voltage step size
	local l_vsource_val = l_vstart --Source value during sweep
	local l_v = 1 --Iteration variable

	--Data tables
	local l_base_curr = {} --Create data table for sourced current
	local l_ce_volt = {} --Create data table for collector-emitter measured voltage
	local l_ce_curr = {} --Create data table for collector-emitter measured current

	smua.reset() --Reset SMU
	smub.reset() --Reset SMU

	errorqueue.clear() --Clear the error queue

	--Configure Collector/Emitter (SMUA) source and measure settings
	smua.source.func = smua.OUTPUT_DCVOLTS
	smua.source.autorangev = smua.AUTORANGE_ON --Enable source autorange
smua.source.levelv = 0
smua.source.limiti = l_icmpl
smua.measure.autorangei = smua.AUTORANGE_ON --Enable measure autorange

smua.measure.autozero = smua.AUTOZERO_AUTO
smua.measure.nplc = l_nplc --Measurement integration rate

	smua.source.output = smua.OUTPUT_ON --Enable Output

	--Configure Base (SMUB) source and measure settings
	smub.source.func = smub.OUTPUT_DCAMPS
smub.source.autorangei = smub.AUTORANGE_ON --Enable source autorange
smub.source.leveli = 0
```

```
smub.source.limitv = l_vcmpl
smub.measure.autorangev = smub.AUTORANGE_ON --Enable measure autorange

smub.measure.autozero = smub.AUTOZERO_AUTO
smub.measure.nplc = l_nplc --Measurement integration rate

       smub.source.output = smub.OUTPUT_ON --Enable Output

       --Execute sweep
       for l_i = 1, l_isteps  do

              smub.source.leveli = l_isource_val

              l_ce_volt[l_i] = {} --Create new row in table
              l_ce_curr[l_i] = {} --Create new row in table

              l_base_curr[l_i] = smub.measure.i() --Measure base current

              for l_v = 1,l_vsteps do


                     if (l_v == 1) then --Intialize start source value
                            l_vsource_val = l_vstart
                     end --if

                     delay(0.001) --Delay
                     l_ce_volt[l_i][l_v] = smua.measure.v() --Measure voltage
                     l_ce_curr[l_i][l_v] = smua.measure.i() --Measure current

                     l_vsource_val = l_vsource_val + l_vstep --Calculate new source value


                     if (l_v == l_vsteps) then --Reinitialize voltage value after last
iteration
                            l_vsource_val = l_vstart
                     end --if

                     smua.source.levelv = l_vsource_val --Increment source

              end --for

              l_isource_val = l_isource_val + l_istep --Calculate new source value

       end--for

       smua.source.output = smua.OUTPUT_OFF --Disable output
       smub.source.output = smub.OUTPUT_OFF --Disable output

       smua.source.levelv = 0 --Return source to bias level
       smub.source.leveli = 0 --Return source to bias level
```

```
      Print_Data(l_isteps,l_vsteps, l_ce_volt, l_ce_curr, l_base_curr)

end--function BJT_Comm_Emit()

function Print_Data(isteps,vsteps, ce_volt,ce_curr, base_curr)
--Print Data to output queue

      --Local Variables
      local l_isteps = isteps
      local l_vsteps = vsteps
      local l_i = 1 --Iteration variable
      local l_v = 1 --Iteration variable
      local l_ce_volt = ce_volt
      local l_ce_curr = ce_curr
      local l_base_curr = base_curr


      for l_i = 1, l_isteps  do
            print("")
            print("Base Current Bias", l_base_curr[l_i])
            print("Emitter Voltage (V)","Emitter Current (A)")

            for l_v = 1, l_vsteps  do
                  print(l_ce_volt[l_i][l_v], l_ce_curr[l_i][l_v])
      end --for
    end --for

 end --function Print_Data()

--BJT_Comm_Emit()
```

Program 5. Gummel Plot

```
--[[
Gummel(): USES TABLES

This program performs a series of voltage sweeps on the base-emitter (VBE) of a BJT at a
fixed collector-emitter voltage (VCE). The base-emitter (IB) and collector-emitter (IC)
currents are measured and printed.

Required equipment:

(1) Dual-channel Keithley Series 2600 System SourceMeter instrument
(1) 2N5089 NPN Transistor

Running this script creates functions that can be used to create a Gummel plot of
transistors. The default values are for an NPN transistor type 2N5089.

       The functions created are:
              1. Gummel(vbestart, vbestop, vbesteps, vcebias)
                 --Default values vbestart = 0V, vbestop = 0.7V, vbesteps = 70, vcebias =
10V
              2. Print_Data(vbesteps,vbe, vcebias, ic, ib)

       See detailed information listed in individual functions.

To run:

1) From Test Script Builder
       - Right-click in the program window, select "Run as TSP"
       - At the TSP> prompt in the Instrument Control Panel, type Gummel()

2) From an external program
       - Send the entire program text as a string using standard GPIB Write calls.

Rev1: JAC 5.30.2007

]]--


------------------ Keithley TSP Function ------------------

function Gummel(vbestart, vbestop, vbesteps, vcebias) --Configure SMUB to perform a voltage
sweep on the
--base (Vbe) from start to stop in a user-defined number of steps while SMUA performs a
fixed voltage bias on the
--collector-emitter. SMUA then increments to next bias value and continues to stop value.
--Returns measured Ib, Ic, and Vbe.

       --Global variables
       local l_icmpl = 100E-3 --Source compliance
```

```
--Shared local variables
local l_nplc = 1 --Integration rate of measurement

--Local sweep variables
local l_vbestart = vbestart --Base sweep start voltage
local l_vbestop =  vbestop --Base sweep stop voltage
local l_vbesteps = vbesteps --Number of steps in sweep

local l_vcebias = vcebias --Collector-emitter voltage

--Default values and level check
if (l_vbestart == nil) then --Use default value
      l_vbestart = 0
end --if

if (l_vbestart > 100E-6) then --Coerce value
      l_vbestart = 100E-6
end --if

if (l_vbestop == nil) then --Use default value
      l_vbestop = 700E-3
end --if

if (l_vbestop > 1) then --Coerce value
      l_vbestop = 1
end --if

if (l_vbesteps == nil) then --Use default value
      l_vbesteps = 70
end --if

if (l_vbesteps > 100) then --Coerce value
      l_vbesteps = 100
end --if

local l_vbestep = (l_vbestop - l_vbestart)/ (l_vbesteps - 1) --Vbe step size
local l_vbesource_val = l_vbestart --Source value during sweep
local l_vbe_i = 1 --Iteration variable

if (l_vce_bias == nil) then --Use default value
      l_vce_bias = 10
end --if

if (l_vce_bias > 40) then --Coerce value
      l_vce_bias = 40
end --if

--Data tables
local l_vbe = {} --Create data table for sourced voltage
```

```
	local l_ic = {} --Create data table for Ic
	local l_ib = {} --Create data table for Ib

	smua.reset() --Reset SMU
	smub.reset() --Reset SMU

	errorqueue.clear() --Clear the error queue

	--Configure Collector/Emitter (SMUA) source and measure settings
	smua.source.func = smua.OUTPUT_DCVOLTS
smua.source.autorangev = smua.AUTORANGE_ON --Enable source autorange
	smua.source.levelv = 0
	smua.source.limiti = l_icmpl
	smua.measure.autorangei = smua.AUTORANGE_ON --Enable measure autorange

	smua.measure.autozero = smua.AUTOZERO_AUTO
	smua.measure.nplc = l_nplc --Measurement integration rate

	smua.source.output = smua.OUTPUT_ON --Enable Output

	--Configure Base (SMUB) source and measure settings
	smub.source.func = smub.OUTPUT_DCVOLTS
	smub.source.autorangev = smub.AUTORANGE_ON --Enable source autorange
	smub.source.levelv = 0
	smub.source.limiti = l_icmpl
	smub.measure.autorangev = smub.AUTORANGE_ON --Enable measure autorange

	smub.measure.autozero = smub.AUTOZERO_AUTO
	smub.measure.nplc = l_nplc --Measurement integration rate

	smub.source.output = smub.OUTPUT_ON --Enable Output

	smua.source.levelv = l_vce_bias

	--Execute sweep
	for l_vbe_i = 1,l_vbesteps do

		if (l_vbe_i == 1) then --Intialize start source value
			l_vbesource_val = l_vbestart
		end --if

		delay(0.01) --Delay
		l_vbe[l_vbe_i] = smub.measure.v() --Measure Vbe
		l_ib[l_vbe_i] = smub.measure.i() --Measure Ib
		l_ic[l_vbe_i] = smua.measure.i() --Measure Ic

		l_vbesource_val = l_vbesource_val + l_vbestep --Calculate new source value
```

```
            if (l_vbe_i == l_vbesteps) then --Reinitialize voltage value after last
iteration
                  l_vbesource_val = l_vbestart
            end --if

            smub.source.levelv = l_vbesource_val --Increment source

      end --for

      smua.source.output = smua.OUTPUT_OFF --Disable output
      smub.source.output = smub.OUTPUT_OFF --Disable output

      smua.source.levelv = 0 --Return source to bias level
      smub.source.levelv = 0 --Return source to bias level

      Print_Data(l_vbesteps, l_vbe,l_vce_bias, l_ic, l_ib)

end--function Gummel()

function Print_Data(vbesteps,vbe, vcebias, ic, ib)
--Print Data to output queue

      --Local Variables
      local l_vbesteps = vbesteps
      local l_vbe_i = 1 --Iteration variable
      local l_vbe = vbe
      local l_vce_bias = vcebias
      local l_ic = ic
      local l_ib = ib

      print("")
      print("Vce", l_vce_bias)
      print("Vbe (V)","Ib (A)","Ic (A)")

      for l_vbe_i = 1, l_vbesteps  do
            print(l_vbe[l_vbe_i],l_ic[l_vbe_i], l_ib[l_vbe_i])
      end --for

 end --function Print_Data()

--Gummel()
```

Section 6. High Power Tests

Program 6. Current Gain

Program 6A. Current Gain (Search Method)

```
--[[
DC_Gain_Search():

This program performs a binary search on the base current (IB) of a BJT at a fixed
collector-emitter voltage (VCE). The base-emitter (IB) and collector-emitter (IC) currents
are measured and the IB, IC, and DC gain values are printed.

Required equipment:

(1) Dual-channel Keithley Series 2600 System SourceMeter instrument
(1) 2N5089 NPN transistor

Running this script creates functions that can be used to create a DC gain search of
transistors. The default values are for an NPN transistor type 2N5089.

       The functions created are:
              1. DC_Gain_Search(vcesource, lowib, highib, targetic)
                 --Default values vcesource = 5V, lowib = 1e-9A, highib = 100e-7A,
targetic = 100e-6A
              2. Check_Comp()

       See detailed information listed in individual functions.

To run:

1) From Test Script Builder
       - Right-click in the program window, select "Run as TSP"
       - At the TSP> prompt in the Instrument Control Panel, type DC_Gain_Search()

2) From an external program
       - Send the entire program text as a string using standard GPIB Write calls.


Rev1: JAC 6.11.2007

]]--


------------------ Keithley TSP Function ------------------

function DC_Gain_Search(vcesource, lowib, highib, targetic) --Configure SMUB to source a
user-defined
--current on the base (Ib) while SMUA performs a fixed voltage bias on the collector-
emitter. SMUB then performs a
```

```
--binary search between a Maximum and Minimum Ib value, and the collector current is
measured. If measured value is
--outside the tolerance, search is performed again until the value falls within the
specified range or the iteration
--limit is reached.
--Returns measured Ib, Ic, and the DC Gain/Beta.

    local l_k --binary search iteration variable
    local l_k_max = 20 --Maximum loop iteration
    local l_vce_source = vcesource --VCEsource value
    local l_high_ib = highib --Start Ib high limit
    local l_low_ib = lowib --Start Ib lo limit
    local l_target_ic = targetic --Target Ic for binary search
    local l_nplc = 1

    local l_ic_meas
    local l_ib_source --Base current
       local l_beta_meas

       --Default values and level check
       if (l_vce_source == nil) then --Use default value
           l_vce_source = 5
       end --if

       if (l_low_ib == nil) then --Use default value
           l_low_ib = 1e-9
       end --if

       if (l_high_ib == nil) then --Use default value
           l_high_ib = 100E-7
       end --if

       if (l_target_ic == nil) then --Use default value
           l_target_ic = 100e-6
       end --if

       smua.reset() --Reset SMU
       smub.reset() --Reset SMU

       errorqueue.clear() --Clear the error queue

    smua.measure.nplc = l_nplc --Measurement integration rate
    smub.measure.nplc = l_nplc --Measurement integration rate

    smua.source.func = smua.OUTPUT_DCVOLTS
    smua.source.autorangev = smua.AUTORANGE_ON --Enable source autorange
    smua.source.limiti = (100 * l_target_ic) --Set compliance value
    smua.measure.autorangev = smua.AUTORANGE_ON --Enable measure autorange

    smub.source.func = smub.OUTPUT_DCAMPS
```

```
    smub.source.autorangei = smub.AUTORANGE_ON --Enable source autorange
    smub.source.limiti = l_high_ib
    smub.source.limitv = 6
    smub.measure.autorangev = smub.AUTORANGE_ON --Enable measure autorange

       --Start test
    smua.source.levelv = l_vce_source --Set source level
    smub.source.leveli = 0 --Set source level

    smua.source.output = smua.OUTPUT_ON --Enable output
    smub.source.output = smub.OUTPUT_ON --Enable output

    delay(0.001) --Delay

    l_comp_val = Check_Comp() --check compliance

    if l_comp_val == true then --If unit is in compliance, end

    smua.source.output = smua.OUTPUT_OFF --Disable output
    smub.source.output = smub.OUTPUT_OFF --Disable output

    smua.source.levelv = 0 --Return source to 0
    smub.source.leveli = 0 --Return source to 0

    else

              --Search for the right base current
       l_k = 0

    repeat --Repeat search until measured Ic is within 5% of target, iteration maximum
reached, or
           --compliance.

              l_k = l_k + 1 --Increment

              l_ib_source = ((l_high_ib-l_low_ib)/2) + l_low_ib --Determine source value
(Binary Search)
              smub.source.leveli = l_ib_source --Program new source value
              delay(0.0001) --Source delay

              l_comp_val = Check_Comp() --check compliance

                     if l_comp_val == true then --If unit is in compliance, end
                     smua.source.output = smua.OUTPUT_OFF --Disable output
                     smub.source.output = smub.OUTPUT_OFF --Disable output

                     smua.source.levelv = 0 --Return source to 0
                     smub.source.leveli = 0 --Return source to 0

                     else
```

```
                        l_ic_meas = smua.measure.i() --Measure Ic

                        if (l_target_ic < l_ic_meas) then --Compare measurement with
target value
                        l_high_ib = l_ib_source
                    else
                        l_low_ib = l_ib_source
                    end --end if
            end --ifelse

if l_ic_meas == nil then --If no measurement taken, initialize to 0 to avoid arithmetic
--error in until statement below
                l_ic_meas = 0
            end --if

        until ((math.abs(l_ic_meas - l_target_ic) < (0.05*l_target_ic))or(l_k>l_k_max))
or(l_comp_val == true) --

          --iteration limit reached
        if (l_k > l_k_max) then
                    print("Iteration Limit Reached!!")
        end --end if

       smua.source.output = smua.OUTPUT_OFF --Disable output
       smub.source.output = smub.OUTPUT_OFF --Disable output

        l_beta_data = l_ic_meas/l_ib_source --Calculate gain

        print("Ic Data:", l_ic_meas) --Print Ic data
        print("Ib Data:", l_ib_source) --Print Ib
        print("Beta Data:",l_beta_data) --Print gain

   end --ifelse

end--function DC_Gain_Search()

function Check_Comp() --Function checks state of compliance, if true, prints warning and
returns
                    --to run_test()

       local l_comp_val = false --Initialize local variable

       l_comp_val = smua.source.compliance --Check compliance

       if l_comp_val == true then
              print("")
              print("SMU Source in Compliance!")
              print("Ensure proper connections, stable device, and settings are correct")
              print("Rerun Test")
              print("")
```

```
	end --if

	return l_comp_val

end --function Check_Comp()

--DC_Gain_Search()
```

Program 6B. Current Gain (Fast Method)

```
--[[
DC_Gain_Fast()

This program applies a bias to the collector/emitter of a BJT (Vce) and sweeps current on
the emitter (IE). The gain for each emitter value is then printed.

Required equipment:

(1) Dual-channel Keithley Series 2600 System SourceMeter instrument
(1) 2N5089 NPN Transistor

Running this script creates functions that can be used to measure the gain characteristics
of transistors. The default values are for an NPN transistor type 2N5089.

	The functions created are:
		1. DC_Gain_Fast(vcesource, istart, istop, isteps)
		   --Default values vcesource = 10V, istart = 1mA, istop = 10mA, isteps = 10
		2. Print_Data(isteps, emitter_curr, base_curr)

	See detailed information listed in individual functions.

1) From Test Script Builder
	- At the TSP> prompt in the Instrument Control Panel, type DC_Gain_Fast()

2) From an external program
	- Send the entire program text as a string using standard GPIB Write calls.


Rev1: JAC 6.11.2007

]]--


------------------ Keithley TSP Function ------------------

function DC_Gain_Fast(vcesource, istart, istop, isteps) --Configure SMUB to source a bias
voltage
--on the base and SMUA performs a current sweep on the emitter from start to stop in a
user-defined number of steps.
```

```
--Returns gain values.

	--Global variables
	local l_irange = 100e-6 --Base current source range
	local l_vcmpl = 11 --Base source compliance

	local l_vrange = 40 --Collector-emitter voltage source range
	local l_icmpl = 100e-3 --Collector-emitter source compliance

	--Shared local variables
	local l_nplc = 1 --Integration rate of measurement

	--Local sweep variables
	local l_istart = istart --Base sweep start current
	local l_istop =  istop --Base sweep stop current
	local l_isteps = isteps --Number of steps in sweep

	local l_vce_source = vcesource --Vce source value

	--Default values and level check
	if (l_vce_source == nil) then --Use default value
		l_vce_source = -10
	end --if

	if (l_vce_source > 0) then --Coerce value
		l_vce_source = -l_vce_source
	end --if

	if (l_istart == nil) then --Use default value
		l_istart = -1e-3
	end --if

	if (l_istart > 0) then --Coerce value
		l_istart = -l_istart
	end --if

	if (l_istop == nil) then --Use default value
		l_istop = -10e-3
	end --if

	if (l_istop > 0) then --Coerce value
		l_istop = -i_stop
	end --if

	if (l_isteps == nil) then --Use default value
		l_isteps = 10
	end --if

	if (l_isteps > 100) then --Coerce value
		l_isteps = 100
```

```
	end --if

	local l_istep = (l_istop - l_istart)/ (l_isteps - 1) --Current step size
	local l_isource_val = l_istart --Source value during sweep
	local l_i = 1 --Iteration variable

	--Data tables
	local l_base_curr = {} --Create data table for sourced current
	local l_emitter_curr = {} --Create data table for emitter current

	smua.reset() --Reset SMU
	smub.reset() --Reset SMU

	errorqueue.clear() --Clear the error queue

	--Configure emitter current (SMUA) source and measure settings
	smua.source.func = smua.OUTPUT_DCAMPS
smua.source.autorangei = smua.AUTORANGE_ON --Enable source autorange
	smua.source.leveli = 0
	smua.source.limitv = l_vcmpl
	smua.source.output = smua.OUTPUT_ON --Enable Output

	--Configure collector/emitter (SMUB) source and measure settings
	smub.source.func = smub.OUTPUT_DCVOLTS
	smub.source.autorangei = smub.AUTORANGE_ON --Enable source autorang
	smub.source.levelv = 0
	smub.source.limiti = l_icmpl
	smub.measure.autorangei = smub.AUTORANGE_ON --Enable measure autorange

	smub.measure.autozero = smub.AUTOZERO_AUTO
	smub.measure.nplc = l_nplc --Measurement integration rate

	smub.source.output = smub.OUTPUT_ON --Enable Output

	smub.source.levelv = l_vce_source --Program source

	--Execute sweep
	for l_i = 1, l_isteps  do

		smua.source.leveli = l_isource_val

		delay(0.01)

		l_base_curr[l_i] = smub.measure.i() --Measure base current

		l_emitter_curr[l_i] = smua.measure.i() --Measure emitter current

		l_isource_val = l_isource_val + l_istep --Calculate new source value

	end--for
```

```
	smua.source.output = smua.OUTPUT_OFF --Disable output
	smub.source.output = smub.OUTPUT_OFF --Disable output

	smua.source.levelv = 0 --Return source to bias level
	smub.source.leveli = 0 --Return source to bias level

	Print_Data(l_isteps, l_emitter_curr, l_base_curr)

end--function DC_Gain_Fast()

function Print_Data(isteps, emitter_curr, base_curr)
--Print Data to output queue

	--Local Variables
	local l_isteps = isteps
	local l_i = 1 --Iteration variable
	local l_emitter_curr = emitter_curr
	local l_base_curr = base_curr
	local l_gain = {} --Gain variable

	print("")
	print("Base Current (A)", "Emitter Current (A)", "Gain")

	for l_i = 1, l_isteps  do

	l_gain[l_i] = (math.abs(l_emitter_curr[l_i]) - math.abs(l_base_curr[l_i]))/math.
abs(l_base_curr[l_i]) ----Calculate gain

		print(math.abs(l_base_curr[l_i]), math.abs(l_emitter_curr[l_i]), l_
gain[l_i])

	end --for

 end --function Print_Data()

--DC_Gain_Fast()
```

Program 7. AC Current Gain

```
--[[
AC_Gain():

This program sources two base currents (IB) on a BJT at a fixed collector-emitter voltage
(VCE). The base-emitter (IB) and collector-emitter (IC) currents are measured and the IB,
IC, and AC gain values are printed.

Required equipment:

(1) Dual-channel Keithley Series 2600 System SourceMeter instrument
(1) 2N5089 NPN transistor

Running this script creates functions that can be used to perform a differential gain
measurement on transistors. The default values are for an NPN transistor type 2N5089.

      The functions created are:
            1. AC_Gain(vcesource, ib1, ib2)
               --Default values vcesource = 5V, ib1 = 1e-7A, ib2= 2e-7A
            2. Check_Comp()

      See detailed information listed in individual functions.

To run:

1) From Test Script Builder
      - Right-click in the program window, select "Run as TSP"
      - At the TSP> prompt in the Instrument Control Panel, type AC_Gain()

2) From an external program
      - Send the entire program text as a string using standard GPIB Write calls.


Rev1: JAC 6.12.2007

]]--


------------------ Keithley TSP Function ------------------

function AC_Gain(vcesource, ib1, ib2) --Configure SMUB to source a user-defined current on
the base (Ib)
--while SMUA performs a fixed voltage bias on the collector-emitter and the Ic is measured.
--SMUB then steps to the next base current and the Ic is measured.
--The AC Gain is then calculated.
--Returns measured Ib1, Ib2, Ic1, Ic2 and the AC Gain/Beta.

      local l_vce_source = vcesource --VCEsource value
    local l_ib1 = ib1 --Ib 1 source value
```

```
    local l_ib2 = ib2 --Ib 2 source value
    local l_nplc = 1

    local l_ic_meas1 = 0 --Ic measurement
    local l_ic_meas2 = 0 --Ic measurement
    local l_beta_meas --Gain calculation variable

       --Default values and level check
       if (l_vce_source == nil) then --Use default value
           l_vce_source = 5
       end --if

       if (l_ib1 == nil) then --Use default value
           l_ib1 = 1.45e-7
       end --if

       if (l_ib2 == nil) then --Use default value
           l_ib2 = 1.6e-7
       end --if

       smua.reset() --Reset SMU
       smub.reset() --Reset SMU

       errorqueue.clear() --Clear the error queue

       smua.measure.nplc = l_nplc --Measurement integration rate
       smub.measure.nplc = l_nplc --Measurement integration rate

       smua.source.func = smua.OUTPUT_DCVOLTS
       smua.source.autorangev = smua.AUTORANGE_ON --Enable source autorange

       smua.measure.autorangev = smua.AUTORANGE_ON --Enable measure autorange

       smub.source.func = smub.OUTPUT_DCAMPS
       smub.source.autorangei = smub.AUTORANGE_ON --Enable source autorange
       smub.source.limitv = 6
       smub.measure.autorangev = smub.AUTORANGE_ON --Enable measure autorange

       --Start test
       smua.source.levelv = l_vce_source --Set source level
       smub.source.leveli = 0 --Set source level

       smua.source.output = smua.OUTPUT_ON --Enable output
       smub.source.output = smub.OUTPUT_ON --Enable output

delay(0.001) --Delay


       smub.source.leveli = l_ib1 --Program new source value
       delay(0.001) --Source delay
```

```
	--l_comp_val = Check_Comp() --check compliance

	if l_comp_val == true then --If unit is in compliance, end
		smua.source.output = smua.OUTPUT_OFF --Disable output
		smub.source.output = smub.OUTPUT_OFF --Disable output

		smua.source.levelv = 0 --Return source to 0
		smub.source.leveli = 0 --Return source to 0

	else
	l_ic_meas1 = smua.measure.i() --Measure Ic 1

		smub.source.leveli = l_ib2 --Program new source value

		l_ic_meas2 = smua.measure.i() --Measure Ic 2

	l_beta_data = (l_ic_meas2 - l_ic_meas1)/(l_ib2 - l_ib1) --Calculate gain

print("") -
print("Ib 1(A) ", "Ic 1(A) ", "Ib 2(A) ", "Ic 2(A) ")
print(l_ib1, l_ic_meas1, l_ib2, l_ic_meas2) --Print Ib and Ic data
		print("") --
		print("Differential Gain")
		print(l_beta_data) --Print gain

	end --ifelse

	smua.source.output = smua.OUTPUT_OFF --Disable output
	smub.source.output = smub.OUTPUT_OFF --Disable output

end --function AC_Gain()

function Check_Comp() --Function checks state of compliance, if true, prints warning and
returns
			--to run_test()

	local l_comp_val = false --Initialize local variable

	l_comp_val = smua.source.compliance --Check compliance

	if l_comp_val == true then
		print("")
		print("SMU Source in Compliance!")
		print("Ensure proper connections, stable device, and settings are correct")
		print("Rerun Test")
		print("")
	end --if

	return l_comp_val
```

```
end --function Check_Comp()

--AC_Gain()
```

Program 8. Transistor Leakage (ICEO)

```
--[[
Iceo(): USES TABLES

This program sweeps the voltage on the collector/emitter (VCE) of a BJT with an open base.
The VCEO and ICEO values are then printed.

Required equipment:

(1) Single-channel Keithley Series 2600 System SourceMeter instrument
(1) 2N3904NPN transistor

Running this script creates functions that can be used to measure open base voltage and
current characteristics of transistors. The default values are for an NPN transistor type
2N3904.

      The functions created are:
            1. Iceo(vstart, vstop, vsteps)
               --Default values vstart = 0V, vstop = 10V, vsteps = 100
            2. Print_Data(vsteps, ce_volt,ce_curr)

      See detailed information listed in individual functions.

1) From Test Script Builder
      - At the TSP> prompt in the Instrument Control Panel, type Vceo()

2) From an external program
      - Send the entire program text as a string using standard GPIB Write calls.


Rev1: JAC 6.12.2007

]]--


------------------ Keithley TSP Function ------------------

function Iceo(vstart, vstop, vsteps) --Configure SMUA to perform a voltage
--sweep from start to stop in a user-defined number on the collector/emitter of a BJT with
an open base. The collector --current (Iceo) is measured at each voltage value.
--Returns programmed voltage and measured current values.
```

```
	--Global variables
	local l_irange = 100E-6 --Base current source range
	local l_vcmpl = 1 --Base source compliance

	local l_vrange = 40 --Collector-emitter voltage source range
	local l_icmpl = 100E-3 --Collector-emitter source compliance

	--Shared local variables
	local l_nplc = 1 --Integration rate of measurement

	--Local sweep variables
	local l_vstart = vstart --Collector-emitter sweep start voltage
	local l_vstop =  vstop --Collector-emitter sweep stop voltage
	local l_vsteps = vsteps --Number of steps in sweep

	--Default values and level check
	local l_i = 1 --Iteration variable

	if (l_vstart == nil) then --Use default value
		l_vstart = 0
	end --if

	if (l_vstart > 100E-3) then --Coerce value
		l_vstart = 100E-3
	end --if

	if (l_vstop == nil) then --Use default value
		l_vstop = 10
	end --if

	if (l_vstop > 40) then --Coerce value
		l_vstop = 40
	end --if

	if (l_vsteps == nil) then --Use default value
		l_vsteps = 100
	end --if

	if (l_vsteps > 2E+2) then --Coerce value
		l_vsteps = 2E+2
	end --if

	local l_vstep = (l_vstop - l_vstart)/ (l_vsteps - 1) --Voltage step size
	local l_vsource_val = l_vstart --Source value during sweep

	--Data tables
	local l_ce_volt = {} --Create data table for collector-emitter measured voltage
local l_ce_curr = {} --Create data table for collector-emitter measured current

	smua.reset() --Reset SMU
```

```
	errorqueue.clear() --Clear the error queue

	--Configure Collector/Emitter (SMUA) source and measure settings
	smua.source.func = smua.OUTPUT_DCVOLTS
smua.source.autorangev = smua.AUTORANGE_ON --Enable source autorange
smua.source.levelv = 0     --Source 0V
smua.source.limiti = l_icmpl --Set compliance level
smua.measure.autorangei = smua.AUTORANGE_ON --Enable measure autorange

smua.measure.autozero = smua.AUTOZERO_AUTO
	smua.measure.nplc = l_nplc --Measurement integration rate

	smua.source.output = smua.OUTPUT_ON --Enable Output

	smua.source.levelv = l_vsource_val --Program source value

	--Execute sweep
	for l_i = 1, l_vsteps  do


		delay(0.01)
		l_ce_volt[l_i] = l_vsource_val --Save programmed voltage
		l_ce_curr[l_i] = smua.measure.i() --Measure current

		l_vsource_val = l_vsource_val + l_vstep --Calculate new source value

		smua.source.levelv = l_vsource_val --Increment source

	end--for

	smua.source.output = smua.OUTPUT_OFF --Disable output
	smua.source.levelv = 0 --Return source to bias level

	Print_Data(l_vsteps,l_ce_volt, l_ce_curr)

end--function Vceo()

function Print_Data(vsteps, ce_volt,ce_curr)
--Print Data to output queue

	--Local Variables
	local l_vsteps = vsteps
	local l_i = 1 --Iteration variable
	local l_ce_volt = ce_volt
	local l_ce_curr = ce_curr

	print("")
	print("Vceo (V)","Iceo (A)")
```

```
      for l_i = 1, l_vsteps  do
            print(l_ce_volt[l_i], l_ce_curr[l_i])
      end --for

 end --function Print_Data()

--Iceo()
```

Section 4. FET Tests

Program 9. Common-Source Characteristics

```
--[[
FET_Comm_Source(): USES TABLES

This program applies a bias to the gate-source of an FET (VGS) and sweeps voltage on the
drain-source (VDS). The VDS and ID values at each VGS bias are then printed.

Required equipment:

(1) Dual-channel Keithley Series 2600 System SourceMeter instrument
(1) SD210 N-Channel MOSFET

Running this script creates functions that can be used to measure the common source
characteristics of
FETs. The default values are for an N-channel MOSFET type SD210.

       The functions created are:
              1. FET_Comm_Source(vgsstart, vgsstop, vgssteps, vdsstart, vdsstop, vdssteps)
                 --Default values vgsstart = 0, vgsstop = 10V, vgssteps = 5, vdstart = 0V,
vdstop = 10V, vdsteps = 100
              2. Print_Data(vgssteps,vdssteps, vds_data,Id_data, vgs_data)

       See detailed information listed in individual functions.

1) From Test Script Builder
       - At the TSP> prompt in the Instrument Control Panel, type FET_Comm_Source()

2) From an external program
       - Send the entire program text as a string using standard GPIB Write calls.


Rev1: JAC 6.18.2007

]]--


------------------ Keithley TSP Function ------------------

function FET_Comm_Source(vgsstart, vgsstop, vgssteps, vdsstart, vdsstop, vdssteps)
--Configure SMUB to source a bias
--voltage on the gate-source (Vgs) and SMUA performs a voltage sweep on the drain-source
(Vds) from start to stop in a --user-defined number of steps. SMUB then increments to next
bias value and continues to the stop value.
--Returns measured Vgs, Vds, and Id values.

       --Global variables
       local l_vrange = 40 --
```

```
local l_icmpl = 100E-3 --

--Shared local variables
local l_nplc = 1 --Integration rate of measurement

--Local sweep variables
local l_vgsstart = vgsstart --Gate-source sweep start voltage
local l_vgsstop =  vgsstop --Gate-source  sweep stop voltage
local l_vgssteps = vgssteps --Number of steps in sweep

local l_vdsstart = vdsstart --Drain-source sweep start voltage
local l_vdsstop =  vdsstop --Drain-source sweep stop voltage
local l_vdssteps = vdssteps --Number of steps in sweep

--Default values and level check
if (l_vgsstart == nil) then --Use default value
      l_vgsstart = 0
end --if

if (l_vgsstart > 10) then --Coerce value
      l_vgsstart = 10
end --if

if (l_vgsstop == nil) then --Use default value
      l_vgsstop = 10
end --if

if (l_vgsstop > 10) then --Coerce value
      l_vgsstop = 10
end --if

if (l_vgssteps == nil) then --Use default value
      l_vgssteps = 5
end --if

if (l_vgssteps > 100) then --Coerce value
      l_vgssteps = 100
end --if

local l_vgsstep = (l_vgsstop - l_vgsstart)/ (l_vgssteps - 1) --Vgs step size
local l_vgssource_val = l_vgsstart --Source value during sweep
local l_vgs_iteration = 1 --Iteration variable

if (l_vdsstart == nil) then --Use default value
      l_vdsstart = 0
end --if

if (l_vdsstart > 10) then --Coerce value
      l_vdsstart = 10
end --if
```

```
	if (l_vdsstop == nil) then --Use default value
		l_vdsstop = 10
	end --if

	if (l_vdsstop > 40) then --Coerce value
		l_vdsstop = 40
	end --if

	if (l_vdssteps == nil) then --Use default value
		l_vdssteps = 100
	end --if

	if (l_vdssteps > 2E+2) then --Coerce value
		l_vdssteps = 2E+2
	end --if

	local l_vdsstep = (l_vdsstop - l_vdsstart)/ (l_vdssteps - 1) --Voltage step size
	local l_vdssource_val = l_vdsstart --Source value during sweep
	local l_vds_iteration = 1 --Iteration variable

	--Data tables
	local l_vgs_data = {} --Create data table for sourced gate-source voltage
	local l_vds_data = {} --Create data table for drain-source voltage
local l_id_data = {} --Create data table for drain-source measured current

	smua.reset() --Reset SMU
	smub.reset() --Reset SMU

	errorqueue.clear() --Clear the error queue

	--Configure Drain-Source (SMUA) source and measure settings
	smua.source.func = smua.OUTPUT_DCVOLTS
smua.source.autorangev = smua.AUTORANGE_ON --Enable source autorange
smua.source.levelv = 0
	smua.source.limiti = l_icmpl
	smua.measure.autorangei = smua.AUTORANGE_ON --Enable measure autorange

	smua.measure.autozero = smua.AUTOZERO_AUTO
	smua.measure.nplc = l_nplc --Measurement integration rate

	smua.source.output = smua.OUTPUT_ON --Enable Output

	--Configure Gate-Source (SMUB) source and measure settings
	smub.source.func = smub.OUTPUT_DCVOLTS
	smub.source.autorangev = smub.AUTORANGE_ON --Enable source autorange
	smub.source.levelv = 0
	smub.source.limiti = l_icmpl
	smub.measure.autorangei = smub.AUTORANGE_ON --Enable measure autorange
```

```
	smub.measure.autozero = smub.AUTOZERO_AUTO
	smub.measure.nplc = l_nplc --Measurement integration rate

	smub.source.output = smub.OUTPUT_ON --Enable Output


	--Execute sweep
	for l_vgs_iteration = 1, l_vgssteps  do

		smub.source.levelv = l_vgssource_val

		l_vds_data[l_vgs_iteration] = {} --Create new row in table
		l_id_data[l_vgs_iteration] = {} --Create new row in table

		l_vgs_data[l_vgs_iteration] = smub.measure.v() --Measure gate-source voltage

		for l_vds_iteration = 1,l_vdssteps do


			if (l_vds_iteration == 1) then --Intialize start source value
				l_vdssource_val = l_vdsstart
			end --if

			--delay(1)
			l_vds_data[l_vgs_iteration][l_vds_iteration] = smua.measure.v()
--Measure sourced voltage
			l_id_data[l_vgs_iteration][l_vds_iteration] = smua.measure.i()
--Measure current
			l_vdssource_val = l_vdssource_val + l_vdsstep --Calculate new source
value

			if (l_vds_iteration == l_vdssteps) then --Reinitialize voltage value
after last iteration
				l_vdssource_val = l_vdsstart
			end --if

			smua.source.levelv = l_vdssource_val --Increment source

		end --for

		l_vgssource_val = l_vgssource_val + l_vgsstep --Calculate new source value

	end--for


	smua.source.output = smua.OUTPUT_OFF --Disable output
	smub.source.output = smub.OUTPUT_OFF --Disable output

	smua.source.levelv = 0 --Return source to bias level
	smub.source.leveli = 0 --Return source to bias level
```

```
      Print_Data(l_vgssteps,l_vdssteps, l_vds_data, l_id_data, l_vgs_data)

end--function FET_Comm_Source()

function Print_Data(vgssteps,vdssteps, vds_data,id_data, vgs_data)
--Print Data to output queue

      --Local Variables
      local l_vgssteps = vgssteps
      local l_vdssteps = vdssteps
      local l_vgs_iteration = 1 --Iteration variable
      local l_vds_iteration = 1 --Iteration variable
      local l_vds_data = vds_data
      local l_id_data = id_data
      local l_vgs_data = vgs_data


      for l_vgs_iteration = 1, l_vgssteps  do
            print("")
            print("Gate-source Bias (V)", l_vgs_data[l_vgs_iteration])
            print("Drain-source Voltage (V)","Drain-source Current (A)")

            for l_vds_iteration = 1, l_vdssteps  do
                  print(l_vds_data[l_vgs_iteration][l_vds_iteration], l_id_data[l_vgs_
iteration][l_vds_iteration])
            end --for
      end --for

 end --function Print_Data()

--FET_Comm_Source()
```

Program 10. Transconductance

```
--[[
Transconductance():

This program sources a voltage bias on a drain-source of a FET (VDS), sources a voltage on
the gate (VGS1), and measures the drain-source current (ID1). Then, another source value
(VGS2) is sourced and the IDS2 is measured.

The transconductance (gfs) is then calculated by taking the change in Ids divided by the
change in VGS.

The drain-source voltage (VDS), Transconductance (gfs), gate-source voltage (VGS), and
drain-source current (ID) are returned.

Required equipment:

(1) Dual-channel Keithley Series 2600 System SourceMeter instrument
(1) SD210 N-channel FET

Running this script creates functions that can be used to create a transconductance test
of FETs. The default values are for an N-channel SD210 FET.

       The functions created are:
              1. Transconductance(vgsstart, vgsstop, vgssteps, vdsbias)
                 --Default values vgsstart = 0V, vgsstop = 5V, vgssteps = 100, vdsbias =
10V
              2. Check_Comp()

       See detailed information listed in individual functions.

To run:

1) From Test Script Builder
       - Right-click in the program window, select "Run as TSP"
       - At the TSP> prompt in the Instrument Control Panel, type Transconductance()

2) From an external program
       - Send the entire program text as a string using standard GPIB Write calls.


Rev1: JAC 6.18.2007

]]--

------------------ Keithley TSP Function ------------------

function Transconductance(vgsstart, vgsstop, vgssteps, vdsbias)--Configure SMUA to source a
user-defined voltage on the
```

```
--drain-source (Vds) while SMUB performs a fixed voltage bias (Vgs)on the gate-source and
the Ids is measured.
--SMUB then steps to the next base current and the Ic is measured.

--Returns measured Vds, Vgs, Id, gfs values are returned.

	--Global variables
	local l_icmpl = 100E-3 --Source compliance

	--Shared local variables
	local l_nplc = 1 --Integration rate of measurement

	--Local sweep variables
	local l_vgsstart = vgsstart --Vgs start voltage
	local l_vgsstop =  vgsstop --Vgs sweep stop voltage
	local l_vgssteps = vgssteps --Number of steps in sweep

	local l_vdsbias = vdsbias --Drain-source voltage

	--Default values and level check
	if (l_vgsstart == nil) then --Use default value
		l_vgsstart = 0
	end --if

	if (l_vgsstart > 10) then --Coerce value
		l_vgsstart = 10
	end --if

	if (l_vgsstop == nil) then --Use default value
		l_vgsstop = 5
	end --if

	if (l_vgsstop > 10) then --Coerce value
		l_vgsstop = 10
	end --if

	if (l_vgssteps == nil) then --Use default value
		l_vgssteps = 20
	end --if

	if (l_vgssteps > 100) then --Coerce value
		l_vgssteps = 100
	end --if

	local l_vgsstep = (l_vgsstop - l_vgsstart)/ (l_vgssteps - 1) --Vbe step size
	local l_vgssource_val = l_vgsstart --Source value during sweep
	local l_i = 1 --Iteration variable

	if (l_vds_bias == nil) then --Use default value
		l_vds_bias = 10
```

```
end --if

if (l_vds_bias > 10) then --Coerce value
      l_vds_bias = 10
end --if

--Data tables
local l_vgs = {} --Create data table for gate-source voltage
local l_id = {} --Create data table for drain-source current
local l_gfs = {} --Create data table for transconductance (gfs)

smua.reset() --Reset SMU
smub.reset() --Reset SMU

errorqueue.clear() --Clear the error queue

--Configure Collector/Emitter (SMUA) source and measure settings
smua.source.func = smua.OUTPUT_DCVOLTS
smua.source.autorangev = smua.AUTORANGE_ON --Enable source autorange
smua.source.levelv = 0
smua.source.limiti = l_icmpl
smua.measure.autorangei = smua.AUTORANGE_ON --Enable measure autorange

smua.measure.autozero = smua.AUTOZERO_AUTO
smua.measure.nplc = l_nplc --Measurement integration rate

smua.source.output = smua.OUTPUT_ON --Enable Output

--Configure Base (SMUB) source and measure settings
smub.source.func = smub.OUTPUT_DCVOLTS
smub.source.autorangev = smub.AUTORANGE_ON --Enable source autorange
smub.source.levelv = 0
smub.source.limiti = l_icmpl
smub.measure.autorangev = smub.AUTORANGE_ON --Enable measure autorange

smub.measure.autozero = smub.AUTOZERO_AUTO
smub.measure.nplc = l_nplc --Measurement integration rate

smub.source.output = smub.OUTPUT_ON --Enable Output

smua.source.levelv = l_vds_bias

--Execute sweep
for l_i = 1,l_vgssteps do

      if (l_i == 1) then --Intialize start source value
            l_vgssource_val = l_vgsstart
      end --if

      --delay(1)
```

```
            l_vgs[l_i] = smub.measure.v() --Measure Vgs
            l_id[l_i] = smua.measure.i() --Measure Id

            l_vgssource_val = l_vgssource_val + l_vgsstep --Calculate new source value


            if (l_i == l_vgssteps) then --Reinitialize voltage value after last
iteration
                  l_vgssource_val = l_vgsstart
            end --if

            smub.source.levelv = l_vgssource_val --Increment source

      end --for

      smua.source.output = smua.OUTPUT_OFF --Disable output
      smub.source.output = smub.OUTPUT_OFF --Disable output

      smua.source.levelv = 0 --Return source to bias level
      smub.source.levelv = 0 --Return source to bias level

      Print_Data(l_vds_bias, l_vgssteps, l_vgs, l_id)

end--function Transconductance()

function Print_Data(vdsbias, vgssteps,vgs, id)
--Calculate Gfs value and print data to output queue

      --Local Variables
      local l_vds_bias = vdsbias --Vds bias value
      local l_vgs_steps = vgssteps --Number of steps in Vgs sweep
      local l_vgs = vgs --Gate-source Voltage data
      local l_id = id --Drain-source current data

      local l_gfs = {} --Table for Transconductance calculations

      local l_i = 1 --Iteration variable

      --Calculate gfs values and populate table
      for l_i = 1,l_vgs_steps do

            if (l_i ~= 1) then --If not the first iteration, calculate gfs
                  l_gfs[l_i] = (l_id[l_i] - l_id[l_i - 1])/(l_vgs[l_i] - l_vgs[l_i - 1])
                  --gfs = dId/dVgs
            end--if

      end --for

      l_i = 1 --Reinitialize Vgs iteration variable
```

```
      print("")
      print("Vds", l_vds_bias)
      print("Vgs (V)","Id (A)","gfs (s)")

      for l_i = 2, l_vgs_steps  do
            print(l_vgs[l_i],l_id[l_i], l_gfs[l_i])
      end --for

 end --function Print_Data()

--Transconductance()
```

Program 11. Threshold

Program 11A. Threshold (Search)

```
--[[
FET_Thres_Search():

This program performs a binary search on the gate-source voltage (VGS) of an FET at a fixed
drain-source voltage (VDS) and searches for a target drain-source current (ID). If the
specified Id is found within the maximum number of iterations, the threshold voltage (VTH)
and drain-source (ID) currents are measured and printed.

If the maximum number of iterations are reached, the program is aborted.

Required equipment:

(1) Dual-channel Keithley Series 2600 System SourceMeter instrument
(1) SD210 N-Channel FET

Running this script creates functions that can be used to create a threshold search of
FETs. The default values are for an NPN transistor type 2N5089.

      The functions created are:
            1. FET_Thres_Search(vdssource, lowvgs, highvgs, targetid)
               --Default values vdssource = 1V, lowvgs = 0.5, highvgs = 2, targetid =
1e-6A
            2. Check_Comp()

      See detailed information listed in individual functions.

To run:

1) From Test Script Builder
      - Right-click in the program window, select "Run as TSP"
      - At the TSP> prompt in the Instrument Control Panel, type FET_Thres_Search()

2) From an external program
      - Send the entire program text as a string using standard GPIB Write calls.
```

```
Rev1: JAC 6.26.2007

]]--


------------------ Keithley TSP Function ------------------

function FET_Thres_Search(vdssource, lowvgs, highvgs, targetid) --Configure SMUA to source
a user-defined voltage on
--the drain-source (Vds) while SMUB sources a voltage on --the gate-source (Vgs). VGS is
varied using a binary
--search algorithm between a maximum and minimum Vgs value, and the drain-source current
(Id) is measured.
--If measured value is outside the tolerance, search is performed again until the value
falls within the
--specified range or the iteration limit is reached.
--Returns measured Vds, Vth, and Id.

    local l_k --binary search loop count variable
    local l_k_max = 20 --Maximum loop counts
    local l_vds_source = vdssource --vdssource value
    local l_high_vgs = highvgs --Start Ib high limit
    local l_low_vgs = lowvgs --Start Ib lo limit
    local l_target_id = targetid --Target Ic for binary search
    local l_nplc = 1

    local l_vgs_source = 0--Gate-sourced voltage
    local l_id_meas --Drain-source measured voltage

       --Default values and level check
       if (l_vds_source == nil) then --Use default value
           l_vds_source = 0.5
       end --if

       if (l_low_vgs == nil) then --Use default value
           l_low_vgs = 0.5
       end --if

       if (l_high_vgs == nil) then --Use default value
           l_high_vgs = 1.1
       end --if

       if (l_target_id == nil) then --Use default value
           l_target_id = 1e-6
       end --if

       smua.reset() --Reset SMU
       smub.reset() --Reset SMU
```

```
      errorqueue.clear() --Clear the error queue

   smua.measure.nplc = l_nplc --Measurement integration rate
   smub.measure.nplc = l_nplc --Measurement integration rate

   smua.source.func = smua.OUTPUT_DCVOLTS
   smua.source.autorangev = smua.AUTORANGE_ON --Enable source autorange
   smua.source.limiti = (100 * l_target_id) --Set compliance value
   smua.measure.autorangev = smua.AUTORANGE_ON --Enable measure autorange

   smub.source.func = smub.OUTPUT_DCVOLTS
   smub.source.autorangev = smub.AUTORANGE_ON --Enable source autorange
   smub.source.limiti = l_high_vgs
   smub.source.limitv = 6
   smub.measure.autorangev = smub.AUTORANGE_ON --Enable measure autorange

   --Start test
   smua.source.levelv = l_vds_source --Set source level
   smub.source.levelv = 0 --Set source level

   smua.source.output = smua.OUTPUT_ON --Enable output
   smub.source.output = smub.OUTPUT_ON --Enable output

   delay(0.001) --Delay

   l_comp_val = Check_Comp() --check compliance

   if l_comp_val == true then --If unit is in compliance, end

      smua.source.output = smua.OUTPUT_OFF --Disable output
      smub.source.output = smub.OUTPUT_OFF --Disable output

      smua.source.levelv = 0 --Return source to 0
      smub.source.levelv = 0 --Return source to 0

   else

            --Search for the right base current
      l_k = 0

      repeat --Repeat search until measured Ic is within 5% of target, or iteration
maximum reached, or compliance.

            l_k = l_k + 1 --Increment

            l_vgs_source = ((l_high_vgs-l_low_vgs)/2) + l_low_vgs --Determine source
value (Binary Search)
            smub.source.levelv = l_vgs_source --Program new source value
            delay(0.01) --Source delay
```

```
            l_comp_val = Check_Comp() --check compliance

                  if l_comp_val == true then --If unit is in compliance, end
                        smua.source.output = smua.OUTPUT_OFF --Disable output
                        smub.source.output = smub.OUTPUT_OFF --Disable output

                        smua.source.levelv = 0 --Return source to 0
                        smub.source.levelv = 0 --Return source to 0

                  else
                        l_id_meas = smua.measure.i() --Measure Id

                        if (l_target_id < l_id_meas) then --Compare measurement with
target value
                        l_high_vgs = l_vgs_source
                   else
                              l_low_vgs = l_vgs_source
                  end --end if
            end --ifelse

            if l_id_meas == nil then --If no measurement taken, initialize to 0 to avoid
arithmatic error
                                                            --in until statement below
                l_id_meas = 0
            end --if

        until ((math.abs(l_id_meas - l_target_id) < (0.05*l_target_id))or(l_k>l_k_max))
or(l_comp_val == true) --

          --iteration limit reached
        if (l_k > l_k_max) then
                  print("Iteration Limit Reached!!")
        end --end if

       smua.source.output = smua.OUTPUT_OFF --Disable output
       smub.source.output = smub.OUTPUT_OFF --Disable output

        print("Id Data:", l_id_meas) --Print Id data
        print("Vgs Data:", l_vgs_source) --Print Vgs
        print("Vds Data:",l_vds_source) --Print Vds

   end --ifelse

end--function FET_Thres_Search()

function Check_Comp() --Function checks state of compliance, if true, prints warning and
returns
                     --to run_test()
```

```
	local l_comp_val = false --Initialize local variable

	l_comp_val = smua.source.compliance --Check compliance

	if l_comp_val == true then
		print("")
		print("SMU Source in Compliance!")
		print("Ensure proper connections, stable device, and settings are correct")
		print("Rerun Test")
		print("")
	end --if

	return l_comp_val

end --function Check_Comp()

--FET_Thres_Search()
```

Program 11B. Threshold (Fast)

```
--[[
FET_Thres_Fast()

This program applies a bias to the drain-source of an FET (VDS) and sweeps current on the
drain-source (ID) and the threshold voltage (VTH) at each ID value is measured.

*NOTE: Due to connection scheme, negative values are to be programmed for the sourced
values. The absolute value of the measurements and sourced values are printed.

Required equipment:

(1) Dual-channel Keithley Series 2600 System SourceMeter instrument
(1) SD210 N-channel FET

Running this script creates functions that can be used to measure the threshold of FETs.
The default values are for an N-channel FET type SD210.

	The functions created are:
		1. FET_Thres_Fast(vdssource, istart, istop, isteps)
		   --Default values vdssource = 0.5V, istart = 0.5uA, istop = 1uA, isteps =
10
		2. Print_Data(isteps, drain_curr, thres_volt)

	See detailed information listed in individual functions.

1) From Test Script Builder
	- At the TSP> prompt in the Instrument Control Panel, type FET_Thres_Fast()

2) From an external program
```

```
      - Send the entire program text as a string using standard GPIB Write calls.

Rev1: JAC 6.26.2007

]]--

------------------ Keithley TSP Function ------------------

function FET_Thres_Fast(vdssource, istart, istop, isteps) --Configure SMUB to source a bias
current
--on the drain-source (Id) and SMUA performs a voltage sweep on the drain-source (Vds)
from start to
--stop in a user-defined number of steps.
--Returns Vth, Vds, and Id values.

      --Global variables
      local l_irange = 100e-6 --Drain current source range
      local l_vcmpl = 11 --Drain source compliance

      local l_vrange = 40 --Drain-source voltage source range
      local l_icmpl = 100e-3 --Drain source compliance

      --Shared local variables
      local l_nplc = 1 --Integration rate of measurement

      --Local sweep variables
      local l_istart = istart --Drain sweep start current
      local l_istop =  istop --Drain sweep stop current
      local l_isteps = isteps --Number of steps in sweep

      local l_vds_source = vdssource --Vds source value

      --Default values and level check
      if (l_vds_source == nil) then --Use default value
            l_vds_source = -0.5
      end --if

      if (l_vds_source > 0) then --Coerce value
            l_vds_source = -l_vds_source
      end --if

      if (l_istart == nil) then --Use default value
            l_istart = -500e-9
      end --if

      if (l_istart > 0) then --Coerce value
            l_istart = -l_istart
      end --if

      if (l_istop == nil) then --Use default value
```

```
      l_istop = -1e-6
end --if

if (l_istop > 0) then --Coerce value
      l_istop = -i_stop
end --if

if (l_isteps == nil) then --Use default value
      l_isteps = 10
end --if

if (l_isteps > 100) then --Coerce value
      l_isteps = 100
end --if

local l_istep = (l_istop - l_istart)/ (l_isteps - 1) --Current step size
local l_isource_val = l_istart --Source value during sweep
local l_i = 1 --Iteration variable

--Data tables
local l_thres_volt = {} --Create data table for threshold voltage
local l_drain_curr = {} --Create data table for emitter current

smua.reset() --Reset SMU
smub.reset() --Reset SMU

errorqueue.clear() --Clear the error queue

--Configure emitter current (SMUA) source and measure settings
smua.source.func = smua.OUTPUT_DCVOLTS
smua.source.autorangev = smua.AUTORANGE_ON --Enable source autorange
smua.source.levelv = 0
smua.source.limiti = l_icmpl
smua.sense = smua.SENSE_REMOTE --Enable Remote (4-wire) sensing
smua.source.output = smua.OUTPUT_ON --Enable Output

--Configure collector/emitter (SMUB) source and measure settings
smub.source.func = smub.OUTPUT_DCAMPS
smub.source.autorangev = smub.AUTORANGE_ON --Enable source autorang
smub.source.levelv = 0
smub.source.limitv = l_vcmpl
smub.measure.autorangev = smub.AUTORANGE_ON --Enable measure autorange

smub.measure.autozero = smub.AUTOZERO_AUTO
smub.measure.nplc = l_nplc --Measurement integration rate

smub.source.output = smub.OUTPUT_ON --Enable Output

smua.source.levelv = l_vds_source --Program source
```

```
	--Execute sweep
	for l_i = 1, l_isteps  do

		smub.source.leveli = l_isource_val

		delay(0.01)

		l_thres_volt[l_i] = smub.measure.v() --Measure threshold voltage (Vt)

		l_drain_curr[l_i] = smua.measure.i() --Measure drain current

		l_isource_val = l_isource_val + l_istep --Calculate new source value

	end--for

	smua.source.output = smua.OUTPUT_OFF --Disable output
	smub.source.output = smub.OUTPUT_OFF --Disable output

	smua.source.levelv = 0 --Return source to bias level
	smub.source.leveli = 0 --Return source to bias level

	Print_Data(l_isteps, l_drain_curr, l_thres_volt, l_vds_source)

end--function DC_Gain_Fast()

function Print_Data(isteps, drain_curr, thres_volt, vdssource)
--Print Data to output queue

	--Local Variables
	local l_isteps = isteps
	local l_i = 1 --Iteration variable
	local l_drain_curr = drain_curr --Drain current table
	local l_thres_volt = thres_volt --Threshold voltage table
	local l_vds_source = vdssource --Drain-source voltage value

	print("")
	print("Drain-source Voltage (V)")
	print(math.abs(l_vds_source))

	print("")
	print("Threshold Voltage (V)", "Drain Current (A)")

	for l_i = 1, l_isteps  do

		print(math.abs(l_thres_volt[l_i]), math.abs(l_drain_curr[l_i]))

	end --for

 end --function Print_Data()
--FET_Thres_Fast()
```

Section 5. Using Substrate Bias

Program 12. Substrate Current vs. Gate-Source Voltage (FET I_{SB} vs. V_{GS})

```
--[[
FET_Isb_Vgs():

This program applies a voltage bias on the drain-source (VDS), a voltage bias on the
substrate-source (VSB) of an FET, then sweeps the gate-source voltage (VGS) from a user-
defined stop, through a defined number of steps.

At each point, the VGS, ID, and ISB are measured and the data is printed.

Required equipment:

(1) Dual-channel Keithley Series 2600 System SourceMeter instrument
(1) Keithley Model 2636 System Sourcemeter instrument (Required for low current
measurement)
(1) Crossover Ethernet Cable
(1) SD210 N-channel FET

- Connect the single-channel SourceMeter instrument to the dual-channel master using a
crossover Ethernet cable.
- Connect the test fixture to both units using appropriate cables.
- Turn on the SourceMeter instruments and allow the units to warm up for two hours for
rated accuracy.

Configure the TSP-Link communications for each instrument:

Slave: A single-channel instrument such as the Model 2601, 2611, or 2635.
      1. Press the MENU key to access MAIN MENU.
      2. Select the COMMUNICATION menu. (Skip this step if the Series 2600 instruments
used have firmware Revision 1.4.0 or later installed.)
      3. Select the TSPLINK_CFG menu. (If the Series 2600 instruments used have firmware
Revision 1.4.0 or later installed, the menu name should be TSPLINK.)
      4. Select the NODE menu.
      5. Set the NODE number to 2 and press ENTER.

Master: A dual-channel instrument such as the Model 2602, 2612, or 2636.
      1. Press the MENU key to access MAIN MENU.
      2. Select the COMMUNICATION menu. (Skip this step if the Series 2600 instruments
used have firmware Revision 1.4.0 or later installed.)
      3. Select the TSPLINK_CFG menu. (If the Series 2600 instruments used have firmware
Revision 1.4.0 or later installed, the menu name should be TSPLINK.)
      4. Select the NODE menu.
      5. Set the NODE number to 1 for the Master and press ENTER.
      6. Select the TSPLINK_CFG menu. (If the Series 2600 instruments used have firmware
Revision 1.4.0 or later installed, the menu name should be TSPLINK.)
      7. Select the RESET to initialize the TSP-Link.
```

```
Running this script creates functions that can be used to measure the Isb v. Vgs
characteristics of FETs.

      The functions created are:
1. FET_Isb_Vgs(vdssource, vsbsource,vgsstart,vgsstop, vgssteps) --Default values vdssource
= 1V,
--vgsstart = 0V,vgsstop = 10V, vgssteps = 10
      2. Print_Data(l_vgs_steps, l_id_curr, l_vgs_volt,l_isb_curr)

      See detailed information listed in individual functions.

To run:

1) From Test Script Builder
      - Right-click in the program window, select "Run as TSP"
      - At the TSP> prompt in the Instrument Control Panel, type FET_Isb_Vgs()

2) From an external program
      - Send the entire program text as a string using standard GPIB Write calls.

Rev1: JAC 5.22.2007

]]--


------------------ Keithley TSP Function ------------------

function FET_Isb_Vgs(vdssource, vsbsource,vgsstart,vgsstop, vgssteps) --Configure node 1
SMUA to source drain-source
--voltage (Vds), node 2 SMUA to apply a voltage bias on the substrate-source (Vsb)and
perform a voltage sweep from
--start to stop in user-defined steps using node 1 SMUB on the gate-source (Vgs). At each
point, Vgs and Isb are
--measured and printed.

      --Global variables
      local l_vds_source = vdssource --Drain-source source voltage
      local l_vsb_source = vsbsource --Substrate-source bias voltage
      local l_icmpl = 100E-3 --Source compliance

      --Shared local variables
      local l_nplc = 1 --Integration rate of measurement

      --Local sweep variables
      local l_vgs_start = vgsstart --Gate-source sweep start voltage
      local l_vgs_stop = vgsstop --Gate-source sweep stop voltage
      local l_vgs_steps = vgssteps --Number of steps in sweep

      --Default values and level check
      if (l_vds_source == nil) then --Use default value
```

```
            l_vds_source = 1
        end --if

        if (l_vds_source > 1) then --Coerce value
            l_vds_source = 1
        end --if

        if (l_vsb_source == nil) then --Use default value
            l_vsb_source = -1
        end --if

        if (l_vsb_source > 0 ) then --Coerce value
            l_vsb_source = -1
        end --if

        if (l_vgs_start == nil) then --Use default value
            l_vgs_start = 0
        end --if

        if (l_vgs_start > 10) then --Coerce value
            l_vgs_start = 10
        end --if

        if (l_vgs_stop == nil) then --Use default value
            l_vgs_stop = 10
        end --if

        if (l_vgs_stop > 10) then --Coerce value
            l_vgs_stop = 10
        end --if

        if (l_vgs_steps == nil) then --Use default value
            l_vgs_steps = 10
        end --if

        if (l_vgs_steps > 1E3) then --Coerce value
            l_vgs_steps = 1E3
        end --if

        local l_step = (l_vgs_stop - l_vgs_start)/ (l_vgs_steps - 1) --Current step size
        local l_source_val = l_vgs_start --Source value during sweep
        local l_i = 1 --Iteration variable

        --Data tables
        local l_isb_curr = {} --Create data table for substrate-source current
        local l_id_curr = {} --Create data table for drain-source current
        local l_vgs_volt = {} --Create data table for gate-substrate voltage
local l_vds_volt = {} --Create data table for drain-substrate voltage

        node[1].smua.reset() --Reset SMU
```

```
	node[1].smub.reset() --Reset SMU
	node[2].smua.reset() --Reset SMU

	errorqueue.clear() --Clear the error queue

	--Configure drain-source SMU (TSP-Link Node[1] SMUA) source and measure settings
	node[1].smua.source.func = node[1].smua.OUTPUT_DCVOLTS
	node[1].smua.source.autorangev = node[1].smua.AUTORANGE_ON --Enable source
autorange
	node[1].smua.source.levelv = l_vds_source
	node[1].smua.source.limiti = l_icmpl
	node[1].smua.measure.autorangev = node[1].smua.AUTORANGE_ON --Enable measure
autorange

	node[1].smua.measure.nplc = l_nplc --Measurement integration rate

	--Configure gate-source SMU (TSP-Link Node[1] SMUB) source and measure settings
	node[1].smub.source.func = node[1].smub.OUTPUT_DCVOLTS
	node[1].smub.source.autorangev = node[1].smub.AUTORANGE_ON --Enable source
autorange
	node[1].smub.source.levelv = l_vds_source
	node[1].smub.source.limiti = l_icmpl
	node[1].smub.measure.autorangev = smub.AUTORANGE_ON --Enable measure autorange

	node[1].smub.measure.nplc = l_nplc --Measurement integration rate

	--Configure substrate-source SMU (TSP-Link Node[2] SMUA) source and measure settings
	node[2].smua.source.func = node[2].smua.OUTPUT_DCVOLTS
	node[2].smua.source.autorangev = node[2].smua.AUTORANGE_ON --Enable source
autorange
	node[2].smua.source.levelv = l_vsb_source
	node[2].smua.source.limiti = l_icmpl
	node[2].smua.measure.autorangev = node[2].smua.AUTORANGE_ON --Enable measure
autorange

	node[2].smua.measure.nplc = l_nplc --Measurement integration rate

	node[1].smua.source.output = smua.OUTPUT_ON --Enable Output
	node[1].smub.source.output = smub.OUTPUT_ON --Enable Output
	node[2].smua.source.output = smua.OUTPUT_ON --Enable Output

	--Execute sweep
	for l_i = 1, l_vgs_steps  do
		--smua.source.leveli = l_source_val
		delay(0.010) --Delay
		l_id_curr[l_i] = node[1].smua.measure.i() --Measure drain-source current
		l_vgs_volt[l_i] = node[1].smub.measure.v() --Measure gate-source voltage
		l_isb_curr[l_i] = node[2].smua.measure.i() --Measure substrate-source
current
```

```
            l_source_val = l_source_val + l_step --Calculate new source value

            node[1].smub.source.levelv = l_source_val --Increment source

      end--for

      node[1].smua.source.output = node[1].smua.OUTPUT_OFF --Disable output
      node[1].smub.source.output = node[1].smub.OUTPUT_OFF --Disable output
      node[2].smua.source.output = node[2].smua.OUTPUT_OFF --Disable output

      Print_Data(l_vgs_steps, l_id_curr, l_vgs_volt,l_isb_curr)

end--function Diode_Fwd_Chr()

function Print_Data(vgssteps,idcurr,vgsvolt,isbcurr)
--Print Data to output queue

      --Local Variables
      local l_vgs_steps = vgssteps
      local l_id_curr = idcurr
      local l_vgs_volt = vgsvolt
      local l_isb_curr = isbcurr

      print("Drain-source current(A):", "Gate-source voltage(V):", "Substrate-source
current(A):" )

      for l_i = 1, l_vgs_steps  do
            print(l_id_curr[l_i], l_vgs_volt[l_i], l_isb_curr[l_i])
      end

end --function Print_Data()

--FET_Isb_Vgs()
```

Program 13. Common-Source Characteristics with Substrate Bias

```
--[[
FET_Comm_Source_Vsb():

This program applies a bias to the substrate-source of an FET (VSB) and a staircase sweep
on the gate-source voltage (VGS). At each VGS value, the drain-source voltage (VDS) is
also swept linearly.

At each point, the VDS and IDS are measured and printed.

Required equipment:

(1) Dual-channel Keithley Series 2600 System SourceMeter instrument
```

(1) Keithley Model 2636 System Sourcemeter instrument (Required for low current measurement)
(1) Crossover Ethernet Cable
(1) SD210 N-Channel MOSFET

- Connect the single-channel SourceMeter instrument to the dual-channel master using a crossover Ethernet cable.
- Connect the test fixture to both units using appropriate cables.
- Turn on the SourceMeter instrument and allow the unit to warm up for two hours for rated accuracy.

Configure the TSP-Link communications for each instrument:

Slave: A single-channel instrument such as the Model 2601, 2611, or 2635.
1. Press the MENU key to access MAIN MENU.
2. Select the COMMUNICATION menu. (Skip this step if the Series 2600 instruments used have firmware Revision 1.4.0 or later installed.)
3. Select the TSPLINK_CFG menu. (If the Series 2600 instruments used have firmware Revision 1.4.0 or later installed, the menu name should be TSPLINK.)
4. Select the NODE menu.
5. Set the NODE number to 2 and press ENTER.

Master: A dual-channel instrument such as the Model 2602, 2612, or 2636.
1. Press the MENU key to access MAIN MENU.
2. Select the COMMUNICATION menu. (Skip this step if the Series 2600 instruments used have firmware Revision 1.4.0 or later installed.)
3. Select the TSPLINK_CFG menu. (If the Series 2600 instruments used have firmware Revision 1.4.0 or later installed, the menu name should be TSPLINK.)
4. Select the NODE menu.
5. Set the NODE number to 1 for the Master and press ENTER.
6. Select the TSPLINK_CFG menu. (If the Series 2600 instruments used have firmware Revision 1.4.0 or later installed, the menu name should be TSPLINK.)
7. Select the RESET to initialize the TSP-Link.

Running this script creates functions that can be used to measure the common source characteristics of an N-channel MOSFET with substrate bias. The default values are for an N-channel MOSFET type SD210.

The functions created are:

1. FET_Comm_Source_Vsb(vgsstart, vgsstop, vgssteps, vdsstart, vdsstop, vdssteps, vsbsource)
   --Default values vgsstart = 0, vgsstop = 10V, vgssteps = 5, vdstart = 0V, vdstop = 10V,
   --vdsteps = 100, vsbsource = -1V
2. Print_Data(vgssteps,vdssteps, vds_data,Id_data, vgs_data, vsbsource)

See detailed information listed in individual functions.

1) From Test Script Builder

```
- At the TSP> p rompt in the Instrument Control Panel, type FET_Comm_Source_Vsb()

2) From an external program
       - Send the entire program text as a string using standard GPIB Write calls.


Rev1: JAC 6.18.2007

]]--

--TODO: Update Node info

------------------ Keithley TSP Function ------------------

function FET_Comm_Source_Vsb(vgsstart, vgsstop, vgssteps, vdsstart, vdsstop, vdssteps,
vsbsource)
--Configure node 1 SMUB to source a bias voltage on the gate-source (Vgs), node 1 SMUA
performs a voltage
--sweep on the drain-source Vds) from start to stop in a user-defined number of steps, and
node 2 SMUA is
--used to bias the substrate (Vsb). Node 1 SMUB then increments to next bias value and
continues to stop
--value.
--Returns measured Vgs, Vds, Vsb, and Id values.

       --Global variables
       local l_vrange = 40 --
       local l_icmpl = 100E-3 --

       --Shared local variables
       local l_nplc = 1 --Integration rate of measurement

       --Local sweep variables
       local l_vgsstart = vgsstart --Gate-source sweep start voltage
       local l_vgsstop =  vgsstop --Gate-source  sweep stop voltage
       local l_vgssteps = vgssteps --Number of steps in sweep

       local l_vdsstart = vdsstart --Drain-source sweep start voltage
       local l_vdsstop =  vdsstop --Drain-source sweep stop voltage
       local l_vdssteps = vdssteps --Number of steps in sweep

       local l_vsbsource = vsbsource --Substrate bias value

       --Default values and level check
       if (l_vgsstart == nil) then --Use default value
             l_vgsstart = 0
       end --if

       if (l_vgsstart > 10) then --Coerce value
             l_vgsstart = 10
```

```
end --if

if (l_vgsstop == nil) then --Use default value
       l_vgsstop = 10
end --if

if (l_vgsstop > 10) then --Coerce value
       l_vgsstop = 10
end --if

if (l_vgssteps == nil) then --Use default value
       l_vgssteps = 5
end --if

if (l_vgssteps > 100) then --Coerce value
       l_vgssteps = 100
end --if

local l_vgsstep = (l_vgsstop - l_vgsstart)/ (l_vgssteps - 1) --Vgs step size
local l_vgssource_val = l_vgsstart --Source value during sweep
local l_vgs_iteration = 1 --Iteration variable

if (l_vdsstart == nil) then --Use default value
       l_vdsstart = 0
end --if

if (l_vdsstart > 10) then --Coerce value
       l_vdsstart = 10
end --if

if (l_vdsstop == nil) then --Use default value
       l_vdsstop = 10
end --if

if (l_vdsstop > 40) then --Coerce value
       l_vdsstop = 40
end --if

if (l_vdssteps == nil) then --Use default value
       l_vdssteps = 100
end --if

if (l_vdssteps > 2E+2) then --Coerce value
       l_vdssteps = 2E+2
end --if

local l_vdsstep = (l_vdsstop - l_vdsstart)/ (l_vdssteps - 1) --Voltage step size
local l_vdssource_val = l_vdsstart --Source value during sweep
local l_vds_iteration = 1 --Iteration variable
```

```
	if (l_vsbsource == nil) then --Use default value
		l_vsbsource = -1
	end --if

	if (l_vsbsource > 0) then --Coerce value
		l_vsbsource = -l_vsbsource
	end --if

	if (l_vsbsource < -40) then --Coerce value
		l_vsbsource = -40
	end --if

	--Data tables
	local l_vgs_data = {} --Create data table for sourced gate-source voltage
	local l_vds_data = {} --Create data table for drain-source voltage
	local l_id_data = {} --Create data table for drain-source measured current

	node[1].smua.reset() --Reset SMU
	node[1].smub.reset() --Reset SMU
	node[2].smua.reset() --Reset SMU

	errorqueue.clear() --Clear the error queue

	--Configure Drain-Source (node [1] SMUA) source and measure settings
	node[1].smua.source.func = node[1].smua.OUTPUT_DCVOLTS
	node[1].smua.source.autorangev = node[1].smua.AUTORANGE_ON --Enable source
autorange
	node[1].smua.source.levelv = 0
	node[1].smua.source.limiti = l_icmpl
	node[1].smua.measure.autorangei = node[1].smua.AUTORANGE_ON --Enable measure
autorange

	node[1].smua.measure.autozero = node[1].smua.AUTOZERO_AUTO
	node[1].smua.measure.nplc = l_nplc --Measurement integration rate

	node[1].smua.source.output = node[1].smua.OUTPUT_ON --Enable Output

	--Configure Gate-Source (node [1] SMUB) source and measure settings
	node[1].smub.source.func = node[1].smub.OUTPUT_DCVOLTS
	node[1].smub.source.autorangev = node[1].smub.AUTORANGE_ON --Enable source
autorange
	node[1].smub.source.levelv = 0
	node[1].smub.source.limiti = l_icmpl
	node[1].smub.measure.autorangei = node[1].smub.AUTORANGE_ON --Enable measure
autorange

	node[1].smub.measure.autozero = node[1].smub.AUTOZERO_AUTO
	node[1].smub.measure.nplc = l_nplc --Measurement integration rate
	node[1].smub.source.output = node[1].smub.OUTPUT_ON --Enable Output
```

```
	--Configure Substrate (node [2] SMUA) source and measure settings
	node[2].smua.source.func = node[2].smua.OUTPUT_DCVOLTS
	node[2].smua.source.autorangev = node[2].smua.AUTORANGE_ON --Enable source
autorange
	node[2].smua.source.levelv = 0
	node[2].smua.source.limiti = l_icmpl
	node[2].smua.measure.autorangei = node[2].smua.AUTORANGE_ON --Enable measure
autorange

	node[2].smua.measure.autozero = node[2].smua.AUTOZERO_AUTO
	node[2].smua.measure.nplc = l_nplc --Measurement integration rate

	node[2].smua.source.output = node[2].smua.OUTPUT_ON --Enable Output

	--Enable Substrate Bias (node [2] SMUA)
	node[2].smua.source.levelv = l_vsbsource

	--Execute sweep
	for l_vgs_iteration = 1, l_vgssteps  do

		node[1].smub.source.levelv = l_vgssource_val

		l_vds_data[l_vgs_iteration] = {} --Create new row in table
		l_id_data[l_vgs_iteration] = {} --Create new row in table

		l_vgs_data[l_vgs_iteration] = node[1].smub.measure.v() --Measure gate-source
voltage

		for l_vds_iteration = 1,l_vdssteps do


			if (l_vds_iteration == 1) then --Intialize start source value
				l_vdssource_val = l_vdsstart
			end --if

			--delay(1)
			l_vds_data[l_vgs_iteration][l_vds_iteration] = node[1].smua.
measure.v()
--Measure sourced voltage
			l_id_data[l_vgs_iteration][l_vds_iteration] = node[1].smua.measure.i()
--Measure current
			l_vdssource_val = l_vdssource_val + l_vdsstep --Calculate new source
value

			if (l_vds_iteration == l_vdssteps) then --Reinitialize voltage value
after last
			--iteration
				l_vdssource_val = l_vdsstart
			end --if
```

```
                  node[1].smua.source.levelv = l_vdssource_val --Increment source

            end --for

            l_vgssource_val = l_vgssource_val + l_vgsstep --Calculate new source value

      end--for


      node[1].smua.source.output = node[1].smua.OUTPUT_OFF --Disable output
      node[1].smub.source.output = node[1].smub.OUTPUT_OFF --Disable output
      node[2].smua.source.output = node[2].smua.OUTPUT_OFF --Disable output

      node[1].smua.source.levelv = 0 --Return source to bias level
      node[1].smub.source.leveli = 0 --Return source to bias level
      node[2].smua.source.levelv = 0 --Return source to bias level

      Print_Data(l_vgssteps,l_vdssteps, l_vds_data, l_id_data, l_vgs_data, l_vsbsource)

end--function FET_Comm_Source_Vsb()

function Print_Data(vgssteps,vdssteps, vds_data,id_data, vgs_data, vsbsource)
--Print Data to output queue

      --Local Variables
      local l_vgssteps = vgssteps
      local l_vdssteps = vdssteps
      local l_vgs_iteration = 1 --Iteration variable
      local l_vds_iteration = 1 --Iteration variable
      local l_vds_data = vds_data
      local l_id_data = id_data
      local l_vgs_data = vgs_data
      local l_vsbsource = vsbsource

      for l_vgs_iteration = 1, l_vgssteps  do
            print("")
            print("Substrate Bias (V)", l_vsbsource)
            print("Gate-source Bias (V)", l_vgs_data[l_vgs_iteration])
            print("Drain-source Voltage (V)","Drain-source Current (A)")

            for l_vds_iteration = 1, l_vdssteps  do
                  print(l_vds_data[l_vgs_iteration][l_vds_iteration], l_id_data[l_vgs_
iteration][l_vds_iteration])
            end --for
      end --for

 end --function Print_Data()

--FET_Comm_Source_Vsb()
```

Program 14. Common-Emitter Characteristics with Substrate Bias

```
--[[
BJT_Comm_Emit_Vsb(): USES TABLES

This program applies a bias to the base of a BJT (IB), a bias to the substrate (VSB), and
sweeps voltage on the collector/emitter (VCE).
The VCE, IB, and IC are then printed.

Required equipment:

(1) Dual-channel Keithley Series 2600 System SourceMeter instrument
(1) Keithley Model 2636 System Sourcemeter instrument (Required for low current
measurement)
(1) Crossover Ethernet Cable
(1) 2N5089 NPN Transistor or equivalent with substrate bias

- Connect the single-channel SourceMeter instrument to the dual-channel master using a
crossover Ethernet cable.
- Connect the test fixture to both units using appropriate cables.
- Turn on the SourceMeter instruments and allow the unit to warm up for two hours for
rated accuracy.

Configure the TSP-Link communications for each instrument:

Slave: A single-channel instrument such as the Model 2601, 2611, or 2635.
      1. Press the MENU key to access MAIN MENU.
      2. Select the COMMUNICATION menu. (Skip this step if the Series 2600 instruments
used have firmware Revision 1.4.0 or later installed.)
      3. Select the TSPLINK_CFG menu. (If the Series 2600 instruments used have firmware
Revision 1.4.0 or later installed, the menu name should be TSPLINK.)
      4. Select the NODE menu.
      5. Set the NODE number to 2 and press ENTER.

Master: A dual-channel instrument such as the Model 2602, 2612, or 2636.
      1. Press the MENU key to access MAIN MENU.
      2. Select the COMMUNICATION menu. (Skip this step if the Series 2600 instruments
used have firmware Revision 1.4.0 or later installed.)
      3. Select the TSPLINK_CFG menu. (If the Series 2600 instruments used have firmware
Revision 1.4.0 or later installed, the menu name should be TSPLINK.)
      4. Select the NODE menu.
      5. Set the NODE number to 1 for the Master and press ENTER.
      6. Select the TSPLINK_CFG menu. (If the Series 2600 instruments used have firmware
Revision 1.4.0 or later installed, the menu name should be TSPLINK.)
      7. Select the RESET to initialize the TSP-Link.


Running this script creates functions that can be used to measure the common emitter
characteristics of transistors. The default values are for an NPN transistor type 2N5089.
```

```
      The functions created are:
            1. BJT_Comm_Emit_Vsb(istart, istop, isteps, vstart, vstop, vsteps,vsbsource)
--Default values istart = 10uA, istop = 50uA, isteps = 5, vstart = 0V, vstop = 10V,
--vsteps =100,vsbsource = 1V
            2. Print_Data(isteps,vsteps, ce_volt,ce_curr, base_curr)

      See detailed information listed in individual functions.

1) From Test Script Builder
- At the TSP> prompt in the Instrument Control Panel, type BJT_Comm_Emit_Vsb()

2) From an external program
      - Send the entire program text as a string using standard GPIB Write calls.


Rev1: JAC 7.23.2007

]]--


------------------ Keithley TSP Function ------------------

function BJT_Comm_Emit_Vsb(istart, istop, isteps, vstart, vstop, vsteps, vsbsource)
--Configure node 1 SMUB to source a --bias current on the base and node 1 SMUA performs a
voltage sweep on the Collector//Emitter from start to stop in a --user-defined number of
steps.
--Node 2 SMUA delivers a user-defined voltage bias to the substrate. Node  1 SMUB then
increments to next bias value
--and continues to stop value.
                                                --Returns measured voltage and current
values.

      --Global variables
      local l_irange = 100E-6 --Base current source range
      local l_vcmpl = 1 --Base source compliance

      local l_vrange = 40 --Collector-emitter voltage source range
      local l_icmpl = 100E-3 --Collector-emitter source compliance

      local l_vsbsource = vsbsource --Substrate bias value

      --Shared local variables
      local l_nplc = 1 --Integration rate of measurement

      --Local sweep variables
      local l_istart = istart --Base sweep start current
      local l_istop =  istop --Base sweep stop current
      local l_isteps = isteps --Number of steps in sweep

      local l_vstart = vstart --Collector-emitter sweep start voltage
```

```
local l_vstop =  vstop --Collector-emitter sweep stop voltage
local l_vsteps = vsteps --Number of steps in sweep

--Default values and level check
if (l_istart == nil) then --Use default value
      l_istart = 10E-6
end --if

if (l_istart > 100E-6) then --Coerce value
      l_istart = 100E-6
end --if

if (l_istop == nil) then --Use default value
      l_istop = 50E-6
end --if

if (l_istop > 500E-6) then --Coerce value
      l_istop = 500E-6
end --if

if (l_isteps == nil) then --Use default value
      l_isteps = 5
end --if

if (l_isteps > 100) then --Coerce value
      l_isteps = 100
end --if

local l_istep = (l_istop - l_istart)/ (l_isteps - 1) --Current step size
local l_isource_val = l_istart --Source value during sweep
local l_i = 1 --Iteration variable

if (l_vstart == nil) then --Use default value
      l_vstart = 0
end --if

if (l_vstart > 100E-3) then --Coerce value
      l_vstart = 100E-3
end --if

if (l_vstop == nil) then --Use default value
      l_vstop = 10
end --if

if (l_vstop > 40) then --Coerce value
      l_vstop = 40
end --if

if (l_vsteps == nil) then --Use default value
      l_vsteps = 100
```

```
	end --if

	if (l_vsteps > 2E+2) then --Coerce value
		l_vsteps = 2E+2
	end --if

	local l_vstep = (l_vstop - l_vstart)/ (l_vsteps - 1) --Voltage step size
	local l_vsource_val = l_vstart --Source value during sweep
	local l_v = 1 --Iteration variable

	if (l_vsbsource == nil) then --Use default value
		l_vsbsource = 1
	end --if

	if (l_vsbsource > 40) then --Coerce value
		l_vsbsource = 40
	end --if

	--Data tables
	local l_base_curr = {} --Create data table for sourced current
	local l_ce_volt = {} --Create data table for collector-emitter measured voltage
local l_ce_curr = {} --Create data table for collector-emitter measured current

	node[1].smua.reset() --Reset SMU
	node[1].smub.reset() --Reset SMU
	node[2].smua.reset() --Reset SMU

	errorqueue.clear() --Clear the error queue

	--Configure Collector/Emitter (Node 1 SMUA) source and measure settings
	node[1].smua.source.func = node[1].smua.OUTPUT_DCVOLTS
	node[1].smua.source.autorangev = node[1].smua.AUTORANGE_ON --Enable source
autorange
	node[1].smua.source.levelv = 0
	node[1].smua.source.limiti = l_icmpl
	node[1].smua.measure.autorangei = node[1].smua.AUTORANGE_ON --Enable measure
autorange

	node[1].smua.measure.autozero = node[1].smua.AUTOZERO_AUTO
	node[1].smua.measure.nplc = l_nplc --Measurement integration rate

	node[1].smua.source.output = node[1].smua.OUTPUT_ON --Enable Output

	--Configure Base (Node 1 SMUB) source and measure settings
	node[1].smub.source.func = node[1].smub.OUTPUT_DCAMPS
	node[1].smub.source.autorangei = node[1].smub.AUTORANGE_ON --Enable source
autorange
	node[1].smub.source.leveli = 0
	node[1].smub.source.limitv = l_vcmpl
```

```
	node[1].smub.measure.autorangev = node[1].smub.AUTORANGE_ON --Enable measure
autorange

	node[1].smub.measure.autozero = node[1].smub.AUTOZERO_AUTO
	node[1].smub.measure.nplc = l_nplc --Measurement integration rate

	node[1].smub.source.output = node[1].smub.OUTPUT_ON --Enable Output

	--Configure Substrate Bias (Node 2 SMUA) source settings
	node[2].smua.source.func = node[2].smua.OUTPUT_DCVOLTS
	node[2].smua.source.autorangev = node[2].smua.AUTORANGE_ON --Enable source
autorange
	node[2].smua.source.levelv = 0
	node[2].smua.source.limiti = l_icmpl
	node[2].smua.measure.autorangei = node[2].smua.AUTORANGE_ON --Enable measure
autorange

	node[2].smua.measure.autozero = node[2].smua.AUTOZERO_AUTO
	node[2].smua.measure.nplc = l_nplc --Measurement integration rate

	node[2].smua.source.output = node[2].smua.OUTPUT_ON --Enable Output

	--Execute sweep
	for l_i = 1, l_isteps  do

		node[2].smua.source.levelv = l_vsbsource
		node[1].smub.source.leveli = l_isource_val

		l_ce_volt[l_i] = {} --Create new row in table
		l_ce_curr[l_i] = {} --Create new row in table

		l_base_curr[l_i] = node[1].smub.measure.i() --Measure base current

		for l_v = 1,l_vsteps do


			if (l_v == 1) then --Intialize start source value
				l_vsource_val = l_vstart
			end --if

			--delay(1)
			l_ce_volt[l_i][l_v] = node[1].smua.measure.v() --Measure sourced
voltage
			l_ce_curr[l_i][l_v] = node[1].smua.measure.i() --Measure current

			l_vsource_val = l_vsource_val + l_vstep --Calculate new source value


			if (l_v == l_vsteps) then --Reinitialize voltage value after last
iteration
```

```
                        l_vsource_val = l_vstart
                    end --if

                    node[1].smua.source.levelv = l_vsource_val --Increment source

              end --for

              l_isource_val = l_isource_val + l_istep --Calculate new source value

       end--for

       node[1].smua.source.output = node[1].smua.OUTPUT_OFF --Disable output
       node[1].smub.source.output = node[1].smub.OUTPUT_OFF --Disable output
       node[2].smua.source.output = node[2].smua.OUTPUT_OFF --Disable output

       node[1].smua.source.levelv = 0 --Return source to bias level
       node[1].smub.source.leveli = 0 --Return source to bias level
       node[2].smua.source.levelv = 0 --Return source to bias level

       Print_Data(l_isteps,l_vsteps, l_ce_volt, l_ce_curr, l_base_curr,l_vsbsource)

end--function BJT_Comm_Emit()

function Print_Data(isteps,vsteps, ce_volt,ce_curr, base_curr, vsbsource)
--Print Data to output queue

       --Local Variables
       local l_isteps = isteps
       local l_vsteps = vsteps
       local l_i = 1 --Iteration variable
       local l_v = 1 --Iteration variable
       local l_ce_volt = ce_volt
       local l_ce_curr = ce_curr
       local l_base_curr = base_curr
       local l_vsbsource = vsbsource


       for l_i = 1, l_isteps  do
              print("")
              print("Base Current Bias (A)", l_base_curr[l_i])
              print("Substrate Bias (V)", l_vsbsource)
              print("Emitter Voltage (V)","Emitter Current (A)")

              for l_v = 1, l_vsteps  do
                     print(l_ce_volt[l_i][l_v], l_ce_curr[l_i][l_v])
              end --for
       end --for

 end --function Print_Data()
```

```
--BJT_Comm_Emit_Vsb()
```

Section 6. High Power Tests

Program 15. High Current with Voltage Measurement

```
--[[
KI2602Example_High_Current.tsp


This program is intended to perform the following:
      1. Set up both SMUs of a Model 2602 for current bias and measure voltage on specific
intervals.
      2. Deliver up to 2A @ 40V (1A @ 40V per SMU) by wiring each SMU in parallel

      Wiring: SMUA Hi to SMUB Hi, SMUA Lo to SMUB Lo

      WARNING: If either SMU reaches a compliance state, the instrument, device, or both
could be damaged.


System Requirements: 260x Firmware version: 1.0.2 or newer

Rev1: JAC 3.21.2006
Rev2: JAC 10.15.2007
        -Change l_sourcei value to sourcei/2. Desired current value at DUT is now
programmed.
--]]

function RunHighCurrent(sourcei, points)

    local l_sourcei = sourcei/2 --Local variable for Source Current Value
    local l_points = points --Local variable for number of points to sample
    local l_cmpl = 40 --compliance must not be reached!

    --Configure display
    display.clear()
    display.screen = display.SMUA_SMUB

    display.smua.measure.func = display.MEASURE_DCVOLTS
    display.smub.measure.func = display.MEASURE_DCVOLTS

    -- Configure source and measure settings.
    smua.source.output = smua.OUTPUT_OFF --Disable Output
    smub.source.output = smub.OUTPUT_OFF --Disable Output

    smua.source.func = smua.OUTPUT_DCAMPS --Set Output function
    smub.source.func = smub.OUTPUT_DCAMPS --Set Output function

    smua.source.leveli = 0 --Set output level
    smub.source.leveli = 0 --Set output level
```

```
    smua.source.rangei = l_sourcei --Set output level
    smub.source.rangei = l_sourcei --Set output level

    smua.source.limitv = l_cmpl --Set compliance level
    smub.source.limitv = l_cmpl --Set compliance level

    smua.measure.nplc = 1 --Set measurement aperture
    smub.measure.nplc = 1 --Set measurement aperture

    smua.measure.autozero = smua.AUTOZERO_AUTO --Set Autozero mode
    smub.measure.autozero = smub.AUTOZERO_AUTO --Set Autozero mode

    -- Setup SMUA buffer to store all the result(s) in and start testing.

    smua.nvbuffer1.clear() --Clear Nonvolatile buffer

    smua.nvbuffer1.appendmode = 0 --Append buffer? 0 = No, 1 = Yes

    smua.nvbuffer1.collecttimestamps = 0 --Collect Timestamps? 0 = No, 1 = Yes

    smua.nvbuffer1.collectsourcevalues = 0 --Collect Source Values? 0 = No, 1 = Yes

    smua.source.output = smua.OUTPUT_ON --Enable outputs
    smub.source.output = smua.OUTPUT_ON --Enable outputs

    smua.source.leveli = l_sourcei  -- Program source to level.
    smub.source.leveli = l_sourcei  -- Program source to level.

    smua.measure.count = l_points --Number of points to collect

    smua.measure.v(smua.nvbuffer1)  -- Measure voltage and store in reading buffer.

    smua.source.output = smua.OUTPUT_OFF
    smub.source.output = smub.OUTPUT_OFF

    -- Update the front panel display and restore modified settings.
    smua.source.leveli = 0
    smub.source.leveli = 0

       printbuffer(1,l_points,smua.nvbuffer1)

end --function RunHighCurrent(sourcei, points)

--RunHighCurrent(1, 10)
```

Program 16. High Voltage with Current Measurement

```
--[[
KI2602Example_High_Voltage.tsp


This program is intended to perform the following:
      1. Set up both SMUs of a Model 2602 for voltage bias and measure current on specific
intervals.
      2. Deliver up to 80V @ 1A (40V @ 1A per SMU) by wiring each SMU Voltage Source in
series.

      Wiring: SMUA Lo to SMUB Hi, SMUA Hi to DUT, SMUB Lo to DUT

      WARNING: If either SMU reaches a compliance state, the instrument, device, or both
could be damaged.


System Requirements: 260x Firmware version: 1.0.2 or newer

Rev1: JAC 3.21.2006
Rev2: JAC 10.15.2007
        -Change l_sourcev value to sourcev/2. Desired voltage value at DUT is now
programmed.
--]]

function RunHighVoltage(sourcev, points)

    local l_sourcev = sourcev/2 --Local variable for Source Voltage Value
    local l_points = points --Local variable for number of points to sample
    local l_cmpl = 1 --compliance

    --Configure display
    display.clear()
    display.screen = display.SMUA_SMUB

    display.smua.measure.func = display.MEASURE_DCAMPS
    display.smub.measure.func = display.MEASURE_DCAMPS

    -- Configure source and measure settings.
    smua.source.output = smua.OUTPUT_OFF --Disable Output
    smub.source.output = smub.OUTPUT_OFF --Disable Output

    smua.source.func = smua.OUTPUT_DCVOLTS --Set Output function
    smub.source.func = smub.OUTPUT_DCVOLTS --Set Output function

    smua.source.levelv = 0 --Set output level
    smub.source.levelv = 0 --Set output level

    smua.source.rangev = l_sourcev --Set output level
```

```
    smub.source.rangev = l_sourcev --Set output level

    smua.source.limiti = l_cmpl --Set compliance level
    smub.source.limiti = l_cmpl --Set compliance level

    smua.measure.nplc = 1 --Set measurement aperture
    smub.measure.nplc = 1 --Set measurement aperture

    smua.measure.autozero = smua.AUTOZERO_AUTO --Set Autozero mode
    smub.measure.autozero = smub.AUTOZERO_AUTO --Set Autozero mode

    -- Setup SMUA buffer to store all the result(s) in and start testing.

    smua.nvbuffer1.clear() --Clear Nonvolatile buffer

    smua.nvbuffer1.appendmode = 0 --Append buffer? 0 = No, 1 = Yes

    smua.nvbuffer1.collecttimestamps = 0 --Collect Timestamps? 0 = No, 1 = Yes

    smua.nvbuffer1.collectsourcevalues = 0 --Collect Source Values? 0 = No, 1 = Yes

    smua.source.output = smua.OUTPUT_ON --Enable outputs
    smub.source.output = smua.OUTPUT_ON --Enable outputs

    smua.source.levelv = l_sourcev  -- Program source to level.
    smub.source.levelv = l_sourcev  -- Program source to level.

    smua.measure.count = l_points --Number of points to collect

    smua.measure.i(smua.nvbuffer1)  -- Measure current and store in reading buffer.

    smua.source.output = smua.OUTPUT_OFF
    smub.source.output = smub.OUTPUT_OFF

    -- Update the front panel display and restore modified settings.
    smua.source.levelv = 0
    smub.source.levelv = 0

       printbuffer(1,l_points,smua.nvbuffer1)

end --function RunHighVoltage(sourcev, points)

--RunHighVoltage(40, 10)
```

KEITHLEY

A GREATER MEASURE OF CONFIDENCE

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

BELGIUM
Sint-Pieters-Leeuw
Ph: 02-3630040
Fax: 02-3630064
info@keithley.nl
www.keithley.nl

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

FRANCE
Saint-Aubin
Ph: 01-64532020
Fax: 01-60117726
info@keithley.fr
www.keithley.fr

GERMANY
Germering
Ph: 089-84930740
Fax: 089-84930734
info@keithley.de
www.keithley.de

INDIA
Bangalore
Ph: 080-26771071, -72, -73
Fax: 080-26771076
support_india@keithley.com
www.keithley.com

ITALY
Peschiera Borromeo (Mi)
Ph: 02-5538421
Fax: 02-55384228
info@keithley.it
www.keithley.it

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

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

MALAYSIA
Penang
Ph: 60-4-643-9679
Fax: 60-4-643-3794
koh_william@keithley.com
www.keithley.com

NETHERLANDS
Gorinchem
Ph: 0183-635333
Fax: 0183-630821
info@keithley.nl
www.keithley.nl

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

SWITZERLAND
Zürich
Ph: 044-8219444
Fax: 044-8203081
info@keithley.ch
www.keithley.ch

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

UNITED KINGDOM
Theale
Ph: 0118-9297500
Fax: 0118-9297519
info@keithley.co.uk
www.keithley.co.uk

Printed in the U.S.A. No. 2911 12.03.09