tek.com/keithley

Keithley Test Environment (KTE)

Programmer's Manual

S500-904-01 Rev. B / January 2019

S500-904-01B

KEITHLEY

A Tektronix Company

Keithley Test Environment (KTE)

Programmer's Manual

© 2019, Keithley Instruments, LLC

Cleveland, Ohio, U.S.A.

All rights reserved.

Any unauthorized reproduction, photocopy, or use of the information herein, in whole or in part, without the prior written approval of Keithley Instruments, LLC, is strictly prohibited.

These are the original instructions in English.

All Keithley Instruments product names are trademarks or registered trademarks of Keithley Instruments, LLC. Other brand names are trademarks or registered trademarks of their respective holders.

Microsoft, Visual C++, Excel, and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Document number: S500-901-01 Rev. B / January 2019

KEITHLEY
A Tektronix Company

Safety precautions

The following safety precautions should be observed before using this product and any associated instrumentation. Although some instruments and accessories would normally be used with nonhazardous voltages, there are situations where hazardous conditions may be present.

This product is intended for use by personnel who recognize shock hazards and are familiar with the safety precautions required to avoid possible injury. Read and follow all installation, operation, and maintenance information carefully before using the product. Refer to the user documentation for complete product specifications.

If the product is used in a manner not specified, the protection provided by the product warranty may be impaired.

The types of product users are:

Responsible body is the individual or group responsible for the use and maintenance of equipment, for ensuring that the equipment is operated within its specifications and operating limits, and for ensuring that operators are adequately trained.

Operators use the product for its intended function. They must be trained in electrical safety procedures and proper use of the instrument. They must be protected from electric shock and contact with hazardous live circuits.

Maintenance personnel perform routine procedures on the product to keep it operating properly, for example, setting the line voltage or replacing consumable materials. Maintenance procedures are described in the user documentation. The procedures explicitly state if the operator may perform them. Otherwise, they should be performed only by service personnel.

Service personnel are trained to work on live circuits, perform safe installations, and repair products. Only properly trained service personnel may perform installation and service procedures.

Keithley products are designed for use with electrical signals that are measurement, control, and data I/O connections, with low transient overvoltages, and must not be directly connected to mains voltage or to voltage sources with high transient overvoltages. Measurement Category II (as referenced in IEC 60664) connections require protection for high transient overvoltages often associated with local AC mains connections. Certain Keithley measuring instruments may be connected to mains. These instruments will be marked as category II or higher.

Unless explicitly allowed in the specifications, operating manual, and instrument labels, do not connect any instrument to mains.

Exercise extreme caution when a shock hazard is present. Lethal voltage may be present on cable connector jacks or test fixtures. The American National Standards Institute (ANSI) states that a shock hazard exists when voltage levels greater than 30 V RMS, 42.4 V peak, or 60 VDC are present. A good safety practice is to expect that hazardous voltage is present in any unknown circuit before measuring.

Operators of this product must be protected from electric shock at all times. The responsible body must ensure that operators are prevented access and/or insulated from every connection point. In some cases, connections must be exposed to potential human contact. Product operators in these circumstances must be trained to protect themselves from the risk of electric shock. If the circuit is capable of operating at or above 1000 V, no conductive part of the circuit may be exposed.

Do not connect switching cards directly to unlimited power circuits. They are intended to be used with impedance-limited sources. NEVER connect switching cards directly to AC mains. When connecting sources to switching cards, install protective devices to limit fault current and voltage to the card.

Before operating an instrument, ensure that the line cord is connected to a properly-grounded power receptacle. Inspect the connecting cables, test leads, and jumpers for possible wear, cracks, or breaks before each use.

When installing equipment where access to the main power cord is restricted, such as rack mounting, a separate main input power disconnect device must be provided in close proximity to the equipment and within easy reach of the operator.

For maximum safety, do not touch the product, test cables, or any other instruments while power is applied to the circuit under test. ALWAYS remove power from the entire test system and discharge any capacitors before: connecting or disconnecting cables or jumpers, installing or removing switching cards, or making internal changes, such as installing or removing jumpers.

Do not touch any object that could provide a current path to the common side of the circuit under test or power line (earth) ground. Always make measurements with dry hands while standing on a dry, insulated surface capable of withstanding the voltage being measured.

For safety, instruments and accessories must be used in accordance with the operating instructions. If the instruments or accessories are used in a manner not specified in the operating instructions, the protection provided by the equipment may be impaired.

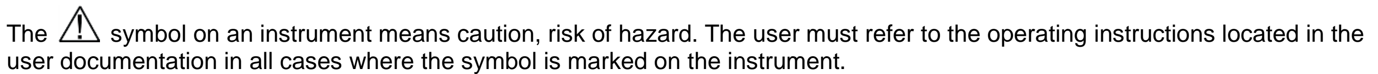
Do not exceed the maximum signal levels of the instruments and accessories. Maximum signal levels are defined in the specifications and operating information and shown on the instrument panels, test fixture panels, and switching cards.

When fuses are used in a product, replace with the same type and rating for continued protection against fire hazard.

Chassis connections must only be used as shield connections for measuring circuits, NOT as protective earth (safety ground) connections.

If you are using a test fixture, keep the lid closed while power is applied to the device under test. Safe operation requires the use of a lid interlock.

If a screw is present, connect it to protective earth (safety ground) using the wire recommended in the user documentation.

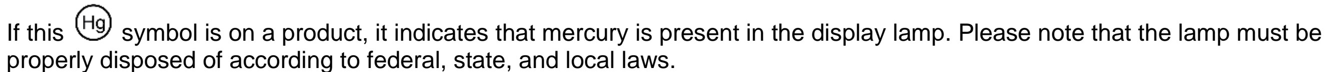


The symbol on an instrument means caution, risk of hazard. The user must refer to the operating instructions located in the user documentation in all cases where the symbol is marked on the instrument.

The symbol on an instrument means warning, risk of electric shock. Use standard safety precautions to avoid personal contact with these voltages.

The symbol on an instrument shows that the surface may be hot. Avoid personal contact to prevent burns.

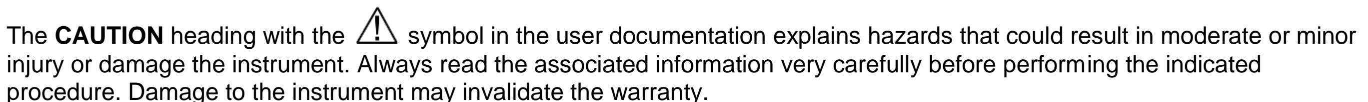
The symbol indicates a connection terminal to the equipment frame.



If this symbol is on a product, it indicates that mercury is present in the display lamp. Please note that the lamp must be properly disposed of according to federal, state, and local laws.

The **WARNING** heading in the user documentation explains hazards that might result in personal injury or death. Always read the associated information very carefully before performing the indicated procedure.

The **CAUTION** heading in the user documentation explains hazards that could damage the instrument. Such damage may invalidate the warranty.



The **CAUTION** heading with the symbol in the user documentation explains hazards that could result in moderate or minor injury or damage the instrument. Always read the associated information very carefully before performing the indicated procedure. Damage to the instrument may invalidate the warranty.

Instrumentation and accessories shall not be connected to humans.

Before performing any maintenance, disconnect the line cord and all test cables.

To maintain protection from electric shock and fire, replacement components in mains circuits — including the power transformer, test leads, and input jacks — must be purchased from Keithley. Standard fuses with applicable national safety approvals may be used if the rating and type are the same. The detachable mains power cord provided with the instrument may only be replaced with a similarly rated power cord. Other components that are not safety-related may be purchased from other suppliers as long as they are equivalent to the original component (note that selected parts should be purchased only through Keithley to maintain accuracy and functionality of the product). If you are unsure about the applicability of a replacement component, call a Keithley office for information.

Unless otherwise noted in product-specific literature, Keithley instruments are designed to operate indoors only, in the following environment: Altitude at or below 2,000 m (6,562 ft); temperature 0 °C to 50 °C (32 °F to 122 °F); and pollution degree 1 or 2.

To clean an instrument, use a cloth dampened with deionized water or mild, water-based cleaner. Clean the exterior of the instrument only. Do not apply cleaner directly to the instrument or allow liquids to enter or spill on the instrument. Products that consist of a circuit board with no case or chassis (e.g., a data acquisition board for installation into a computer) should never require cleaning if handled according to instructions. If the board becomes contaminated and operation is affected, the board should be returned to the factory for proper cleaning/servicing.

Safety precaution revision as of June 2017.

Table of contents

Introduction 1-1

Contact information 1-1

Systems documentation 1-1

Overview 1-2

Conventions used in this manual 1-2

LPTLib command reference 2-1

Introduction 2-1

Categorized command lists 2-1

- Combination commands 2-2
- Dual-site commands 2-2
- General commands 2-2
- GPIB commands 2-3
- Matrix commands 2-3
- Measure commands 2-3
- Pulse generator commands 2-4
- Range commands 2-4
- Scope card commands 2-5
- Spectrum analyzer commands 2-5
- Source commands 2-5
- Timing commands 2-6

Using the LPTLib 2-6

- Measuring 2-6
- Sourcing and limits 2-7
- Ranging 2-8
- Matrix operations 2-9
- Sweeping 2-10
- Triggers 2-11
- GPIB 2-12
- Instruments and instrument drivers 2-12
- Instrument and terminal IDs 2-15
- Optimizing test sequences 2-16
- Error handling 2-16

LPTLib command descriptions 2-24

- addcon 2-24
- adelay 2-25
- asweepX 2-26
- avgX 2-28
- bmeasX 2-30
- bsweepX 2-32
- clrcon 2-33
- clrscn 2-34
- clrtrg 2-35
- conpin 2-37
- conpth 2-39
- delay 2-40
- delcon 2-41
- devclr 2-42
- devint 2-42
- disable 2-44
- enable 2-44

forceX.... 2-45
getlpterr.... 2-46
getstatus.... 2-47
imeast.... 2-48
insbind.... 2-49
intgX.... 2-50
kibdefclr.... 2-52
kibdefint.... 2-53
kibrcv.... 2-54
kibsnd.... 2-55
kibspl.... 2-56
kibsplw.... 2-57
limitX.... 2-57
lorangeX.... 2-59
measX.... 2-61
mpulse.... 2-63
pgu_current_limit.... 2-64
pgu_delay.... 2-64
pgu_fall.... 2-65
pgu_halt.... 2-66
pgu_height.... 2-66
pgu_init.... 2-67
pgu_load.... 2-67
pgu_mode.... 2-68
pgu_offset.... 2-69
pgu_period.... 2-69
pgu_range.... 2-70
pgu_rise.... 2-71
pgu_select.... 2-71
pgu_trig.... 2-72
pgu_trig_burst.... 2-72
pgu_trig_unit.... 2-73
pgu_width.... 2-74
pulseX.... 2-75
rangeX.... 2-77
rdelay.... 2-79
refctrl.... 2-79
rsa_close.... 2-80
rsa_detect_peaks.... 2-81
rsa_init.... 2-83
rsa_measure.... 2-83
rsa_measure_next.... 2-84
rsa_selftest.... 2-85
rsa_setup.... 2-86
rtfary.... 2-88
rttrigary.... 2-88
savgX.... 2-89
scp_close.... 2-90
scp_detect_peaks.... 2-91
scp_init.... 2-93
scp_measure.... 2-93
scp_measure_next.... 2-94
scp_selftest.... 2-95
scp_setup.... 2-96
searchX.... 2-97
setauto.... 2-100
setmode.... 2-101
setmode modifier tables.... 2-102
setXmtr.... 2-108
sintgX.... 2-109
site_disable.... 2-110

site_enable ... 2-111
site_mapping ... 2-112
site_status ... 2-113
smeasX ... 2-114
ssmeasX ... 2-116
sweepX ... 2-118
trigXg, trigXl ... 2-121
tstsel ... 2-124

PARLib command reference ... 3-1

Introduction ... 3-1

How to use the library reference ... 3-2

Categorized subroutine lists ... 3-4
Bipolar subroutines ... 3-4
FET and JFET subroutines ... 3-4
Math and support subroutines ... 3-5
MOSFET subroutines ... 3-5
Resistors, diodes, capacitors, and special structure subroutines ... 3-6

Subroutine descriptions ... 3-6
beta1 ... 3-6
beta2 ... 3-8
beta2a ... 3-9
beta3a ... 3-11
bice ... 3-13
bkdn ... 3-14
bvcbo ... 3-15
bvcbo1 ... 3-17
bvceo ... 3-18
bvceo2 ... 3-19
bvces ... 3-21
bvces1 ... 3-22
bvdss ... 3-24
bvdss1 ... 3-25
bvebo ... 3-26
cap ... 3-28
deltl1 ... 3-29
deltw1 ... 3-30
ev ... 3-31
fimv ... 3-33
fnddat ... 3-34
fndtrg ... 3-35
fvmi ... 3-36
gamma1 ... 3-37
gd ... 3-39
gm ... 3-40
ibic1 ... 3-42
icbo ... 3-43
iceo ... 3-44
ices ... 3-45
id1 ... 3-46
idsat ... 3-48
idss ... 3-49
idvsvd ... 3-51
idvsvg ... 3-52
iebo ... 3-53
isubmx ... 3-55
kdelay ... 3-56

leak ... 3-57
logstp ... 3-58
rcsat ... 3-59
re ... 3-61
res ... 3-63
res2 ... 3-64
res4 ... 3-65
resv ... 3-66
rvdp ... 3-67
tdelay ... 3-68
tox ... 3-69
vbes ... 3-70
vf ... 3-71
vg2 ... 3-72
vgsat ... 3-74
vp ... 3-76
vp1 ... 3-78
vt14 ... 3-79
vtati ... 3-80
vtext ... 3-82
vtext2 ... 3-85
vtext3 ... 3-87

HVLib command reference ... 4-1

Introduction ... 4-1

How to use the library reference ... 4-1

High-Voltage Library commands ... 4-4
gate_charge ... 4-4
hv_bvsweep ... 4-6
hvcv_3term ... 4-8
hvcv_3term_basic ... 4-11
hvcv_comp ... 4-13
hvcv_genCompData ... 4-14
hvcv_genCompFreq ... 4-16
hvcv_getData ... 4-18
hvcv_intgcg ... 4-19
hvcv_measure ... 4-21
hvcv_storeData ... 4-23
hvcv_sweep ... 4-24
hvcv_sweep_basic ... 4-27
hvcv_test ... 4-29
hvcv_test_basic ... 4-32

KI_MultiSite command reference ... 5-1

Introduction ... 5-1
multi_site_clear_mapping() ... 5-1
multi_site_mapping() ... 5-3

Prober and prober driver command reference ... 6-1

Introduction ... 6-1
PrAbsMove ... 6-1
PrAdjustZHeight ... 6-2
PrAutoAlign ... 6-2
PrCassetteMap ... 6-2
PrCassetteMask ... 6-3

PrCheckOptions 6-4
PrChuck 6-4
PrClearAll 6-5
PrClearPipeline 6-5
PrError 6-5
PrGetNxtWafer 6-6
PrGetProduct 6-6
PrGetWafer 6-7
PrInit 6-7
PrLoad 6-8
PrLoadProduct 6-8
PrLowerBoat 6-9
PrMove 6-9
PrNeedleClean 6-9
PrProfile 6-10
PrPutNxtSlot 6-10
PrPutWafer 6-11
PrQueryChuckTemp 6-11
PrQueryZHeight 6-12
PrReadId 6-12
PrRelMove 6-12
PrRelReturn 6-13
PrSerialPoll 6-13
PrSetChuckTemp 6-13
PrSetDiam 6-14
PrSetDieSize 6-14
PrSetFlat 6-14
PrSetMode 6-14
PrSetPipeline 6-15
PrSetQuadrant 6-15
PrSetRefDie 6-15
PrSetSlotStatus 6-16
PrSetTime 6-17
PrSetUnits 6-17
PrSetZHeight 6-17
PrSmifClamp 6-18
PrSmifLock 6-18
PrSmifStatus 6-19
PrStart 6-20
PrStatus 6-20
PrStop 6-21
PrUnLoad 6-21
PrWriteRead 6-21
PrWriteReadSRQ 6-22
PrZParams 6-24
PrZTravel 6-24

Keithley data files (KDF) library command reference 7-1

Overview 7-1

Data logging routines 7-2
PutLot 7-2
PutWafer 7-3
PutSite 7-3
PutParam 7-4
PutParamList 7-5
EndLot 7-6
EndWafer 7-6
EndSite 7-6
GetLot 7-7

GetWafer 7-8
GetSite 7-9
GetParam 7-10
GetParamList 7-11
GetLotData 7-11
MatchParam2Limit 7-12
FileExist 7-12
LotExist 7-12
GetStartTime 7-13
DeleteLot 7-13
DeleteWafer 7-13
DeleteSite 7-14
DeleteParam 7-14
DeleteLimitCode 7-15
DeleteLimit 7-15

Update comment routines 7-15
GetComment 7-15
PutComment 7-16

Update limits routines 7-16
GetLimitCode 7-16
GetLimit 7-16
PutLimit 7-17

Structure handling routines 7-17
AddNew[STRUCTURE] 7-17
CreateNew[STRUCTURE] 7-18
FindFirst[STRUCTURE] 7-19
FindLast[STRUCTURE] 7-19
FindNext[STRUCTURE] 7-20
FindPrev[STRUCTURE] 7-21
InsertNew[STRUCTURE] 7-22
Remove[STRUCTURE] 7-23
LimitExist 7-24

KTXE_RP zone-based testing command reference 8-1

Introduction 8-1
KTXE_RP_CleanUpWDF 8-1
KTXE_RP_CreateRandomWDF 8-1
KTXE_RP_CreateWPF 8-2
KTXE_RP_GetUsrArgs 8-2
KTXE_RP_RemoveWPF 8-2

KTXE_AT result-based testing command reference 9-1

Introduction 9-1
KTXE_AT_alternate_site_site_end() 9-1
KTXE_AT_alternate_site_test_end() 9-1
KTXE_AT_AlterWWP() 9-2
KTXE_AT_CheckResWithLimits() 9-2
KTXE_AT_cleanup_site() 9-3
KTXE_AT_debug_print() 9-3
KTXE_AT_demo_data_func() 9-3
KTXE_AT_enable_kdf() 9-3
KTXE_AT_FindAltSite() 9-4
KTXE_AT_generate_val() 9-4
KTXE_AT_LogResultList() 9-5
KTXE_AT_more_sites_cur_wafer_site_end() 9-5
KTXE_AT_more_tests_curr_wafer_site_end() 9-5

KTXE_AT_more_tests_curr_wafer_wafer_begin() 9-5
KTXE_AT_more_tests_next_wafer_site_end() 9-6
KTXE_AT_wafer_begin() 9-6

Keithley User Interface Library command reference 10-1

Introduction 10-1
GetProgramArgs 10-1
InitUINew 10-3
InputMsgDlg 10-4
LotDlg 10-4
OkCancelAbortMsgDlg 10-5
OkCancelMsgDlgDialog 10-5
OkMsgDlg 10-5
QuitUI 10-6
ScrollMsgDlg 10-6
ScrollMsgDlgClr 10-6
ScrollMsgDlgMsg 10-7
StatusDlg 10-7
UpdateModelessDlgs 10-8
UpdateStatusDlg 10-8
VarMsgDlg 10-9
WfrIdsDlgDialog 10-10
WfrIdDlg 10-11
YesNoAbortMsgDlg 10-13
YesNoCancelMsgDlg 10-13
ContSkipAbortDlg 10-13
LBoxDlg 10-14

Data pool command reference 11-1

Introduction 11-1
dpAddData 11-1
dpAddPointer 11-2
dpAddArray 11-3
*dpGetDataPtr 11-3
*dpGetPointer 11-4
*dpGetArrayElement 11-4
dpRemoveData 11-5
dpPrintData 11-5
dpPrintAllData 11-6

Index .... Index- 1

Section 1

Introduction

In this section:

Contact information 1-1
Systems documentation 1-1
Overview 1-2
Conventions used in this manual 1-2

Contact information

If you have any questions after you review the information in this documentation, please contact your local Keithley Instruments office, sales partner, or distributor. You can also call the corporate headquarters of Keithley Instruments (toll-free inside the U.S. and Canada only) at 1-800-935-5595, or from outside the U.S. at +1-440-248-0400. For worldwide contact numbers, visit the Keithley Instruments website (tek.com/keithley).

Systems documentation

Documentation for your system is available at tek.com/keithley. Following is a list of documentation for Keithley systems, including the document part numbers.

- S530 Parametric Test System Administrative Guide (S530-924-01)
- S530 Parametric Test System Reference Manual (S530-901-01)
- S535 Wafer Acceptance Test System Administrative Guide (S535-924-01)
- S535 Wafer Acceptance Test System Reference Manual (S535-901-01)
- S540 Power Semiconductor Test System Administrative Guide (S540-924-01)
- S540 Power Semiconductor Test System Reference Manual (S540-901-01)
- Keithley Test Environment (KTE) Programmer's Manual (S500-904-01)
- KIGEM Automation Software Reference Manual (KIGEM-901-01)
- KIGEM Automation Software User's Manual (KIGEM-900-01)

Overview

This manual contains detailed descriptions of commands you can use to configure and control your parametric test system.

The following command libraries are described in this manual:

- Linear Parametric Test Library (LPTLib)
- Parametric Test Subroutine Library (PARLib)
- High-Voltage Library (HVLib; S540 systems only)
- KI_MultiSite User Library (S535 systems only)
- Prober and Prober Driver Library (PRBLib)
- Keithley Data Files Library (KDF)
- KTXE_RP Zone-Based Testing User Library
- KTXE_AT Result-Based Testing User Library
- Keithley User Interface Library (KUI)
- Data Pool User Library

Conventions used in this manual

Throughout this manual, the following conventions are used when explaining the commands:

- All LPTLib commands are case-sensitive and must be entered as lower case when writing programs.
- Parameters that are user-supplied are shown in *`monospace italic`* font.
- A capital letter *`X`* shown in a command name indicates that you must select from a list of replacement suffixes. Using the example `force`*`X`* command, the *`X`* can be replaced with either a `v` for voltage or `i` for current.

The following table contains a list of valid suffixes, the parameter each represents, and the units used throughout the LPTLib for that parameter.

Suffix	**Parameter**	**Unit**
c	Capacitance	Farads
f	Frequency	Hertz
g	Conductance	Siemens
i	Current	Amperes
q	Charge	Coulombs
r	Resistance	Ohms
rh	Relative humidity	Percent
temp	Temperature	Degree Celsius
t	Time	Seconds
v	Voltage	Volts

- **Brackets** `[]` are used to enclose optional arguments of a command.
- **Period strings (. . .)** indicate additional arguments or commands that can be added.
- **Periods (.)** indicate data not shown in an example because it is not necessary to help explain the command.

Section 2

LPTLib command reference

In this section:

Introduction 2-1
Categorized command lists 2-1
Using the LPTLib.. 2-6
LPTLib command descriptions 2-23

Introduction

The Keithley line of semiconductor test systems uses function libraries to control the instruments in the system. These libraries are called test control libraries. A test control library is the lowest level software interface to a parametric tester.

The Linear Parametric Test Library (LPTLib) is one of the primary libraries that you can use with your system. This section contains detailed information about using the LPTLib and documents the Linear Parametric Test Library (LPTLib) commands.

Categorized command lists

The tables that follow contain all of the commands grouped by function, with a brief description of the purpose of the command and a hyperlink to the full command description.

Combination commands

Command	Description
asweepX (on page 2-26)	Sweep with a user-defined force array (i, v).
bmeasX (on page 2-29)	Block measurement; take a series of readings as quickly as possible (i, v).
bsweepX (on page 2-32)	Sweep and shutdown source if device meets trigger condition (i, v).
clrscn (on page 2-34)	Clear any sweep measurements.
clrtrg (on page 2-35)	Clear any set triggers.
mpulse (on page 2-63)	Force a pulse and measure voltage and current.
rtfary (on page 2-88)	Return forced array after sweep.
savgX (on page 2-89)	Average each point of associated sweep (i, v).
searchX (on page 2-97)	Search for a specific current or voltage.
sintgX (on page 2-109)	Integrate each point of associated sweep (i, v, c, g).
smeasX (on page 2-114)	Measure each point of associated sweep (i, t, v).
sweepX (on page 2-118)	Sweep a specified range of current or voltage.
trigXg, trigXl (on page 2-121)	Trigger if a measurement is greater than a specific value (i, t, v).
	Trigger if a measurement is less than a specific value (i, t, v).

Dual-site commands

NOTE

These commands are only compatible with S535 test systems.

Command	Description
site_disable (on page 2-110)	Disable dual-site mode for the specified `siteid`.
site_enable (on page 2-111)	Enable dual-site mode for the specified `siteid`.
site_mapping (on page 2-112)	Establish a new pin mapping between Site_0 and Site_1.
site_status (on page 2-113)	Read the state of the specified site and places it in the `state` variable.

General commands

Command	Description
devclr (on page 2-42)	Set all sources to a zero state.
devint (on page 2-42)	Reset all instruments and clear the system.
getlpterr (on page 2-46)	Get last LPTLib error since `devint` command.
getstatus (on page 2-47)	Return operating status of instrument.
insbind (on page 2-49)	Establish a cooperative relationship between two instruments.
setmode (on page 2-101)	Set operating mode.

GPIB commands

Command	Description
kibdefclr (on page 2-52)	Clear instrument on `devclr` command.
kibdefint (on page 2-53)	Clear instrument on `delay` command.
kibrcv (on page 2-54)	Read device-dependent string.
kibsnd (on page 2-55)	Send device-dependent command.
kibspl (on page 2-56)	Serial poll an instrument.
kibsplw (on page 2-57)	Synchronous serial poll an instrument.

Matrix commands

Command	Description
addcon (on page 2-24)	Add a connection.
clrcon (on page 2-33)	Disconnect all crosspoint connections.
conpin (on page 2-37)	Connect a pin or instrument terminal.
conpth (on page 2-39)	Connect pins and instruments using a specific pathway.
delcon (on page 2-41)	Remove specific matrix connections

Measure commands

Command	Description
avgX (on page 2-28)	Averages measurements of voltage, current, conductance, or capacitance.
bmeasX (on page 2-29)	Block measurement; make a series of readings as quickly as possible.
intgX (on page 2-50)	Integrates a measurement of voltage, current, conductance, or capacitance.
measX (on page 2-61)	Measure a voltage, current, conductance, or capacitance.
refctrl (on page 2-79)	Enable or disable automatic reference measurements.
setXmtr (on page 2-108)	Set the source to operate as a voltmeter or current meter.
ssmeasX (on page 2-116)	Steady state measurement (i, v).

Pulse generator commands

NOTE

These commands are only compatible with systems that have 4220-PGU pulse cards.

Command	Description
pgu_current_limit (on page 2-64)	Force a voltage or current.
pgu_delay (on page 2-64)	Set the trigger delay time.
pgu_fall (on page 2-65)	Set the fall time of the pulse.
pgu_halt (on page 2-66)	Stop all the pulse channels.
pgu_height (on page 2-66)	Set the peak-to-peak height of the pulse.
pgu_init (on page 2-67)	Initialize communication with pulse card and set pulse generator to default conditions.
pgu_load (on page 2-67)	Set the load impedance of a pulse.
pgu_mode (on page 2-68)	Set the pulse mode of the pulse generator.
pgu_offset (on page 2-69)	Set the peak-to-peak height and DC offset of the pulse.
pgu_period (on page 2-69)	Set the period of the pulse in seconds.
pgu_range (on page 2-70)	Set the voltage range of a pulse generator channel.
pgu_rise (on page 2-70)	Set the rise time of the pulse.
pgu_trig (on page 2-72)	Trigger first pulse generator unit and output waveforms.
pgu_trig_burst (on page 2-72)	Trigger a specified number of pulses.
pgu_trig_unit (on page 2-73)	Trigger a specified pulse generator unit, or units, to output waveforms.
pgu_width (on page 2-74)	Set the width of the pulse.

Range commands

Command	Description
lorangeX (on page 2-59)	Define lowest range an instrument should use during autorange operation (i, v).
rangeX (on page 2-77)	Put a measuring instrument on a specific range (c, i, v).
setauto (on page 2-100)	Re-enable autorange mode.

Scope card commands

NOTE

These commands are only compatible with systems that include 4200-SCP2HR scope cards.

Command	Description
scp_close (on page 2-90)	Disconnect communications to the scope card.
scp_detect_peaks (on page 2-91)	Return frequencies in signal amplitude order.
scp_init (on page 2-93)	Initialize the scope card to a default state.
scp_measure (on page 2-93)	Measure frequency and amplitude of the strongest signal.
scp_measure_next (on page 2-94)	Get the frequency and amplitude of next highest peak in frequency spectrum.
scp_selftest (on page 2-95)	Run an internal self-test of the scope card.
scp_setup (on page 2-96)	Set the start, stop, and step frequencies of a scan.

Spectrum analyzer commands

NOTE

These commands are only compatible with systems that include an RSA306B USB Spectrum Analyzer. In Keithley systems, the RSA306B functions as a replacement for discontinued scope cards. Spectrum analyzer capabilities may be added in the future.

Command	Description
rsa_close (on page 2-80)	Disconnect communications to the spectrum analyzer.
rsa_detect_peaks (on page 2-81)	Return frequencies in signal amplitude order.
rsa_init (on page 2-83)	Initialize spectrum analyzer to its default state.
rsa_measure (on page 2-83)	Measure the frequency and amplitude of the strongest signal.
rsa_measure_next (on page 2-84)	Return the frequency and amplitude of the next highest peak.
rsa_selftest (on page 2-85)	Runs the specified self-test and returns a status.
rsa_setup (on page 2-86)	Sets the start, stop, and step frequencies of a scan.

Source commands

Command	Description
forceX (on page 2-45)	Program a source instrument to output voltage or current at a specified level.
limitX (on page 2-57)	Limit an instrument to a set voltage or current other than the instrument default.
pulseX (on page 2-75)	Force voltage or current at a specified level for a specified amount of time.

Timing commands

Command	Description
adelay (on page 2-25)	Specify an array of delay points to use in a sweep.
delay (on page 2-40)	Set a user-defined delay in a test sequence (in milliseconds).
disable (on page 2-44)	Disable the timer.
enable (on page 2-44)	Initialize and start the timer.
imeast (on page 2-48)	Read the timer (immediate measure time).
rdelay (on page 2-79)	Set a user-defined delay (in seconds).

Using the LPTLib

The following topics describe how to use the Linear Parametric Test Library (LPTLib).

Measuring

The most important part of a parametric tester is its ability to make measurements. There are three types of measurements you can make with the Linear Parametric Test Library (LPTLib):

- Ordinary measurements (on page 2-6)
- Averaged measurements (on page 2-6)
- Integrated measurements (on page 2-6)

The type of measurement you make depends on the type of noise you are trying to eliminate from your measurement.

Ordinary measurements

Ordinary measurements are made with the `meas`*X* LPTLib command. This is the fastest single-point measurement you can make. Use this type of measurement when speed is most important or when noise is not significant.

Averaged measurements

Averaged measurements are made with the `avg`*X* LPTLib command. The `avg`*X* command makes an averaged measurement by making several single-point measurements and averaging them. Averaged measurements reduce the effects of random noise.

Integrated measurements

Integrated measurements are made with the `intg`*X* LPTLib command. Integrated measurements examine the signal over a longer period and reduce the effects of AC noise. Like averaged measurements, integrated measurements reduce the effects of random noise, but they also reject noise with a period that is an integer multiple of the integration period or aperture.

You can change the integration aperture of some instruments. Integration apertures are commonly defined in units of power line cycles (PLCs). A PLC is the time it takes for one complete AC cycle of the main power supplied to the system. The default integration aperture is one PLC. For 60 Hz power, one PLC is 16.667 ms; for 50 Hz, one PLC is 20 ms. An integration aperture of 0.1 PLC on a 60 Hz system is (0.1)(16.667 ms) = 1.667 ms.

A common mistake is to try to use large aperture integrated measurements to eliminate the effect of random noise. Although noise can be reduced this way, it is typically more productive to use an averaged measurement. The integrated measurement is generally as stable, but averaged measurements usually can be made more quickly than integrated measurements.

This is especially true when making autoranged measurements. When an instrument makes autoranged measurements, all of the measurements it makes when determining the best range are done at the same aperture. If the instrument discards any measurements because they are on a suboptimal range, time is wasted if they are made with a large aperture. With averaged measurements, the instrument typically spends less time finding the optimal range.

Some instruments allow combinations of integration and averaging by allowing the behavior of one or more of the three types of measurements to be altered temporarily. In this case, the instrument is capable of performing an averaged integrated measurement.

Sourcing and limits

Source instruments normally force 0.0 by default. You can change the source value with the `forceX` LPTLib command. All current and voltage sources restrict the complementary function to the one they are sourcing.

Current is the complementary function of voltage, and voltage is the complementary function of current. For example, when a voltage source is forcing voltage, it restricts how much current it allows to flow (including when it is sourcing its default 0.0 value). This is called the limit (also known as the compliance limit). A current limit is used for a voltage source and a voltage limit is used for a current source.

When this limit is reached, the source reduces its force value. In the preceding example, when the voltage source reaches the limit, it reduces the voltage being forced so that the current does not exceed the limit. When this happens, the source is said to be in compliance. For this example, the voltage source is in current compliance.

All sources have default limits, but you can change limit values with the `limitX` LPTLib command.

When there are active sources in a test sequence, you can reset all source levels to the default of 0.0 with the `devclr` LPTLib command. This resets the source level only. If a limit or any other instrument setting has been changed from its default, it is not reset by the `devclr` LPTLib command.

To reset all instrument settings to their initial or default state, use the `devint` LPTLib command. The `devint` command also clears all sources by internally calling the `devclr` command before resetting all instruments to default settings.

Ranging

The default mode of operation for all instruments is to automatically select the best range available for sourcing and for measuring. This is known as autoranging. For sources, the range is picked when the source value is changed. When measuring, the instrument may need to make several measurements, each on a different range, until it finds the best range for the measurement.

Autoranged measurements can take much longer to make than fixed-range measurements because the instrument may need to change ranges and make extra measurements before finding the correct range. There are two features that instruments may support that improve the performance of autoranged measurements. These are smart ranging and sticky ranging.

Smart ranging

An instrument without smart ranging successively upranges or downranges until the correct range is found. For instruments with many ranges, a large change in signal causes the instrument to scan through many ranges before finding the correct one. With smart ranging, the instrument uses the measured value on the incorrect range to try and determine what the correct range will be. If the measurement is really small, the instrument skips ranges and tries to downrange directly to the correct range.

When upranging, the instrument upranges once. If it is still on the wrong range it goes straight to its highest allowable range (source limits affect the highest allowable range). If this range is too large, it uses the smart method of downranging to the correct range. Note that as instrument technologies evolve, new instruments may actually use variations of this technique.

Sticky ranging

The other special ranging feature an instrument may have is sticky ranging. Often an instrument is required to make many measurements on the same range. If the instrument starts on the default range each time (for example, the highest allowable range) and goes through its autorange algorithm for each requested measurement, the instrument must make extra measurements to get to the correct range each time.

Sticky ranging causes the instrument to stay on the last range it was on. If the next measurement it must make is on this range, the instrument only makes one measurement. This feature is most useful during sweeps where most of the measurements are on the same range. Sticky ranging also coordinates with fixed ranging. When an instrument is put on a fixed range and then switched back to autorange, the instrument starts autoranging on that range.

Settling time

Another issue associated with range changes is settling time. The lowest ranges of an instrument can have significantly larger settling times than the higher ranges. Both sticky ranging and smart ranging help with this problem.

You may want autoranged measurements but do not care about resolution once the signal falls below a certain value. In this case, having an autoranged measurement go to the lowest range an instrument has wastes time.

You can use the `lorangeX` LPTLib command to specify the lowest range an instrument uses when autoranging. The instrument then gives you the resolution you need without wasting time trying to make a measurement on a more accurate range.

Fixed ranging

Often, autoranged measurements are not required. If the range on which a measurement will be made is known before the measurement is made, fixed ranging can be used. You can select a fixed range on an instrument with the `rangeX` LPTLib command. Fixed-range measurements are made more quickly than autoranged measurements.

When making fixed range measurements, you must be careful that the range is not set too low. If the range is set too low, the signal may be too large to measure on the specified range. If the signal is larger than the full scale of the range, the instrument goes into an overrange condition. When this happens, the instrument returns a special value to indicate the error instead of the actual measurement.

Range limits

When an instrument is on a range lower than its compliance limit, it limits at full scale of range. For example, a voltage source that is fixed on the 10 µA current measurement range limits at 10 µA, even though the compliance may be programmed to a larger value. Because a fixed-range measurement will not automatically uprange, it cannot resolve this artificial compliance. This is known as range limit. This can affect measurements made on another instrument because the source is not forcing the programmed value. The system automatically resolves problems like this, but only for instruments that are on autorange.

Matrix operations

Most instruments require their terminals to be connected to a device under test (DUT) before they can be used. This involves the use of matrix commands. A typical test sequence consists of making connections, sourcing, measuring, and then calling the `devint` LPTLib command to restore the entire system, including the matrix, to its default condition.

The command most used to make connections is the `conpin` LPTLib command. Normally, several `conpin` calls are made together at the beginning of a test sequence. These grouped `conpin` calls are called a conpin sequence.

If you need to clear all matrix connections in the middle of a test sequence, you can call the `clrcon` LPTLib command to do this explicitly. If you start a new pin connection sequence in the middle of a test sequence, the `conpin` command automatically performs a `clrcon` command before making any new connections.

To make new connections or remove connections in the middle of a test sequence, use the `addcon` LPTLib and `delcon` commands. These commands do not clear the matrix like the `conpin` command does.

To prevent damage to matrix relays, all switching is done with sources off. The `addcon`, `delcon`, and `clrcon` commands internally call the `devclr` LPTLib command before opening or closing any relays. The `devint` command does not directly call the `devclr` command, as described in the sourcing section above. It calls the `clrcon` command, which then calls the `devclr` command as part of its normal processing.

Sweeping

The Linear Parametric Test Library (LPTLib) can automatically perform multiparameter sweeps. Sweeps are more efficient than programmatically changing the source value on an instrument and performing a set of individual measurements.

To set up a sweep, use the `smeasX`, `sintgX`, and `savgX` LPTLib commands to populate a measurement scan table. Each call to one of these commands adds an entry to the table. You can then use the `sweepX` LPTLib command to step a source through a range of source values. At each step, a measurement is made for each entry in the measurement scan table.

Before sweeping the source, the source range is automatically set to the range appropriate for the largest source value in the sweep. This prevents a source range change in the middle of the sweep.

NOTE

When ranging, the absolute value of the source value is used to determine range. For example, if a voltage source is swept from –10 V to +0.5 V, the range appropriate for ±10 V (the 10 V range) is used.

After the sweep has completed, the measurement scan table is not cleared. If another sweep is performed, more measurements are made for each entry in the measurement scan table, which may cause unexpected results. If you do not want this to happen, call the `clrscn` command between sweeps to rebuild the measurement scan table before another sweep is performed.

Calling the `smeasX`, `sintgX`, or `savgX` command does not clear the measurement scan table. The new entries are added to the existing measurement scan table. The measurement scan table is automatically cleared when the `devint` LPTLib command is called, but it can be explicitly cleared by calling the `clrscn` LPTLib command.

Triggers

Several Linear Parametric Test Library (LPTLib) commands use triggers. Triggers are Boolean conditions, but to fully understand them you must understand how the LPTLib processes them.

Triggers are registered with the system using the `trigXY` LPTLib commands. This creates an entry in a trigger table. During processing of multiple LPTLib commands, the system evaluates the triggers. This is done by examining each entry in the trigger table to see if its condition is true or false. If any of the triggers are true, the trigger evaluation is true. Based on the evaluation of the triggers, the LPTLib command may alter how it continues processing.

The trigger table is cleared automatically when the `devint` LPTLib command is called. Note that previous entries are not cleared by adding a new trigger. The new trigger is appended to the existing trigger table and it is used with the rest of the trigger table entries the next time triggers are evaluated. This is a common mistake among even experienced LPTLib users. The trigger table can be cleared explicitly by calling the `clrtrg` LPTLib command.

Most triggers are set up to monitor when a measurement on a specified instrument goes above or below a specified value. For example, calling `trigvg (SMU1, 2.0)` registers a trigger that is true when the voltage measured by SMU1 goes above 2 V.

When evaluating this type of trigger, the system uses an existing measurement if possible. It requests a fresh reading from the triggering instrument if the existing measurement is of a different type (for example, `measi` instead of `intgi`). It then compares it to the threshold value. This means that when evaluating triggers, the system may make a separate measurement for each entry in the trigger table.

For Test Script Processor (TSP®) instruments (2636, 2461-SYS, 2657A, and DMM7510), triggering has been simplified to use the sweep measurement function when possible. For routines that do not have sweep measurements defined, the measurement is made using the `measX` command of the triggering instrument.

For non-TSP instruments (in older systems with a 2410), you can use the `setmode` command to force the LPTLib to use the `intgX` or `avgX` command of the triggering instrument instead. To do this, all triggering instruments must support the `intgX` or `avgX` command.

This type of trigger uses real numbers for the comparison. You can use the `setmode` command to make all triggers evaluate based on the absolute value of the measurement instead of the polarized measurement. You can clear this `setmode` option and the other triggering options mentioned above using the `clrtrg` and `devint` commands.

GPIB

The kibXXX commands allow you to control general purpose interface bus (GPIB) instruments that do not have Linear Parametric Test Library (LPTLib) drivers. Because the instruments controlled this way do not have drivers that automatically interact with matrix control, you must clear all sources before performing any matrix operations.

You must also ensure that the instruments are returned to their default state at the end of a test sequence. You can use the `kibdefclr` and `kibdefint` LPTLib commands to do this. These commands allow you to define strings that are sent to GPIB instruments any time the system is executing a `devclr` or `devint` LPTLib command.

CAUTION

Failure to clear active GPIB-based sources before a matrix operation is performed could result in damage to the matrix.

NOTE

Because of the slow speed of GPIB communication and the order in which instruments are cleared, do not use GPIB instruments when performing `bsweepX` LPTLib tests.

NOTE

Multiple GPIB interfaces are not currently supported. However, future systems may support multiple interfaces.

Instruments and instrument drivers

The Linear Parametric Test Library (LPTLib) provides the lowest level of instrument control available, though it does not control the hardware directly. Instead, LPTLib relies on hardware drivers to control instruments.

Each of these drivers only supports LPTLib commands appropriate for their function. For example, the driver for a voltmeter would not support the function to measure frequency. The following tables provide information about which LPTLib commands are supported by the driver for each type instrument.

Commands supported by all drivers

The following commands are supported by all drivers:

▪ `delay` ▪ `devint`		

Commands supported for SMUs

The following commands are supported by source-measure unit (SMU) drivers:

- adelay
- asweep*X*
- avgi, avgv (avg*X*)
- bmeas*X*
- bsweep*X*
- clrscn
- clrtrg
- devclr
- forcei, forcev (force*X*)
- getstatus
- insbind
- intgi, intgv (intg*X*)
- limit*X*
- lorange*X*
- measi, measv (meas*X*)
- pulse*X*
- rangei, rangev (range*X*)
- refctrl
- rtfary
- rttrigary (26xx only)
- savgi, savgv (savg*X*)
- search*X*
- setauto
- setimtr (26xx only)
- setmode
- setvmtr (26xx only)
- sintgi, sintgv (sintg*X*)
- smeasi, smeasv (smeas*X*)
- sweepi, sweepv (sweep*X*)
- trigig, trigtg, trigvg (trig*X*g)
- trigil, trigtl, trigvl (trig*X*l)

Commands supported for CVUs

The following commands are supported by capacitance-voltage unit (CVU) drivers:

- avgc, avgcg, avgg (avg*X*)
- clrscn
- devclr
- forcev
- getstatus
- insbind
- intgc, intgcg, intgg (intg*X*)
- measc, meascg, measg (meas*X*)
- rangec
- rangei
- rtfary
- setauto
- setmode
- smeasc
- smeasg
- sweepv

Commands supported for the RSA306B spectrum analyzer

The following commands are supported for the RSA306B USB Spectrum Analyzer.

NOTE

In Keithley systems, the RSA306B functions as a replacement for discontinued scope cards. Spectrum analyzer capabilities may be added in the future.

- rsa_close
- rsa_detect_peaks
- rsa_init
- rsa_measure
- rsa_measure_next
- rsa_selftest
- rsa_setup

Commands supported for PGUs

The following commands are supported by pulse-generator unit (PGU) drivers:

NOTE

These commands are deprecated, but are listed here for users with older systems.

■ pgu_current_limit ■ pgu_delay ■ pgu_fall ■ pgu_halt ■ pgu_height ■ pgu_init	■ pgu_load ■ pgu_mode ■ pgu_offset ■ pgu_period ■ pgu_rise ■ pgu_range	■ pgu_select ■ pgu_trig ■ pgu_trig_burst ■ pgu_trig_unit ■ pgu_width

Commands supported for scope cards

The following commands are supported by scope card (SCP) drivers (systems with 4200-SCP2HR scope cards):

■ scp_init ■ scp_setup ■ scp_measure	■ scp_measure_next ■ scp_detect_peaks ■ scp_selftest	■ scp_close

Commands supported for systems

The following commands are supported by system drivers:

■ setmode	■ tstsel	■ devint

Commands supported for DMMs

The following commands are supported by digital multimeter (DMM) drivers:

■ avgi, avgv (avg*X*) ■ clrtrg ■ devclr ■ getstatus ■ intgi, intgv (intg*X*) ■ lorange*X* ■ measi, measv (meas*X*)	■ rangei, rangev (range*X*) ■ refctrl ■ savgv (savg*X*) ■ setauto ■ setmode ■ sintgv (sintgX)	■ smeast, smeasi, smeasv (smeasX) ■ trigtg, trigvg (trig*X*g) ■ trigtl, trigvl (trig*X*l) ■ tstsel

Commands supported for switch mainframes

The following commands are supported by switch mainframe drivers:

▪ `addcon` ▪ `clrcon` ▪ `conpin`	▪ `conpth` ▪ `delcon` ▪ `setmode`	▪ `tstsel`

Commands that support timer functions

The following commands support timer functions:

▪ `disable`	▪ `enable`	▪ `imeast`

Commands for USB instruments not supported by systems drivers

The following commands control USB instruments that are not supported by S530, S535, or S540 drivers:

▪ `kibdefclr` ▪ `kibdefint`	▪ `kibsnd` ▪ `kibrcv`	▪ `kibspl` ▪ `kibsplw`

Instrument and terminal IDs

The Linear Parametric Test Library (LPTLib) uses instrument identification codes to refer to the instruments in the system. An instrument identification code is an integer value. This manual never refers to the actual numbers used to identify the instruments. Instead, it refers to the mnemonic codes that you can use in your programs to refer to the various instruments.

An instrument ID typically consists of a mnemonic string that identifies the type of instrument and a number that specifically identifies an individual instrument of this type. For example, `SMU2` is an instrument ID that refers to the second source-measure unit (SMU) in the system.

This manual often refers to a SMU instrument as `SMU`*`n`* (the *`n`* represents a number), however, it does not matter which SMU is being used. Anywhere an instrument ID is required by a library command, a specific instrument ID (mnemonic) can be used directly or as an integer variable that was assigned the value of an instrument ID. For example, `SMU`*`n`* can indicate `SMU1`, `SMU2`, `SMU3`, and so on.

Most instruments have terminals that must be connected to your circuit before the instrument can be used in a test sequence. For example, a simple voltmeter has two terminals: A high terminal and a low terminal. The terminals of an instrument also have identification codes similar to the instrument IDs; in some places an instrument ID and terminal ID can be used interchangeably.

In some cases, specific terminal IDs are associated with the instrument. Where appropriate, they are listed in the `instr_id` parameter description. There are also some special instrument IDs and terminal IDs that the system recognizes but are not associated with any specific instrument.

Special instrument IDs:

- **KI_SYSTEM:** This refers to the system itself. Note that `KI_SYSTEM` is a special pseudo-instrument and the instrument ID does not refer to the collection of instruments in the system but to the pseudo-instrument itself.

Special terminal IDs:

- **CHUCK:** The chuck connection. The sense pin option cannot be used with `CHUCK`. Use `CHUCKM` instead.
- **KI_EOC:** This is a special terminal ID used to terminate a list of terminals in a connection subroutine. This value is `0`; you can use the value `0` instead of `KI_EOC`.

Optimizing test sequences

There are several things you can do to optimize your system for faster operation.

Fixed range versus autorange measurements

Use fixed-range measurements whenever possible. Depending on the measurement, this can significantly increase the speed of the test sequence.

Fix-range trigger instruments

When you are using triggers, you should fix-range the triggering instrument with the value of the trigger. If you do not do this, the trigger command forces a measurement to be made after each sweep point, and the measuring instrument autoranges. Autoranging noticeably increases the total test execution time.

Use combination commands

Do not use individual `forceX` and `measX` LPTLib commands to sweep a set of points. Use the `sweepX` LPTLib command, which incurs significantly less overhead.

Error handling

Error handling is done on two levels. The first is Linear Parametric Test Library (LPTLib) command return values. The other is error message processing performed by the Keithley Test Environment (KTE) tools. For more information about how the KTE tools process error messages, see the documentation for the tool you are using.

Calling the getlpterr function

You can also use the `getlpterr` LPTLib function to get error values. It returns the first error encountered since the last `devint` LPTLib command.

Error messages

There are two parts to an error message generated by Keithley test systems. These are the error header and the error text. An example error message looks like the following:

```
2017/03/01 12:00 - E0101
Argument #2 is not a pin in the current configuration.
```

In this example `2017/03/01 12:00 - E0101` is the error header and `Argument #2 is not a pin in the current configuration.` is the text. The error header provides information about the error and the error text explains the cause of the error.

The first part of the header is the date and time the error occurred. Next is the letter `E` followed by the error number. In the example, the error number is `101`.

Special error values returned

The Linear Parametric Test Library (LPTLib) commands may return error values in place of actual measurement values. These values are summarized in Error definitions. For example, the following Keithley Interactive Test Tool (KITT) macro will generate a matrix error before trying to make a measurement.

```
conpin(SMU1, 999, 0);
measv(SMU1,V);
```

Rather than a true voltage reading, the actual value returned in the `measv` variable is `1E+23`. This indicates that no measurement was made due to an error. In this case, it was the matrix connection error.

Result values indicating an error

The following table contains errors that are returned as measured results (not as Linear Parametric Test Library (LPTLib) return status codes). Each value has a specific meaning. If one of these values is returned from any measurement subroutine, the test was not performed and action is required by the Keithley programmer or manager.

Table: Errors

Value	Description
1.0E22	Instrument Overrange. Measurement instrument performed the measurement, but the resulting value was inappropriate for the range specified. This is either from selecting the wrong range for manual ranging or from autoranged measurement on a dynamically changing signal (slowly charging device).
1.0E23	Measurement Not Performed. If an error causes a measurement not to be performed, this result is returned. Previously, if a measurement was not taken, nothing was returned, so the result available to the user (from the execution of the test sequence) was the previous reading. This could be confusing. Now, whenever possible, it will return this error value as a default value.
1.138E26	Site Inactive. Measurement not performed because site was inactive.
2.0E22	Current Overload. Measurement was not performed due to a shutdown condition. The source went into a current overload condition and shutdown.
3.0E22	Oscillation Detection. Instrument was forced into a shutdown state because oscillations were detected at the instrument.
4.0E22	Thermal Shutdown. Instrument is dissipating too much heat. The instrument is shutdown to keep from self-destructing.
5.0E22	SOA Exceeded. The instrument is designed to operate within the Safe Operating Area. If the programming conditions exceed this area, this error value is returned.
6.0E22	Pulse Width Too Short. The specified pulse width is less than the minimum specified for the instrument.
7.0E22	Source Limit. Measurement was not performed because source was in limit.

Error messages

This section lists all of the error messages. The error number, error description, and additional remarks are given for each error. Note that Linear Parametric Test Library (LPTLib) commands return negative values, but the messages indicate them as positive values.

3 LPT_NOCOMCHAN

Message: **This is a host-side only error. No message will be generated by the tester.

Remarks: Linear Parametric Test Library (LPTLib) calls may only be sent after a successful call to the `tstsel` command. Make sure the `tstsel` command is called before any other LPTLib command.

5 SYS_MEM_ALLOC_ERR

Message: Memory allocation failure.

Remarks: The tester does not have enough memory for the Keithley system software to run correctly.

20 LPT_PREVERR

Message: Command not executed because a previous error was encountered.

Remarks: There was an error encountered during a test sequence. All Linear Parametric Test Library (LPTLib) commands after that point will generate this error. Correct the problem in the test sequence causing the first error.

21 LPT_FATAL

Message: Tester is in a fatal error state.

Remarks: The tester is in a state requiring user attention. Correct the problem requiring attention.

22 LPT_FATALINTEST

Message: Fatal condition detected while in testing state.

Remarks: The tester entered a fatal state while a test sequence was in progress. Results generated during this test sequence may not be valid.

24 LPT_TOMANYARGS

Message: Too many arguments.

Remarks: A Linear Parametric Test Library (LPTLib) command has passed more arguments than it can handle. Reduce the number of arguments by trying to split the call into two or more smaller calls.

100 MX_INVLDCNT

Message: Invalid connection count, number of connections passed was NNN.

Remarks: The matrix driver could not determine what to connect because there were not enough terminal IDs or pins specified. This is usually caused by passing `-1` for all but one argument to a matrix function.

101 MX_NOPIN

Message: Argument #NNN is not a pin in the current configuration.

Remarks: A request was made to make a connection to a pin that does not exist.

102 MX_MULTICON

Message: Multiple connections on XXX.

Remarks: Certain terminals can only be connected to one pin at a time. An attempt was made to connect this terminal to more than one pin at a time.

109 MX_ILLGLTSN

Message: Illegal test station: NNN.

Remarks: An internal system software error has occurred.

113 MX_NOSWITCH

Message: There are no switching instruments in the system configuration.

Remarks: The system did not detect any matrix hardware during system configuration.

114 MX_ILLGLCON

Message: Illegal connection.

Remarks: An attempt was made to make a connection that physically cannot be made or is disallowed.

122 UT_INVLDPRM

Message: Illegal value for parameter #NNN.

Remarks: An invalid value was passed as the NNNth argument to a Linear Parametric Test Library (LPTLib) command.

126 UT_NOURAM

Message: Insufficient user RAM for dynamic allocation.

Remarks: The system could not allocate memory for user data. This could be caused by sweeps with an unusually large number of steps or by large sweeps that measure too many parameters.

129 UT_TMRIVLD

Message: Timer not ENABLED.

Remarks: Time measurements can only be made on a timer when the timer is enabled.

137 UT_INVLDVAL

Message: Invalid value for modifier.

Remarks: An invalid option was used for the `setmode` or `getstatus` command.

152 CB_BADFUNC

Message: Function not supported by XXX (NNN).

Remarks: The driver for this instrument does not support the Linear Parametric Test Library (LPTLib) command used.

156 CB_NOFILE

Message: Configuration file does not exist: XXX.

Remarks: A required configuration file is missing.

157 CB_FORMAT

Message: Configuration file format error. File: XXX, Section: XXX, Key: XXX.

Remarks: A system configuration file has a missing entry or an entry that was formatted in an unexpected way.

162 CB_INVLDERROR

Message: Invalid error number: NNN.

Remarks: The logging `.ini` file has a format error.

163 CB_INVLDEVENT

Message: Invalid event number: NNN.

Remarks: The `logging.ini` file has a format error.

166 CB_INSNOTREC

Message: Instrument with model code XXX is not recognized.

Remarks: An instrument is not recognized by the system. The instruments may be misread over the 170 CB_INITFAIL.

173 CB_MULTITIMER

Message: System supports only four timer(TIMER1, ..., TIMER4).

Remarks: The LPT function was expecting `TIMER1`, `TIMER2`, `TIMER3`, or `TIMER4` for the *`instr_id`* parameter, but some other instrument ID was sent (for example, `TIMER5` or `SMU1`).

194 MX_INVLDTRM

Message: Invalid terminal: XXX.

Remarks: The terminal specified is invalid for the command.

233 FM_NOCON

Message: Cannot force when not connected.

Remarks: The instrument must be connected to a device under test (DUT) before it can be used.

455 ECP_PROTOVER

Message: Protocol version mismatch.

Remarks: The application controller software version and the tester software version do not match. Either the system software was not installed correctly, or the application controller is being used to control an older or newer tester not intended to be used with this application controller.

601 SYS_INTERNAL_ERR

Message: System software internal error.

Remarks: An internal system software error has occurred.

610 SYS_SPAWN_ERR

Message: Could not start XXX.

Remarks: An internal system software error has occurred.

611 SYS_NETWORK_ERR

Message: Network error.

Remarks: The system is having problems communicating over the network. Make sure that all network connections are secure, all network cables are intact, and any network routers are functioning properly.

612 SYS_PROTOCOL_ERR

Message: Protocol error.

Remarks: An internal system software error has occurred.

650 TAPI_BADCHANNEL

Message: Request to open unknown channel type XXX.

Remarks: An internal system software error has occurred.

651 TAPI_BADTESTER

Message: **This is a host-side only error. No message will be generated by the tester.

Remarks: There is no network node corresponding to the tester address. Make sure the network configuration is correct.

652 TAPI_NOTFOUND

Message: **This is a host-side only error. No message will be generated by the tester.

Remarks: The tester cannot be located on the network. Make sure the tester is powered on and has started correctly. This problem can also be caused by network problems.

653 TAPI_REFUSED

Message: **This is a host-side only error. No message will be generated by the tester.

Remarks: The tester is in use. This can be caused by another process running diagnostics or a test plan.

656 TAPI_CHANLIMIT

Message: Channel limit exceeded.

Remarks: There are too many active network connections to the tester. Close some of the tools that communicate with the tester and try again.

657 TAPI_BUFOFLOW

Message: **This is a host-side only error. No message will be generated by the tester.

Remarks: An internal system software error has occurred.

LPTLib command descriptions

Detailed descriptions of the LPTLib commands are in the following topics.

addcon

This command adds connections without clearing existing connections.

Models supported

S530, S535, S540

Usage

```
int addcon(int exist_connect, int connect1, [connectn, [...]] 0);
```

exist_connect	A pin number or instrument terminal ID; this instrument or terminal may have been, but is not required to have been, previously connected with the `addcon`, `conpin`, or `conpth` command
connect1	A pin number or an instrument terminal ID
connectn	A pin number or an instrument terminal ID

Details

The test system has an I-V matrix. The `addcon` command can be used to make additional connections on this matrix. I-V connections are direct connections.

When used with the I-V matrix, the `addcon` command connects every item in the argument list and there is no real distinction between the *exist_connect* parameter and the rest of the connection list. When using the `addcon` command with the I-V matrix, the command functions like the `conpin` command, except previous connections are never cleared.

The value `-1` is ignored by the `addcon` command and is considered valid as both a connection list and the *exist_connect* parameter. There must be at least two parameters with a value other than `-1`.

Matrix errors are generated if a dangerous connection is detected, such as connecting a source-measure unit (SMU) high terminal directly to ground.

Before making the new connections, the `addcon` command clears all active sources by calling the `devclr` command.

You can use this command in dual-site mode (S535 systems only).

NOTE

If you have a chuck connected in an S535 system that is in dual-site mode, the pin the chuck is on cannot be mirrored.

Also see

clrcon (on page 2-33)
conpin (on page 2-37)
conpth (on page 2-39)
delcon (on page 2-41)

adelay

This command specifies an array of delay points to use with `asweepX` command calls.

Models supported

S530, S535, S540

Usage

```
int adelay(unsigned int delaypoints, double *delayarray);
```

delaypoints	The number of separate delay points defined in the array
delayarray	The name of the array defining the delay points; this is a single-dimension floating-point array that is *delaypoints* long and contains the individual delay times; units of the delays are seconds

Details

The delay is specified in units of seconds, with a resolution of 1 ms. The minimum delay is 0 s.

Each delay in the array is added to the delay specified in the `asweepX` command. For example, if the array contains four delays (0.04 s, 0.05 s, 0.06 s, and 0.07 s) and the delay specified in the `asweepX` command is 0.1 s, then the resulting delays are 0.14 s, 0.15 s, 0.16 s, and 0.17 s.

You can use this command in dual-site mode (S535 systems only).

Also see

asweepX (on page 2-26)

asweepX

This command generates a waveform based on a user-defined forcing array (logarithmic sweep or other custom forcing commands).

Models supported

S530, S535, S540

Usage

```
int asweepi(int instr_id, unsigned int num_points, double delay_time, double
    *force_array);
int asweepv(int instr_id, unsigned int num_points, double delay_time, double
    *force_array);
```

instr_id	The instrument identification code of the sourcing instrument
num_points	The number of separate current and voltage force points defined in the array
delay_time	The delay, in seconds, between each step and the measurements defined by the active measure list
force_array	The name of the user-defined force array; this is a single dimension array that contains all force points

Details

The asweep*X* command is used with the smeas*X*, sintg*X*, or savg*X* commands.

The trig*X*l or trig*X*g command can also be used with the asweep*X* command. However, once a trigger point is reached, the sourcing device stops moving through the array. The output is held at the last forced point for the duration of the asweep*X* command. Data resulting from each step is stored in an array, as noted above, with smeas*X*. After the trigger point is reached, measurements are made at each subsequent point. Results are approximately equal because the source is held at a constant output.

The asweepv and asweepi commands are sourcing-type commands. When called, an automatic limit is imposed on the sourcing device. Refer to the limitX (on page 2-57) command for additional information.

The maximum number of times data is measured (using the smeas*X*, sintg*X*, or savg*X* command) is determined by the num_points argument in the asweep*X* command. A one-dimensional result array with the same number of data elements as the selected value of the *num_points* parameter must be defined in the test program.

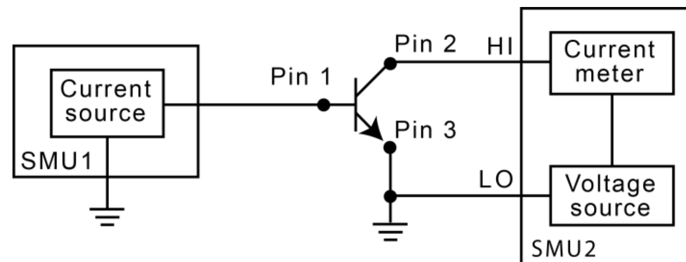
The clrscn command is used to eliminate previous buffers for the second sweep. Using the smeas*X*, sintg*X*, and savg*X* commands after calling the clrscn command causes the appropriate new measures to be defined and used.

This command returns dual-site data for SITE0 and SITE1 if available (S535 systems only).

Example

```
double icmeas[10], ifrc[10];
.
.
ifrc[0]=1.0e-10;
for (i=1; i<10; i++) /* Create decade array from */
   /* 1.0E-10 to 1.0E-1. */
    ifrc[i]=10.0*ifrc[i-1];
.
.
conpin(SMU1, 1, 0); /* Base connection. */
conpin(SMU2, 2, 0); /* Collector connection. */
conpin(GND, 3, 0);
limiti(SMU2, 200.0E-3); /* Reset I limit to maximum. */
smeasi(SMU2, icmeas); /* Define collector current */
 /* array. */
forcev(SMU2, 5.0); /* Force vce bias. */
asweepi(SMU1, 10, 10.0E-3, ifrc); /* SweepIB, 10 points, 10 ms */
 /* apart. */
```

This example gathers data to construct a graph showing the gain of a bipolar device over a wide range of base currents. A fixed collector-emitter bias is generated by SMU2. A logarithmic base current from 1.0E-10 to 1.0E-1A is generated by SMU1 using the `asweepi` command. The collector current applied by SMU2 is measured 10 times by the `smeasi` command. The data gathered is then stored in the *`icmeas`* array.



Also see

limitX (on page 2-57)
savgX (on page 2-89)
sintgX (on page 2-109)
smeasX (on page 2-114)
trigXg, trigXl (on page 2-121)

avgX

This command makes a series of measurements and averages the results.

Models supported

S530, S535, S540

Usage

```
int avgc(int instr_id, double *result, unsigned int stepno, double steptime);
int avgcg(int instr_id, double *c, double *g, unsigned int stepno, double steptime);
int avgg(int instr_id, double *result, unsigned int stepno, double steptime);
int avgi(int instr_id, double *result, unsigned int stepno, double steptime);
int avgv(int instr_id, double *result, unsigned int stepno, double steptime);
```

instr_id	The instrument identification code of the measuring instrument; `SMUn`, `CMTRn`, `VMTRn`
result	The variable assigned to the result of the measurement
stepno	The number of steps averaged in the measurement (1 to 32,767)
steptime	The interval in seconds between each measurement; the minimum practical time is approximately 2.5 ms
c	The variable assigned to the capacitance measurement
g	The variable assigned to the conductance measurement

Details

The `avgX` command is used primarily to get measurements when:

- The device under test (DUT) being tested acts in an unstable manner.
- Electrical interference is higher than can be tolerated if the `measX` command is used.

The programmer specifies the number of samples and the duration between each sample.

After this command executes, all closed relay matrix connections remain closed and the sources continue to generate voltage or current. This allows additional sequential measurements.

In general, measurement commands that return multiple results are more efficient than performing multiple measurement commands. For example, calling a single `avgcg` command is faster than calling the `avgc` command followed by the `avgg` command.

The `rangeX` command directly affects the operation of the `avgX` command. The use of the `rangeX` command prevents the addressed instrument from automatically changing ranges. This can result in an overrange condition similar to what would occur when measuring 10 V on a 2 V range. An overrange condition returns the value 1.0e+22 as the result of the measurement.

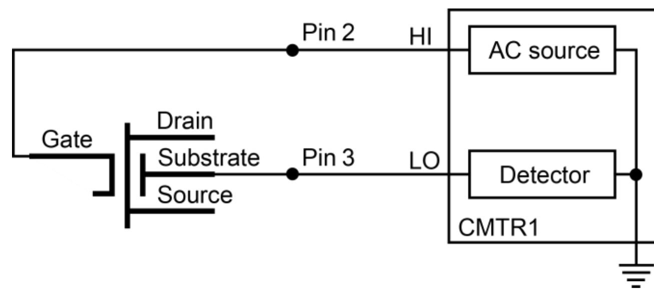
If the `rangeX` command is not in the test sequence before the `avgX` call, the measurements performed automatically select the optimum range.

The `avgi` and `avgv` commands return dual-site data for both `SITE0` and `SITE1` if available (S535 systems only).

Example

```
double ciss;
.
.
conpin(CMTR1L, 3, 0);
conpin(CMTR1H, 2, 0);
rangec(CMTR1, 2.0E-12); /* Select range for 2.0 pF. */
avgc(CMTR1, &ciss, 10, 2.0E-3); /* Measure capacitance ten */
 /* times with 2 ms between each;*/
 /* return average of results to*/
 /* ciss. */
```

This example shows a test sequence used to measure the capacitance between a MOSFET gate and substrate. The capacitance returned is the average of the result of ten measurements, each separated by 2 ms.



Also see

measX (on page 2-61)
rangeX (on page 2-77)

bmeasX

This command makes a series of readings as quickly as possible. This measurement mode allows for waveform capture and analysis (within the resolution of the measurement instrument).

Models supported

S530, S535, S540

Usage

```
int bmeasi(int instr_id, double *result, unsigned int numrdg, double delay, int timerid,
    double *timerdata);
int bmeasv(int instr_id, double *result, unsigned int numrdg, double delay, int timerid,
    double *timerdata);
```

instr_id	The instrument identification code of the measuring instrument; `SMUn`, `VMTRn`
result	The result name of the array to receive readings; the array must be large enough to hold the readings
numrdg	The number of readings to return in the array
delay	The delay between points to wait (in seconds)
timerid	The device name of the timer to use (`0` = no timer data)
timerdata	The array used to receive the time points at which the readings were made; if *timerID* = `0`, the timer is not read and this array is not updated; if used, the array must be large enough to hold the readings

Details

This command collects data using the presently selected range. The measurement range is typically the same as the force range. If you need a different range, you must change the measurement range before calling the `bmeasX` command.

When used with the time module, the measurements and the times for each measurement are stored. The specific timer is defined in the command, and the time array is returned with the *results* array.

Each parametric test system has a single timer. For compatibility with older systems, you can use `TIMER1` through `TIMER4` for the *instr_id* parameter, but any use of `TIMER2`, `TIMER3`, or `TIMER4` refers internally to `TIMER1`.

Example 1

```
double irange, volts, rdng[5], timer[5];
:
.
.
enable(TIMER1); /* Enable the timer module. */
.
.
conpin(GND, 11, 0); /* Make connections. */
conpin(SMU3, 14, 0);
.
.
forcev(SMU3, volts); /* Perform the test. */
measi(SMU3, &irange); /* Set the I range of the SMU based */
rangei(SMU3, irange); /* on the initial measurement. */
.
forcev(SMU3, volts);
bmeasi(SMU3, rdng, 5, 0.0001, TIMER1, timer); /* gather a block of    measurements
   */
 /* I measurement of 5 */
 /* readings using SMU3 with */
 /* 100 us delay between */
 /* readings, using TIMER1 with */
 /* time data labeled timer. */
```

This example shows how the `bmeasX` command is used with a timer. Each measurement is associated with a timestamp. This timestamp marks the interval when each reading is made. This information is useful when determining how much time was required to obtain a specific reading.

Example 2

```
double volts, rdng[5];
:
.
conpin(GND, 11, 0); /* Make connections. */
conpin(SMU3, 14, 0);
.
forcev(SMU3, volts); /* Perform the test. */
.
bmeasi(SMU3, rdng, 5, 0, 0, 0); /* Block current measurement */
 /* of 5 readings using SMU3. */
```

This example shows how the `bmeasX` command is used without a timer. When used without a timer, the returned measurement is not associated with a timestamp.

Also see

None

bsweepX

This command supplies a series of ascending or descending voltages or currents and shuts down the source when a trigger condition is encountered.

Models supported

S530, S535, S540

Usage

```
int bsweepi(int instr_id, double startval, double endval, unsigned int num_points,
   double delay_time, double *result);
int bsweepv(int instr_id, double startval, double endval, unsigned int num_points,
   double delay_time, double *result);
```

instr_id	The instrument identification code of the sourcing instrument; `SMUn`
startval	The initial voltage or current level applied as the first step in the sweep; this value can be positive or negative
endval	The final voltage or current level applied as the last step in the sweep; this value can be positive or negative
num_points	The number of separate current and voltage force points between the *startval* and *endval* parameters (`1` to `8,000`)
delay_time	The delay in seconds between each step and the measurements defined by the active measure list
result	Assigned to the result of the trigger; this value represents the source value applied at the time of the trigger or breakdown

Details

The `bsweepX` command is used with the `trigXg` or `trigXl` command. These trigger commands provide the termination point for the sweep. At the time of trigger or breakdown, all sources are shut down to prevent damage to the device under test. Typically, this termination point is the test current required for a given breakdown voltage.

Once triggered, the `bsweepX` command terminates the sweep and clears all sources by executing a `devclr` command internally. The standard `sweepX` command continues to force the last value. This is useful for device characterization curves but can cause problems when used in device breakdown conditions.

The `bsweepX` command can also be used with the `smeasX`, `sintgX`, `savgX`, or `rtfary` command. Measurements are stored in a one-dimensional array in the order in which they were made.

The system maintains a measurement scan table consisting of devices to test. This table is maintained using calls to the `smeasX`, `sintgX`, `savgX`, or `clrscn` command. As multiple calls to `sweepX` commands are made, these commands are appended to the measurement scan table. Measurements are made after the time programmed by the *delay_time* parameter has elapsed at the beginning of each `bsweepX` command step.

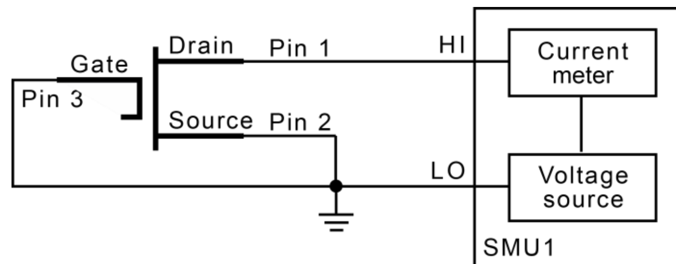
New measurements are defined and used by calling the `smeasX`, `sintgX`, or `savgX` command after a `clrscn` command.

This command returns dual-site data for SITE0 and SITE1 if available (S535 systems only).

Example

```
double bvdss;
.
.
conpin(SMU1, 1, 0);
conpin(GND, 2, 3, 0);
limiti(SMU1, 100e-6); /* Define the I limit for the device. */
rangei(SMU1, 100e-6); /* Select a fixed range */
 /* measurement. */
trigil(SMU1, -10e-6); /* Set the trigger point to -10 uA. */
bsweepv(SMU1, 10.0, 50.0, 40, 10.0e-3, &bvdss); /* Sweep */
 /* from 10 V to 50 V in 40 */
 /* steps with 10 ms settling */
 /* time per step. */
```

This example measures the drain to source breakdown voltage of a field-effect transistor (FET). A linear voltage sweep is generated from 10.0 V to 50.0 V by SMU1 using the `bsweepv` command. The breakdown current is set to 10 mA by using the `trigil` command. The voltage at which this current is exceeded is stored in the variable `bvdss`.



Also see

clrscn (on page 2-34)
devclr (on page 2-42)
rtfary (on page 2-88)
savgX (on page 2-89)
sintgX (on page 2-109)
smeasX (on page 2-114)
sweepX (on page 2-118)
trigXg, trigXl (on page 2-121)

clrcon

This command opens or de-energizes all device under test (DUT) pins and instrument matrix relays, disconnecting all crosspoint connections.

Models supported

S530, S535, S540

Usage

```
int clrcon(void);
```

Details

The `clrcon` command is called automatically by the `devint` command. The first in a series of one or more connection-type commands automatically calls a `clrcon` command. Because this command is automatically called, it is not normally used by a programmer.

If any sources are actively generating current or voltage, the `devclr` command is automatically called before the relay matrix is de-energized.

You can use this command in dual-site mode (S535 systems only).

Also see

devclr (on page 2-42)
devint (on page 2-42)

clrscn

This command clears the measurement scan tables associated with a sweep.

Models supported

S530, S535, S540

Usage

```
int clrscn(void);
```

Details

The `clrscn` command is only required when multiple sweeps and multiple sweep measurements are used in a single test sequence.

You can use this command in dual-site mode (S535 systems only).

Example

```
double res1[14], res2[14];
.
conpin(SMU1, 1, 0);
conpin(SMU2, 2, 0);
conpin(GND, 3, 0);
forcev(SMU1, 4.0); /* Apply 4 V to gate. */
smeasi(SMU2, res1); /* Measure drain current in */
 /* each step; store results */
 /* in res1 array. */
sweepv(SMU2, 0.0, 14.0, 13, 2.0E-2); /* Make */
 /* 14 measurements */
 /* over a range of 0 V to 14 V. */
clrscn(); /* Clear smeasi. */
forcev(SMU1, 5.0); /* Apply 5 V to gate. */
smeasi(SMU2, res2); /* Measure drain current in */
 /* each step; store results in */
 /* res2 array. */
sweepv(SMU2, 0.0, 14.0, 13, 2.0E-2); /* Perform */
  /*14 measurements */
 /* over a range 0 V through 14 V. */
```

In this example, the `sweepX` command configures SMU2 to source a voltage that sweeps from 0 V through +14 V in 14 steps. The results of the first `sweepv` command are stored in an array called `res1`. Because of the `clrscn` command, the data and pointers associated with the first `sweepv` command are cleared. Then 5 V is forced to the gate, and the measurement process is repeated. Results from these second measurements are stored in an array called `res2`.

This example gets the measurement data needed to create a graph showing the gate voltage-to-drain current characteristics of a field-effect transistor (FET). The program samples the current generated by SMU2 14 times. This is done in two phases: First with 4 V applied to the gate, and then with 5 V applied. The gate voltages are generated by SMU1.

Also see

sweepX (on page 2-118)

clrtrg

This command clears the user-selected voltage or current level that is used to set trigger points. This permits the use of the `trigXl` or `trigXg` command more than once with different levels in a single test sequence.

Models supported

S530, S535, S540

Usage

```
int clrtrg(void);
```

Details

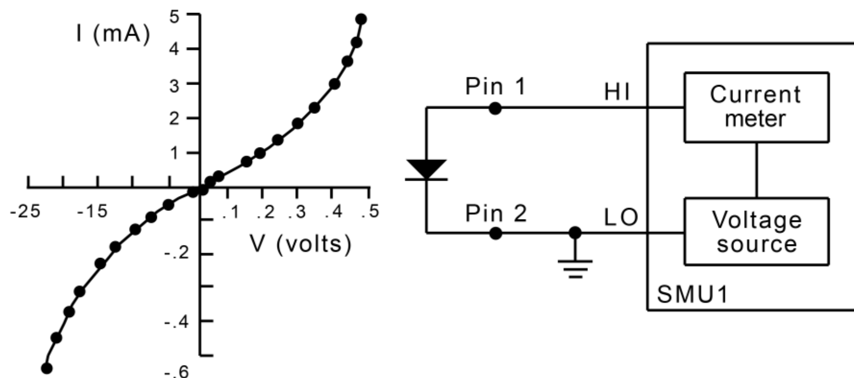
The `searchX`, `sweepX`, `asweepX`, or `bsweepX` command, each with different voltage or current levels, may be used repeatedly within a command if each is separated by a `clrtrg` command.

You can use this command in dual-site mode (S535 systems only).

Example

```
double forcur[11], revcur[11]; /* Defines arrays. */
.
.
conpin(SMU1, 1, 0);
conpin(GND, 2, 0);
trigil(SMU1, 5.0e-3); /* Increase ramp to I = 5 mA.*/
smeasi(SMU1, forcur); /* Measure forward */
 /* characteristics; */
 /* return results to forcur */
 /* array. */
sweepv(SMU1, 0.0, 0.5, 10, 5.0e-3); /* Output */
 /* 0 V to 0.5 V in 11 */
 /* steps, each 5 ms duration. */
clrtrg(); /* Clear 5 mA trigger point. */
clrscn(); /* Clear sweepv. */
trigil(SMU1, -0.5e-3); /* Decrease ramp to */
 /* I = -0.5 mA. */
smeasi(SMU1, revcur); /* Measure reverse */
 /* characteristics; */
 /* return results to revcur */
 /* array. */
sweepv(SMU1, 0.0, -30.0, 10, 5.00e-3); /* Output */
 /* 0 V to -30 V in 11 steps */
 /* each 5 ms in duration. */
```

This example collects data and creates a graph that shows the forward and reverse conduction characteristics of a diode. The `clrtrg` command allows multiple triggers to be programmed twice in the same test sequence. Each result is returned to a separate array.



Also see

asweepX (on page 2-26)
bsweepX (on page 2-32)
searchX (on page 2-97)
sweepX (on page 2-118)
trigXg, trigXl (on page 2-121)

conpin

This command connects pins and instruments.

Models supported

S530, S535, S540

Usage

```
int conpin(int connect1, [connectn, [...]] 0);
```

connect1	A pin number or an instrument terminal ID
connectn	A pin number or an instrument terminal ID

Details

The `conpin` command is used to make connections in Keithley Test Environment (KTE) systems. The S540 3 kV system has two kinds of pins: High-voltage and low-voltage pins. Pins 1 through 12 are high-voltage (3 kV) pins, and the rest of the pins are low-voltage (100 V) pins.

The S540 contains interconnect pathways that allow 200 V source-measure units (SMUs) to connect to the 3 kV pins in a protected circuit. The `conpin` command recognizes this and allows such connections.

When used with an I-V matrix, the `conpin` command connects every item in the argument list together. Because I-V connections do not require pathways to be allocated, a pin or terminal may be used in more than one `conpin` command call. If there are no connection rules violated, the pin or terminal is connected to the additional items and everything to which it is already connected.

NOTE

High-voltage systems only: If you want to use high-voltage SMUs and low-voltage SMUs simultaneously, connect the high-voltage SMUs first using the `conpin` command. This is important because the source-measure units (SMUs) are connected starting with pathway A, and only pathways A and B are high-voltage pathways. If you try to connect a high-voltage SMU using a low-voltage pathway (for example, connecting a 2410 using pathway C in an S530 system), an error is generated.

The first `conpin` or `conpth` command after any other LPT library call clears all sources by calling the `devclr` command, and then clears all matrix connections by calling `clrcon` command before making the new connections.

The value `-1` is ignored by the `conpin` command and is considered a valid entry in the connection list. However, there must be at least two entries in the list with a value other than `-1`.

Matrix errors are generated under the following conditions:

- A dangerous connection is detected, such as connecting a source-measure unit (SMU) high terminal directly to ground.
- The user attempts to connect a high-voltage SMU to a 200 V pin, which is not allowed.

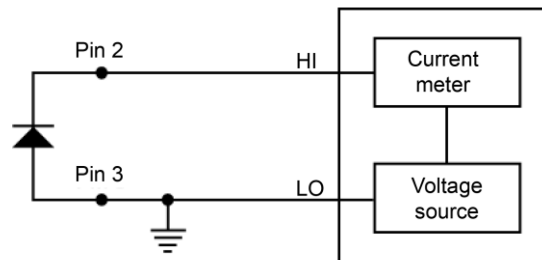
You can use this command in dual-site mode (S535 systems only).

NOTE

If you have a chuck connected in an S535 system that is in dual-site mode, the pin the chuck is on cannot be mirrored.

Example

```
conpin(3, GND, 0); /* Connect pin 3 to GND (through interconnect pathway) */
 /* and ground. */
conpin(2, HVSMU1, 0); /* Connect pin 2 to HVSMU1. */
.
.
```



Also see

addcon (on page 2-24)
clrcon (on page 2-33)
conpth (on page 2-39)
devclr (on page 2-42)

conpth

This command connects pins and instruments using a specific pathway.

Models supported

S530, S535, S540

Usage

```
int conpth(int path, int connect1, int connect2, [connectn, [...]] 0);
```

path	Pathway number to use for the connections
connect1	A pin number or an instrument terminal ID
connect2	A pin number or an instrument terminal ID
connectn	A pin number or an instrument terminal ID

Details

The system has an I-V matrix; I-V connections are usually direct connections.

The first `conpin` or `conpth` command after any other LPT library call clears all sources by calling the `devclr` command and then clears all matrix connections by calling the `clrcon` command before making the new connections.

The value `-1` for any item in the connection list is ignored by `conpth` and is considered a valid entry in the connection list.

The `conpth` command is not valid in the row-column connection scheme of the HVM1212A 3-kV matrix (S540 systems only).

Matrix errors are generated under the following conditions:

- I-V connections are included in the connection list, except as noted above.
- High-voltage pins or 2657A source-measure units (SMUs) are used in arguments in the `conpth` command (S540 systems with only an HVM1212A).

You can use this command in dual-site mode (S535 systems only).

NOTE

If you have a chuck connected in an S535 system that is in dual-site mode, the pin the chuck is on cannot be mirrored.

Also see

addcon (on page 2-24)
clrcon (on page 2-33)
conpin (on page 2-37)
devclr (on page 2-42)

delay

This command provides a user-programmable delay in a test sequence.

Models supported

S530, S535, S540

Usage

```
int delay(unsigned int n);
```

n	The duration of the delay in milliseconds

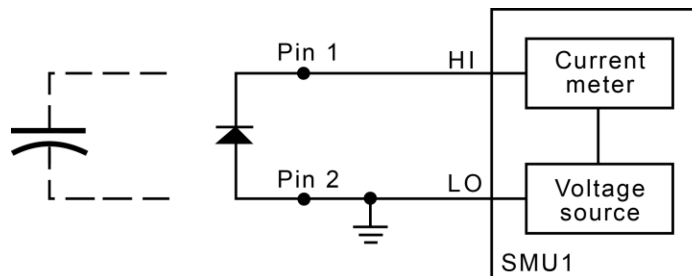
Details

The `delay` command can be called anywhere in the test sequence.

Example

```
double ir4;
.
.
conpin(SMU1, 1, 0);
conpin(GND, 2, 0);
forcev(SMU1, 60.0); /* Generate 60 V from SMU1. */
delay(20); /* Pause for 20 ms. */
measi(SMU1, &ir4); /* Measure current; return */
 /* result to ir4. */
```

This example measures the leakage current of a variable-capacitance diode. SMU1 applies 60 V across the diode. This device is always configured in the reverse bias mode, so the high side of SMU1 is connected to the cathode. Because this type of diode has very high capacitance and low leakage current, a 20 ms delay is added. After the delay, current through SMU1 is measured and stored in the variable IR4.



Also see

rdelay (on page 2-79)

delcon

This command removes specific matrix connections.

Usage

```
int delcon(int exist_connect, [int exist_connectn, [...]] 0);
```

exist_connect	A pin number or an instrument terminal ID
exist_connectn	A pin number or an instrument terminal ID

Details

This command disconnects all connections to each terminal or pin listed. Before disconnecting the pins or terminals, the `delcon` command clears all active sources by calling the `devclr` command.

A programmer can run a series of tests in a single test sequence using the `addcon` and `delcon` commands together without breaking existing connections. Only the required terminal and pin changes are made before the next sourcing and measuring operations.

If you use the `delcon` command to disconnect an instrument terminal but do not include all the pins it was connected to in the command, those omitted pins continue to be connected and continue to take up a matrix row. If you later use the `addcon` command to reconnect the instrument terminal, another matrix row is used (the pins left in the initial connection are not automatically reconnected).

Example

```
double i1,i2;
conpin(GND, 3, 4, 0);
conpin(SMU1, 1, 0);
conpin(SMU2, 2, 0);
forcev(SMU1, 1.0)
forcei(SMU2, 0.001);
measi(SMU1,&i1);
delcon(GND, 3, 4); /* disconnect GND and both pins connected to GND and clear both
    SMUs */
forcev(SMU1, 1.0);
measi(SMU1, &i2);
```

Also see

addcon (on page 2-24)
clrcon (on page 2-33)
conpin (on page 2-37)
conpth (on page 2-39)
devclr (on page 2-42)

devclr

This command sets all sources to a zero state.

Models supported

S530, S535, S540

Usage

```
int devclr(void);
```

Details

This command clears all sources sequentially in the reverse order from which they were originally forced. Before clearing all Keithley supported instruments, GPIB-based instruments are cleared by sending all strings defined with the `kibdefclr` command.

You can use this command in dual-site mode (S535 systems only). For more information about using dual-site mode, see "Dual-site operation" in the *S535 Reference Manual* (part number S535-901-01).

The `devclr` command is implicitly called by the `clrcon` and `devint` commands.

Also see

clrcon (on page 2-33)
devint (on page 2-42)
kibdefclr (on page 2-52)

devint

This command resets all active instruments in the system to their default states.

Models supported

S530, S535, S540

Usage

```
int devint(void);
```

Details

Resets all active instruments in the system to their default states. It clears the system by opening all relays and disconnecting the pathways. Meters and sources are reset to their default states. Refer to the hardware manuals for the instruments in your system for listings of available ranges and the default conditions and ranges.

Before resetting the instruments, this command:

1. Clears all sources by calling the `devclr` command.
2. Clears the matrix cross-points by calling the `clrcon` command.
3. Clears the trigger tables by calling the `clrtrg` command.
4. Clears the sweep tables by calling the `clrscn` command.
5. Resets GPIB instruments by sending the string defined with the `kibdefint` command.

System defaults after a `devint` command is called are shown in the following tables.

Source-measure unit (SMU) settings after a devint command (defaults)

Item	2410 (S530 systems only)	2636B	2657A (S540 systems only)	2461-SYS (S535 systems only)
rangei	105 µA	10 mA	10 mA	10 mA
rangev	21 V	20 V	20 V	20 V
limiti	10 mA	10 mA	10 mA	10 mA
limitv	1110 V	20 V	20 V	20 V
lorangei	1 µA	1 nA	1 nA	1 µA
lorangev	0.21 V	0.20 V	0.20 V	200 mV
Default measurement NPLC	0.01	0.01	0.01	0.01
Maximum NPLC	10	25	25	10
Minimum NPLC	0.01	0.001	0.001	0.01

CMTR settings after a devint command (defaults)

Item	4210-CVU
Measurement mode	System
Cable compensation	Off
AC V HI	HCUR/HPOT
DC V HI	HCUR/HPOT
AC drive level	0.045 V
Measurement model	Cp, Gp
Measurement speed	FAST
Frequency	100 kHz
DC voltage offset	0.0 V
Sample hold time	0.0 s
DC presoak voltage	0.0 V
DC bias	0.0 V
DC sample count	1
Sample interval	0.0 s
Current range	1 mA

You can use this command in dual-site mode (S535 systems only). Dual-site settings specified by the `site_enable` and `site_disable` commands are not reset by the `devint` command.

Also see

clrcon (on page 2-33)
clrscn (on page 2-34)
clrtrg (on page 2-35)
devclr (on page 2-42)
kibdefint (on page 2-53)

disable

This command stops the timer and sets the time value to zero (0).

Models supported

S530, S535, S540

Usage

```
int disable(int instr_id);
```

instr_id	The instrument identification code of the timer module (`TIMERn`)

Details

Timer reading is also stopped.

Sending `disable(TIMERn)` stops the timer and resets the time value to zero (0).

Also see

enable (on page 2-44)

enable

This command provides correlation of real time to measurements of voltage, current, conductance, and capacitance.

Models supported

S530, S535, S540

Usage

```
int enable(int instr_id);
```

instr_id	The instrument identification code of the timer module (`TIMERn`)

Details

Sending `enable(TIMERn)` initializes and starts the timer and allows other measurements to read the timer. The time starts at zero (0) at the time of the enable call.

Each parametric test system has a single timer. For compatibility with older systems, you can use `TIMER1` through `TIMER4` for the *`instr_id`* parameter, but any use of `TIMER2`, `TIMER3`, or `TIMER4` refers internally to `TIMER1`.

Also see

disable (on page 2-44)

forceX

This command programs a sourcing instrument to generate a voltage or current at a specific level.

Models supported

S530, S535, S540

Usage

```
int forcei(int instr_id, double value);
int forcev(int instr_id, double value);
```

instr_id	The instrument identification code; `SMUn`, `CMTRn`; for `forcev` it can also be `CMTR1H` and `CMTR1L` (see **Details**)
value	The level of the bipolar voltage or current forced in volts or amperes

Details

The `forcev` and `forcei` commands generate either a positive or negative voltage, as directed by the sign of the value argument. With both `forcev` and `forcei` commands:

- Positive values generate positive voltage or current from the high terminal of the source relative to the low terminal.
- Negative values generate negative voltage or current from the high terminal of the source relative to the low terminal.

The `forcev` command accepts both `CMTR1H` and `CMTR1L` for the *`instr_id`* parameter to support differential CVU biasing. By forcing one polarity on CMTR1H and an opposite polarity on CMTR1L, total bias can be up to 60 V, centered in relationship to ground. Note that it is not possible to exceed ± 30 V in relationship to ground.

When using the `limitX`, `rangeX`, and `forceX` commands on the same source at the same time in a test sequence, call the `limitX` and `rangeX` commands before the `forceX` command.

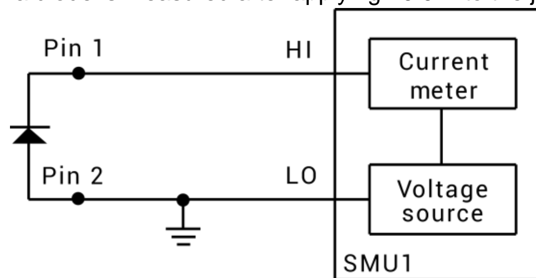
The ranges of currents and voltages available from a voltage or current source vary with the instrument type. For more detailed information, refer to the hardware manual for each instrument.

You can use this command in dual-site mode (SMUs only).

Example

```
double ir12;
.
.
conpin(2, GND, 0);
conpin(SMU1, 1, 0);
limiti(SMU1, 2.0E-4); /* Limit 1 mA to 200 uA. */
forcev(SMU1, 40.0); /* Apply 40.0 V. */
measi(SMU1, &ir12); /* Measure leakage; */
 /* return results to ir12. */
```

The reverse bias leakage of a diode is measured after applying 40.0 V to the junction.



Also see

None

getlpterr

This command returns the first LPT library error since the last `devint` command.

Models supported

S530, S535, S540

Usage

```
int getlpterr(void);
```

Details

This command returns the error code of the first error encountered since the last call to the `devint` command.

Also see

devint (on page 2-42)

getstatus

This command returns the operating state of a specified instrument.

Models supported

S530, S535, S540

Usage

```
int getstatus(int instr_id, unsigned int parameter, double *result);
```

instr_id	The instrument identification code; `SMU`*n* or `VMTR`*n*
parameter	The parameter of query
result	The data returned from the instrument; the `getstatus` command returns one item in single-site mode and two in dual-site mode

Details

The `getstatus` command returns instrument-specific operating states.

If you see the `UT_INVLDPRM` invalid parameter error returned from the `getstatus` command, it indicates that the status item parameter is illegal for this device. The requested status code is invalid for the selected device.

A list of supported `getstatus` command values for *`parameter`* for a source-measure unit (SMU) are provided in the following table.

This command returns dual-site data for SITE0 and SITE1 if available (S535 systems only).

Supported SMU getstatus query parameters

SMU parameter	Returns	Comment
`KI_IPVALUE`	The presently programmed output value	Current value (I output value)
`KI_VPVALUE`		Voltage value (V output value)
`KI_IPRANGE`	The presently programmed range	Current range (full-scale range value, or 0.0 for autorange)
`KI_VPRANGE`		Voltage range (full-scale range value, or 0.0 for autorange)
`KI_MAX_VOLTAGE`	The presently programmed maximum voltage	For systems with 2657A source-measure units (SMUs) only; a value between 300 V and 3000 V
`KI_IARANGE`	The presently active range	Current range (full-scale range value)
`KI_VARANGE`		Voltage range (full-scale range value)
`KI_IMRANGE`	The present range used when last measurement was performed	For autorange, the range at which the previous current measurement was made
`KI_VMRANGE`		For autorange, the range at which the previous voltage measurement was made
`KI_MEAS_DELAY`	The presently programmed SMU measure delay	`OFF` = No delay `AUTO` = The instrument sets the delay automatically; default setting <*`value`*> = User-specified value in seconds

SMU parameter	Returns	Comment
KI_MEAS_DELAY_FACTOR	2636 only: The presently programmed delay multiplier	For 26xx SMUs when the KI_MEAS_DELAY modifier is set to KI_DELAY_AUTO; the value that stored SMU delay values are multiplied by
KI_INTGPLC	The presently programmed period to average measurements	AC power line cycles: CMTR: 0.006 to 10.002 24xx: 0.01 to 10 (S530 only) 26xx: 0.001 to 25 2461-SYS: 0.01 to 10
KI_COMPLNC	Active compliance status for fixed range	In range compliance if 1
KI_2600_ANALOG_FILTER	The active state of the analog filter for a specified SMU	For 26xx SMUs; on or off
KI_SYSTEM_SPEED_MODE	The presently programmed speed mode	FAST or CUSTOM mode
KI_SETTLE_MODE	The presently programmed settling mode	For 26xx SMUs (2461-SYS not supported): SMOOTH = Additional settings options off FAST_RANGE = Faster range changes FAST_POLARITY = Polarity changes without going to zero DIRECT_IRANGE = SMU changes range directly (default) FAST_ALL = All settle fast modes enabled
KI_CALDATE	The date the instrument was last calibrated	

Also see

None

imeast

This command forces a reading of the timer and returns the result.

Models supported

S530, S535, S540

Usage

```
int imeast(int instr_id, double *result);
```

instr_id	The instrument identification code of the device
result	The variable assigned to the measurement

Details

This command applies to all timers.

This command returns dual-site data for SITE0 and SITE1 if available (S535 systems only).

Also see

None

insbind

This command establishes a cooperative relationship between two instruments.

Models supported

S530, S535, S540

NOTE

This command is valid only for CMTR2 on S540 systems with two CMTRs.

Usage

```
int insbind(int instr1, int instr2);
```

instr1	The first instrument to bind
instr2	The second instrument to bind

Details

Some instruments are designed to be used with other instruments to provide complementary or enhanced functionality to the other instrument. For example, some capacitance meters (CMTRs) cannot generate bias voltages by themselves; they rely on a source-measure unit (SMU) to act as the bias source.

This command closes relays on the HVM1212 matrix to connect the specified high-voltage SMU to the bias tee connected to CMTR2. This allows the high-voltage SMU to bias a higher voltage than that normally available from CMTR2.

NOTE

This command works differently than it does on the S400 and S600 test systems. Linear Parametric Test Library (LPTLib) code imported from these systems must be modified to accommodate differences. For example, executing the `forceV` command (`CMTR2, x`) will have no effect; this is a code compatibility issue for S400 and S600 code.

The `devint` command restores all instruments to normal operation.

Example

```
double capval;

conpin(CMTR2H, 1, 0);
conpin(CMTR2L, 2, 0);
insbind(HVGND, CMTR2L);
insbind(HVSMU1, CMTR2H);
forcev(HVSMU1, 1500);
measc(CMTR2, &capval);
devint();
```

Connects CMTR2 to pins 1 and 2 and binds HVSMU1 to CMTR2. HVSMU1 then biases voltage and measures capacitance. The `devint` command unbinds the instruments after the measurement is made.

Also see

devint (on page 2-42)

intgX

This command performs voltage or current measurements averaged over a user-defined period (usually one AC line cycle).

Models supported

S530, S535, S540

Usage

```
int intgc(int instr_id, double *result);
int intgcg(int instr_id, double *capacitance, double *conductance);
int intgg(int instr_id, double *result);
int intgi(int instr_id, double *result);
int intgv(int instr_id, double *result);
```

instr_id	The instrument identification code of the measuring instrument; `SMUn`, CMTR*n*, `VMTRn`
result	The variable assigned to the result of the measurement
capacitance	The variable assigned the capacitance measurement
conductance	The variable assigned the conductance measurement

Details

The averaging is done in hardware by integration of the analog measurement signal over a specified period of time. The integration is automatically corrected for 50 Hz or 60 Hz power mains.

The default integration time is one AC line cycle (1 PLC). This default time can be overridden with the `KI_INTGPLC` option of `setmode`. The integration time can be set from 0.01 PLC to 10.0 PLC. The `devint` command resets the integration time to the one AC line cycle default value.

The `rangeX` command directly affects the operation of the `intgX` command. The use of the `rangeX` command prevents the instrument addressed from automatically changing ranges. This can result in an overrange condition that would occur when measuring 10.0 V on a 2.0 V range. An overrange condition returns the value as the measurement result.

If used, the `rangeX` command must be in the test sequence before the associated `intgX` command.

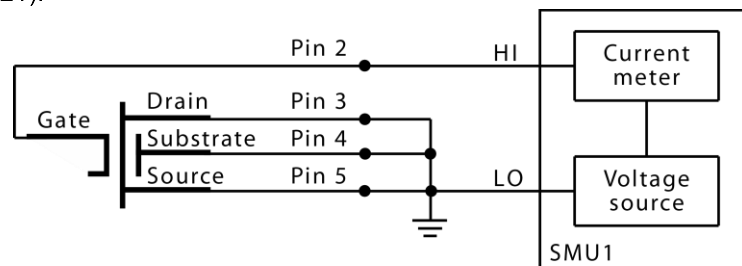
In general, measurement commands that return multiple results are more efficient than performing multiple measurement commands. For example, performing a single call to the `intgcg` command is faster than calling the `intgc` command followed by the `intgg` command.

The `intgi` and `intgv` commands return dual-site data for SITE0 and SITE1 if available (S535 systems only).

Example

```
double idss;
.
.
conpin(GND, 5, 4, 3, 0);
conpin(SMU1, 2, 0);
limiti(SMU1, 2.0E-8); /* Limits to 20.0 nA. */
rangei(SMU1, 2.0E-8); /* Select range for 20.0 nA */
forcev(SMU1, 25.0); /* Apply 25 V to the gate. */
intgi(SMU1, &idss); /* Measure gate leakage; */
 /* return results to idss. */
```

This example measures the relatively low leakage current of a metal-oxide semiconductor field-effect transistor (MOSFET).



Also see

devint (on page 2-42)
measX (on page 2-61)
rangeX (on page 2-77)
setmode (on page 2-101)

kibdefclr

This command defines the device-dependent command sent to an instrument connected to the GPIB1 interface.

Models supported

S530, S535, S540

Usage

```
int kibdefclr(int pri_addr, int sec_addr, unsigned int timeout, double delay, unsigned
    int snd_size, char *sndbuffer);
```

pri_addr	The primary address of the instrument (0 to 30; the controller uses address 31)
sec_addr	The secondary address of the instrument (1 to 30; if the instrument device does not support secondary addressing, this parameter must be -1)
timeout	The GPIB timeout for the transfer in 100 ms units (for example, timeout = 40 = 4.0 s)
delay	The time to wait after the device-dependent string is sent to the device, in seconds
snd_size	The number of bytes to send over the GPIB interface
sndbuffer	The physical byte buffer containing the data to send over the bus (the physical CLEAR string); a maximum of 1024 bytes is allowed

Details

This string is sent during any normal tester-based `devclr` command. It ensures that if the tester is calling the `devclr` command internally, any external GPIB device is cleared with the given string.

Each call to the `kibdefclr` and `kibdefint` commands copies parameters into a data structure within the tester memory. These data structures are allocated dynamically. These tables are cleared when the `devint` command executes. Any strings previously defined must be redefined.

The tester system allows you to define a maximum of 20 clear and 20 initialization strings. Each string may contain up to a maximum of 1024 bytes.

Strings are sent over the GPIB interface in a first-in, first-out queue. This means that the first call to the `kibdefclr` or `kibdefint` command is the first string sent over the GPIB. The `devclr` (`kibdefclr`) strings are always sent before initialization.

The KIBLIB `devclr` strings are sent before the `devclr` and `devint` commands execute. This may be a problem when communicating with any Keithley-supported GPIB instruments. This may also have an effect on the `bsweepX` command, because the `bsweepX` command sends a call to the `devclr` command to clear active sources. It is not recommended to use GPIB instruments when performing tests with the `bsweepX` command.

Also see

devclr (on page 2-42)
devint (on page 2-42)
kibdefint (on page 2-53)

kibdefint

This command defines a device-dependent command sent to an instrument connected to the GPIB1 interface.

Models supported

S530, S535, S540

Usage

```
int kibdefint(int pri_addr, int sec_addr, unsigned int timeout, double delay, unsigned
    int snd_size, char *snd_buff);
```

pri_addr	The primary address of the instrument (0 to 30; the controller uses address 31)
sec_addr	The secondary address of the instrument (0 to 31; if the instrument device does not support secondary addressing, this parameter must be -1)
timeout	The GPIB timeout for the transfer in 100 ms units (for example, timeout = 40 = 4.0 s)
delay	The time to wait after the device-dependent string is sent to the device, in seconds
snd_size	The number of bytes to send over the GPIB interface
snd_buff	The physical byte buffer containing the data to send over the bus (the INITIALIZE string); a maximum of 1024 bytes is allowed

Details

This string is sent during any normal tester-based call to the `devint` command. It ensures that if the tester is calling the `devint` command internally, any external GPIB device is initialized with the rest of the known instruments.

Each call to the `kibdefclr` and `kibdefint` commands copies parameters into a data structure within the tester memory. These data structures are allocated dynamically. These tables are cleared when the `devint` command executes. Any strings previously defined must be redefined.

The tester system allows you to define a maximum of 20 clear and 20 initialization strings. Each string may contain up to a maximum of 1024 bytes.

Strings are sent over the GPIB interface in a first-in, first-out queue. This means that the first call to the `kibdefclr` or `kibdefint` command is the first string sent over the GPIB. The `devclr` (`kibdefclr`) strings are always sent before initialization.

The KIBLIB `devclr` strings are sent before the `devclr` and `devint` commands execute. This may be a problem when communicating with any Keithley-supported GPIB instruments. This may also have an effect on the `bsweepX` command, because the `bsweepX` command sends a call to the `devclr` command to clear active sources. It is not recommended to use GPIB instruments when performing tests with the `bsweepX` command.

Also see

devclr (on page 2-42)
devint (on page 2-42)
kibdefclr (on page 2-52)

kibrcv

This command reads a device-dependent string from an instrument connected to the GPIB interface.

Models supported

S530, S535, S540

Usage

```
int kibrcv(int pri_addr, int sec_addr, char term, unsigned int timeout, unsigned int
    rcv_size, unsigned int *rcv_len, char *rcv_buff);
```

pri_addr	The primary address of the instrument (0 to 30; the controller uses address 31)
sec_addr	The secondary address of the instrument (1 to 30; if the instrument device does not support secondary addressing, this parameter must be -1)
term	The ASCII delimiter character of the returned string; this is the byte used for terminating data buffer reading
timeout	The GPIB timeout for the transfer in 100 ms units (for example, timeout = 40 = 4.0 s)
rcv_size	The physical receive buffer size; this is the maximum number of bytes that can be read from the device
rcv_len	The number of bytes that are read from the device on the GPIB interface; this variable is returned by the tester after all bytes are read from the device
rcv_buff	The physical byte buffer destined to receive the data from the device connected to the GPIB interface

Details

The `kibrcv` command receives a buffer from the GPIB interface by doing the following:

1. Assert attention (ATN).
2. Send device LISTEN address.
3. Send device TALK address.
4. Send secondary address (if not -1).
5. De-assert ATN.
6. Read byte array from the device *rcv_buff* parameter until end-or-identify (EOI) or the delimiter is received.
7. Assert ATN.
8. Send UNTalk (UNT).
9. Send UNListen (UNL).
10. De-assert ATN.

The *rcv_size* parameter defines the maximum number of bytes physically allowed in the buffer. If the *rcv_size* parameter is greater than the byte string returned by the instrument, the device is short-cycled and only the maximum number of bytes is returned.

Also see

None

kibsnd

This command sends a device-dependent command to an instrument connected to the GPIB interface.

Models supported

S530, S535, S540

Usage

```
int kibsnd(int pri_addr, int sec_addr, unsigned int timeout, unsigned int send_len, char
    *send_buff);
```

pri_addr	The primary address of the instrument (0 to 30; the controller uses address 31)
sec_addr	The secondary address of the instrument (0 to 31; if the instrument device does not support secondary addressing, this parameter must be -1)>
timeout	The GPIB timeout for the transfer in 100 ms units (for example, timeout = 40 = 4.0 s)
send_len	The number of bytes to send over the GPIB interface
send_buff	The physical byte buffer containing the data to send over the bus

Details

The `kibsnd` command sends a buffer out through the GPIB interface by doing the following:

1. Assert attention (ATN).
2. Send device LISTEN address.
3. Send secondary address (if not -1).
4. Send my TALK address.
5. De-assert ATN.
6. Send the *send_buff* parameter with end-or-identify (EOI) asserted with the last byte.
7. Assert ATN.
8. Send UNTalk (UNT).
9. Send UNListen (UNL).
10. De-assert ATN.

Also see

None

kibspl

This command serial polls an instrument connected to the GPIB interface.

Models supported

S530, S535, S540

Usage

```
int kibspl(int pri_addr, int sec_addr, unsigned int timeout,
    int *serial_poll_byte);
```

pri_addr	The primary address of the instrument (0 to 30; the controller uses address 31)
sec_addr	The secondary address of the instrument; numbers 0 through 31 are valid; if the instrument device does not support secondary addressing, this parameter must be -1
timeout	The GPIB polling timeout in 100 ms units (for example, timeout = 40 = 4.0 s)
serial_poll_byte	The variable name for the serial poll byte returned by the current; must be 4 bytes long

Details

The `kibspl` command does the following:

1. Assert attention (ATN).
2. Send serial poll enable (SPE).
3. Send LISTEN address.
4. Send device TALK address.
5. Send secondary address (if not -1).
6. De-assert ATN.
7. Poll GPIB interface until data is available.
8. Read the *serial_poll_byte* parameter from the device (if data is available)
9. Assert ATN.
10. Send serial poll disable (SPD).
11. Send UNTalk (UNT).
12. Send UNListen (UNL).
13. De-assert ATN.

Also see

kibsplw (on page 2-57)

kibsplw

This command synchronously serial polls an instrument connected to the GPIB interface.

Models supported

S530, S535, S540

Usage

```
int kibsplw(int pri_addr, int sec_addr, unsigned int timeout, int *serial_poll_byte);
```

pri_addr	The primary address of the instrument (0 to 30)
sec_addr	The secondary address of the instrument (0 to 31; if the instrument device does not support secondary addressing, this parameter must be –1)
timeout	The GPIB polling timeout in 100 ms units (for example, a timeout of 40 = 4.0 s)
serial_poll_byte	The serial poll status byte variable name returned by the device presently being polled

Details

This command waits for SRQ to be asserted on the GPIB by any device. After SRQ is asserted, a serial poll sequence is initiated for the device and the serial poll status byte is returned.

The `kibsplw` command does the following:

1. Waits with timeout for general SRQ assertion on the GPIB.
2. Calls the `kibspl` command.

Also see

kibspl (on page 2-56)

limitX

This command allows the programmer to specify a current or voltage limit other than the default limit of the instrument.

Models supported

S530, S535, S540

Usage

```
int limiti(int instr_id, double limit_val);
int limitv(int instr_id, double limit_val);
```

instr_id	The instrument identification code of the instrument on which to impose a source value limit; `SMUn`
limit_val	The maximum level of the current or voltage; see **Details**

Details

The parameter *limit_val* is bidirectional. For example, the command `limitv(SMU1, 10.0)` limits the voltage of the current source SMU1 to ±10.0 V. The command `limiti(SMU1, 1.5E-3)` limits the current of the voltage source SMU1 to ±1.5 mA.

Use the `limiti` command to limit the current of a voltage source. Use the `limitv` command to limit the voltage of a current source.

NOTE

If the instrument is ranged below the programmed limit value, the instrument will temporarily limit to full scale of range.

This command must be called in the test sequence before the associated `forceX`, `sweepX`, or `searchX` command is used to generate the voltage or current. The `limitX` command also sets the top measurement range of an autoranged measurement.

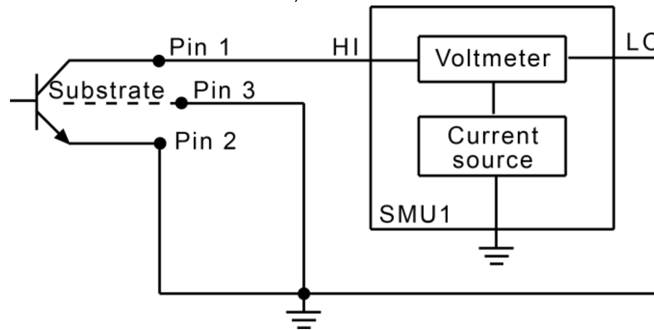
The limits set within a particular test sequence are cleared when the `devint` command is called.

You can use this command in dual-site mode (S535 systems only).

Example

```
double ibceo, vbceo;
.
.
conpin(2, 3, GND, 0);
conpin(SMU1, 1, 0);
limitv(SMU1, 150.0); /* Limit voltage at 150 V. */
forcei(SMU1, ibceo); /* Force current through the DUT. */
measv(SMU1, &vbceo); /* Measure breakdown voltage; */
. /* return results to vbceo. */
.
```

This example measures the breakdown voltage of a device. The limit is set at 150 V. This limit is necessary to override the default limit of the SMU, which would otherwise be in effect.



Also see

devint (on page 2-42)
forceX (on page 2-45)
measX (on page 2-61)
rangeX (on page 2-77)
searchX (on page 2-97)
sweepX (on page 2-118)

lorangeX

This command defines the bottom autorange limit.

Models supported

S530, S535, S540

Usage

```
int lorangei(int instr_id, double range);
int lorangev(int instr_id, double range);
```

instr_id	The instrument identification code
range	The value of the instrument range, in volts or amperes

Details

The `lorangeX` command is used with autoranging to limit the number of range changes, which saves test time.

If the instrument is on a range lower than the one specified by the `lorangeX` command, the range is changed. The system automatically provides any settling delay for the range change that may be necessary due to this potential range change.

NOTE

When the force and measure functions are both current (I) or both voltage (V), specifying a lower force range overrides the `lowrangeX` command.

Once defined, the `lorangeX` command is in effect until a `devclr` or `devint` command, or another `lorangeX` command executes.

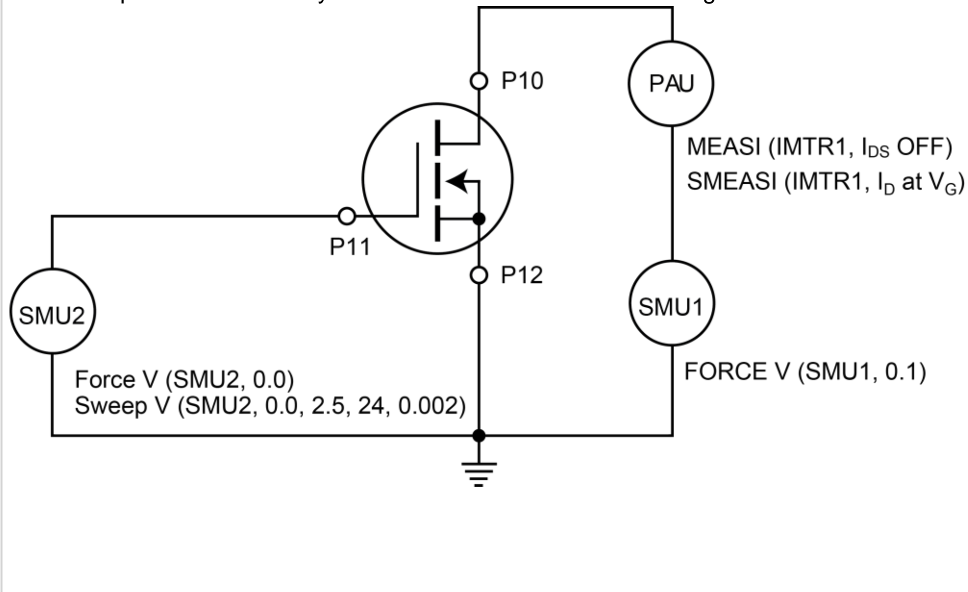
You can use this command in dual-site mode (S535 systems only).

Example

```
double idatvg[25];

.
.
conpin(SMU1, 10, 0);
conpin(SMU2, 11, 0):
conpin(12, GND, 0);
lorangei(SMU1, 2.0E-6); /* Select 2 uA as minimum */
 /* range during autoranging. */
smeasi(SMU1, idatvg); /* Set up sweep measurement */
 /* of IDS. */
sweepv(SMU2, 0.0, 2.5, 24, 0.002); /* Sweep */
 /* gate from 0 V to 2.5 V. */
```

This example illustrates how you would select the bottom autorange limit.



Also see

devclr (on page 2-42)
devint (on page 2-42)

measX

This command allows the measurement of voltage, current, charge capacitance, or conductance.

Models supported

S530, S535, S540

Usage

```
int measc(int instr_id, double *result);
int meascg(int instr_id, double *c, double *g);
int measg(int instr_id, double *result);
int measi(int instr_id, double *result);
int measv(int instr_id, double *result);
```

instr_id	The instrument identification code
c	The variable assigned to the capacitance of the measurement
g	The variable assigned to the conductance of the measurement
result	The variable assigned to the result of the measurement

Details

After the command is called, all relay matrix connections remain closed, and the sources continue to generate voltage or current. For this reason, two or more measurements can be made in sequence.

The `rangeX` command directly affects the operation of the `measX` command. Using the `rangeX` command prevents the instrument addressed from automatically changing ranges when the `measX` command is called. This can result in an overrange condition such that would occur when measuring 10 V on a 2 V range. An overrange condition returns the value as the result of the measurement.

If used, the `rangeX` command must be in the test sequence before the associated `measX` command.

All measurements except the `meast` command invoke a timer snapshot measurement to be made by all enabled timers. This timer snapshot can then be read with the `meast` command.

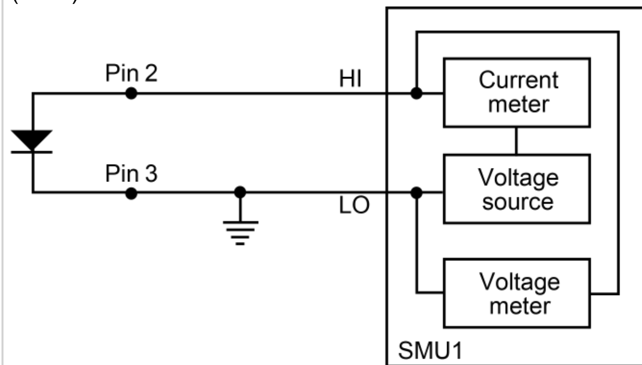
In general, measurement functions that return multiple results are more efficient than performing multiple measurement functions. For example, calling a single `meascg` command is faster than calling the `measc` command followed by the `measg` command.

The `measi` and `measv` commands return dual-site data for SITE0 and SITE1 if available (S535 systems only).

Example

```
double if46, vf47;
.
.
if46 = 50e-3;
.
.
conpin(3, GND, 0);
conpin(SMU1, 2, 0);
forcei(SMU1, if46); /* Forward bias the diode; */
 /* set SMU current */
 /* limit to 50 mA. */
measv(SMU1, &vf47); /* Measure forward bias; */
 /* return result to vf47. */
```

In this example, the forward bias voltage of the diode is obtained from a single source-measure unit (SMU).



Also see

rangeX (on page 2-77)

mpulse

This command uses a source-measure unit (SMU) to force a voltage pulse and measure both the voltage and current for exact device loading.

Models supported

S530, S535, S540

Usage

```
int mpulse(int instr_id, double pulse_amplitude, double pulse_duration, double *vmeas,
    double *imeas);
```

instr_id	The instrument identification code of the instrument under control
pulse_amplitude	The pulse height in volts
pulse_duration	The pulse width in seconds; the measurements are made at the end of the pulse before the `mpulse` command is shut down
vmeas	The variable used to receive the voltage on the output of the SMU at the time the pulse terminates; this reading is buffered internally
imeas	The variable used to receive the current drawn from the SMU; this measurement is made simultaneously with the voltage, so the combined values are an exact representation of the device load at pulse termination

Details

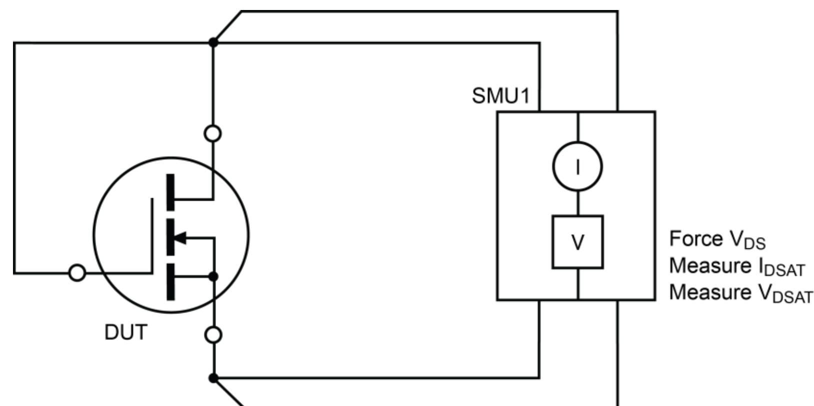
Voltage and current are measured just before the pulse terminates. Pulse mode is used because the device under test will be destroyed if the voltage is applied for a long period of time, such as with GaAs type devices or high-power bipolar.

This command returns dual-site data for SITE0 and SITE1 if available (S535 systems only).

Example

```
.
.
mpulse(SMU1, vds, 1.0E-3, vdsat, idsat)
```

This example measures the drain current of a metal-oxide semiconductor field-effect transistor (MOSFET) when drain-source voltage (V_{DS}) equals gate-source voltage (V_{GS}). A voltage pulse, V_{DS}, is applied to the drain. The pulse duration is 1 ms. Voltage across the MOS transistor, V_{DSAT}, and drain current, I_{DSAT}, are measured.



Also see

None

pgu_current_limit

This command sets the maximum amount of current that the pulse generator unit (PGU) channel can supply because of the pulse amplitude and load impedance.

Models supported

S530, S540 (systems with pulse-generator units (PGUs))

Usage

```
istat = int pgu_current_limit(int instr_id, double limit);
```

intstr_id	The instrument identification code of the PGU
limit	The current limit value

Details

This command returns a `0` if executed without error; a negative number indicates an error.

Example

`pgu_current_limit(PGU2, 1e-3)`
Sets the current limit of PGU2 to 1 mA.

Also see

pgu_load (on page 2-67)

pgu_delay

This command sets the amount of time to wait after a trigger signal or command is received before outputting a pulse (trigger delay time).

Models supported

S530, S540 (systems with pulse-generator units (PGUs))

Usage

```
istat = int pgu_delay(int instr_id, double delay_time);
```

instr_id	The instrument identification code of the pulse generator
delay_time	The trigger delay time in seconds (input)

Details

This command returns a `0` if executed without error; a negative number indicates an error.

Example

Code	Description
`istat = pgu_delay(PGU1A,0.0);`	Set channel 1 to have no pulse delay.
`...`	
`istat = pgu_delay(PGU1B,10e-06);` `istat = pgu_trig();`	Set channel 2 to trigger a pulse after a delay of 10 µs.

Also see

pgu_period (on page 2-69)
pgu_range (on page 2-70)
pgu_trig (on page 2-72)

pgu_fall

This command sets the fall time of a pulse.

Models supported

S530, S540 (systems with pulse-generator units (PGUs))

Usage

```
istat = int pgu_fall(int instr_id, double fall_time);
```

Parameter	Description
instr_id	The instrument identification code of the pulse generator unit (PGU)
fall_time	The desired fall time in seconds (input)

Details

This command returns a `0` if executed without error; a negative number indicates an error.

The fall time must be greater than 10 ns and less than 33 ms.

Example

```
pgu_fall(PGU2, 50e-9)
```

Sets the pulse fall time of PGU2 to 50 ns.

Also see

pgu_range (on page 2-70)
pgu_rise (on page 2-70)
pgu_trig (on page 2-72)

pgu_halt

This command stops all the pulse channels in the identified instrument card.

Models supported

S530, S540 (systems with pulse-generator units (PGUs))

Usage

```
istat = int pgu_halt(int instr_id);
```

instr_id	The instrument identification code of the pulse generator unit (PGU)

Details

This command returns a `0` if executed without error; a negative number indicates an error.

Example

`pgu_halt(PGU1)`
Stops pulse output on PGU1.

Also see

pgu_trig (on page 2-72)

pgu_height

This command sets the peak-to-peak height of the pulse.

Models supported

S530, S540 (systems with pulse-generator units (PGUs))

Usage

```
istat = int pgu_height(int instr_id, double height);
```

instr_id	The instrument identification code
height	The pulse height in volts (input)

Details

This command returns a `0` if executed without error; a negative number indicates an error.

The pulse height must be greater than -20 V and less than 20 V.

Example

`pgu_height(PGU2, 2)`
Sets the height of the pulse on PGU2 to 2 V.

Also see

None

pgu_init

This command initializes communication with the pulse card and sets the pulse generator to a specific set of default conditions.

Models supported

S530, S540 (systems with pulse-generator units (PGUs))

Usage

```
istat = int pgu_init(int instr_id);
```

instr_id	The instrument identification code

Details

This command returns a `0` if executed without error; a negative number indicates an error.

The `pgu_init` command sets the pulse generators to the following states:

- Set to single pulse mode
- Set output impedance to 50 Ω
- Set output polarity to normal
- Enable software triggering
- Set rise time, fall time to 100 ns
- Set pulse delay to 0 s
- Set pulse height to 0.2 V
- Set pulse width to 500 ns

Example

`pgu_init(PGU2)`
Initializes PGU2 and resets it to default settings.

Also see

None

pgu_load

This command sets the load impedance of a pulse.

Models supported

S530, S540 (systems with pulse-generator units (PGUs))

Usage

```
istat = int pgu_load(int instr_id, double load);
```

instr_id	The instrument identification code
load	The output (load) impedance in ohms (input)

Details

This command returns a `0` if executed without error; a negative number indicates an error.

The load must be greater than 1 Ω and less than 10 MΩ.

Example

`pgu_load(PGU1, 100)`
Sets the output impedance of PGU2 to 100 Ω.

Also see

None

pgu_mode

This command sets the pulse mode of the pulse generator.

Models supported

S530, S540 (systems with pulse-generator units (PGUs))

Usage

```
istat = int pgu_mode(int instr_id, int mode);
```

instr_id	The instrument identification code
mode	The pulse mode of the pulse generator: ■ Single = `0`; single-pulse output ■ Continuous = `1`; continuous stream of pulses ■ Burst = `2`; a burst of a specified number of pulses

Details

This command returns a `0` if executed without error; a negative number indicates an error.

Mode numbers less than 0 and greater than 2 are not accepted.

NOTE

To stop the generation of continuous pulses and reset the pulse generator to its default state (single-pulse output), send the pgu_init (on page 2-67) command.

Example

`pgu_mode(PGU2, 1)`
Sets PGU2 to the continuous pulse mode.

Also see

pgu_init (on page 2-67)

pgu_offset

This command sets the peak-to-peak height and DC offset of the pulse.

Models supported

S530, S540 (systems with pulse-generator units (PGUs))

Usage

```
istat = int pgu_offset(int instr_id, double amplitude, double offset);
```

instr_id	The instrument identification code
amplitude	The peak-to-peak amplitude in volts (input); a positive number
offset	The DC offset of the pulse in volts (input); a positive or negative number

Details

This command returns a `0` if executed without error; a negative number indicates an error.

The amplitude and offset combined must be greater than -20 V and less than +20 V.

Example

```
pgu_offset(PGU1, 10, 5)
```

Sets the peak-to-peak amplitude and DC offset on PGU1.

Also see

None

pgu_period

This command sets the period of a pulse.

Models supported

S530, S540 (systems with pulse-generator units (PGUs))

Usage

```
istat = int pgu_period(int instr_id, double period);
```

instr_id	The instrument identification code
period	The pulse period in seconds (input)

Details

This command returns a `0` if executed without error; a negative number indicates an error.

Example

```
pgu_period(PGU1, 200e-9)
```

Sets the pulse period of PGU1 to 200 ns.

Also see

pgu_trig (on page 2-72)

pgu_range

This command sets the voltage range of a pulse generator channel.

Models supported

S530, S540 (systems with pulse-generator units (PGUs))

Usage

```
istat = int pgu_range(int instr_id, double range);
```

instr_id	The instrument identification code
range	The voltage range of the pulse (input); `5` or `20`

Details

This command returns a `0` if executed without error; a negative number indicates an error.

Example

`pgu_range(PGU2, 20)`
Sets the voltage range of the PGU2 to 20 V.

Also see

pgu_fall (on page 2-65)
pgu_period (on page 2-69)
pgu_rise (on page 2-70)
pgu_width (on page 2-74)

pgu_rise

This command sets the rise time of a pulse.

Models supported

S530, S540 (systems with pulse-generator units (PGUs))

Usage

```
istat = int pgu_rise(int instr_id, double rise_time);
```

instr_id	The instrument identification code
rise_time	The rise time in seconds (input)

Details

This command returns a `0` if executed without error; a negative number indicates an error.

The rise time must be greater than 10 ns and less than 33 ms.

Example

`pgu_rise(PGU1, 50e-9)`
Sets the pulse rise time of PGU1 to 50 ns.

Also see

pgu_fall (on page 2-65)
pgu_range (on page 2-70)
pgu_trig (on page 2-72)

pgu_select

This deprecated command selects a pulse generator unit on which to modify a pulse output channel.

Models supported

S530 (systems with pulse-generator units (PGUs))

Usage

```
int pgu_select(int instr_id)
```

instr_id	The instrument identification code

Details

This function has been deprecated, but it remains to support older code. This command should not be used in any new code that you develop.

pgu_trig

This command triggers the first pulse generator unit and outputs the waveforms (this command is for compatibility with previous versions of this software).

Models supported

S530, S540 (systems with pulse-generator units (PGUs))

Usage

```
istat = int pgu_trig(int instr_id);
```

instr_id	The instrument identification code

Details

This command returns a `0` if executed without error; a negative number indicates an error.

This command is similar to using the `pgu_trig_unit(1)` command.

Example

`pgu_trig(PGU1)`
Triggers the first pulse generator unit (PGU1).

Also see

pgu_delay (on page 2-64)
pgu_trig_burst (on page 2-72)
pgu_trig_unit (on page 2-73)

pgu_trig_burst

This command triggers a specified number of pulses on the selected pulse generator unit.

Models supported

S530, S540 (systems with pulse-generator units (PGUs))

Usage

```
istat = int pgu_trig_burst(int instr_id, int unit, long count);
```

instr_id	The instrument identification code
unit	Pulse generator unit number (input): `1`, `2`, `3`, or `4`, depending on system configuration
count	The number of pulses to output: `1` to `65535` (input).

Details

This command triggers a burst of pulses that have been previously defined. Unlike the `pgu_trig` command, this command takes a unit number and a number of pulses as arguments so that you can trigger a stream of pulses on a specific pulse generator unit.

To trigger multiple pulse generator units, you can sum the unit numbers designated to the pulse generators as listed in the following table.

Number	Pulse generator unit
4096	PTRIG1
8192	PTRIG2
16384	PTRIG3
32768	PTRIG4

This command returns a `0` if executed without error; a negative number indicates an error.

Example

```
istat = pgu_trig_burst(2, 107)
```

Triggers 107 pulses on pulse generator unit number 2.

Also see

pgu_trig (on page 2-72)
pgu_trig_unit (on page 2-73)

pgu_trig_unit

This command triggers a specified pulse generator unit or units to output waveforms.

Models supported

S530, S540 (systems with pulse-generator units (PGUs))

Usage

```
istat = int pgu_trig_unit(int unit);
```

unit	Pulse generator unit number (input): `1`, `2`, `3`, or `4`, depending on system configuration

Details

This command returns a `0` if executed without error; a negative number indicates an error.

To trigger multiple pulse generator units, you can sum the unit numbers designated to the pulse generators as listed in the following table.

Number	Pulse generator unit
4096	PTRIG1
8192	PTRIG2
16384	PTRIG3
32768	PTRIG4

Example 1

`istat = pgu_trig_unit(2)`
Triggers pulses on pulse generator unit 2.

Example 2

`istat = pgu_trig_unit(12288)`
Triggers pulses on pulse generator units 1 and 2.

Example 3

`istat = pgu_trig_unit(3)`
Triggers the pulses on pulse generator unit 3 (not units 1 and 2).

Also see

pgu_trig (on page 2-72)
pgu_trig_burst (on page 2-72)

pgu_width

This command sets the width of a pulse.

Models supported

S530, S540 (systems with pulse-generator units (PGUs))

Usage

```
istat = int pgu_width(int instr_id, double width);
```

instr_id	The instrument identification code
width	The pulse width in seconds (input)

Details

This command returns a `0` if executed without error; a negative number indicates an error.

The width must be greater than 250 ns and less than 999 ms.

Example

`istat pgu_width(PGU2, 10e-06)`
Sets the pulse width on PGU2 to 10 µs.

Also see

pgu_period (on page 2-69)
pgu_range (on page 2-70)
pgu_trig (on page 2-72)
pgu_trig_burst (on page 2-72)
pgu_trig_unit (on page 2-73)

pulseX

This command directs a sourcing instrument to force a voltage or current at a specific level for a predetermined length of time.

Models supported

S530, S535, S540

Usage

```
int pulsei(int instr_id, double forceval, double time);
int pulsev(int instr_id, double forceval, double time);
```

instr_id	The instrument identification code; `SMUn`
forceval	The level of voltage in volts or current in amperes to force; see **Details**
time	The pulse duration in seconds; for example, a time of 0.5 initiates a time of 0.5 s, and a time of 2.0e-2 initiates a time of 20 ms; the minimum practical time for a source-measure unit (SMU) source is dependent on the voltage or current level being sourced and the impedance of the device under test (DUT)

Details

The *forceval* parameter can be positive or negative. For example, sending `pulsev(SMU1, 10.0, 10e-3)` generates +10 V for 10 ms, and sending `pulsei(SMU1, -1.5e-3, 10e-3)` generates -1.5 mA for 10 ms.

The ranges of current and voltage available vary with the instrument type. For more detailed information, refer to the hardware manuals of the instruments in your system.

The `pulsev` and `pulsei` commands generate either a positive or negative voltage, as specified by the sign of the value argument. With both the `pulseV` and `pulseI` commands:

- A positive value generates a positive voltage from the high terminal of the source.
- A negative value generates a negative voltage from the high terminal of the source.

After the `pulseX` command is executed, other commands may be used to make measurements. The `measX` command can measure:

- Residual voltage or current as it decays after removal of the initial application.
- Capacitance between DUT pins as the residual voltage or current decays.

All measurements made using the `pulseX` and `measX` commands are processed after the pulse has completed.

Whenever the `pulseX` command is run, either a default or a programmed current or voltage limit is in effect. The type of limit depends on the type of `pulseX` command:

- The `pulsev` command has an automatic current limit default. For example, a SMU used as a voltage source defaults to a current limit of 10 mA. You can call the `limiti` command before the `pulseV` command to override the default.

- The `pulsei` command has an automatic voltage limit default. For example, a SMU used as a current source defaults to a voltage limit of 20 V. You can call the `limitv` command before the `pulsei` command to override the default.

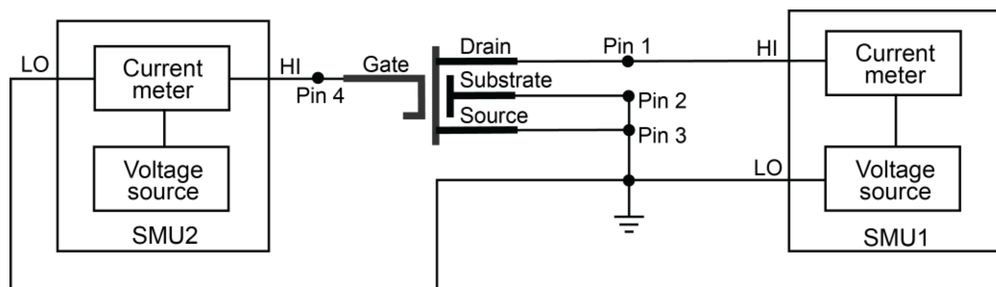
When using the `limitX` and `pulseX` commands on the same source at the same time in a test sequence, call the `limitX` command first, then call the `pulseX` command.

You can use this command in dual-site mode (S535 systems only).

Example

```
float res1, res2;
.
.
conpin(GND, 2, 3, 0);
conpin(SMU1, 1, 0)
conpin(SMU2, 4, 0)
forcev(SMU1, .5)
trigil(SMU1, -1e-5) /* Set the trigger point for -10 mA. */
searchv(SMU2, 0.0, 3.0, 7, 2.0E-5, &res1) /* Increase */
 /* voltage until trigger */
 /* point occurs. Return results to res1. */
pulsev(SMU2, 20.0, 5e-1) /* Apply a 20 V pulse to the */
 /* gate for 500 ms. */
searchv(SMU2, 0.0, 3.0, 7, 2.0E-5, &res2) /* Increase */
 /* voltage until trigger */
 /* point occurs. Return results */
 /* to res2. */
```

This example measures the threshold voltage shift of an FET by calling two `searchv` commands:
1. The `searchv` command measures the gate voltage required to initiate a drain current of 10 µA.
2. The `searchv` command measures the gate voltage required to initiate a drain current of 10 µA immediately after a 20 V pulse is applied to the gate.
Note that the second `searchv` command was called without reprogramming the `trigil` command. This is possible because the clear trigger command (`clrtrg`) was not used.



Also see

None

rangeX

This command selects a range and prevents the selected instrument from autoranging.

Models supported

S530, S535, S540

Usage

```
int rangec(int instr_id, double range);
int rangei(int instr_id, double range);
int rangev(int instr_id, double range);
```

instr_id	The instrument identification code; `SMUn`, `CMTRn`
range	The value of the highest measurement to be made (the most appropriate range for this measurement is selected); if *range* is set to 0, the instrument autoranges

Details

Use the `rangeX` command to eliminate the time required by automatic range selection on a measuring instrument. Because the `rangeX` command prevents autoranging, an overrange condition can occur (for example, when measuring 10 V on a 2 V range). The value 1.0e+22 is returned when this occurs.

The `rangeX` command can also reference a source, because a source-measure unit (SMU) can be either of the following:

- Simultaneously a voltage source, voltmeter, and ammeter.
- Simultaneously a current source, ammeter, and voltmeter.

The range of a SMU is the same for the source and the measure commands.

When selecting a range below the limit value, whether it is explicitly programmed or the default value, an instrument temporarily uses the full-scale value of the range as the limit. This does not change the programmed limit value, and if the instrument range is restored to a value higher than the programmed limit value, the instrument again uses the programmed limit value.

When changing the instrument range, be careful not to overrange the instrument. For example, a test initially performed on the 10 mA range with a 5 mA limit is changed to test in the 1 mA range with a 1 mA limit. Notice that the limit is lowered from 5 mA to 1 mA to avoid overranging the 1 mA setting.

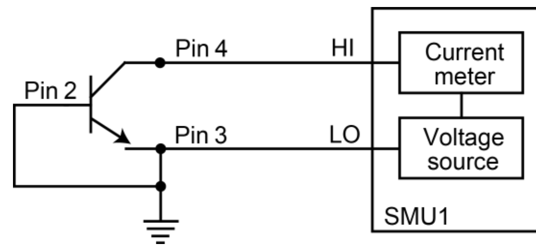
You can use this command in dual-site mode (S535 systems only).

Example

```
double icer2;
.
.
conpin(GND, 3, 2, 0);
conpin(SMU1, 4, 0);
limiti(SMU1, 1.0E-3); /* Limit current to 1.0 mA. */
rangei(SMU1, 2.0E-3); /* Select range for 2 mA. */
forcev(SMU1, 35.0); /* Force 35 V. */
measi(SMU1, &icer2); /* Measure leakage; return */
 /* results to icer2. */
```

This example specifies connections, sets a 1 mA limit on the 2 mA range and forces 35 V, then measures current leakage and returns the results to the variable `icer2`.

Equation 1: SMU measure and source range function



Also see

None

rdelay

This command sets a user-programmable delay.

Models supported

S530, S535, S540

Usage

```
int rdelay(double n);
```

n	The delay duration in seconds

Example

```
double ir4;
.
.
conpin(SMU1, 1, 0);
conpin(GND, 2, 0);
forcev(SMU1, 60.0); /* Generate 60 V from SMU1. */
rdelay(0.02); /* Pause for 20 ms. */
measi(SMU1, &ir4); /* Measure current; return */
 /* result to ir4. */
```

This example measures the leakage current of a variable-capacitance diode. SMU1 presets 60 V across the diode. The device is configured in reverse-bias mode with the high side of SMU1 connected to the cathode. This type of diode has high capacitance and low-leakage current. Because of this, a 20 ms delay is added. After the delay, current through SMU1 is measured and stored in the variable `ir4`.

Also see

delay (on page 2-40)

refctrl

This command enables or disables automatic reference measurements.

Models supported

S530, S535, S540

Usage

```
int refctrl(int instr_id, int auto_ref)
```

instr_id	The instrument identification code
auto_ref	Automatic reference measurement on or off: REF_ON = 1 REF_OFF = 2

Details

Use this command to turn off automatic reference measurements in sweeps and other test sequences in which measurement timing is critical.

NOTE

When automatic reference measurements are disabled, the instrument may gradually drift out of specification.

You can use this command in dual-site mode (S535 systems only).

Example

`int refctrl(SMU1, 2)`
Turn off automatic reference measurements on source-measure unit 1.

Also see

None

rsa_close

This command disconnects communications to the spectrum analyzer.

Models supported

S530, S540 (systems with an RSA306B USB Spectrum Analyzer)

NOTE

In Keithley systems, the RSA306B functions as a replacement for discontinued scope cards. Spectrum analyzer capabilities may be added in the future.

Usage

```
int rsa_close(int instr_id);
```

instr_id	The instrument identification code of the spectrum analyzer

Details

This command returns a `0` if executed without error; a negative number indicates an error.

Example

`rsa_close(RSA1)`
Disconnects communications to spectrum analyzer.

Also see

None

rsa_detect_peaks

This command returns frequencies in signal amplitude order.

Models supported

S530, S540 (systems with an RSA306B USB Spectrum Analyzer)

NOTE

In Keithley systems, the RSA306B functions as a replacement for discontinued scope cards. Spectrum analyzer capabilities may be added in the future.

Usage

```
int rsa_detect_peaks(int instr_id, double min_level, double lower_bound, double
   upper_bound, double *freq_array, int nPeaks, double *amp_array, int nPeak1);
```

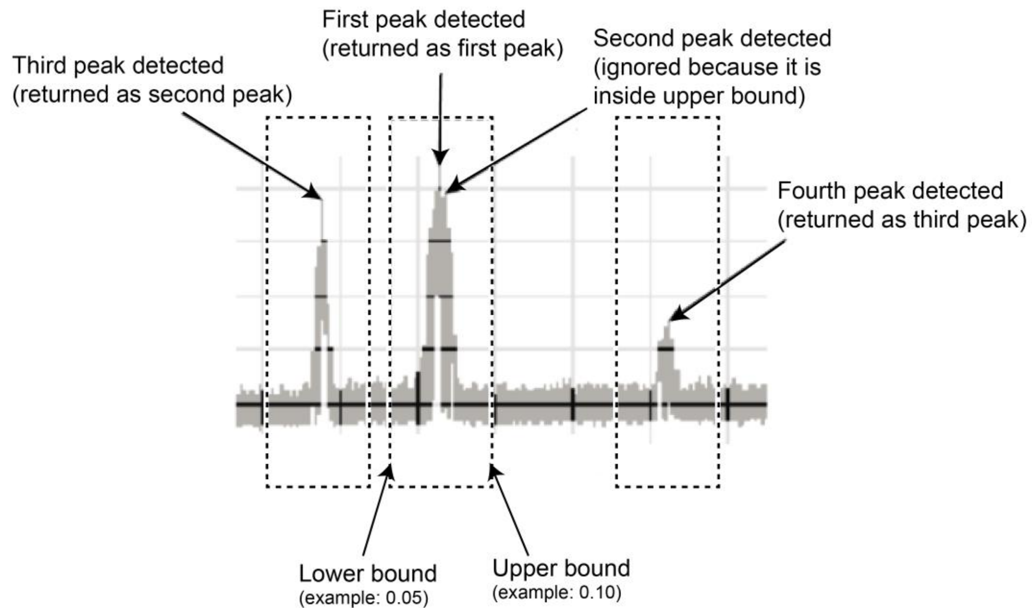
instr_id	The instrument identification code
min_level	The minimum amplitude in decibels; valid values: –50 dB to 20 dB; the peak detected must be higher than this parameter (input)
lower_bound	The measured peak is ignored if the ratio of measured frequencies versus returned peaks is greater than or equal to this parameter; valid values 0.1% (`0.001`) to 100% (`1.000`) (input)
upper_bound	The measured peak is ignored if the ratio of measured frequencies versus returned peaks is less than or equal to this parameter; valid values 0.1% (`0.001`) to 100% (`1.000`) (input)
freq_array	The array output of measured peak frequencies in Hertz (output)
nPeaks	The number of peak frequencies to return (valid values 1 to 5); zero (`0`) is returned to fill the output array when the number of detected peaks is less than this parameter (input)
amp_array	The amplitude of the measured peak frequencies in dB (output)
nPeak1	This parameter must be the same as the *nPeaks* parameter (input)

Details

The spectrum analyzer must be initialized before using this command.

Use this command to return a specified number of frequencies in signal amplitude order. The highest peak is returned first, then the second highest peak is returned, and so on. If multiple peaks are between the upper and lower bounds, the highest peak is returned and the others are ignored (see the following figure).

Figure 1: Upper and lower bounds



If any detected peak frequency is less than or equal to the *min_level* parameter, that peak is dropped and the frequency is returned as zero. If a detected peak frequency is a part of a single-frequency spectrum, the peak is ignored.

Each measured peak frequency has a corresponding measured signal level returned in the `amp_array` parameter.

Example

```
double freq[5];
double ampltd[5];
.
.
.
Status1 = rsa_init(RSA1);
Status2 = rsa_setup(RSA1, 20e6, 850e6, 1e6);
Status3 = rsa_detect_peaks(RSA1, 6e-4, 0.7, 0.4, freq_array, 5, amp_array, 5);
```

This example sets up a spectrum analyzer to scan a 20 MHz to 850 MHz signal in 1 MHz steps and return five detected peak frequencies that are within the specified values of the *lower_bound* and *upper_bound* parameters.

Also see

rsa_init (on page 2-83)
rsa_setup (on page 2-86)

rsa_init

This command initializes the spectrum analyzer to its default state.

Models supported

S530, S540 (systems with an RSA306B USB Spectrum Analyzer)

NOTE

In Keithley systems, the RSA306B functions as a replacement for discontinued scope cards. Spectrum analyzer capabilities may be added in the future.

Usage

`int rsa_init(int instr_id)`

instr_id	The instrument identification code of the spectrum analyzer

Details

The spectrum analyzer must be initialized before using the `rsa_detect_peaks` command.

This command returns a `0` if executed without error; a negative number indicates an error.

Example

```
double freq;
double ampltd;
.
.
.
Status1 = rsa_init(RSA1);
Status2 = rsa_setup(RSA1, 20e6, 850e6, 1e6);
Status3 = rsa_measure(RSA1, freq, ampltd);
```

This example shows how to initialize the spectrum analyzer to its default state.

Also see

rsa_detect_peaks (on page 2-81)

rsa_measure

This command measures the frequency and amplitude of the strongest signal.

Models supported

S530, S540 (systems with an RSA306B USB Spectrum Analyzer)

NOTE

In Keithley systems, the RSA306B functions as a replacement for discontinued scope cards. Spectrum analyzer capabilities may be added in the future.

Usage

```
int rsa_measure(int instr_id, double *freq_result, double *amp_result)
```

instr_id	The instrument identification code of the spectrum analyzer
freq_result	The measured frequency in Hertz (Hz)
amp_result	The measured amplitude in decibels (dB)

Details

The spectrum analyzer must be initialized before using this command.

This command returns a `0` if executed without error; a negative number indicates an error.

Example

```
double freq;
double ampltd;
.
.
.
Status1 = rsa_init(RSA1);
Status2 = rsa_setup(RSA1, 20e6, 850e6, 1e6);
Status3 = rsa_measure(RSA1, freq, ampltd);
```

This example shows how to measure the amplitude of the strongest signal.

Also see

rsa_init (on page 2-83)
rsa_setup (on page 2-86)

rsa_measure_next

This command returns the frequency and amplitude of the next highest peak in the frequency spectrum.

Models supported

S530, S540 (systems with an RSA306B USB Spectrum Analyzer)

NOTE

In Keithley systems, the RSA306B functions as a replacement for discontinued scope cards. Spectrum analyzer capabilities may be added in the future.

Usage

```
int rsa_measure_next(int instr_id, double *freq_result, double *amp_result)
```

instr_id	The instrument identification code of the spectrum analyzer
freq_result	The measured frequency in Hertz (Hz)
amp_result	The measured amplitude in decibels (dB)

Details

The spectrum analyzer must be initialized before using this command.

This command returns a `0` if executed without error; a negative number indicates an error.

Example

```
double freq;
double ampltd;
double freq_next;
double ampltd_next;
.
.
.
Status1 = rsa_init(RSA1);
Status2 = rsa_setup(RSA1, 20e6, 850e6, 1e6);
Status3 = rsa_measure(RSA1, freq, ampltd);
Status4 = rsa_measure_next(RSA1, freq_next, ampltd_next);
```

This example shows how to return the frequency and amplitude of the next highest peak in a frequency spectrum.

Also see

rsa_init (on page 2-83)
rsa_measure (on page 2-83)
rsa_setup (on page 2-86)

rsa_selftest

This command runs the specified RSA306B spectrum analyzer self-test and returns a status for the test.

Models supported

S530 (systems with an RSA306B USB Spectrum Analyzer)

NOTE

In Keithley systems, the RSA306B functions as a replacement for discontinued scope cards. Spectrum analyzer capabilities may be added in the future.

Usage

```
int rsa_selftest(int instr_id, int testname, int *teststatus)
```

instr_id	The instrument identification code of the spectrum analyzer; valid value RSA1
testname	The name of a specific test to run; valid values are: ■ RSA_ST_SV = Supply voltage ■ RSA_ST_TEMP = Temperature sensor ■ RSA_ST_AD_SPI = A/D SPI ■ RSA_ST_LO1_TUN = LO1 tuning ■ RSA_ST_LO2_TUN = LO2 tuning ■ RSA_ST_PREAMP = Preamplifier ■ RSA_ST_RF_ATT = RF attenuator ■ RSA_ST_ME_AMP = ME amplifier
teststatus	The status of the specified test: 0 = Test failed to complete 1 = Test completed successfully -1 = Not supported

Details

The spectrum analyzer must be initialized before using this command.

This command allows you to run specific self-tests on the RSA306B and get the success-failure status of the test.

Example

```
status = rsa_selftest(RSA1, RSA_ST_SV, teststatus)
```

In the Keithley Interactive Test Tool (KITT), runs a self-test of the spectrum analyzer (RSA1) supply voltage and returns a success (1) or failure (0) status.

Also see

rsa_init (on page 2-83)

rsa_setup

This command sets the start, stop, and step (scan resolution) frequencies of a scan.

Models supported

S530 (systems with an RSA306B USB Spectrum Analyzer)

NOTE

In Keithley systems, the RSA306B functions as a replacement for discontinued scope cards. Spectrum analyzer capabilities may be added in the future.

Usage

```
int rsa_setup(int instr_id, double min_freq, double max_freq, double res_bandwidth)
```

instr_id	The instrument identification code
min_freq	Start frequency of the scan in Hertz (Hz)
max_freq	Stop frequency of the scan in Hz
res_bandwidth	Step size (scan resolution in Hz): `1e6`, `3e5`, `1e5`, `3e4`, `1e4`, `3e3`, or `1000`

Details

The spectrum analyzer must be initialized before using this command.

The recommended relationship of the `res_bandwidth` parameter for different spans of a scan is listed in the following table.

Relationship of res_bandwidth and span

Frequency span	res_bandwidth
60 MHz ≤ Span	1 MHz
20 MHz ≤ Span ≤ 60 MHz	300 kHz
6 MHz ≤ Span ≤ 20 MHz	100 kHz
2 MHz ≤ Span ≤ 6 MHz	30 kHz
300 kHz ≤ Span ≤ 2 MHz	10 kHz
100 kHz ≤ Span ≤ 300 kHz	3 kHz

This command returns a `0` if executed without error; a negative number indicates an error.

Example

```
double freq;
double ampltd;
.
.
.
Status1 = rsa_init(RSA1);
Status2 = rsa_setup(RSA1, 20e6, 850e6, 1e6);
Status3 = rsa_measure(RSA1, freq, ampltd);
```

This example sets the start, stop, and step frequencies of a scan.

Also see

rsa_init (on page 2-83)
rsa_measure (on page 2-83)

rtfary

This command returns the force array determined by the instrument action.

Models supported

S530, S535, S540

Usage

```
int rtfary(double *results);
```

results	The floating point array where the force values are stored

Details

This command eliminates the need to calculate the forced array in the application.

When this command is used with one of the sweep routines, you can determine the exact forced value for each point in the sweep.

When the test sequence is executed, the sweep command initiates the first step of the voltage or current sweep. The sweep then logs the force point that the buffer specified by the `rtfary` command.

Locate the `rtfary` command before the sweep. The number of data points returned by the `rtfary` command is determined by the number of force points generated by the sweep.

This command returns dual-site data for SITE0 and SITE1 if available (S535 systems only).

Also see

smeasX (on page 2-114)
sweepX (on page 2-118)

rttrigary

This command returns the measured values from a trigger operation

Models supported

S530, S535, S540

Usage

```
int rttrigary(double *results);
```

results	The floating point array where the measured values are stored

Details

When used with the `bsweepX` or `sweepX` commands, this command allows you to view the data results from the measure command used for a trigger.

Using this command with the `bsweepX` command allows you to determine the measured value for each point in the sweep.

When the test sequence is executed, the sweep command initiates the first step of the voltage or current sweep. The sweep then logs the measure point in the array specified by the `rttrigary` command.

Place the `rttrigary` command before the sweep. The number of data points returned by the `rttrigary` command is determined by the number of measure points generated by the sweep.

This command returns dual-site data for SITE0 and SITE1 if available (S535 systems only).

Also see

bsweepX (on page 2-32)
sweepX (on page 2-118)

savgX

This command makes an averaging measurement for every point in a sweep.

Models supported

S530, S535, S540

Usage

```
int savgi(int instr_id, double *result, unsigned int count, double delay);
int savgv(int instr_id, double *result, unsigned int count, double delay);
int savgc(int instr_id, double *results, unsigned int count, double delay);
int savgg(int instr_id, double *results, unsigned int count, double delay);
```

instr_id	The instrument identification code of the measuring instrument; `SMUn`, `CMTRn`, `VMTRn`
result	The floating point array where the results are stored
count	The number of measurements made at each point before the average is computed
delay	The time delay in seconds between each measurement within a given ramp step

Details

This command creates an entry in the measurement scan table. During any of the sweeping commands, a measurement scan is done for every force point in the sweep. During each scan, a measurement is made for every entry in the scan table. The measurements are made in the same order in which the entries were made in the scan table.

The `savgX` command sets up the new scan table entry to make an averaging measurement. The measurement results are stored in the array specified by the *result* parameter. The scan table is cleared by an explicit call to the `clrscn` command or implicitly when the `devint` command is called.

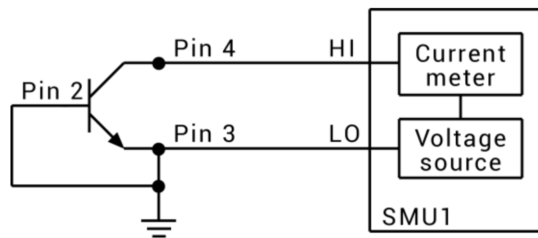
When making each averaged measurement, the number of actual measurements specified by the *count* parameter is made on the instrument at the interval specified by the *delay* parameter, and then the average is calculated. This average is the value that is stored in the results array.

The `savgc` and `savgg` commands return dual-site data for SITE0 and SITE1 if available (S535 systems only).

Example

```
double res1[26];
.
.
conpin(GND, 3, 2, 0);
conpin(SMU1, 4, 0);
savgi(SMU1, res1, 8, 1.0E-3); /* Measure average */
 /* current 8 times per */
 /* sample; return results to */
 /* res1 array. */
sweepv(SMU1, 0.0, -50.0, 25, 2.0E-2); /* Generate */
    /* a voltage from 0 V */
 /* to -50 V over 25 steps.*/
```

This example gets the measurement data that is needed to create a graph that shows the capacitance versus voltage characteristics of a variable-capacitance diode. This diode is operated in reverse-biased mode. SMU1 outputs a voltage that sweeps from 0 through -50 V. Capacitance is measured 26 times during the sweep. The results are stored in an array called `res1`.



Also see

clrscn (on page 2-34)
devint (on page 2-42)

scp_close

This command disconnects communications to the scope card.

Models supported

S530, S540 (systems with a Model 4200-SCP2HR scope card)

Usage

`int scp_close(int instr_id);`

instr_id	The instrument identification code of the scope card

Details

This command returns a `0` if executed without error; a negative number indicates an error.

Example

```
scp_close(SCP1)
```

Disconnects communications to scope card 1.

Also see

None

scp_detect_peaks

This command returns frequencies in signal amplitude order.

Models supported

S530, S540 (systems with a Model 4200-SCP2HR scope card)

Usage

```
int scp_detect_peaks(int instr_id, double min_level, double lower_bound, double
   upper_bound, double *freq_array, int nPeaks, double *amp_array, int nPeak1);
```

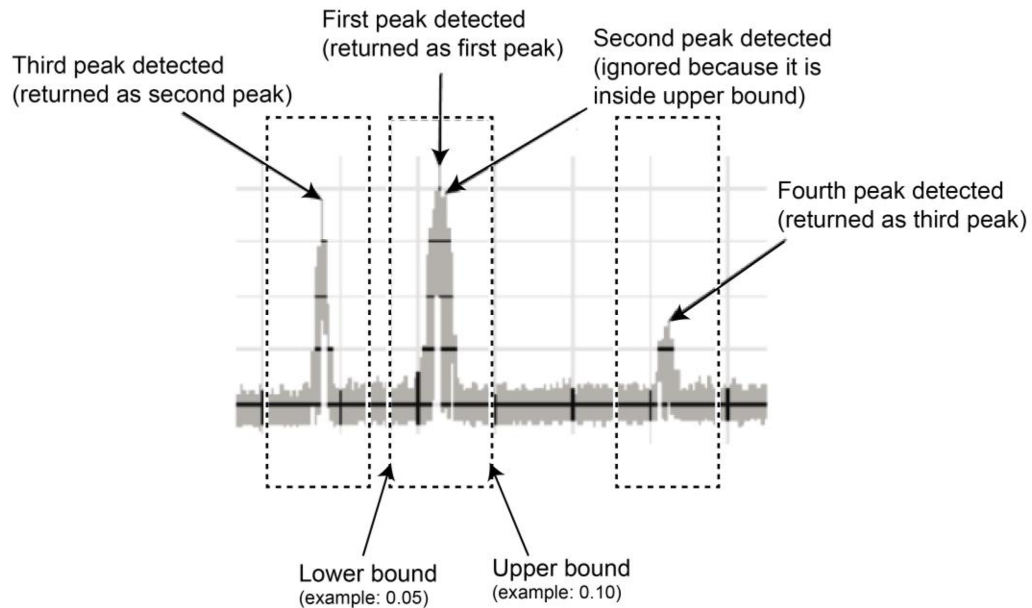
instr_id	The instrument identification code
min_level	The minimum amplitude in decibels; valid values: –50 dB to 20 dB; the peak detected must be higher than this parameter (input)
lower_bound	The measured peak is ignored if the ratio of measured frequencies versus returned peaks is greater than or equal to this parameter; valid values 0.1% (`0.001`) to 100% (`1.000`) (input)
upper_bound	The measured peak is ignored if the ratio of measured frequencies versus returned peaks is less than or equal to this parameter; valid values 0.1% (`0.001`) to 100% (`1.000`) (input)
freq_array	The array output of measured peak frequencies in Hertz (output)
nPeaks	The number of peak frequencies to return (valid values 1 to 5); zero (`0`) is returned to fill the output array when the number of detected peaks is less than this parameter (input)
amp_array	The amplitude of the measured peak frequencies in dB (output)
nPeak1	This parameter must be the same as the *nPeaks* parameter (input)

Details

The scope card must be initialized before using this command.

Use this command to return a specified number of frequencies in signal amplitude order. The highest peak is returned first, then the second highest peak is returned, and so on. If multiple peaks are between the upper and lower bounds, the highest peak is returned and the others are ignored (see the following figure).

Figure 2: Upper and lower bounds



If any detected peak frequency is less than or equal to the *min_level* parameter, that peak is dropped and the frequency is returned as zero. If a detected peak frequency is a part of a single-frequency spectrum, the peak is ignored.

Each measured peak frequency has a corresponding measured signal level returned in the `amp_array` parameter.

Example

```
double freq[5];
double ampltd[5];
.
.
.
Status1 = scp_init(SCP1);
Status2 = scp_setup(SCP1, 20e6, 850e6, 1e6);
Status3 = scp_detect_peaks(SCP1, 6e-4, 0.7, 1.4, freq_array, 5, amp_array, 5);
```

This example sets up a spectrum analyzer to scan a 20 MHz to 850 MHz signal in 1 MHz steps and return five detected peak frequencies that are within the specified values of the *lower_bound* and *upper_bound* parameters.

Also see

scp_init (on page 2-93)
scp_setup (on page 2-96)

scp_init

This command initializes the scope card to its default state.

Models supported

S530, S540 (systems with a Model 4200-SCP2HR scope card)

Usage

`int scp_init(int instr_id)`

instr_id	The instrument identification code of the scope card

Details

The scope card must be initialized before using the `scp_detect_peaks` command.

This command returns a `0` if executed without error; a negative number indicates an error.

Example

```
double freq;
double ampltd;
.
.
.
Status1 = scp_init(SCP1);
Status2 = scp_setup(SCP1, 20e6, 850e6, 1e6);
Status3 = scp_measure(SCP1, freq, ampltd);
```

This example shows how to initialize a scope card to its default state.

Also see

scp_detect_peaks (on page 2-91)

scp_measure

This command measures the frequency and amplitude of the strongest signal.

Models supported

S530, S540 (systems with a Model 4200-SCP2HR scope card)

Usage

`int scp_measure(int instr_id, double *freq_result, double *amp_result)`

instr_id	The instrument identification code of the scope card
freq_result	The measured frequency in Hertz (Hz)
amp_result	The measured amplitude in decibels (dB)

Details

The scope card must be initialized before using this command.

This command returns a `0` if executed without error; a negative number indicates an error.

Example

```
double freq;
double ampltd;
.
.
.
Status1 = scp_init(SCP1);
Status2 = scp_setup(SCP1, 20e6, 850e6, 1e6);
Status3 = scp_measure(SCP1, freq, ampltd);
```

This example shows how to measure the amplitude of the strongest signal.

Also see

scp_init (on page 2-93)
scp_setup (on page 2-96)

scp_measure_next

This command returns the frequency and amplitude of the next highest peak in the frequency spectrum.

Models supported

S530, S540 (systems with a Model 4200-SCP2HR scope card)

Usage

```
int scp_measure_next(int instr_id, double *freq_result, double *amp_result)
```

instr_id	The instrument identification code of the scope card
freq_result	The measured frequency in Hertz (Hz)
amp_result	The measured amplitude in decibels (dB)

Details

The scope card must be initialized before using this command.

This command returns a `0` if executed without error; a negative number indicates an error.

Example

```
double freq;
double ampltd;
double freq_next;
double ampltd_next;
.
.
.
Status1 = scp_init(SCP1);
Status2 = scp_setup(SCP1, 20e6, 850e6, 1e6);
Status3 = scp_measure(SCP1, freq, ampltd);
Status4 = scp_measure_next(SCP1, freq_next, ampltd_next);
```

This example shows how to return the frequency and amplitude of the next highest peak in a frequency spectrum.

Also see

scp_init (on page 2-93)
scp_measure (on page 2-93)
scp_setup (on page 2-96)

scp_selftest

This command does an internal self-test of the scope card.

Models supported

S530, S540 (systems with a Model 4200-SCP2HR scope card)

Usage

```
int scp_selftest(int instr_id)
```

instr_id	The instrument identification code of the scope card

Details

The scope card must be initialized before using this command.

This command returns a `0` if executed without error; a negative number indicates an error.

Example

```
Status1 = scp_selftest(SCP2);
```

Does a self-test of scope card 2 (SCP2).

Also see

scp_init (on page 2-93)

scp_setup

This command sets the start, stop, and step (scan resolution) frequencies of a scan.

Models supported

S530, S540 (systems with a Model 4200-SCP2HR scope card)

Usage

```
int scp_setup(int instr_id, double min_freq, double max_freq, double res_bandwidth)
```

instr_id	The instrument identification code
min_freq	Start frequency of the scan in Hertz (Hz)
max_freq	Stop frequency of the scan in Hz
res_bandwidth	Step size (scan resolution in Hz): `1e6`, `3e5`, `1e5`, `3e4`, `1e4`, `3e3`, or `1000`

Details

The scope card must be initialized before using this command.

The recommended relationship of the `res_bandwidth` parameter for different spans of a scan is listed in the following table.

Relationship of res_bandwidth and span

Frequency span	res_bandwidth
60 MHz ≤ Span	1 MHz
20 MHz ≤ Span ≤ 60 MHz	300 kHz
6 MHz ≤ Span ≤ 20 MHz	100 kHz
2 MHz ≤ Span ≤ 6 MHz	30 kHz
300 kHz ≤ Span ≤ 2 MHz	10 kHz
100 kHz ≤ Span ≤ 300 kHz	3000 Hz
50 kHz ≤ Span ≤ 100 kHz	1000 Hz

This command returns a `0` if executed without error; a negative number indicates an error.

Example

```
double freq;
double ampltd;
.
.
.
Status1 = scp_init(SCP1);
Status2 = scp_setup(SCP1, 20e6, 850e6, 1e6);
Status3 = scp_measure(SCP1, freq, ampltd);
```

This example sets the start, stop, and step frequencies of a scan.

Also see

scp_init (on page 2-93)
scp_measure (on page 2-93)

searchX

This command is used to determine the voltage or current required to get a current, voltage, capacitance, or conductance. It is useful in finding initial threshold points such as junction breakdown or transistor turn on.

Models supported

S530, S535, S540

Usage

```
int searchi(int instr_id, double min_val, double max_val, unsigned int iterate_no,
    double iterate_time, double *result);
int searchv(int instr_id, double min_val, double max_val, unsigned int iterate_no,
    double iterate_time, double *result);
```

instr_id	The instrument identification code of the sourcing instrument; `SMUn` or `CMTRn`
min_val	The lower limit of the source range
max_val	The upper limit of the source range
iterate_no	The number of separate current or voltage levels to generate; the range of iterations is from 1 through 16
iterate_time	The duration, in seconds, of each iteration
result	The floating point variable assigned to the search operation result; it represents the voltage, with the `searchv` command, or current, with the `searchi` command, applied during the last search operation

Details

The `trigXg` or `trigXl` command must be used with the `searchX` command. Triggers and the `searchX` command together initiate a search operation consisting of a series of steps referred to as iterations. During each iteration, the following events occur:

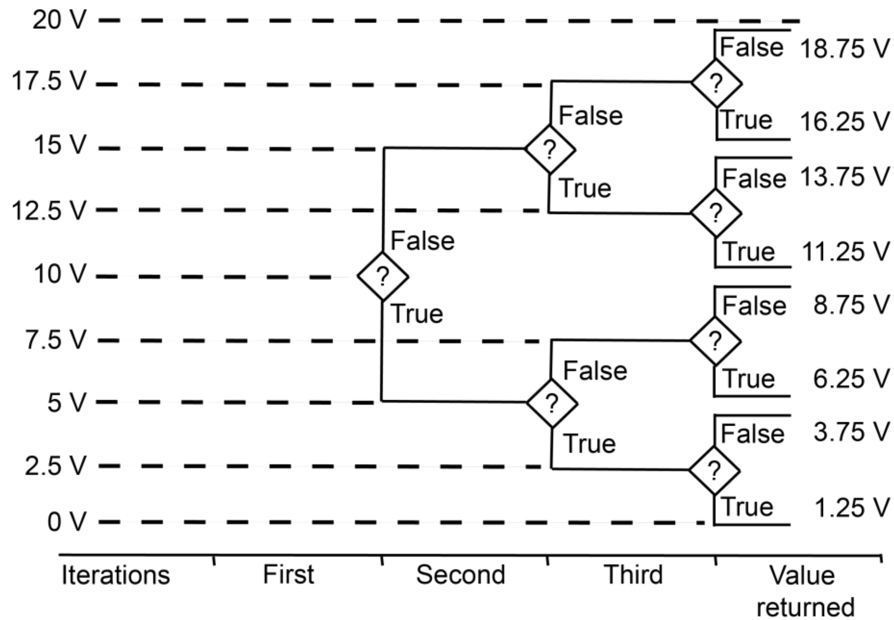
- A voltage or current is applied to a circuit node of the device under test (DUT).
- All triggers are evaluated.
- If the triggers evaluate true, the source value is moved toward the value specified in the *min_val* parameter. If the triggers do not evaluate true, the source value is moved toward the value specified in the *max_val* parameter. The source range is then divided in half for the next iteration.

A total of 16 iterations can be programmed. When all iterations are completed, a value of voltage or current is returned as the result of the search operation. This value is the voltage or current level required to match the trigger point.

The following example shows all binary search possibilities where the minimum and maximum source values are 0 and 20 V, respectively. Note the following:

- Three iterations, numbered one through three, are shown. Within a given iteration, the values of possible sourcing voltages are indicated.
- During the first iteration of the binary search process, 10 V is applied. This represents the midpoint of the minimum and maximum values.
- At the end of each iteration, the program determines whether to increase or decrease the source voltage. The determination is dependent on the evaluation of the trigger point.

Figure 3: Minimum and maximum source values



The question mark (?) is the true or false determination.

As shown in the above figure, the true or false decision determines the voltage generated in the next step of the binary progression.

Because the command initiates a current or voltage from a source, its placement in a test sequence is critical. Therefore:

- Call the `limitX` and `rangeX` commands before the `searchX` command when all three refer to the same instrument.
- Call the `trigXg` or `trigXl` command before the `searchX` command.

The search operation determines the source voltage or current required at one circuit node to generate a trigger point value at a second node. The resolution of the result depends on the number of iterations or steps and the actual current or voltage range used by the instrument.

$$\frac{\text{voltage or current range}}{2^{(\text{iteration}+1)}}$$

For example, assume the minimum and maximum values of the source range are from 0 V to 20 V, and the number of iterations is 16. The 20 V level automatically initiates a source-measure unit (SMU) 20 V source range. As a result, the resolution of the final source voltage returned is:

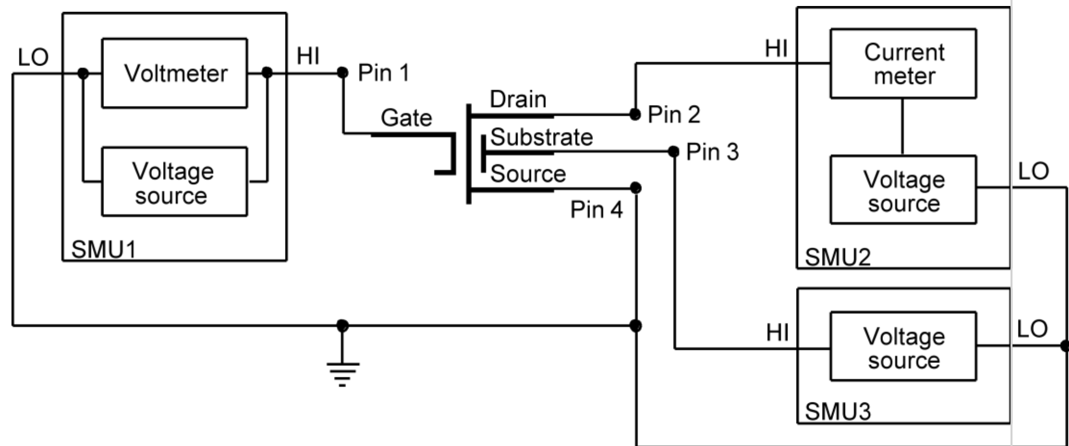
$$\frac{20}{2^{(16+1)}} = 1.2\text{ mV}$$

This command is not supported in dual-site mode (S535 systems only).

Example

```
double ssbiasv, vgs1, vds1;
.
conpin(SMU1, 1, 0);
conpin(SMU2, 2, 0);
conpin(SMU3, 3, 0);
conpin(GND, 4, 0);
trigig(SMU2, +1.0E-6); /* Set trigger point for 1 uA. */
forcev(SMU3, ssbiasv); /* Apply a substrate bias */
 /* voltage ssbiasv. */
forcev(SMU2, vds1); /* Apply a drain voltage of */
 /* vds1. */
searchv(SMU1, 0.6, 1.7, 8, 1.0E-3, &vgs1); /* Set */
 /* for 8 steps from 0.6 to */
 /* 1.7 V at 1 ms.*/
 /* per iteration; return the */
 /* result to vgs1. */
```

This example searches for the gate voltage required to generate a drain current of 1 µA. Eight separate gate voltages within the range of 0.6 V through 1.7 V are specified by the `searchv` command. After the eight iterations complete, the drain current is close to 1 µA, and the `searchv` operation is terminated. The gate voltage generated at this time by SMU1 is returned in the variable `vgs1`.



Also see

None

setauto

This command re-enables autoranging and cancels any previous `rangeX` command for the specified instrument.

Models supported

S530, S535, S540

Usage

`int setauto(int instr_id);`

instr_id	The instrument identification code

Details

Due to the dual mode operation of the SMU (v versus i), `setauto` places both voltage and current ranges in autorange mode.

The specific range is not changed until a measurement is made. When a measurement is made, the autorange software evaluates the data and changes the range if necessary.

You can use this command in dual-site mode (S535 systems only).

Example

```
.
rangei(SMU1, 2.0E-9); /* Select manual range. */
delay(200); /* Delay after range change. */
measi(SMU1, &icer1);       /* Measure leakage; return. */
.
.
setauto(SMU1); /* Enable autorange mode. */
lorangei(SMU1, 2.0E-6); /* Select 2 uA as minimum range */
 /* during autoranging. */
delay(200); /* Delay after range change. */
smeasi(SMU1, idatvg); /* Setup sweep measurement of IDS. */
sweepv(SMU2, 0.0, 2.5, 24, 0.002);      /* Sweep gate from 0 V to 2.5 V. */
```

Also see

None

setmode

This command sets instrument-specific operating mode parameters.

Models supported

S530, S535, S540

Usage

`int setmode(int instr_id, long modifier, double value);`

instr_id	The instrument identification code of the instrument being operated on
modifier	The instrument-specific operating characteristic to change; refer to setmode modifier tables (on page 2-102)
value	The value of the *modifier* parameter

Details

The `setmode` command allows you to control certain instrument-specific operating characteristics.

A special instrument ID named `KI_SYSTEM` sets the setmode modifier for all instruments that support the specified setmode.

All modifiers listed in the setmode modifier tables (on page 2-102) can be used with the `KI_SYSTEM` pseudo-instrument.

You can use this command in dual-site mode (S535 systems only).

Example 1

```
double ic[10];
double vb[10];

conpin(SMU1, 1, 0);
conpin(SMU2, 2, 0);
conpin(GND, 3, 0);

setmode(SMU1, KI_INTGPLC, 0.025);

forcev(SMU2, 5.0);
sintgv(SMU1, vb);
smeasi(SMU2, ic);
sweepi(SMU1, 0.0, 1.0e-6, 9, 0.0);
```

The `setmode` command in this example specifies the period (0.025 AC line cycles) over which measurements on SMU1 are averaged.

Example 2

```
setmode(KI_SYTEM, KI_MAX_VOLTAGE, 2000);
```

S540 system only: Sets the maximum voltage for all 2657A SMUs in the system to 2000 V.

Also see

setmode modifier tables (on page 2-102)

setmode modifier tables

Modifiers that affect the system		
Modifier	**Value**	**Comment**
KI_MAX_VOLTAGE	S540 system, 2657A SMUs only: *value* Where: *value* = 300 to 3000	Sets the maximum voltage for all 2657A SMUs in the S540 system. The default value is 3000 V. Calling the devint command resets this value to the MAX_HV value defined in the icconfig_<QMO>.ini file. This setmode only affects the 2657A SMU; it does nothing on other models.
KI_SYSTEM_SPEED_MODE	KI_SYSTEM_SPEED_CUSTOM Deprecated: KI_SYSTEM_SPEED_LEGACY	CUSTOM mode allows you to specify the default speed mode. This setting is reset to the mode defined in the icconfig_<QMO>.ini file with a call to the devint command. **NOTE:** LEGACY mode was deprecated in KTE version 5.8.0. If you use KTE version 5.8.0 or later, the LEGACY setting is ignored and the default system settings are used. In KTE versions later than 5.6.0 but earlier than 5.8.0, LEGACY mode could be used if you had problems with correlation of data using the default mode.

Modifiers that affect source-measure unit (SMU) behavior		
Modifier	**Value**	**Comment**
KI_AVGMODE	26xx and 2461-SYS: KI_INTEGRATE KI_MEASX	Controls the behavior of the avg*X* command call. KI_INTEGRATE replaces the avg*X* command with the intg*X* command. KI_MEASX replaces the avg*X* command with the meas*X* command.
KI_HIGHC_MODE	26xx and 2461-SYS: KI_ON and KI_OFF	KI_OFF = Disable High C mode KI_ON = Enable High C mode
KI_IMTR		Sets up a source-measure unit (SMU) as a current meter. The ranges used are representative of the type of instrument being simulated. Note that this setmode turns on the source.
	KI_S400	Sets a SMU to use ranges equivalent to the S400.
	KI_DMM	Sets a SMU to use ranges equivalent to a DMM. Provides a lower resolution, fast measurement. Used for high-current applications.
	KI_ELECTROMETER	Sets a SMU to use ranges equivalent to an electrometer. Provides the best measurement resolution but slower measurement time. Used for low-current applications.
KI_INTGPLC	CMTR: 0.006 to 10.002 24xx: 0.01 to 10 (S530 only) 26xx: 0.001 to 25 2461-SYS: 0.01 to 10	Defines a period in terms of AC line cycles over which measurements are averaged. Note that the default NPLC value differs when using certain commands. See "Using NPLCs to adjust speed and accuracy" in the reference manual for your system.
KI_LIM_INDCTR	Any number	Controls what measure value is returned if the SMU is at its programmed limit. The devint command default is SOURCE_LIMIT (7.0e22). Note that the SMU always returns INST_OVERRANGE (7.0e22) if it is on a fixed range that is too low for the signal being measured.

Modifiers that affect source-measure unit (SMU) behavior (continued)		
Modifier	**Value**	**Comment**
KI_LIM_MODE	KI_INDICATOR KI_VALUE	Controls whether the SMU returns an indicator value when in limit or overrange, or the actual value measured. The default mode after a devint command is to return a value.
KI_MEAS_DELAY	26xx (2461-SYS not supported): KI_DELAY_OFF KI_DELAY_AUTO *value*	Controls measurement delays on a SMU. You can choose to turn the delay off, have the instrument set the delay, or specify a value in seconds. KI_DELAY_AUTO is the default setting. Note that reducing measure delays may cause the SMU to return unsettled or inaccurate readings.
KI_MEAS_DELAY_FACTOR	26xx (2461-SYS not supported): *value*	When the KI_MEAS_DELAY modifier is set to KI_DELAY_AUTO, specifying this modifier multiplies the delay values that are stored in the 26xx SMU by the specified value.
KI_SENSE	26xx and 2461-SYS: KI_SENSE_LOCAL or 0 KI_SENSE_REMOTE or 1	Controls the sense mode of the 26xx SMU (2-wire or 4-wire mode). 0 = Local sense mode (2-wire) 1 = Remote sense mode (4-wire)
KI_SETTLE_MODE	26xx (2461-SYS not supported):	Changes the source settling mode.
	KI_SETTLE_SMOOTH	Turns off additional settling operations.
	KI_SETTLE_FAST_RANGE	Sets the SMU to use a faster procedure when changing ranges
	KI_SETTLE_FAST_POLARITY	Sets the SMU to change polarity without going to zero.
	KI_SETTLE_DIRECT_IRANGE	Default setting; sets the SMU to change the current range directly.
	KI_SETTLE_FAST_ALL	Enables all settle fast modes.

Modifiers that affect source-measure unit (SMU) behavior (continued)		
Modifier	**Value**	**Comment**
KI_VMTR		Sets up a SMU as a voltmeter. The ranges used are representative of the type of instrument being simulated. Note that this setmode turns on the source.
	KI_S400	Sets a SMU to use ranges equivalent to the S400.
	KI_DMM	Sets a SMU to use ranges equivalent to a DMM. Provides a low impedance, fast measurements. Used for low-voltage applications.
	KI_ELECTROMETER	Sets a SMU to use ranges equivalent to an electrometer. Provides high-input impedance but has slower measurement time. Used for high-resistance measurements.
KI_VMTR_FUNC	DMM7510 KI_VMTR_FUNC_ACV KI_VMTR_FUNC_DCV	Sets up the DMM7510 as a voltmeter using the DCV or ACV measure function. The default value is KI_VMTR_FUNC_DCV.
KI_2600_ANALOG_FILTER	26xx (2461-SYS not supported): KI_ON KI_OFF	Turns the analog filter on or off for a specified SMU.

Modifiers that affect triggering		
Modifier	**Value**	**Comment**
KI_AVGNUMBER	4200A and 2410 *value*	Number of readings to make when KI_TRIGMODE is set to KI_AVERAGE. **NOTE:** This setmode modifier is valid for the 4200A and 2410 only. If used with any other model, you will get a "NO-OP" error.
KI_AVGTIME	*value*	Time between readings when KI_TRIGMODE is set to KI_AVERAGE. **NOTE:** This setmode modifier is valid for the 4200A and 2410 only. If used with any other model, you will get a "NO-OP" error.

Modifiers that affect triggering		
Modifier	**Value**	**Comment**
KI_TRIGMODE	4200A and 2410: KI_MEASX KI_INTEGRATE KI_AVERAGE	Redefines all existing triggers to use a new method of measurement. **NOTE:** These setmode modifier values are valid for the 4200A and 2410 only. If used with any other model, you will get a "NO-OP" error.
	KI_ABSOLUTE KI_NORMAL	Use absolute value or polarized readings for trigger condition.

Modifiers that affect CVU measurements		
Modifier	**Value**	**Comment**
KI_CVU_ACV	*value*	10 mV to 100 mV (45 mV is default)
KI_CVU_ACZ_RANGE	*value*	0 = Autorange (default) 1e-6 = 1 µA range 30e-6 = 30 µA range 1e-3 = 1 mA range
KI_CVU_CORRECT LOAD	0 or 1	0 = Off (default) 1 = On
KI_CVU_CORRECT_OPEN	0 or 1	0 = Off (default) 1 = On
KI_CVU_CORRECT_SHORT	0 or 1	0 = Off (default) 1 = On
KI_CVU_FREQ	*value*	1 kHz to 2 MHz (100 kHz is default)
KI_CVU_LENGTH	KI_CVU_CABLE_CORR_LV KI_CVU_CABLE_CORR_HV KI_CVU_CABLE_CORR_LV_HV	Specifies the type of cables used in the system so that cable-length corrections can be automated.
KI_CVU_MODE	0 or 1	0 = User mode 1 = System mode (default)
KI_CVU_MODEL	0 to 5	0 = Z, theta 1 = R, jx 2 = Cp, Gp (default) 3 = Cs, Rs 4 = Cp, D 5 = Cs, D
KI_CVU_SPEED	0 to 2	0 = Fast (default) 1 = Normal 2 = Quiet Does not affect the intgc, intgg, or intgcg LPT commands. To adjust the speed of these LPT commands, use the KI_INTGPLC setmode.

Modifiers that affect matrix operation		
Modifier	**Value**	**Comment**
KI_AUTO_GND_PINS	70xB: 0 or 1	Enable or disable the feature that automatically connects all unused pins to GND (including the CHUCK). 0 = Disable 1 = Enable

setXmtr

This command allows a source to operate as a voltmeter or current meter. The source function is disabled after calling the `setXmtr` command.

Models supported

S530, S535, S540

Usage

```
int setimtr(int instr_id);
int setvmtr(int instr_id);
```

instr_id	The instrument identification code of the instrument to control; `SMUn`

Details

Use *x* = `v` for volts and `i` for current.

The `setXmtr` command does not affect any existing connections.

Note that the `setvmtr` command operates as a current source with 0.0 A output. It also activates the voltmeter function of the instrument. Additionally, the `setimtr` command operates as a voltage source with 0.0 V output; it also activates the current meter function of the instrument.

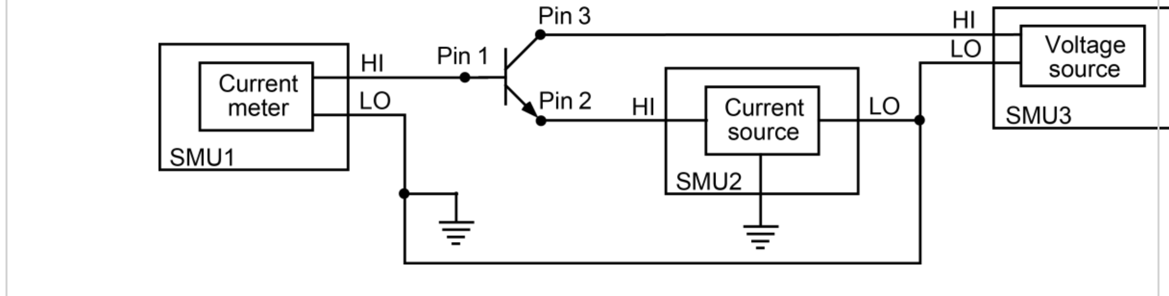
The effects of the `setXmtr` command are also cleared when a `devint` or `device initialize` command is called at the end of the test sequence.

You can use this command in dual-site mode (S535 systems only).

Example

```
float vcc12, icc8, ib47;
.
.
conpin(SMU1H, 1, 0);
conpin(SMU2H, 2, 0);
conpin(SMU3H, 3, 0);
setimtr(SMU1); /* Set SMU1 as a current meter only. */
forcev(SMU3, vcc12); /* Apply vcc12V to collector. */
forcei(SMU2, icc8); /* Enable icc8 current through emitter. */
measi(SMU1, &ib47); /* Measure base current return result */
 /* to ib47. */
```

This figure shows a transistor beta measurement at a specified emitter current and collector-base voltage.



Also see

devint (on page 2-42)

sintgX

This command makes an integrated measurement for every point in a sweep.

Models supported

S530, S535, S540

Usage

```
int sintgi(int instr_id, double *result);
int sintgv(int instr_id, double *result);
int sintgc(int instr_id, double *result);
int sintgg(int instr_id, double *result);
```

instr_id	The instrument identification code of the measuring instrument; SMU*n*, VMTR*n*, CMTR*n*
result	The floating point array where the results are stored

Details

Use this command to create an entry in the measurement scan table. During any of the sweeping commands, a measurement scan is performed for every force point in the sweep. During each scan, a measurement is made for every entry in the scan table. The measurements are made in the same order in which the entries were made in the scan table.

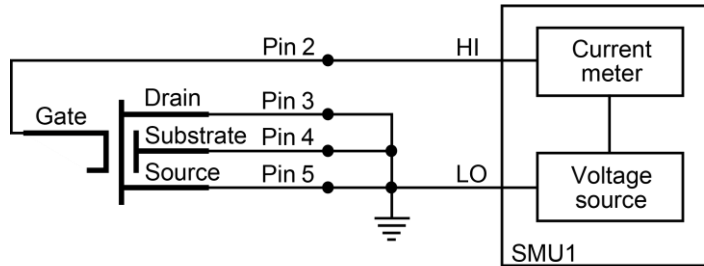
The `sintgX` command sets up the new scan table entry to make an integrated measurement. The scan table is cleared by an explicit call to the `clrscn` command or implicitly when the `devint` command is called.

The `sintgi` and `sintgv` commands return dual-site data for SITE0 and SITE1 if available (S535 systems only).

Example

```
float idss[16];
.
.
conpin(SMU1, 2, 0);
conpin(GND, 5, 4, 3, 0);
limiti(SMU1, 1.5E-8);
rangei(SMU1, 2.0E-8); /* Select range for 20 nA. */
sintgi(SMU1, idss); /* Measure current with SMU1;*/
 /* return results to idss. */
.
.
sweepv(SMU1, 0.0, 25.0, 15, /* Perform 16 measurements */
 1.0E-3); /* (steps) from 0 through */
. /* 25 V; each step 1 ms in */
. /* duration. */
```

This example collects information on the low-level gate leakage current of a metal-oxide field-effect transistor (MOSFET). Sixteen integrated measurements are made as the voltage is increased from 0 V to 25 V.



Also see

clrscn (on page 2-34)
devint (on page 2-42)
sweepX (on page 2-118)

site_disable

This command disables the specified site ID.

Models supported

S535

Usage

`int site_disable(int siteid);`

siteid	Specifies a site to disable (input); valid values: `KI_SITE0` = Disable the anchor site `KI_SITE1` = Disable the mirror site

Details

Set *siteid* to `KI_SITE0` to specify the anchor site and `KI_SITE1` to specify the mirror site. Any measurements for disabled sites return `1.138E26`, a unique value indicating `SITE_INACTIVE`.

Dual-site settings specified by the `site_enable` and `site_disable` commands are not reset by the `devint` command.

When dual-site mode is disabled, this command returns error code `KTE_FEATURE_DISABLED`.

NOTE

Calls to this command are built into the Keithley Test Execution Engine (KTXE) if you specify this parameter in the wafer description file (`.wdf`).

To enable a site, see site_enable (on page 2-111).

For more information about using dual-site mode, see "Dual-site operation" in the *S535 Reference Manual* (part number S535-901-01).

Example

```
site_enable(KI_SITE0);
site_disable(KI_SITE1);
conpin(SMU1, 1);
conpin(GND, 2);
forcev(SMU1, 10.0);
measi(SMU1, &i[0]);
```

This example enables the anchor site and disables the mirror site, then forces voltage and measures current. `i[KI_SITE0]` contains measured data from the anchor site; `i[KI_SITE1]` contains `SITE_INACTIVE`.

site_enable

This command enables dual-site mode for the specified site ID.

Models supported

S535

Usage

`int site_enable(int siteid);`

siteid	Specifies a site to enable (input); valid values are: `KI_SITE0` = Enables the anchor site `KI_SITE1` = Enables the mirror site

Details

Set *siteid* to `KI_SITE0` to specify the anchor site and `KI_SITE1` to specify the mirror site. When a site is enabled, measurements are stored in the result data array entry for that site. To access the data for the specified site, use `siteid` as the index for the results data array.

Dual-site settings specified by the `site_enable` and `site_disable` commands are not reset by the `devint` command.

When dual-site mode is disabled, this command returns error code `KTE_FEATURE_DISABLED`.

NOTE

Calls to this command are built into the Keithley Test Execution Engine (KTXE) if you specify this parameter in the wafer description file (`.wdf`).

To disable a site, see the site_disable (on page 2-110) command.

For more information about using dual-site mode, see "Dual-site operation" in the *S535 Reference Manual* (part number S535-901-01).

Example

```
site_enable(KI_SITE0);
site_enable(KI_SITE1);
conpin(SMU1, 1, 0);
conpin(GND, 2, 0);
forcev(SMU1, 10.0);
measi(SMU1, &i[0]);
```

This example enables both the anchor site and the mirror site. `i[KI_SITE0]` contains data from the anchor site; `i[KI_SITE1]` contains data from the mirror site.

site_mapping

This command creates a new pin mapping between SITE0 and SITE1.

Models supported

S535

Usage

`int site_mapping(int site_id, int *site_pinlist, int num_pins);`

site_id	An integer value representing the site to add to the pin mapping (input); valid values: `KI_SITE0` = Specifies a new pin map for the anchor site `KI_SITE1` = Specifies a new pin map for the mirror site
site_pinlist	Integer array containing a list of pins for the site (input)
num_pins	The number of pins in the pinlist arrays (input); valid values: `1` to `X`

Details

Pins of matching index in each array become dual pins. When you call the `conpin` command in test code, the dual pins close automatically to connect SITE1 instruments to the dual site.

When dual-site mode is disabled, this command returns error code `KTE_FEATURE_DISABLED`.

All pins in the pinlist must be valid pins that are defined in the `icconfig_<QMO>.ini` file. Pins in `site0_pinlist` must not have entries in the `site1_pinlist`, and conversely, pins in the `site1_pinlist` must not have entries in the `site0_pinlist`.

NOTE

If you have a chuck connected in an S535 system that is in dual-site mode, the pin the chuck is on cannot be mirrored.

Any given pin may have only one entry in the array. This function inspects each array value to make sure that no invalid pins are specified. If any of these error cases are detected, the function generates error `MX_INVALID_PIN_MAPPING`.

NOTE

Unlike other LPT commands, dual-site pin mapping changes are permanent and do not reset to default when the `devint` command is called.

For more information about using dual-site mode, see "Dual-site operation" in the *S535 Reference Manual* (part number S535-901-01).

Example

```
int site0[4] = {1, 2, 3, 4};
int site1[4] = {5, 6, 7, 8};
site_mapping(KI_SITE0, &site0[0], 4);
site_mapping(KI_SITE1, &site1[0], 4);

conpin(SMU1, 1, 0);
forcev(SMU1, 10.0);
```

In this example, SMU1 forces 10 V on pin 1 and the corresponding mirror SMU forces 10 V on pin 5.

site_status

This command reads the state of the specified site and places it in the `state` variable.

Models supported

S535

Usage

```
int site_status(int siteid, int *state);
```

siteid	Specifies the site to read (input); valid values: `KI_SITE0` = The anchor site `KI_SITE1` = The mirror site
state	The variable that receives the site status (output) `KI_ON` = Site enabled `KI_OFF` = Site disabled

Details

This function sets the `state` variable to `KI_ON` if the site is enabled and `KI_OFF` if it is disabled.

Dual-site settings specified by the `site_enable` and `site_disable` commands are not reset by the `devint` command.

When dual-site mode is disabled, this command returns error code `KTE_FEATURE_DISABLED`.

For more information about using dual-site mode, see "Dual-site operation" in the *S535 Reference Manual* (part number S535-901-01).

Example

```
int state = KI_OFF;
site_enable(KI_SITE0);
site_status(KI_SITE0, &state);
/* state is now set to KI_ON.*/
```

This example enables SITE0 and reads the *state* variable (`KI_ON`).

smeasX

This command allows a number of measurements to be made by a specified instrument during a `sweepX` command. The results of the measurements are stored in the defined array.

Models supported

S530, S535, S540

Usage

```
int smeasi(int instr_id, double *result);
int smeast(int instr_id, double *result);
int smeasv(int instr_id, double *result);
int smeasc(int instr_id, double *result);
int smeasg(int instr_id, double *result);
```

instr_id	The instrument identification code of the measuring instrument
result	The floating point array that stores the results

Details

This command is used to create an entry in the measurement scan table. During any of the sweep commands, a measurement scan is done for every force point in the sweep. During each scan, a measurement is made for every entry in the scan table. The measurements are made in the same order in which the entries were made in the scan table.

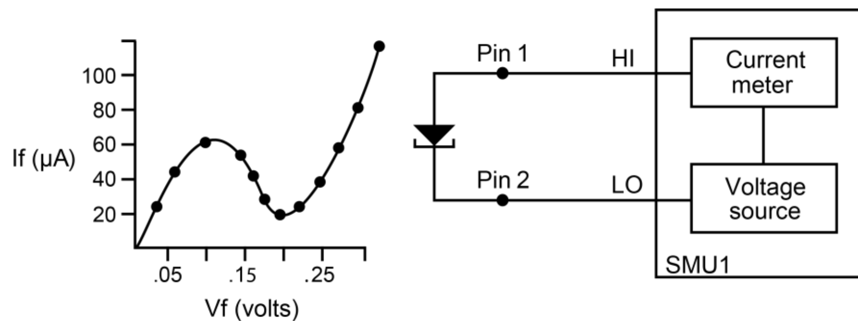
The `smeasX` command sets up the new scan table entry to make an ordinary measurement. The scan table is cleared by an explicit call to the `clrscn` command or implicitly when the `devint` command is called.

The `smeasi`, `smeast`, and `smeasv` commands return dual-site data for SITE0 and SITE1 if available (S535 systems only).

Example

```
double resi[13]; /*  Defines array. */
.
.
conpin(SMU1, 1, 0);
conpin(GND, 2, 0);
smeasi(SMU1, resi); /* Make a series of */
 /* measurements; */
 /* return the results to the */
 /* resi array. */
sweepv(SMU1, 0.0, 0.3, 12, 25.0E-3); /* Make 13 measurements as the */
 /* voltage ranges from 0 to */
 /* 0.3V. */
```

This example determines the measurement data needed to create a graph showing the negative resistance characteristics of a tunnel diode. SMU1 generates a voltage ramp ranging from 0 to 0.3 V. The current through the diode is sampled 13 times with a duration of 25 ms at each step. The results are stored in an array named `resi`.



Also see

clrscn (on page 2-34)
devint (on page 2-42)
sweepX (on page 2-118)

ssmeasX

This command makes a series of readings until the change (delta) between readings is within a specified percentage.

Models supported

S530, S535, S540

Usage

```
int ssmeasi(int instr_id, double *result, double delta, unsigned int max_read, double
    delay);
int ssmeasv(int instr_id, double *result, double delta, unsigned int max_read, double
    delay);
```

instr_id	The instrument identification code of the measuring instrument; SMU*n* or VMTR*n*
result	The floating point variable assigned to the result of the measurement
delta	The termination definition; this is the percentage of the first reading that defines the steady-state condition
max_read	The maximum number of readings made to determine whether or not the reading is steady
delay	The delay between readings to wait in seconds

Details

This command is used when device stability is uncertain. It continually reads the instrument until the resulting measurement is stable and provides the fastest measurement possible.

If the reading never stabilizes because of factors such as oscillations or charge and discharge, this reading count expires and a reading of MEAS_NOT_PERFORMED (1.00E23) is returned.

Any instrument that uses the meas*X* command can use the ssmeas*X* command. This command calls the meas*X* command for each reading. Any range*X* command rule applies to this command.

The ssmeas*X* command is used when making single-point readings. It is not used for any of the combination measurements, such as the *X*sweep*Y* and trig*XY* commands.

Under certain test conditions, the ssmeas*X* command is not ideal. For example, an oscillation where two contiguous measurements are within the given percentage will return a stable reading, even though the device cannot be measured.

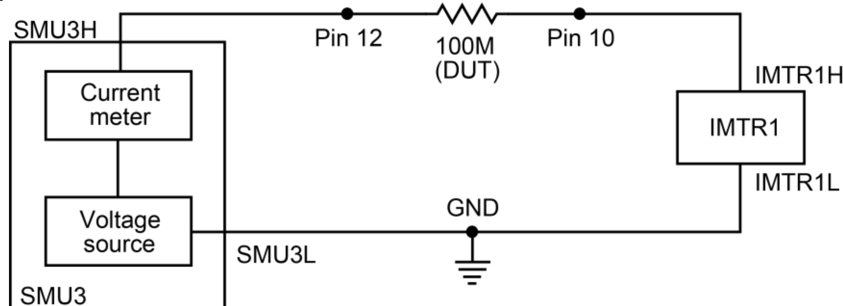
This command returns dual-site data for SITE0 and SITE1 if available (S535 systems only).

Example

```
double meascur;
.
.
conpin(SMU3, 12, 0);       /* Make connections. */
conpin(SMU2, 10, 0);
setimtr(SMU2);
.
.
forcev(SMU3, 0.1);  /* Perform the test. */
ssmeasi(SMU2, &meascur, 0.1, 300, 0.015);      /* Steady */
    /* state measurement /*
 /* with delta of 0.1%, with */
 /* maximum of 300 readings */
. /* before error, wait 15 ms */
. /* between readings. */
```

This example makes a series of measurements and tests to verify if the present measurement and the previous measurement are within 0.1%. If the measurements are within 0.1%, the result of the last measurement is stored and the program continues. If the measurements are not within 0.1%, the program waits 15 ms before making another measurement. It then compares this measurement with previous measurements. If the measurements are within 0.1%, the result of the last measurement is stored and the program continues. If the measurements are not within 0.1% it repeats the comparison until the change is within 0.1%. If, after 300 attempts, the change is not within the specified limit, the following error is returned:

`"MEAS_NOT_PERFORMED (1.000E23)."`



Also see

measX (on page 2-61)
rangeX (on page 2-77)
smeasX (on page 2-114)

sweepX

This command generates a ramp consisting of ascending or descending voltages or currents. The sweep consists of a sequence of steps, each with a user-specified duration.

Models supported

S530, S535, S540

Usage

```
int sweepi(int instr_id, double startval, double endval, long stepno, double
    step_delay);
int sweepv(int instr_id, double startval, double endval, long stepno, double
    step_delay);
```

instr_id	The instrument identification code of the sourcing instrument
startval	The initial voltage or current level output from the sourcing instrument, which is applied for the first sweep measurement; this value can be positive or negative
endval	The final voltage or current level applied in the last step of the sweep; this value can be positive or negative
stepno	The number of current or voltage changes in the sweep; the actual number of forced data points is one greater than the number of steps specified
step_delay	The delay in seconds between each step and the measurements defined by the active measure list

Details

The `sweepX` command is always used with the `smeasX`, `sintgX`, `savgX`, or `rtfary` command.

The `sweepX` command causes a sourcing instrument to generate a series of ascending or descending voltages or current changes called steps. During this source time, a measurement scan is done at each step.

NOTE

The actual number of forced data points is one more than the number of steps specified. This means that the number of measurements made is the number of steps specified plus one. This is important when dimensioning the size of the results array. Failure to make sure the array is big enough will produce operating system access violation errors.

Measurements are stored in an array in the order they were made.

The `trigXg` and `trigXl` commands can be used with the `sweepX` command, even though they are also used with the `smeasX`, `sintgX`, or `savgX` commands. In this case, data resulting from each of the steps is stored in an array, as noted above. However, once a trigger point (for example, a level of current or voltage) is reached, the sourcing device stops incrementing or decrementing and is held at a steady output level for the remainder of the sweep.

The `clrscn` command is used to eliminate the previous measures for the second sweep. Using the `smeasX`, `sintgX`, or `savgX` command after a `clrscn` command causes the appropriate new measures to be defined and used.

When the first sweep point is nonzero, it may be necessary to precharge the circuit so that the `sweepX` command will return a stable value for the first measured point without penalizing remaining points in the sweep. For example:

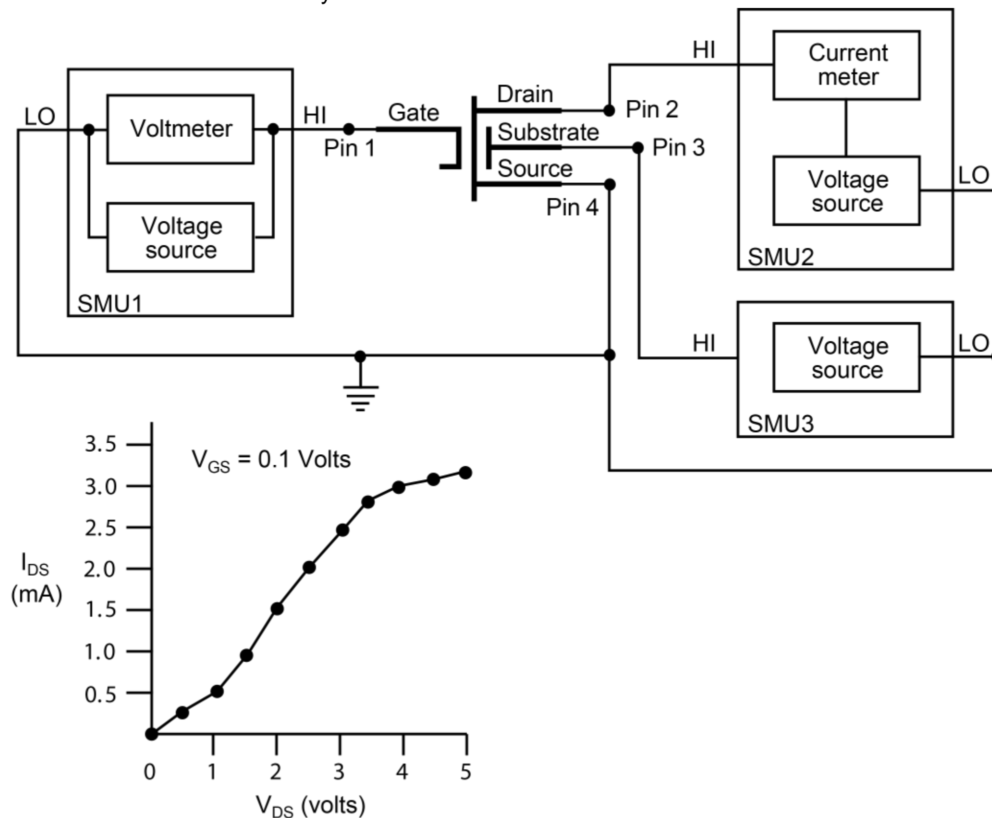
```
double ires[6];
conpin(SMU1, 10);
conpin(2, GND, 0);
forcev(SMU1, 5.0);       /* Force 5 V to charge. */
delay(10);       /* Wait for precharge. */
smeasi(SMU1, &ires); /* Set up the measurement. */
sweepv(SMU1, 5.0, 10.0, 5, /* Make the real measurement. */
 2.5E-3);
```

You can use this command in dual-site mode (S535 systems only).

Example

```
double resi[11], ssbiasv;
.
.
conpin(SMU1, 1, 0);
conpin(SMU2, 2, 0);
conpin(SMU3, 3, 0);
conpin(GND, 4, 0);
forcev(SMU3, ssbiasv); /* Apply substrate bias */
 /* voltage ssbiasv. */
forcev(SMU1, -.1); /* Apply a gate-to-source */
 /* voltage of -0.1 V. */
smeasi(SMU2, resi); /* Perform a series of current */
 /* measurements; return */
 /* the results to the array */
 /* resi. */
sweepv(SMU2, 0.0, 5.0, 10, 2.5E-3); /* Generate */
 /* 11 steps and 11 */
 /* points each 2.5 ms duration, */
 /* ranging from 0 V to 5V. */
```

This example gathers data to create a graph showing the common drain-source characteristics of a field-effect transistor (FET). A fixed gate-to-source voltage is generated by SMU1. A voltage ramp from 0 V to 5 V is generated by SMU2. Drain current applied by SMU2 is measured 11 times by the `smeasi` command. Data is stored in the array `resi`.



Also see

rtfary (on page 2-88)
savgX (on page 2-89)
sintgX (on page 2-109)
smeasX (on page 2-114)

trigXg, trigXl

This command monitors for a predetermined level of voltage, current, conductance, capacitance, or time.

Models supported

S530, S535, S540

Usage

```
int trigig(int instr_id, double value);
int trigil(int instr_id, double value);
int trigtg(int instr_id, double value);
int trigtl(int instr_id, double value);
int trigvg(int instr_id, double value);
int trigvl(int instr_id, double value);
```

instr_id	The instrument identification code of the monitoring instrument
value	The voltage, current, conductance, capacitance, or time specified as the trigger point; this trigger point value is reached when either of the following occurs: ■ The measured value is equal to or greater than the value argument of the `trigXg` command ■ The measured value is less than the value argument of the `trigXl` command

Details

The `trigXl` and `trigXg` commands are used with the `searchX` command or with one of the sweep measurement commands: `smeasX`, `sintgX`, or `savgX`.

- The `trigXg` or `trigXl` command provides the `sweepX` command the digital feedback to allow for the increase or decrease in sourcing values.
- The `trigXl` and `trigXg` commands must be located before any associated `searchX` commands.
- Triggers are not automatically reset by the `searchX` or `sweepX` command. A single call to the `trigXl` or `trigXg` command can be followed by two or more calls to the `searchX` or `sweepX` commands.

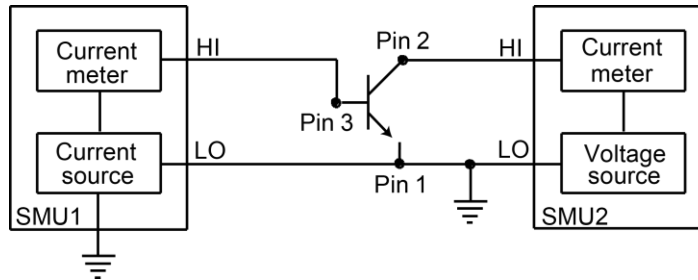
The specified trigger point is automatically cleared when a `clrtrg` or `devint` command is executed.

You can use this command in dual-site mode (S535 systems only).

Example 1

```
double res22, vcc8;
.
.
conpin(SMU1, 3, 0);
conpin(SMU2, 2, 0);
conpin(GND, 1, 0);
forcev(SMU2, vcc8); /* Apply collector voltage to vcc8. */
trigig(SMU2, +5.0E-3); /* Search for a collector */
 /* current of 5 mA. */
searchi(SMU1, 5.0E-5, 2.0E-4, 15, 1.0E-3, &res22); /* Generate */
    /* a current ranging */
 /* from 50 uA to 200 uA in */
 /* 15 iterations. Return the */
 /* current resulting from the */
 /* last iteration as res22. */
```

This example uses the `trigig` and `searchi` commands together to generate and search for a specific current level. A search is initiated to find the base current needed to produce 5 mA of collector current. The collector-emitter voltage supplied by SMU2 is defined by the variable `vcc8`. The `searchi` command generates the base current from SMU1. This current ranges between 50 mA and 200 mA in 15 iterations. The `trigig` command continuously monitors the current through SMU1. The base current supplied by SMU1 is stored as the result `res22`.

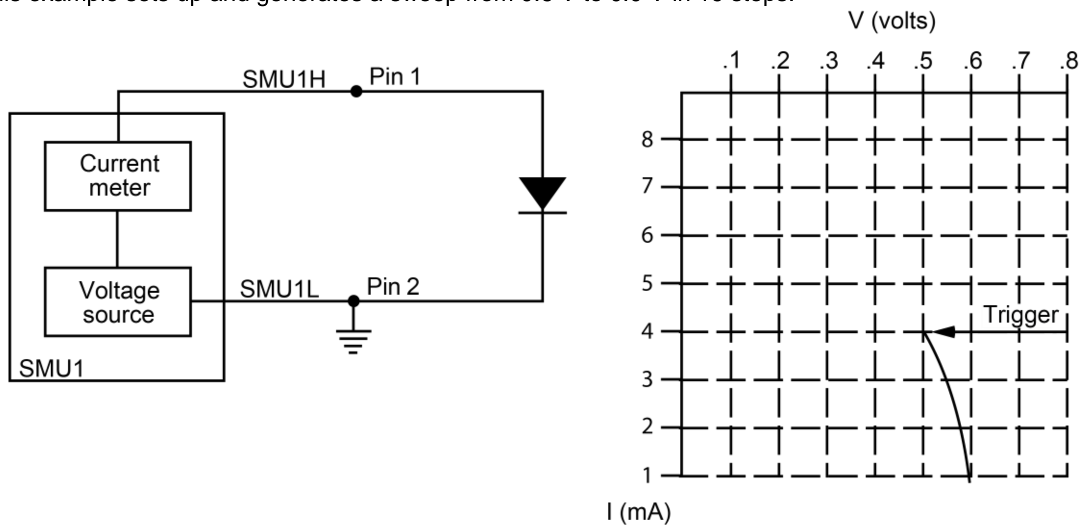


Example 2

```
double res1[19];
.
.
conpin(SMU1, 1, 0);
conpin(GND, 2, 0);
trigil(SMU1, +5.0E-3); /* If greater than -5 mA, */
 /* stop ramping. */

smeasi(SMU1, res1); /* Measure current at each of */
 /* the 19 levels; return */
 /* results to the res1 array. */
sweepv(SMU1, 0.6, 0.0, 18, 1.00E-3); /* Generate*/
     /* 0.6 V to 0.0 V in 19 steps. */
```

This example sets up and generates a sweep from 0.6 V to 0.0 V in 19 steps.



Also see

savgX (on page 2-89)
searchX (on page 2-97)
sintgX (on page 2-109)
smeasX (on page 2-114)
sweepX (on page 2-118)

tstsel

This command enables or disables a test station.

Models supported

S530, S535, S540

Usage

```
tstsel(long x);
```

x	The test station number: `1`

Details

Only one test station can be active at a time.

The `tstsel` command is normally called at the beginning of a test program. Only one call to the `tstsel` command per program is recommended.

Calling a new test station within a test program cancels any previous `tstsel` calls.

To relinquish control of an individual test station so another station can access the system, call the `tstsel` command with a value of zero (`0`). A new test station must then be selected before any subsequent test control commands are run.

Attempting to run a test program on an already active test station causes the message "Error: `tstsel` failed with status = -653. Exiting" to be displayed.

NOTE

The `tstsel` command is not required for use in a user test module (UTM).

You can use this command in dual-site mode (S535 systems only).

Also see

None

Section 3

PARLib command reference

In this section:

Introduction 3-1
How to use the library reference 3-2
Categorized subroutine lists 3-4
Subroutine descriptions 3-6

Introduction

The Keithley Test Environment (KTE) Parametric Test Subroutine Library (PARLib) is a parameter extraction and data analysis software system. The PARLib subroutines are used to analyze data associated with parametric tests.

This section contains detailed descriptions of the PARLib subroutines. It is intended as a reference guide for experienced users.

The PARLib subroutines use the C programming language to make measurements for very specific applications. The PARLib subroutines are grouped in the following categories:

- Bipolar subroutines
- Resistors, diodes, capacitors, and special structure subroutines
- MOSFET subroutines
- FET and JFET subroutines
- Math and support subroutines

The Subroutine descriptions (on page 3-6) contain information specific to the type of structure you are testing, including voltage and current polarities between pins on the device under test (DUT).

The descriptions also include (where applicable) a description of what each source-measure unit (SMU) in the test configuration does and a simplified schematic showing the configuration of instruments and devices in the subroutine.

How to use the library reference

The subroutines in the Test subroutine library reference are in the C programming language. Each subroutine is presented in a standard format that follows the pattern below:

- **Purpose statement:** The first line of text under the subroutine heading contains a brief explanation of what the subroutine does.

Figure 4: Example purpose statement

icbo

This subroutine measures leakage when the collector-base junction is reverse-biased (common base) .

- **Usage:** A line of code representing the prototype of the subroutine, followed by a table listing the input and output parameters for the subroutine.

  Parameters that you specify are shown in *`monospace italic`* font. Parameters preceded by an asterisk (`*`) are character parameters that are passed into the function (input) or pointers to information that is returned (output).

  Each parameter is preceded by one of the following declarations that specifies the data type for the parameter: `int` (integer), `double` (double-precision floating-point), and `char` (a character string).

Figure 5: Example syntax and parameter definition

Usage

```
double icbo(int e, int b, int c, int sub, double vcbo, double vsub)
```

e	Input	The emitter pin of the device
b	Input	The base pin of the device
c	Input	The collector pin of the device
sub	Input	The substrate pin of the device
vcbo	Input	Forced collector-base voltage, in volts
vsub	Input	The forced substrate bias, in volts
Returns	Output	The measured collector-base current

- **Details:** Additional information about using the subroutine.

Figure 6: Example details

Details

This subroutine measures the collector-base leakage current at a specified collector-base voltage (V_{CB}) and substrate bias (V_{SUB}) for a bipolar transistor. The emitter pin is not connected (floating), and the base is grounded.

If a zero or negative substrate pin is specified, the substrate is left floating. If the pin number is greater than 0 and V_{SUB} is less than 0.9 mV, the substrate is grounded. In all other cases, it is connected and forced.

- **V/I polarities:** The polarities of the current or voltage flow between the pins of the device; based on whether you are using an NPN or PNP transistor. This information is only applicable in some subroutines.

Figure 7: Example V/I polarities information

V/I polarities

NPN $+V_{CB}$, $-V_{SUB}$

PNP $-V_{CB}$, $-V_{SUB}$

- **Source-measure units (SMUs):** A description of what each SMU in the test configuration does in the subroutine. This information is only applicable in some subroutines.

Figure 8: Example SMUs description

Source-measure units (SMUs)

SMU1: Forces V_{CB}, default current limit, measures I_{CBO}

SMU2: Forces V_{SUB}, default current limit

- **Example:** A line of code showing what a call to the command might look like in actual use.

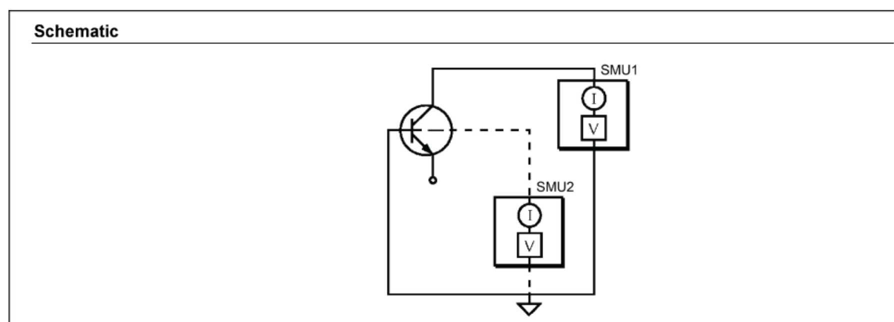
Figure 9: Example of actual command call

Example

```
result = icbo(e, b, c, sub, vcbo, vsub)
```

- **Schematic:** A simplified schematic showing the configuration of the instruments and devices for the subroutine. This information is only applicable in some subroutines.

Figure 10: Example schematic



Categorized subroutine lists

The subroutine descriptions are listed in alphabetical order in Subroutine descriptions (on page 3-6). The tables that follow contain all of the subroutines grouped by function, with a brief description of the purpose of the subroutine and a hyperlink to the full subroutine description.

Bipolar subroutines

Subroutine	Description
`beta1`	Calculate DC β at specified I_E and V_{CB}
`beta2`	Calculate DC β and V_{BE} at specified I_C and V_{CE}
`beta2a`	Calculate β at V_{CB} and I_{CE} with search on I_E
`beta3a`	Calculate β at V_{CE} and I_{CE} with search on I_{BE}
`bice`	Calculate β when V_{BE} swept, at V_{CE} and V_{SUB}
`bvcbo`	Measure collector-base breakdown voltage, emitter open
`bvcbo1`	Measure collector-base breakdown voltage using LPTLib `bsweepv` subroutine
`bvceo`	Measure collector-emitter breakdown voltage, base open
`bvceo2`	Measure collector-emitter breakdown voltage using LPTLib `bsweepv` subroutine
`bvces`	Measure collector-emitter breakdown voltage
`bvebo`	Measure emitter-base breakdown voltage, collector open
`ibic1`	Measure I_{CE}, I_{BE} and calculate β at V_{CE}, V_{BE}, V_{SUB}
`icbo`	Measure collector-base leakage at V_{CB} and V_{SUB}
`iceo`	Measure collector-emitter leakage at V_{CE} and V_{SUB}
`ices`	Measure emitter-collector leakage at V_{CES} and V_{SUB}
`iebo`	Measure emitter-base leakage at V_{EB} and V_{SUB}
`rcsat`	Estimate `rcsat` when I_C and I_B swept at constant β
`re`	Estimate emitter resistance
`vbes`	Measure base-emitter voltage at specified I_E ($V_C = V_B$)

FET and JFET subroutines

Subroutine	Description
`gm`	Estimate MESFET transconductance at V_{DS}, V_{GS}
`idss`	Estimate MESFET I_{DSS} and V_{DSAT} at V_{DSS}
`vp`	Estimate FET pinch-off voltage for a MESFET
`vp1`	Estimate MESFET pinch-off at I_{DS} (I_P) and V_{DS}

Math and support subroutines

Subroutine	Description
fnddat	Search an array, return a new array
fndtrg	Determine which native mode trigger to use
kdelay	Delay, based on current and voltage values
logstp	Create an array using logarithmic steps
tdelay	Return calculated delay time

MOSFET subroutines

Subroutine	Description
bvdss	Measure drain-source breakdown voltage (V_G = 0)
bvdss1	Measure drain-source breakdown voltage using LPTLib bsweepv subroutine
deltl1	Estimate delta L MOSFET parameter
deltw1	Estimate delta W for a MOSFET
gamma1	Estimate body effect (γ)
gd	Calculate drain conductance of a MOSFET
id1	Measure drain current at specified V_{GS}, V_{DS}, and V_{BS}
idsat	Measure drain current at V_{DS}, V_{BS} (V_D = V_G)
idvsvg	Measure I_{DS} when V_{GS} is swept at constant V_{DS} and V_{BS}
isubmx	Find peak substrate current at V_{DS},V_{BS}
vg2	Measure gate-source voltage at I_{DS}, V_{DS}, V_{BS}
vgsat	Measure V_{GSAT} at specified I_{DS} (V_{GS} = V_D)
vt14	Estimate V_T using two-point technique
vtati	Find V_T to produce specified I_{DS}
vtext	Extrapolate gate-source threshold voltage
vtext2	Estimate V_T using modified vtext subroutine method
vtext3	Calculate V_T using max slope method

Resistors, diodes, capacitors, and special structure subroutines

Subroutine	Description
bkdn	Measure breakdown voltage (force I, measure V)
cap	Measure two-terminal capacitance
fimv	Force current and measure voltage on device with four high pins and four ground pins
fvmi	Force voltage and measure current on device with four input pins and four ground pins
leak	Measure leakage current at specified voltage
res	2-terminal resistance (force I, measure V)
res2	2-terminal resistance with voltage limit
res4	4-terminal resistance (force I, measure V)
resv	2-terminal resistance (force V, measure I)
rvdp	4-terminal van der Pauw measurement
tox	Calculate oxide thickness from capacitance
vf	Measure the forward junction voltage of a diode

Subroutine descriptions

The following topics contain detailed descriptions of the Parametric Test Subroutine Library (PARLib) commands.

beta1

This subroutine calculates the DC beta (β) of a test device at constant emitter current (I_E) and collector-base bias (V_{CB}). The device is in the common-base configuration.

Usage

```
double beta1(int e, int b, int c, int sub, double ie, double vcb, char type);
```

e	Input	The emitter pin of the device
b	Input	The base pin of the device
c	Input	The collector pin of the device
sub	Input	The substrate pin of the device
ie	Input	The forced emitter current, in amperes
vcb	Input	The forced *c* to *b* bias, in volts
type	Input	Type of transistor: 'N' or 'P'
Returns	Output	The calculated beta of the device: ■ -1.0 = TYPE not 'N' or 'P' ■ -2.0 = SMU2 overload ■ -3.0 = Divide by 0, or β < 0.01 ■ -4.0 = β > 10 K or I_B wrong sign ■ -5.0 = Emitter voltage limit reached; developed emitter voltage is within 98% of the 3 V voltage limit

Details

If a positive substrate pin is specified, the substrate is grounded. If a positive substrate pin is not specified, the substrate is left floating.

A delay is incorporated into the `beta1` subroutine; this delay is the calculated time required for stable forcing of emitter current with a 3 V voltage limit.

V/I polarities

The polarities of V_{CB} and I_E are determined by device type.

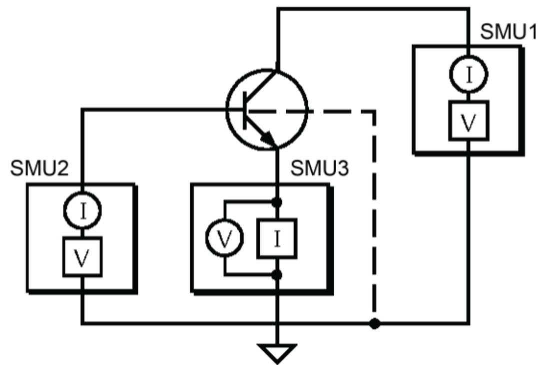
Source-measure units (SMUs)

- SMU1: Forces V_{CB}, default current limit
- SMU2: Forces 0.0 V, measures base current (I_B)
- SMU3: Forces I_E, 3 V voltage limit

Example

```
result = beta1(e, b, c, sub, ie, vcb, type);
```

Schematic



beta2

This subroutine calculates beta (β) and base-emitter voltage (V_{BE}) at a specified collector current (I_C) and collector-emitter bias (V_{CE}).

Usage

```
double beta2(int e, int b, int c, int sub, double ice, double vce, double *vbeout, double
    *icout, char type);
```

e	Input	The emitter pin of the device
b	Input	The base pin of the device
c	Input	The collector pin of the device
sub	Input	The substrate pin of the device
ice	Input	The targeted collector current, in amperes
vce	Input	The forced collector-emitter voltage, in volts
type	Input	Type of transistor: `'N'` or `'P'`
vbeout	Output	The measured base voltage
icout	Output	The measured collector current
Returns	Output	The calculated beta: ▪ `-1.0` = TYPE not 'N' or 'P' ▪ `-2.0` = SMU2 overload ▪ `-3.0` = Divide by 0, or β < 0.01 ▪ `-4.0` = β > 10 K ▪ `-5.0` = Too many iterations ▪ `-6.0` = Emitter voltage limit reached; developed emitter voltage is within 98% of the 3 V voltage limit

Details

If a positive substrate pin is specified, the substrate is grounded. If a positive substrate pin is not specified, the substrate is left floating.

A delay is incorporated into the `beta2` subroutine; this delay is the calculated time required for stable forcing of emitter current with a 3 V voltage limit.

A faster and simpler subroutine to use is beta2a (on page 3-9).

V/I polarities

The polarities of V_{CE} and I_E are determined by device type.

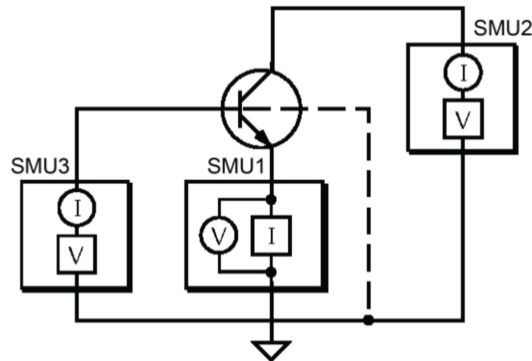
Source-measure units (SMUs)

- SMU1: Forces I_E, 3 V voltage limit
- SMU2: Forces V_{CE}, maximum current limit, measures I_{CE}
- SMU3: Forces 0.0 V, measures base current (I_B)

Example

```
result = beta2 (e, b, c, sub, ice, vce, &vbeout, &icout, type);
```

Schematic



beta2a

This subroutine calculates beta (β) at collector-base voltage (V_{CB}) and collector-emitter current (I_{CE}) using the `searchi` and `trig` LPTLib functions to search emitter current (I_E) until the target I_{CE} is reached. The device is in the common-base configuration.

Usage

```
double beta2a(int e, int b, int c, int sub, double ice, double vcb, double ie1, double
    ie2, double vsub, double *icmeas, double *ieout, double *error);
```

e	Input	The emitter pin of the device
b	Input	The base pin of the device
c	Input	The collector pin of the device
sub	Input	The substrate pin of the device
ice	Input	The targeted collector current, in amperes
vcb	Input	The forced *c* to *b* bias, in volts
ie1	Input	The start of the emitter current search, in amperes
ie2	Input	The end of the emitter current search, in amperes
vsub	Input	The forced substrate bias, in volts
icmeas	Output	The final measured collector-emitter current
ieout	Output	The final forced value of emitter current
error	Output	The percent error between the target collector current (I_{CE}) and the final measured collector current (I_{CMEAS})
Returns	Output	The calculated beta: `-1.0` = Target I_{CE} = 0.0 `-2.0` = Collector current limit reached `-3.0` = Emitter voltage limit reached

Details

This subroutine is a revised version of the beta2 (on page 3-8) subroutine that uses the LPTLib `searchi` and `trig` functions to search I_E until the target I_{CE} is reached.

This subroutine sets the current trigger on SMU1 at the specified I_{CE}. The emitter current is searched until the trigger is set. The emitter current is then forced, the collector current measured, and β is calculated.

The percent error (*error*) is calculated between the target I_{CE} and the final measured I_{CE} and returned.

If a zero or negative substrate pin is specified, the substrate is left floating. If the pin number is greater than 0 and V_{SUB} is less than 0.9 mV, the substrate is grounded. In all other cases, it is connected and forced.

This subroutine is not supported in dual-site mode (S535 systems).

V/I polarities

NPN $+I_{CE}$, $+V_{CB}$, I_E, $-V_{SUB}$

PNP $-I_{CE}$, $-V_{CB}$, $+I_E$, $-V_{SUB}$

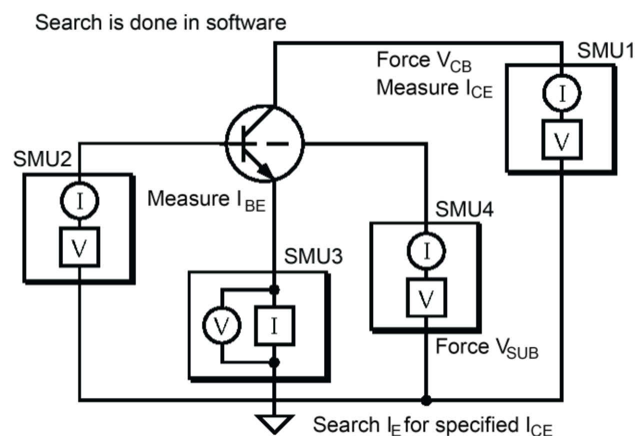
Source-measure units (SMUs)

- SMU1: Forces V_{CB}, maximum current limit, triggers on I_{CE}
- SMU2: Forces 0.0 V, measures I_{BE}
- SMU3: Searches I_E, 3 V voltage limit
- SMU4: Forces V_{SUB}, default current limit

Example

```
result = beta2a(e, b, c, sub, ice, vcb, ie1, ie2, vsub, &icmeas, &ieout, &error);
```

Schematic



beta3a

This subroutine calculates beta (β) at collector-emitter voltage (V_{CE}) and collector-emitter current (I_{CE}) using the `searchi` and `trig` LPTLib functions to search base-emitter current (I_{BE}) until the target I_{CE} is reached. The device is in the common-emitter configuration.

Usage

```
double beta3a(int e, int b, int c, int sub, double ice, double vce, double ibe1, double
    ibe2, double vsub, double *ibe, double *cmeas, double *error);
```

e	Input	The emitter pin of the device
b	Input	The base pin of the device
c	Input	The collector pin of the device
sub	Input	The substrate pin of the device
ice	Input	The targeted collector current, in amperes
vce	Input	The forced collector-emitter voltage, in volts
ibe1	Input	The start of the base-emitter current (I_{BE}) search, in amperes
ibe2	Input	The end of the base-emitter current (I_{BE}) search, in amperes
vsub	Input	The forced substrate bias, in volts
ibe	Output	The final measured emitter-base current
icmeas	Output	The final measured collector-emitter current
error	Output	The percent error between the target collector current (I_{CE}) and the final measured collector current (I_{CMEAS})
Returns	Output	The calculated beta: `-1.0` = Target I_{CE} = 0.0 `-2.0` = Base voltage limit reached

Details

This subroutine sets the current trigger on SMU1 at the specified I_{CE}. The base current is searched until the trigger is set. The base current is then forced, the collector current measured, and β is calculated.

The percent error (*error*) is calculated between the target I_{CE} and the final measured I_{CE} and returned.

If a zero or negative substrate pin is specified, the substrate is left floating. If the pin number is greater than 0 and V_{SUB} is less than 0.9 mV, the substrate is grounded. In all other cases, it is connected and forced.

This subroutine is not supported in dual-site mode (S535 systems).

V/I polarities

NPN $+I_{CE}$, $+V_{CE}$, $+I_{BE}$, $-V_{SUB}$

PNP $-I_{CE}$, $-V_{CE}$, $-I_{BE}$, $-V_{SUB}$

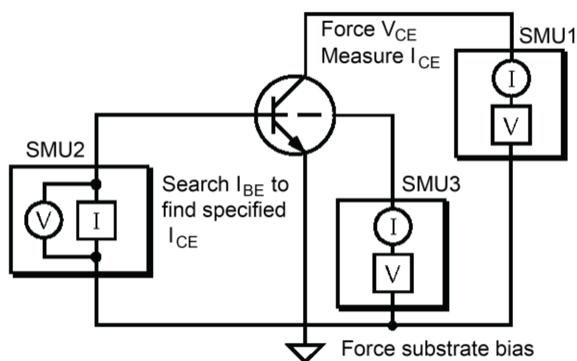
Source-measure units (SMUs)

- SMU1: Forces V_{CE}, maximum current limit, triggers on I_{CE}
- SMU2: Searches I_{BE}, 3 V voltage limit
- SMU3: Forces V_{SUB}, default current limit

Example

```
result = beta3a(e, b, c, sub, ice, vce, ibe1, ibe2, vsub, &ibe, &cmeas, &error);
```

Schematic



bice

This subroutine sweeps the emitter-base voltage (V_{BE}), measures the resulting collector-emitter current (I_{CE}), and calculates beta (β) at each value of V_{BE} for a bipolar transistor. The device is connected in the common-emitter configuration.

Usage

```
void bice(int e, int b, int c, int sub, double vce, double vbe1, double vbe2, double
    vsub, int npts, double ice_last, double *beta_last, double *beta_max, double
    *ic_max);
```

e	Input	The emitter pin of the device
b	Input	The base pin of the device
c	Input	The collector pin of the device
sub	Input	The substrate pin of the device
vce	Input	The forced collector-emitter voltage, in volts
vbe1	Input	The start point of the V_{BE} sweep, in volts
vbe2	Input	The end point of the V_{BE} sweep, in volts
vsub	Input	Substrate bias, in volts
npts	Input	The number of points in the sweep
ice_last	Output	The measured I_{CE} array
beta_last	Output	The calculated beta array
beta_max	Output	The maximum beta in the array
ic_max	Output	The I_{CE} at maximum beta

Details

The collector-emitter voltage (V_{CE}) and the substrate voltage (V_{SUB}) are held constant.

In addition to the β and I_{CE} return arrays, the maximum β (*beta_max*) and collector current at maximum β (*ic_max*) are returned.

If a zero or negative substrate pin is specified, the substrate is left floating. If the pin number is greater than 0 and V_{SUB} is less than 0.9 mV, the substrate is grounded. In all other cases, it is connected and forced.

V/I polarities

NPNs $+V_{CE}$, $+I_{BE}$, and $-V_{SUB}$

PNPs $-V_{CE}$, $-I_{BE}$, and $-V_{SUB}$

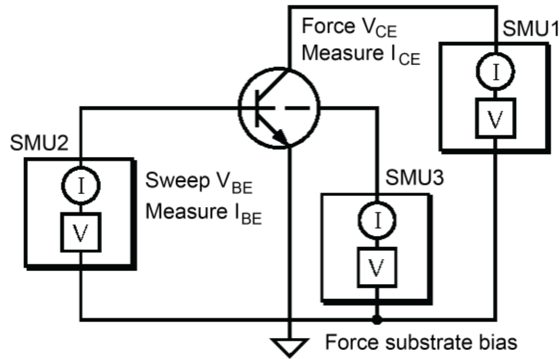
Source-measure units (SMUs)

- SMU1: Forces V_{CE}, maximum current limit, measures I_{CE}
- SMU2: Sweeps V_{BE}, maximum current limit, measures I_{BE}
- SMU3: Forces V_{SUB}, default current limit

Example

```
result = bice(e, b, c, sub, vce, vbe1, vbe2, vsub, npts, ice_last, &beta_last,
   &beta_max, &ic_max);
```

Schematic



bkdn

This subroutine forces a current and measures breakdown voltage on a two-terminal device.

Usage

`double bkdn(int hi, int lo, int sub, double ipgm, double vlim);`

hi	Input	The high pin of the device
lo	Input	The low pin of the device
sub	Input	The substrate pin of the device
ipgm	Input	The forced current, in amperes
vlim	Input	The voltage limit, in volts
Returns	Output	The measured breakdown voltage: `+2.0E + 21` = Measured voltage is within 98% of the specified voltage limit

Details

If a positive substrate pin is specified, the substrate is grounded. If a positive substrate pin is not specified, the substrate is left floating.

A delay is incorporated into the `bkdn` subroutine; this delay is the calculated time required for stable forcing of *ipgm* within the *vlim* voltage limit.

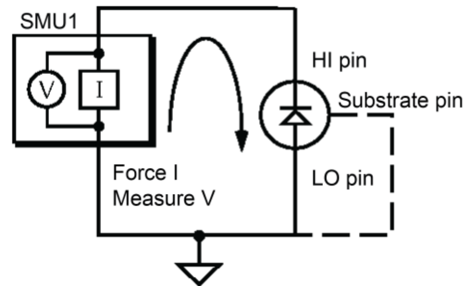
Source-measure units (SMUs)

- SMU1: Forces *ipgm*, programmable voltage limit, measures breakdown voltage

Example

```
result = bkdn(hi, lo, sub, ipgm, vlim);
```

Schematic



bvcbo

This subroutine forces a collector current (I_{CBO}) and measures the collector-base breakdown voltage (V_{CB}) with the emitter open.

Usage

```
double bvcbo(int e, int b, int c, int sub, double ipgm, double vlim, char type);
```

e	Input	The emitter pin of the device
b	Input	The base pin of the device
c	Input	The collector pin of the device
sub	Input	The substrate pin of the device
ipgm	Input	The forced collector-base current (I_{CB}), in amperes
vlim	Input	The collector voltage limit, in volts
type	Input	Type of transistor: `'N'` or `'P'`
Returns	Output	Collector-base voltage: ▪ `-1.0` = TYPE not 'N' or 'P' ▪ `+2.0E + 21` = Voltage limit reached; measured voltage is within 98% of the specified voltage limit (*vlim*)

Details

If a positive substrate pin is specified, the substrate is grounded. If a positive substrate pin is not specified, the substrate is left floating.

A delay is incorporated into the `bvcbo` subroutine; this delay is the calculated time required for stable forcing of *ipgm* within the *vlim* voltage limit.

V/I polarities

The polarity of *ipgm* is determined by the device type.

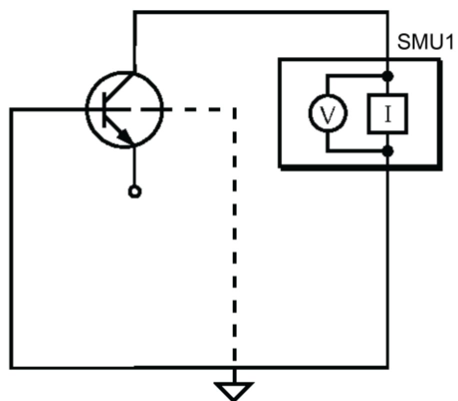
Source-measure units (SMUs)

- SMU1: Forces I_{CBO}, programmed voltage limit, measures `bvcbo`

Example

```
result = bvcbo(e, b, c, sub, ipgm, vlim, type);
```

Schematic



bvcbo1

This subroutine uses the `bsweepv` LPTLib function to measure collector-base breakdown voltage at a specified current with the emitter open.

Usage

```
double bvcbo1(int e, int b, int c, int sub, double vcbmin, double vcbmax, int nstep,
   double ipgm, double udelay, char type);
```

e	Input	The emitter pin of the device
b	Input	The base pin of the device
c	Input	The collector pin of the device
sub	Input	The substrate pin of the device
vcbmin	Input	The starting collector-base voltage (V_{CB}), in volts
vcbmax	Input	The ending V_{CB}, in volts
nstep	Input	The number of voltage steps
ipgm	Input	The targeted collector-base current (I_{CB}), in amperes
udelay	Input	Delay between V_{CB} steps, in seconds
type	Input	Type of transistor: `'N'` or `'P'`
Returns	Output	Collector-base voltage: ▪ `-1.0` = TYPE not 'N' or 'P' ▪ `+1.0E + 21` = Device triggered on *vcbmin* ▪ `+2.0E + 21` = Device triggered on *vcbmax*

Details

This subroutine sweeps the collector-base voltage from *vcbstart* to *vcbstop* while monitoring the collector current with the emitter open. When the programmed current level (*ipgm*) is reached, the last collector-base voltage increment is returned as `BVCBO1`.

If a positive substrate pin is specified, the substrate is grounded. If a positive substrate pin is not specified, the substrate is left floating.

The *udelay* parameter should be programmed to approximate `c` * *vcbmax* / *ipgm*, where `c` = junction capacitance of the device under test.

V/I polarities

The polarities of *vcbmin*, *vcbmax*, and *ipgm* are determined by device type.

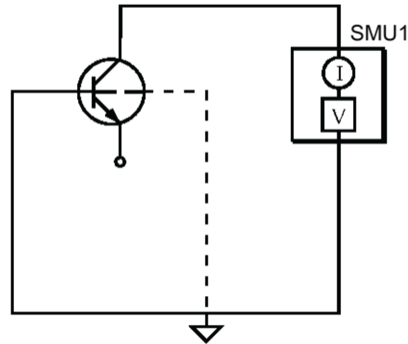
Source-measure units (SMUs)

- SMU1: Forces V_{CB}, programmed current limit = 1.25 * *ipgm*, measures collector current (I_{CBO})

Example

```
Result = bvcbo1(e, b, c, sub, vcbmin, vcbmax, nstep, ipgm, udelay, type);
```

Schematic



bvceo

This subroutine measures the collector-emitter breakdown voltage (V_{CE}) when the collector current (I_C) is forced with the base terminal left open.

Usage

```
double bvceo(int e, int b, int c, int sub, double ipgm, double vlim, char type);
```

e	Input	The emitter pin of the device
b	Input	The base pin of the device
c	Input	The collector pin of the device
sub	Input	The substrate pin of the device
ipgm	Input	The forced collector-emitter current (I_{CE}), in amperes
vlim	Input	The collector voltage limit, in volts
type	Input	Type of transistor: `'N'` or `'P'`
Returns	Output	Collector-emitter voltage: ■ `-1.0` = TYPE not 'N' or 'P' ■ `+2.0E + 21` = Voltage limit reached; measured voltage is within 98% of the specified voltage limit (*vlim*)

Details

If a positive substrate pin is specified, the substrate is grounded. If a positive substrate pin is not specified, the substrate is left floating.

A delay is incorporated into the `bvceo` subroutine; this delay is the calculated time required for stable forcing of *ipgm* within the *vlim* voltage limit.

V/I polarities

The polarity of *ipgm* is determined by the device type.

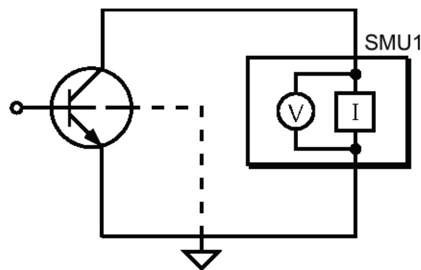
Source-measure units (SMUs)

- SMU1: Forces I_{CEO}, programmed voltage limit, measures `bvceo`

Example

```
result = bvceo(e, b, c, sub, ipgm, vlim, type);
```

Schematic



bvceo2

This subroutine measures collector-emitter breakdown voltage using the `bsweepV` LPTLib function.

Usage

```
double bvceo2(int e, int b, int c, int sub, double vcemin, double vcemax, int nstep,
    double ipgm, double udelay, char type);
```

e	Input	The emitter pin of the device
b	Input	The base pin of the device
c	Input	The collector pin of the device
sub	Input	The substrate pin of the device
vcemin	Input	The starting collector-emitter voltage (V_{CE}), in volts
vcemax	Input	The ending V_{CE}, in volts
nstep	Input	The number of voltage steps
ipgm	Input	The targeted collector-emitter current (I_{CE}), in amperes
udelay	Input	The delay between V_{CE} steps, in seconds
type	Input	Type of transistor: `'N'` or `'P'`
Returns	Output	Collector-emitter voltage: ▪ `-1.0` = TYPE not 'N' or 'P' ▪ `+1.0E + 21` = Device triggered on *vcemin* ▪ `+2.0E + 21` = Device triggered on *vcemax*

Details

This subroutine sweeps V_{CE} from *vcemin* to *vcemax* while monitoring the collector current with the base open. When the specified current level (*ipgm*) is reached, the last collector-emitter voltage increment is returned as `bvceo2`.

If a positive substrate pin is specified, the substrate is grounded. If a positive substrate pin is not specified, the substrate is left floating.

Set the *udelay* parameter to approximate `C * vcemax / ipgm`, where `C` = junction capacitance of the device under test.

V/I polarities

The polarities of *vcemin*, *vcemax*, and *ipgm* are determined by device type.

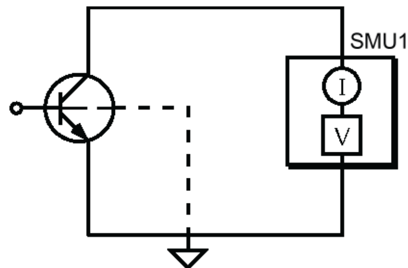
Source-measure units (SMUs)

- SMU1: Forces V_{CE}, programmed current limit = 1.25 **ipgm*, measures I_{CEO}

Example

```
result = bvceo2(e, b, c, sub, vcemin, vcemax, nstep, ipgm, udelay, type);
```

Schematic



bvces

This subroutine measures the collector-emitter/base breakdown voltage by forcing a collector current (I_{CES}).

Usage

```
double bvces(int e, int b, int c, int sub, double ipgm, double vlim, char type);
```

e	Input	The emitter pin of the device
b	Input	The base pin of the device
c	Input	The collector pin of the device
sub	Input	The substrate pin of the device
ipgm	Input	The forced I_{CES}, in amperes
vlim	Input	The collector voltage limit, in volts
type	Input	Type of transistor: `'N'` or `'P'`
Returns	Output	Collector-emitter/base voltage: ▪ `-1.0` = TYPE not 'N' or 'P' ▪ `+2.0E + 21` = Voltage limit reached; measured voltage is within 98% of the specified voltage limit (*vlim*)

Details

This subroutine measures collector-to-emitter breakdown voltage at a specified current with the base shorted to the emitter.

If a positive substrate pin is specified, the substrate will be grounded. If a positive substrate pin is not specified, it is left floating.

A delay is incorporated into the `bvces` subroutine; this delay is the calculated time required for stable forcing of *ipgm* within the *vlim* voltage limit.

V/I polarities

The polarity of *ipgm* is determined by the device type.

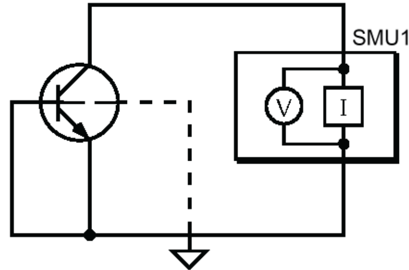
Source-measure units (SMUs)

- SMU1: Forces I_{CES}, programmed voltage limit, measures `bvces`

Example

```
result = bvces(e, b, c, sub, ipgm, vlim, type);
```

Schematic



bvces1

This subroutine measure the collector-emitter breakdown voltage using the `bsweepV` LPTLib function.

Usage

```
double bvces1(int e, int b, int c, int sub, double vcemin, double vcemax, int nstep,
    double ipgm, double udelay, char type);
```

e	Input	The emitter pin of the device
b	Input	The base pin of the device
c	Input	The collector pin of the device
sub	Input	The substrate pin of the device
vcemin	Input	The starting collector-emitter voltage (V_{CE}), in volts
vcemax	Input	The ending V_{CE}, in volts
nstep	Input	The number of voltage steps
ipgm	Input	The targeted collector-emitter current (I_{CE}), in amperes
udelay	Input	The delay between V_{CE} steps, in seconds
type	Input	Type of transistor: `'N'` or `'P'`
Returns	Output	Collector-emitter voltage: ▪ `-1.0` = TYPE not 'N' or 'P' ▪ `+1.0E + 21` = Device triggered on *vcemin* ▪ `+2.0E + 21` = Device triggered on *vcemax*

Details

This subroutine sweeps the collector-emitter voltage from *vcemin* to *vcemax* while monitoring the collector current with the base shorted to the emitter. When the programmed current level (*ipgm*) is reached, the last collector-emitter voltage increment is returned as `bvces1`.

If a positive substrate pin is specified, the substrate is grounded. If a positive substrate pin is not specified, the substrate is left floating.

Set the *udelay* parameter to approximate `C * vcemax / ipgm`, where `C` = junction capacitance of the device under test.

V/I polarities

The polarities of *vcemin*, *vcemax*, and *ipgm* are determined by device type.

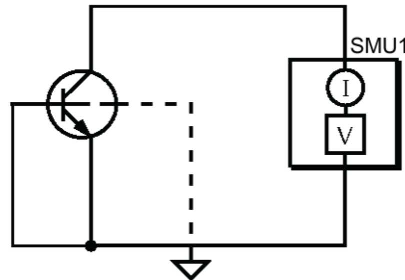
Source-measure units (SMUs)

- SMU1: Forces V_{CE}, programmed current limit = 1.25 **ipgm*, measures I_{CEO}

Example

```
result = bvces1(e, b, c, sub, vcemin, vcemax, nstep, ipgm, udelay, type);
```

Schematic



bvdss

This subroutine measure drain-source breakdown voltage ($V_G = 0$) when the gate is grounded with the source.

Usage

```
double bvdss(int d, int g, int s, int sub, double ipgm, double vlim);
```

d	Input	The drain pin of the device
g	Input	The gate pin of the device
s	Input	The source pin of the device
sub	Input	The substrate pin of the device
ipgm	Input	The forced drain current, in amperes
vlim	Input	The drain voltage limit, in volts
Returns	Output	Measured breakdown voltage: ▪ `+2.0E + 21` = Voltage limit reached; measured voltage is within 98% of the specified voltage limit (*vlim*)

Details

This subroutine measures the drain-to-source breakdown voltage of a field-effect transistor (FET) with the gate grounded with the source, at a specified current (magnitude and polarity).

If a positive substrate pin is specified, the substrate is grounded. If a positive substrate pin is not specified, the substrate is left floating.

A delay is incorporated into the bvdss subroutine; this delay is the calculated time required for stable forcing of *ipgm* within the *vlim* voltage limit.

V/I polarities

N-channel +Ipgm

P-channel -Ipgm

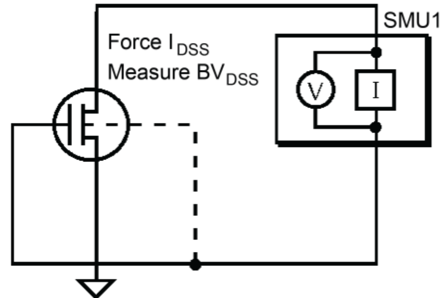
Source-measure units (SMUs)

- SMU1: Forces *ipgm*, programmed voltage limit, measures `bvdss`

Example

```
result = bvdss(d, g, s, sub, ipgm, vlim);
```

Schematic



bvdss1

This subroutine measures the drain-source breakdown voltage using the `bsweepv` LPTLib function.

Usage

```
double bvdss1 (int d, int g, int s, int sub, double vdsmin, double vdsmax, int nstep,
    double ipgm, double udelay, char type);
```

d	Input	The drain pin of the device
g	Input	The gate pin of the device
s	Input	The source pin of the device
sub	Input	The substrate pin of the device
vdsmin	Input	The starting drain-source voltage (V_{DS}), in volts
vdsmax	Input	The ending V_{DS}, in volts
nstep	Input	The number of voltage steps
ipgm	Input	Target drain-source current (V_{CS}), in amperes
udelay	Input	The delay between V_{DS} steps, in seconds
type	Input	Type of transistor: `'N'` or `'P'`
Returns	Output	Measured breakdown voltage: ■ `-1.0` = TYPE not 'N' or 'P' ■ +1.0E+21 = Device triggered on *vdsmin* ■ +2.0E+21 = Device triggered on *vdsmax*

Details

This subroutine sweeps the drain-source voltage from *vdsmin* to *vdsmax* while monitoring the drain current with the gate grounded to the source. When the specified current level (*ipgm*) is reached, the last drain-source voltage increment is returned as `bvdss1`.

If a positive substrate pin is specified, the substrate is grounded. If a positive substrate pin is not specified, the substrate is left floating.

Set the *udelay* parameter to approximate C * *vdsmax* / *ipgm*, where C = junction capacitance of the device under test.

V/I polarities

The polarities of *vdsmin*, *vdsmax*, and *ipgm* are determined by device type.

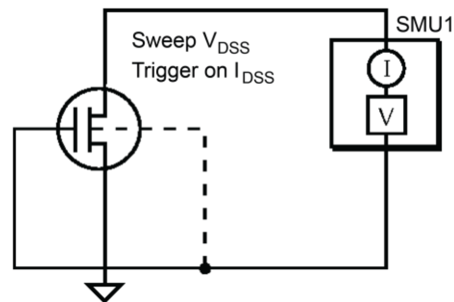
Source-measure units (SMUs)

- SMU1: Forces V_{DS}, programmed current limit = 1.25**ipgm*, measures I_{DS}

Example

```
result = bvdss1(d, g, s, sub, vdsmin, vdsmax, nstep, ipgm, udelay, type);
```

Schematic



bvebo

This subroutine measures emitter-base breakdown voltage at a specified current with the collector open.

Usage

```
double bvebo(int e, int b, int c, int sub, double ipgm, double vlim, char type);
```

e	Input	The emitter pin of the device
b	Input	The base pin of the device
c	Input	The collector pin of the device
sub	Input	The substrate pin of the device
ipgm	Input	The forced emitter-base current (I_{EB}), in amperes
vlim	Input	The emitter voltage limit, in volts
type	Input	Type of transistor: 'N' or 'P'
Returns	Output	Emitter-base voltage: ▪ -1.0 = TYPE not 'N' or 'P' ▪ +2.0E + 21 = Voltage limit reached; measured voltage is within 98% of the specified voltage limit (*vlim*)

Details

This subroutine measures the emitter-base breakdown voltage by forcing an emitter current with the collector pin open. Always call this subroutine last when testing transistors.

At high values of I_{EBO}, degradation of the emitter-base junction can occur, which will lower beta (β).

If a positive substrate pin is specified, the substrate is grounded. If a positive substrate pin is not specified, the substrate is left floating.

A delay is incorporated into the `bvebo` subroutine; this delay is the calculated time required for stable forcing of *`ipgm`* within the *`vlim`* voltage limit.

V/I polarities

The polarity of *`ipgm`* is determined by the device type.

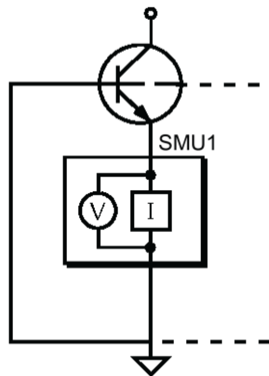
Source-measure units (SMUs)

- SMU1: Forces I_{EBO}, programmed voltage limit, measures `bvebo`

Example

```
result = bvebo(e, b, c, sub, ipgm, vlim, type);
```

Schematic



cap

This subroutine measures the capacitance of a two-terminal device.

Usage

```
double cap(int hi, int lo, int sub, double vbias);
```

hi	Input	The high pin of the device
lo	Input	The low pin of the device
sub	Input	The substrate pin of the device
vbias	Input	The voltage bias on the device, in volts
Returns	Output	Measured capacitance

Details

This subroutine measures the capacitance of a two-terminal capacitor at a specified voltage. The voltage is provided by the internal capacitance meter bias supply. The result is returned in farads.

If a positive substrate pin is specified, the substrate is grounded. If a positive substrate pin is not specified, the substrate is left floating.

When using this routine for junction capacitance measurements, make sure the junction never becomes forward biased. To prevent this, make sure the forward voltage is less than one-half the barrier potential (for silicon, this means that the forward voltage (V_F) should not exceed 300 mV to 350 mV).

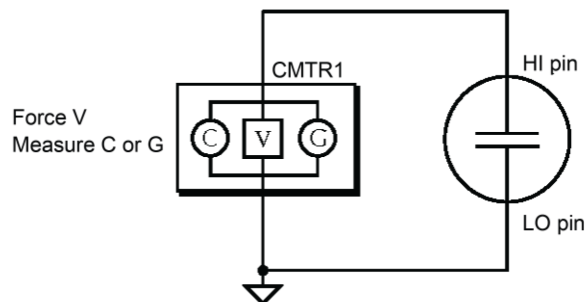
NOTE

At the onset of conduction in a forward-biased diode (V > 300 mV), the current flow causes unpredictable readings in the capacitance meter (usually overrange, 10^{22}).

Example

```
result = cap(hi, lo, sub, vbias);
```

Schematic



deltl1

This subroutine estimates MOSFET gate length reduction (ΔL) using transconductance (g_m) data obtained from the `vtext2` subroutine for two different transistors.

Usage

```
double deltl1(int d1, int g1, int s1, int sub1, double l1, int d2, int g2, int s2, int
    sub2, double l2, double vlow, double vhigh, double vds, double vbs, double ithr,
    double vstep, int npts, int *kflag)
```

d1	Input	The drain pin of the first transistor
g1	Input	The gate pin of the first transistor
s1	Input	The source pin of the first transistor
sub1	Input	The substrate pin of the first transistor
l1	Input	Drawn gate length of the first transistor, in microns
d2	Input	The drain pin of the second transistor
g2	Input	The gate pin of the second transistor
s2	Input	The source pin of the second transistor
sub2	Input	The substrate pin of the second transistor
l2	Input	Drawn gate length of the second transistor, in microns
vlow	Input	Start of the gate-source voltage (V_{GS}) search, in volts
vhigh	Input	End of the V_{GS} search, in volts
vds	Input	Drain bias, in volts
vbs	Input	Substrate bias, in volts
ithr	Input	Drain-source trigger current (I_{DS}), in amperes
vstep	Input	V_{GS} step size, in volts
npts	Input	Number of points in the V_{GS} sweep
kflag	Output	Returned status flag: ■ 0 = Normal completion ■ 1 = First g_m measurement failed ■ 2 = Second g_m measurement failed
Returns	Output	Estimated gate length reduction

Details

The *npts* parameter must be greater than 5. If a value less than 5 is used, the subroutine uses 5 points by default.

The equation used for this calculation is:

$\Delta L = ((Slope_1 / Slope_2) \times (L_1 - L_2) / (Slope_1 / Slope_2 - 1.0)$

Use this subroutine to infer the variability in the channel length based on the transconductance comparison of two devices, where the reference device is considerably larger than the second device.

Source-measure units (SMUs)

See the vtext2 (on page 3-85) subroutine.

Example

```
Result = deltl1(d1, g1, s1, sub1, l1, d2, g2, s2, sub2, l2, vlow, vhigh, vds,
       vbs, ithr, vstep, npts, &kflag)
```

deltw1

This subroutine estimates the gate width reduction parameter (ΔW) for a MOSFET using two values of threshold voltage (V_T) obtained from the `vtext2` subroutine.

Usage

```
double deltw1(int d1, int g1, int s1, int sub1, double w1, int d2, int g2, int s2, int
    sub2, double w2, double vlow, double vhigh, double vds, double vbs, double ithr,
    double vstep, int npts, int *kflag)
```

d1	Input	The drain pin of the first transistor
g1	Input	The gate pin of the first transistor
s1	Input	The source pin of the first transistor
sub1	Input	The substrate pin of the first transistor
w1	Input	The drawn gate width of the first transistor, in microns
d2	Input	The drain pin of the second transistor
g2	Input	The gate pin of the second transistor
s2	Input	The source pin of the second transistor
sub2	Input	The substrate pin of the second transistor
w2	Input	The drawn gate width of the second transistor, in microns
vlow	Input	Start of the gate-source voltage (V_{GS}) search, in volts
vhigh	Input	End of the V_{GS} search, in volts
vds	Input	Drain bias, in volts
vbs	Input	Substrate bias, in volts
ithr	Input	Drain-source trigger current (I_{DS}), in amperes
vstep	Input	V_{GS} step size, in volts
npts	Input	Number of points in the V_{GS} sweep
kflag	Output	Returned status flag: ■ 0 = Normal completion ■ 1 = First V_T measurement failed ■ 2 = Second V_T measurement failed
Returns	Output	Estimated gate width reduction

Details

ΔW is calculated using the following equation:

$\Delta L = ((Slope_2 / Slope_1) W_1 - W_2) / (Slope_2 / Slope_1 - 1.0)$

The *npts* parameter must be greater than 5. If a value less than 5 is used, the subroutine uses 5 points by default.

Source-measure units (SMUs)

See the vtext2 (on page 3-85) subroutine.

Example

```
result = deltw1(d1, g1, s1, sub1, w1, d2, g2, s2, sub2, w2, vlow, vhigh, vds,
      vbs, ithr, vstep, npts, &kflag)
```

ev

This subroutine calculates the early voltage at constant base-emitter current (IBE).

Usage

```
void ev(int e, int b, int c, int sub, double ibe, double vstart, double vstop, int npts,
    double vsub, double *slope, double *iflag, double *r, double *early);
```

e	Input	The emitter pin of the device
b	Input	The base pin of the device
c	Input	The collector pin of the device
sub	Input	The substrate pin of the device
ibe	Input	Forced base current, in amperes
vstart	Input	The start of the collector-emitter voltage (VCE) sweep
vstop	Input	The end of the VCE sweep
npts	Input	The number of points in the sweep
vsub	Input	Substrate bias, in volts
slope	Output	The calculated inductance
iflag	Output	Status flag: ■ 0 = Normal completion ■ 1 = No valid data for fit ■ 2 = Calculated slope = 0.0 ■ 3 = Developed base voltage is within 98% of the 3 V voltage limit
r	Output	Correlation coefficient
early	Output	Calculated early voltage

Details

This subroutine estimates the forward early voltage of a bipolar device at constant IBE. The device is connected in the common-emitter configuration, and a collector-emitter voltage (VCE) and collector-emitter current (ICE) data set is generated. A linear least squares (LLSQ) line is fit to the data, and the X-intercept is returned as the forward early voltage. The correlation coefficient is returned as an estimate of the fit.

When calling this routine, make sure the VCE start and stop values have the device well into saturation.

If a zero or negative substrate pin is specified, the substrate is left floating. If the pin number is greater than 0 and V_{SUB} is less than 0.9 mV, the substrate is grounded. In all other cases, it is connected and forced.

V/I polarities

NPN +VCE, +IBE and -VSUB

PNP -VCE, -IBE and -VSUB

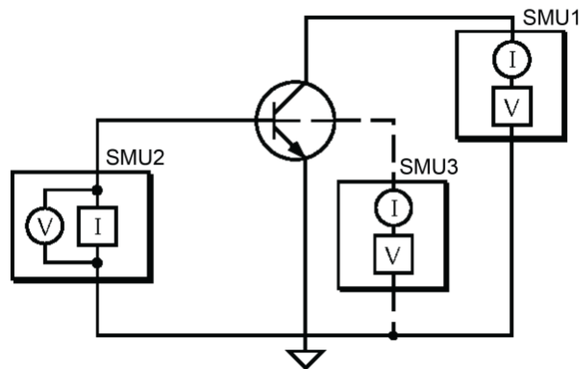
Source-measure units (SMUs)

- SMU1: Sweeps VCE, default current limit, measures ICE
- SMU2: Forces ibe, 3 V voltage limit
- SMU3: Forces vsub, default current limit

Example

```
ev(e, b, c, sub, ibe, vstart, vstop, npts, vsub, &slope, &iflag, &r, &early);
```

Schematic



fimv

This subroutine forces a current and measures a voltage on a device with four high (source) pins and four ground pins. This is an alternate version of the `fvmi` subroutine.

Usage

```
double fimv(int h1, int h2, int h3, int h4, int l1, int l2, int l3, int l4, double *v,
    double i);
```

h1	Input	High pin 1
h2	Input	High pin 2
h3	Input	High pin 3
h4	Input	High pin 4
l1	Input	Low pin 1
l2	Input	Low pin 2
l3	Input	Low pin 3
l4	Input	Low pin 4
v	Output	Measured voltage
i	Input	Forced current, in amperes
Returns	Output	Measured voltage: 0.0 = All high or low pins are <1

Details

Input a `-1` if the pin is not to be used.

A delay is incorporated into the `fimv` subroutine; this delay is the calculated time required for stable forcing of *i* with a 30 V voltage limit (default).

Source-measure units (SMUs)

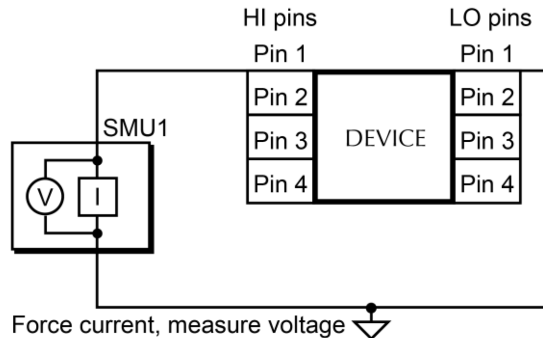
- SMU1: Forces current, default voltage limit, measures voltage

Example

```
result = fimv(h1, h2, h3, h4, l1, l2, l3, l4, &v, i);
```

Schematic

Figure 11: Schematic for the fimv subroutine



fnddat

This subroutine searches an array and returns a new array.

Usage

```
void fnddat(double *x, int npts, double *y, int npts1, double x1, double x2, double *xnew,
    int np1, double *ynew, int np2, int *np, char code)
```

x	Input	Input x array
npts	Input	Number of points in the input array
y	Input	Input y array
npts1	Input	Number of points in the input array
x1	Input	Minimum valid point
x2	Input	Maximum valid point
xnew	Output	Screened x array
np1	Input	Number of points in the output array
ynew	Output	Screened y array
np2	Input	Number of points in the output array
np	Output	Number of points in the output array
code	Input	Search "x" or "y" data array

Details

This subroutine searches a data set of x and y values for a specified range of data and returns two new arrays with the screened data. Use this routine in other routines to remove unwanted or bad points from measured data.

The *x*, *y*, *xnew*, and *ynew* parameters are all adjustable dimensioned arrays. The calling routine should dimension them all the same.

The *code* parameter searches either the x data or y data. For example, in most measurement routines, either the voltage or current is fixed, and the other variable is measured. The measured variable is normally what is screened.

Example

```
fnddat(&x, npts, &y, npts1, x1, x2, &xnew, np1, &ynew, np2, &np, code)
```

fndtrg

This subroutine determines which native mode trigger to use.

Usage

`int fndtrg(double low, double high)`

low	Input	The low value
high	Input	The high value
Returns	Output	`TRUE` = Use the Less Than trigger `FALSE` = Use the Greater Than trigger

Details

This subroutine compares the algebraic magnitudes of the input parameters and sets its return value `TRUE` if TRIGL should be used or `FALSE` if TRIGH should be used.

Example

```
result = fndtrg(low, high)
```

fvmi

This primitive subroutine forces a voltage and measures a current on a device with four input pins and four ground pins.

Usage

```
double fvmi(int h1, int h2, int h3, int h4, int l1, int l2, int l3, int l4, double v,
    double *i);
```

h1	Input	High pin 1
h2	Input	High pin 2
h3	Input	High pin 3
h4	Input	High pin 4
l1	Input	Low pin 1
l2	Input	Low pin 2
l3	Input	Low pin 3
l4	Input	Low pin 4
v	Input	Forced voltage, in volts
i	Output	Measured current
Returns	Output	Measured current: ▪ 0.0 = All high or low pins are < 1 ▪ +4.0E+21 = Measured voltage is within 98% of the default current limit

Details

This subroutine is normally used for defect structures with multiple high pins and ground pins.

Source-measure units (SMUs)

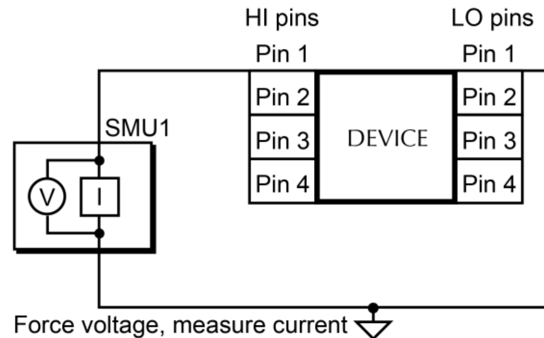
- SMU1: Forces voltage, default current limit, measures current

Example

```
result = fvmi(h1, h2, h3, h4, l1, l2, l3, l4, v, &i);
```

Schematic

Figure 12: Schematic for the fvmi subroutine



gamma1

This subroutine returns the value of the body effect parameter gamma obtained from two measurements of the threshold voltage (V_T) at different substrate bias voltages (V_{BS}).

Usage

```
double gamma1(int d, int g, int s, int sub, double vlow, double vhigh, double vds, double
    vbs1, double vbs2, double phip, double ithr, double vstep, int npts, int *kflag)
```

d	Input	The drain pin of the device
g	Input	The gate pin of the device
s	Input	The source pin of the device
sub	Input	The substrate pin of the device
vlow	Input	Low limit of the expected threshold
vhigh	Input	High limit of the expected threshold
vds	Input	Drain-source voltage, in volts
vbs1		First substrate to source voltage
vbs2		Second substrate to source voltage
phip		Surface potential, in volts
ithr		Drain-source trigger current (I_{DS}), in amperes
vstep		Gate-source voltage (V_{GS}) step size, in volts
npts		The number of points in the sweep
kflag	Output	Returned status flag: ■ 0 = Normal completion ■ 1 = First V_T measurement failed ■ 2 = second V_T measurement failed
Returns	Output	Estimated body effect

Details

This subroutine estimates body effect from V_T measured at two V_{BS} values. The body effect parameter characterizes the effect of the substrate bias on threshold voltage. The V_T data is obtained using the `vtext2` subroutine.

The equation used in this subroutine:

$\gamma = V_{T2}-V_{T1}/\sqrt{}V_{T1} + \phi_p - \sqrt{}(V_{T2}+\phi_p)$

Where:

- γ = MOSFET body effect constant
- V_{T1} = Threshold voltage at the first value of V_{BS}
- V_{T2} = Threshold voltage at the second value of V_{BS}
- ϕ_p = Surface potential (twice the Fermi level)

The *npts* parameter must be greater than 5. If a value less than 5 is used, the subroutine uses 5 points by default.

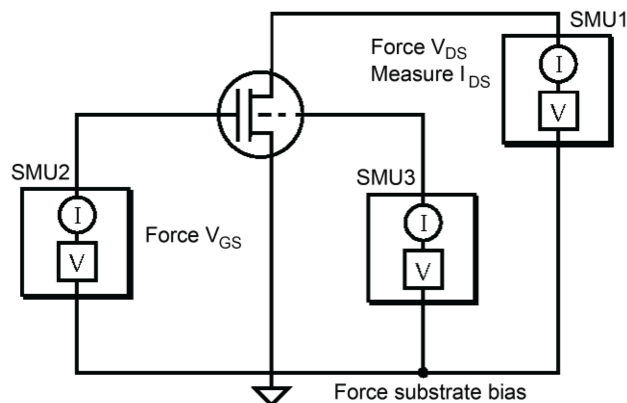
Source-measure units (SMUs)

See the vtext2 (on page 3-85) subroutine.

Example

```
result = gamma1(d, g, s, sub, vlow, vhigh, vds, vbs1, vbs2, phip, ithr, vstep,
   npts, &kflag)
```

Schematic



gd

This subroutine calculates the drain conductance of a MOSFET.

Usage

```
double gd(int d, int g, int s, int sub, double vds, double vgs, double vbs, double *ids)
```

d	Input	The drain pin of the device
g	Input	The gate pin of the device
s	Input	The source pin of the device
sub	Input	The substrate pin of the device
vds	Input	Drain-source voltage, in volts
vgs	Input	Gate voltage, in volts
vbs	Input	Substrate bias, in volts
ids	Output	The measured drain current
Returns	Output	The measured drain conductance

Details

This subroutine calculates the drain conductance (g_D) at drain-source voltage (V_{DS}), gate-source voltage (V_{GS}), and substrate bias voltage (V_{BS}) for a MOSFET.

The drain conductance is calculated by:

$g_D = I_{DS} / V_{DS}$

If a zero or negative substrate pin is specified, the substrate is left floating. If the pin number is greater than 0 and V_{BS} is less than 0.9 mV, the substrate is grounded. In all other cases, it is connected and forced.

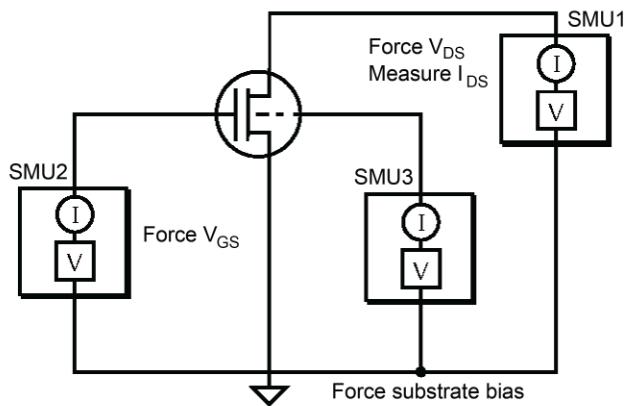
Source-measure units (SMUs)

- SMU1: Forces *vds*, default current limit, measures ids
- SMU2: Forces *vgs*, default current limit
- SMU3: Forces *vbs*, default current limit

Example

```
result = gd(d, g, s, sub, vds, vgs, vbs, &ids)
```

Schematic



gm

This subroutine estimates transconductance of a metal-semiconductor field-effect transistor (MESFET) at a specified drain voltage (V_{DS}) and gate voltage (V_{GS}).

Usage

```
double gm(int d, int g, int s, int sub, double vds, double idlim, double vgs, double
    vgstep, double iglim, int *iflag)
```

d	Input	The drain pin of the device
g	Input	The gate pin of the device
s	Input	The source pin of the device
sub	Input	The substrate pin of the device
vds	Input	Drain voltage, in volts
idlim	Input	Drain current limit, in amperes
vgs	Input	Gate voltage, in volts
vgstep	Input	V_{GS} step size, in volts
iglim	Input	Gate current limit, in amperes
iflag	Output	Return status flag: ■ 0 = Normal completion ■ 1 = Not enough valid data for LLSQ ■ 2 = Current limit reached (98% of *iglim* or *idlim*)
Returns	Output	Estimated MESFET transconductance

Details

This subroutine estimates the transconductance of a MESFET at a specified V_{DS} and V_{GS}. A drain voltage is forced, and then five V_{GS} to I_{DS} (drain-source current) data points are taken around the specified V_{GS} (the V_{GS} step size is defined by the input parameter *vgstep*). Then a linear least squares (LLSQ) line is fit through the data and the transconductance is estimated from the slope of the line.

V/I polarities

N-channel $+V_{DS}$, $+V_{GS}$

P channel $-V_{DS}$, $-V_{GS}$

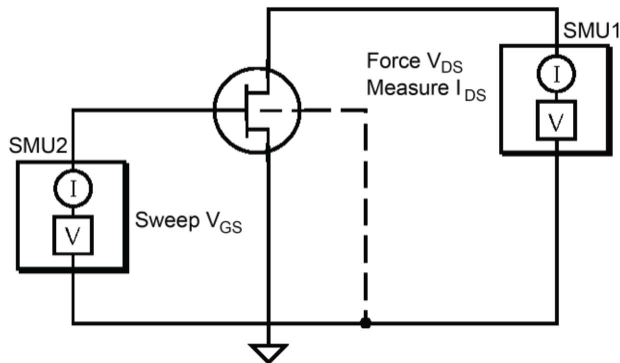
Source-measure units (SMUs)

- SMU1: Forces *vds*, programmable current limit, measures I_{DS}
- SMU2: Sweeps *vgs*, programmable current limit

Example

```
result = gm(d, g, s, sub, vds, idlim, vgs, vgstep, iglim, &iflag)
```

Schematic



ibic1

This subroutine measures collector current (I_{CE}) and base current (I_B) and calculates beta (β) at a fixed collector voltage (V_{CE}), base voltage (V_{BE}), and substrate bias (V_{SUB}). The device is in the common-emitter configuration.

Usage

```
void ibic1(int e, int b, int c, int sub, double vce, double vbe, double vsub, double
    *ibe, double *ice, double *beta)
```

e	Input	The emitter pin of the device
b	Input	The base pin of the device
c	Input	The collector pin of the device
sub	Input	The substrate pin of the device
vce	Input	Forced collector voltage, in volts
vbe	Input	Forced base voltage, in volts
vsub	Input	The forced substrate bias, in volts
ibe	Output	Measured base current: ▪ 4.0E+21 = Current limit reached; measured current is within 98% of the 200 mA limit. β is returned as 0.0
ice	Output	Measured collector current: ▪ 4.0E+21 = Current limit reached; measured current is within 98% of the 200 mA limit. β is returned as 0.0
beta	Output	Current gain, collector current (I_C) / base current (I_B)

Details

If a zero or negative substrate pin is specified, the substrate is left floating. If the pin number is greater than 0 and V_{SUB} is less than 0.9 mV, the substrate is grounded. In all other cases, it is connected and forced.

V/I polarities

NPN $+V_{BE}$, $+V_{CE}$, and $-V_{SUB}$

PNP $-V_{BE}$, $-V_{CE}$, and $-V_{SUB}$

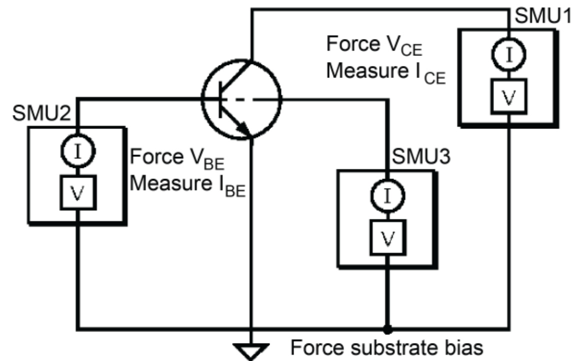
Source-measure units (SMUs)

- SMU1: Forces `vce`, maximum current limit, measures `ice`
- SMU2: Forces `vbe`, maximum current limit, measures `ibe`
- SMU3: Forces `vsub`, default current limit

Example

```
ibic1(e, b, c, sub, vce, vbe, vsub, &ibe, &ice, &beta)
```

Schematic



icbo

This subroutine measures leakage when the collector-base junction is reverse-biased (common base).

Usage

```
double icbo(int e, int b, int c, int sub, double vcbo, double vsub)
```

e	Input	The emitter pin of the device
b	Input	The base pin of the device
c	Input	The collector pin of the device
sub	Input	The substrate pin of the device
vcbo	Input	Forced collector-base voltage, in volts
vsub	Input	The forced substrate bias, in volts
Returns	Output	The measured collector-base current

Details

This subroutine measures the collector-base leakage current at a specified collector-base voltage (V_{CB}) and substrate bias (V_{SUB}) for a bipolar transistor. The emitter pin is not connected (floating), and the base is grounded.

If a zero or negative substrate pin is specified, the substrate is left floating. If the pin number is greater than 0 and V_{SUB} is less than 0.9 mV, the substrate is grounded. In all other cases, it is connected and forced.

V/I polarities

NPN $+V_{CB}$, $-V_{SUB}$

PNP $-V_{CB}$, $-V_{SUB}$

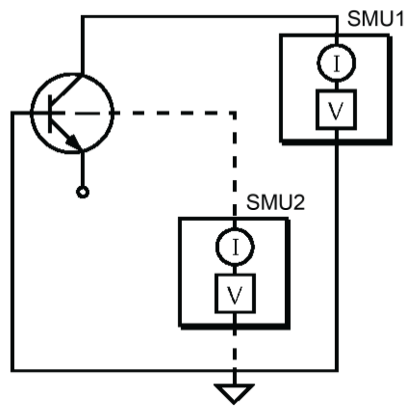
Source-measure units (SMUs)

- SMU1: Forces V_{CB}, default current limit, measures `icbo`
- SMU2: Forces *vsub*, default current limit

Example

```
result = icbo(e, b, c, sub, vcbo, vsub)
```

Schematic



iceo

This subroutine measures collector-emitter leakage at collector voltage (V_{CE}) and substrate bias (V_{SUB}).

Usage

```
double iceo(int e, int b, int c, int sub, double vce, double vsub)
```

e	Input	The emitter pin of the device
b	Input	The base pin of the device
c	Input	The collector pin of the device
sub	Input	The substrate pin of the device
vce	Input	The forced collector-emitter voltage, in volts
vsub	Input	The forced substrate bias, in volts
Returns	Output	The measured leakage current

Details

This subroutine forces a V_{CE} and V_{SUB} and measures the leakage current. The base terminal is open and the emitter is grounded.

If a zero or negative substrate pin is specified, the substrate is left floating. If the pin number is greater than 0 and V_{SUB} is less than 0.9 mV, the substrate is grounded. In all other cases, it is connected and forced.

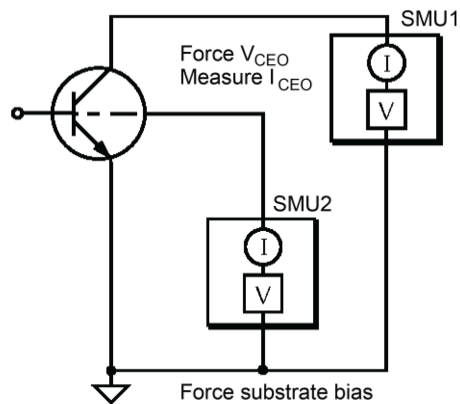
Source-measure units (SMUs)

- SMU1: Forces *vce*, default current limit, measures `iceo`
- SMU2: Forces *vsub*, default current limit

Example

```
result = iceo(e, b, c, sub, vce, vsub)
```

Schematic



ices

This subroutine measures collector-emitter/base leakage when the collector-base junction is reverse-biased (common base).

Usage

```
double ices(int e, int b, int c, int sub, double vces, double vsub)
```

e	Input	The emitter pin of the device
b	Input	The base pin of the device
c	Input	The collector pin of the device
sub	Input	The substrate pin of the device
vces	Input	The forced collector-emitter voltage, in volts
vsub	Input	Substrate bias, in volts
Returns	Output	The measured collector-emitter/base leakage current

Details

This subroutine measures the collector-emitter/base leakage at a specified collector-emitter voltage (V_{CES}) and substrate bias (V_{SUB}). The base and emitter terminals are shorted to ground.

If a zero or negative substrate pin is specified, the substrate is left floating. If the pin number is greater than 0 and V_{SUB} is less than 0.9 mV, the substrate is grounded. In all other cases, it is connected and forced.

V/I polarities

NPN $+V_{CES}$, $-V_{SUB}$

PNP $-V_{CES}$, $-V_{SUB}$

Source-measure units (SMUs)

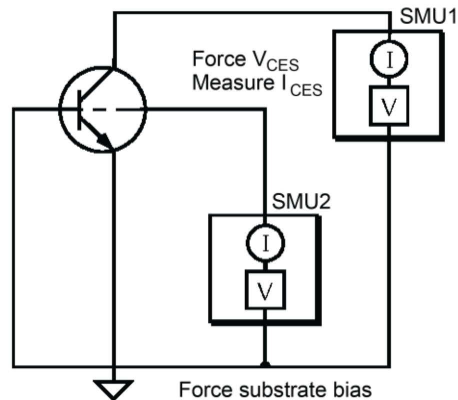
SMU1: Forces *vces*, default current limit, measures `ices`

SMU2: Forces *vsub*, default current limit

Example

```
result = ices(e, b, c, sub, vces, vsub)
```

Schematic



id1

This subroutine measures drain current (I_{DS}) at a specified gate-source voltage (V_{GS}), drain-source voltage (V_{DS}), and substrate-source voltage (V_{BS}).

Usage

```
double id1(int d, int g, int s, int sub, double vgs, double vds, double vbs)
```

d	Input	The drain pin of the device
g	Input	The gate pin of the device
s	Input	The source pin of the device
sub	Input	The substrate pin of the device
vgs	Input	The forced gate voltage, in volts
vds	Input	Drain voltage, in volts
vbs	Input	Substrate bias, in volts
Returns	Output	The measured drain current

Details

If a zero or negative substrate pin is specified, the substrate is left floating. If the pin number is greater than 0 and V_{BS} is less than 0.9 mV, the substrate is grounded. In all other cases, it is connected and forced.

V/I polarities

N-channel $+V_{DS}$, $+V_{GS}$, $-V_{BS}$

P-channel $-V_{DS}$, $-V_{GS}$, $+V_{BS}$

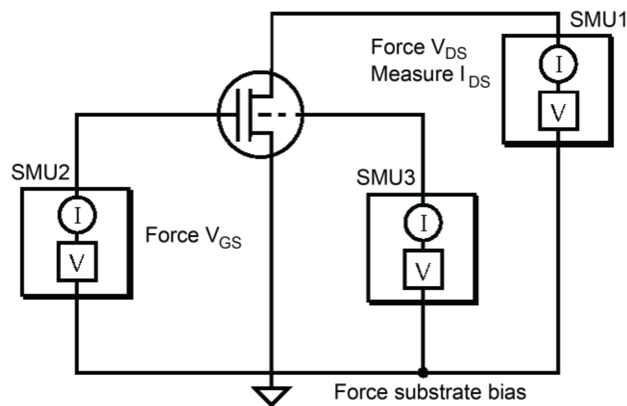
Source-measure units (SMUs)

- SMU1: Forces *vds*, default current limit, measures I_{DS}
- SMU2: Forces *vgs*, default current limit
- SMU3: Forces *vbs*, default current limit

Example

```
result = id1(d, g, s, sub, vgs, vds, vbs)
```

Schematic



idsat

This subroutine measures drain-source current (I_{DS}) at a specified drain-source voltage (V_{DS}) and substrate-source voltage (V_{BS}). The gate is tied to the drain.

Usage

```
double idsat(int d, int g, int s, int sub, double vds, double vbs)
```

d	Input	The drain pin of the device
g	Input	The gate pin of the device
s	Input	The source pin of the device
sub	Input	The substrate pin of the device
vds	Input	Drain-source voltage, in volts
vbs	Input	Substrate bias, in volts
Returns	Output	The measured drain current

Details

If a zero or negative substrate pin is specified, the substrate is left floating. If the pin number is greater than 0 and V_{BS} is less than 0.9 mV, the substrate is grounded. In all other cases, it is connected and forced.

V/I polarities

N-channel $+V_{DS}$, $-V_{BS}$

P-channel $-V_{DS}$, $+V_{BS}$

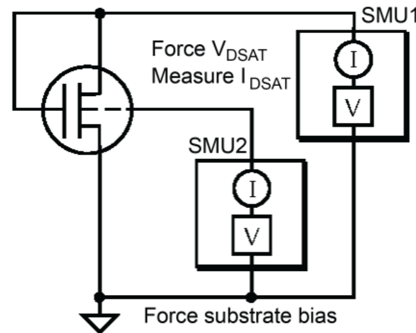
Source-measure units (SMUs)

- SMU1: Forces `vds`, default current limit, measures I_{DS}
- SMU2: Forces `vbs`, default current limit

Example

```
result = idsat(d, g, s, sub, vds, vbs)
```

Schematic



idss

This subroutine estimates the saturated drain current (I_{DSS}) and saturation voltage (V_{DSAT}) at forced drain voltage (V_{DSS}) for a metal-semiconductor field effect transistor (MESFET).

Usage

```
double idss(int d, int g, int s, int sub, double vdss, double idlim, double f, double
    *idsat, double *vdsat)
```

d	Input	The drain pin of the device
g	Input	The gate pin of the device
s	Input	The source pin of the device
sub	Input	The substrate pin of the device
vdss	Input	The forced drain voltage, in volts
idlim	Input	Drain current limit, in amperes
f	Input	Fraction of I_{DSS}
idsat	Output	Target saturation current
vdsat	Output	Saturation voltage
Returns	Output	Measured drain current: ▪ 0.0 = If *f* ≤ 0.0 or > 1.0 ▪ 2.0E+21 = If measured `vdsat` is within 98% of the gate voltage limit ▪ 4.0E+21 = If `idss` is within 98% of specified drain current limit (*idlim*)

Details

This subroutine measures the drain current of a field effect transistor (FET) when the gate is shorted to the source at a specified drain voltage (V_{DS}). It also estimates the V_{DSAT} by measuring I_{DSS} and then finding the V_{DS} that forces a fraction of I_{DSS} (usually 0.9).

If a positive substrate pin is specified, the substrate is grounded. If a positive substrate pin is not specified, the substrate is left floating.

The `f` parameter is normally set to 0.9.

A delay is included in the `idss` subroutine; this delay is the calculated time required for stable forcing of drain current with a 30 V voltage limit.

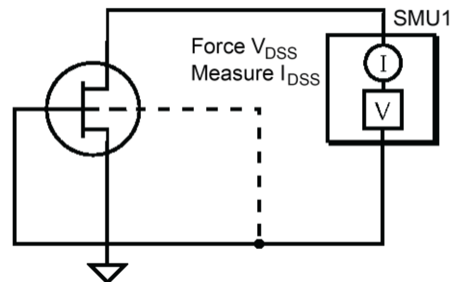
Source-measure units (SMUs)

- SMU1: Forces *vdss*, programmable current limit, measures `idss`

Example

```
result = idss(d, g, s, sub, vdss, idlim, f, &idsat, &vdsat)
```

Schematic



idvsvd

This subroutine returns an array of drain-source current (I_{DS}) and drain-source voltage (V_{DS}) values for a given V_{DS} sweep range, where gate to source bias (V_{GS}) and substrate to source voltage (V_{BS}) are held constant.

Usage

```
void idvsvd(int d, int g, int s, int sub, double vlow, double vhigh, double vgs, double
   vbs, int npts, double *id, int idSize, double *vd, int vdSize);
```

d	Input	The drain pin of the device
g	Input	The gate pin of the device
s	Input	The source pin of the device
sub	Input	The substrate pin of the device
vlow	Input	The start of the V_{DS} sweep, in volts
vhigh	Input	The end the V_{DS} sweep, in volts
vgs	Input	The gate bias, in volts
vbs	Input	Substrate bias, in volts
npts	Input	The number of points in the sweep
id	Output	The array of measured I_{DS} values
idSize	Input	The size of the I_D array
vd	Output	The array of forced V_{DS} values
vdSize	Input	The size of the V_D array

Details

If a zero or negative substrate pin is specified, the substrate is left floating. If the pin number is greater than 0 and V_{BS} is less than 0.9 mV, the substrate is grounded. In all other cases, it is connected and forced.

The value of the *npts*, *idSize*, and *vdSize* parameters must be the same.

V/I polarities

N-channel $+V_{low}$, $+V_{high}$, $+V_{GS}$, $-V_{BS}$

P-channel $-V_{low}$, $-V_{high}$, $-V_{GS}$, $+V_{BS}$

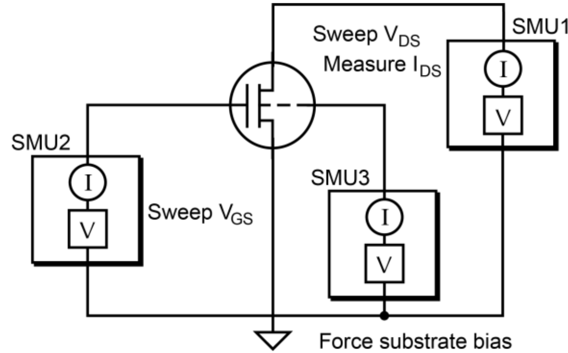
Source-measure units (SMUs)

- SMU1: Sweeps V_{DS}, maximum current limit, measures I_{DS}
- SMU2: Forces *vgs*, maximum current limit
- SMU3: Forces *vbs*, default current limit

Example

```
idvsvd(d, g, s, sub, vlow, vhigh, vgs, vbs, npts, &id, idSize, &vd, vdSize);
```

Schematic



idvsvg

This subroutine measures drain-source current (I_{DS}) when gate-source voltage (V_{GS}) is swept and drain-source voltage (V_{DS}) and forced substrate bias voltage (V_{BS}) are held constant.

Usage

```
void idvsvg(int d, int g, int s, int sub, double vlow, double vhigh, double vds, double
    vbs, int npts, double *id, int idSize, double *vg, int vgSize);
```

d	Input	The drain pin of the device
g	Input	The gate pin of the device
s	Input	The source pin of the device
sub	Input	The substrate pin of the device
vlow	Input	The start of the V_{GS} sweep, in volts
vhigh	Input	The end the V_{GS} sweep, in volts
vds	Input	The forced drain voltage, in volts
vbs	Input	Substrate bias, in volts
npts	Input	The number of points in the sweep
id	Output	The array of measured I_{DS} values
idSize	Input	The size of the I_D array
vg	Output	The array of calculated V_{GS} values
vgSize	Input	The size of the V_G array

Details

If a zero or negative substrate pin is specified, the substrate is left floating. If the pin number is greater than 0 and V_{BS} is less than 0.9 mV, the substrate is grounded. In all other cases, it is connected and forced.

V/I polarities

N-channel $+V_{low}$, $+V_{high}$, $+V_{DS}$, $-V_{BS}$

P-channel $-V_{low}$, $-V_{high}$, $-V_{DS}$, $-V_{BS}$

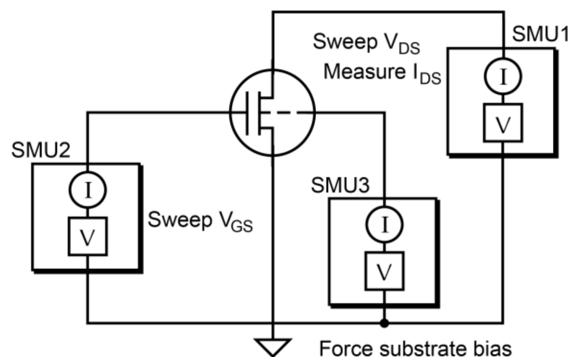
Source-measure units (SMUs)

- SMU1: Forces *vds*, maximum current limit, measures I_{DS}
- SMU2: Sweeps V_{GS}, maximum current limit
- SMU3: Forces *vbs*, default current limit

Example

```
idvsvg(d, g, s, sub, vlow, vhigh, vds, vbs, npts, &id, idSize, &vg, vgSize);
```

Schematic



iebo

This subroutine measures the reverse-bias leakage current through the emitter-base diode of a bipolar transistor with the base grounded and collector terminal floating.

Usage

```
double iebo(int e, int b, int c, int sub, double vebo, double vsub)
```

e	Input	The emitter pin of the device
b	Input	The base pin of the device
c	Input	The collector pin of the device
sub	Input	The substrate pin of the device
vebo	Input	Emitter-base voltage, in volts
vsub	Input	Substrate bias, in volts
Returns	Output	Reverse-bias leakage current: ■ +4.0E+21 = Current limit reached, measured current is within 98% of the 1 mA limit

Details

If a zero or negative substrate pin is specified, the substrate is left floating. If the pin number is greater than 0 and V_{SUB} is less than 0.9 mV, the substrate is grounded. In all other cases, it is connected and forced.

V/I polarities

NPN $+V_{EB}$ and $-V_{SUB}$

PNP $-V_{EB}$ and $-V_{SUB}$

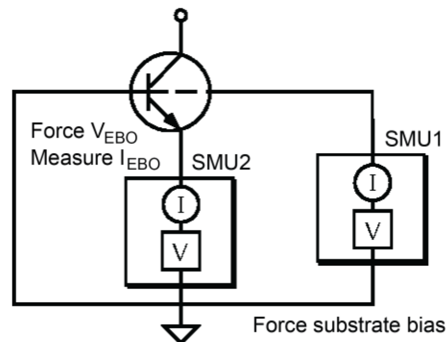
Source-measure units (SMUs)

- SMU1: Forces *vebo*, 1 mA current limit, measures leakage current
- SMU2: Forces *vsub*, default current limit

Example

```
result = iebo(e, b, c, sub, vebo, vsub)
```

Schematic



isubmx

This subroutine finds peak substrate current at drain-source voltage (V_{DS}) and finds substrate bias voltage (V_{BS}).

Usage

```
void isubmx(int d, int g, int s, int sub, double vds, double vbs, double vlow, double
    vhigh, int npts, double *ismax, double *vgmax)
```

d	Input	The drain pin of the device
g	Input	The gate pin of the device
s	Input	The source pin of the device
sub	Input	The substrate pin of the device
vds	Input	The forced drain voltage, in volts
vbs	Input	Substrate bias, in volts
vlow	Input	The start of the gate voltage (V_{GS}) sweep
vhigh	Input	The end of the V_{GS} sweep
npts	Input	The number of points in the sweep
ismax	Output	Peak substrate current: ▪ -1.0 = Substrate not specified ▪ +4.0E+21 = Measured drain current (I_{DS}) is within 98% of the drain current limit (200 mA)
vgmax	Output	V_{GS} at `ismax`

Details

This subroutine measures the substrate current when the gate voltage (V_{GS}) is swept with V_{DS} and V_{BS} held constant. Maximum current measured is returned as the function result. The gate voltage at maximum current is also returned. In some cases, peak substrate current is referred to as I_{SX}.

A substrate connection is mandatory; the *ismax* parameter returns -1.0 if the *sub* parameter is less than 1.

The typical value for the *npts* parameter is 10 to 20.

V/I polarities

N-channel $+V_{DS}$, $+V_{low}$ and V_{high}, $-V_{BS}$

P-channel $-V_{DS}$, $-V_{low}$ and V_{high}, $+V_{BS}$

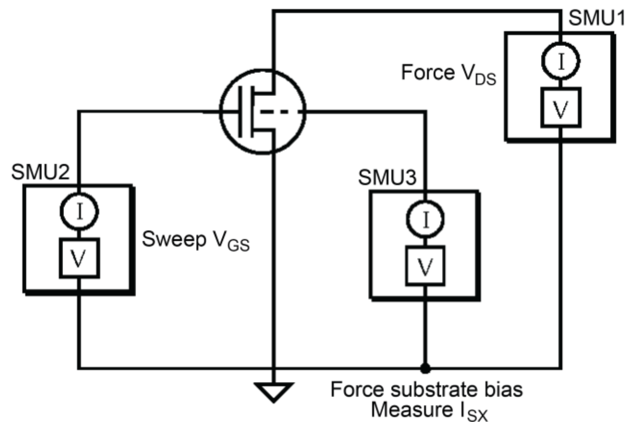
Source-measure units (SMUs)

- SMU1: Forces *vds*, maximum current limit
- SMU2: Sweeps V_{GS}, default current limit
- SMU3: Forces *vbs*, default current limit, measures I_{SX}

Example

```
isubmx(d, g, s, sub, vds, vbs, vlow, vhigh, npts, &ismax, &vgmax)
```

Schematic



kdelay

This subroutine provides an appropriate delay time based on current and voltage values.

Usage

```
void kdelay(int npin, double i, double v)
```

npin	Input	The number of pins connected to the charging node
i	Input	The current, in amperes
v	Input	The voltage, in volts

Details

This subroutine provides an appropriate delay time for a current source to reach a specified voltage by using an equation that accounts for the system capacitance, leakage currents, and number of pins to which the source is connected. The linear capacitance charging equation used by this subroutine:

IDELAY = 1 ms + ABS(*npin* * CDELAY * *v*) / MAX((ABS(*i*)-ILEAK), ILEAK) * 1000 ms

Where:

- IDELAY = The calculated required delay, in milliseconds
- *npin* = The number of pins connected to the charging node
- CDELAY = A constant representing the capacitance of the system
- *v* = The voltage, in volts
- *i* = The current, in amps
- ILEAK = The leakage current

The `kdelay` subroutine defaults to 1 ms for calculated delays less than 1 ms; it defaults to 30 s for calculated delays greater than 30 s.

Example

```
kdelay(npin, i, v)
```

leak

This subroutine measures leakage current of a two-terminal device (diode) at a specified voltage.

Usage

```
double leak(int hi, int lo, int sub, double v, double ilim)
```

hi	Input	The HI pin of the device (anode)
lo	Input	The LO pin of the device (cathode)
sub	Input	The substrate pin of the device
v	Input	The forced voltage, in volts
ilim	Input	The current limit, in amperes
Returns	Output	Measured leakage current: ▪ +4.0E+21 = Measured current is within 98% of the specified current limit

Details

This subroutine measures the leakage current by forcing a specified voltage and measuring the resulting current flow.

If a positive substrate pin is specified, the substrate is grounded. If a positive substrate pin is not specified, the substrate is left floating.

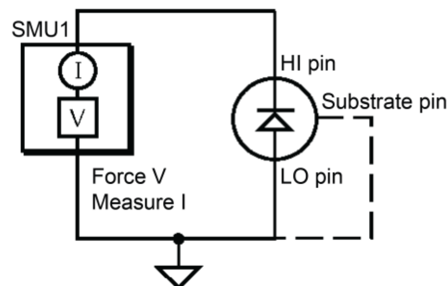
Source-measure units (SMUs)

SMU1: Forces *v*, programmable current limit, measures current

Example

```
result = leak(hi, lo, sub, v, ilim)
```

Schematic



logstp

This subroutine creates an array using logarithmic steps.

Usage

```
int logstp(double xstart, double xstop, double *steps, int npts)
```

xstart	Input	The start point of the sweep
xstop	Input	The end point of the sweep
steps	Output	The step array
npts	Input	The number of steps in the sweep
Returns	Output	The valid range status flag: ■ 1 = The *xstart* and *xstop* parameters are valid ■ 0 = Limits cross zero or equal 0.0

Details

This subroutine creates an array of logarithmic-based steps from an input range (*xstart* and *xstop*) and the number of steps (*npts*). The array of values is returned in the *steps* output parameter.

The logstp subroutine is often used instead of the `sweepi` native-mode subroutine call. The `sweepi` subroutine uses linear-based steps, which should not be used when sweeping current across more than three decades of current. Many of the bipolar routines that collect beta-I_{CE} type data use the `logstp` subroutine to calculate the proper current values to force.

The sweep range cannot cross 0.0.

Negative sweep start and stop points generate an array of negative numbers.

Example

```
result = logstp(xstart, xstop, &steps, npts)
```

rcsat

This subroutine estimates the collector resistance (R_C) modeling parameter when collector current (I_C) and base current (I_B) are swept at a constant beta (β).

Usage

```
double rcsat(int e, int b, int c, int sub, double ice1, double ice2, double beta, double
    vsub, int npts, double *r, int *iflag)
```

e	Input	The emitter pin of the device
b	Input	The base pin of the device
c	Input	The collector pin of the device
sub	Input	The substrate pin of the device
ice1	Input	The start of the collector-emitter current (I_{CE}) sweep, in amperes
ice2	Input	The end of the I_{CE} sweep, in amperes
beta	Input	The current ratio, I_C/I_B
vsub	Input	The forced substrate bias, in volts
npts	Input	The number of points in the sweep
r	Output	The correlation coefficient
iflag	Output	The status flag: ▪ 0 = Normal completion ▪ 1 = Insufficient points for LLSQ analysis ▪ 2 = Calculated LLSQ slope is 0.0
Returns	Output	The collector resistance modeling parameter

Details

This subroutine estimates the modeling parameter RC in the saturation region of a transistor using Getreu's method (Ian Getreu, *Modeling the Bipolar Transistor*, Tektronix, 1976). Current is stepped into the base and collector at a specified β (normally 10). The developed collector-emitter voltage (V_{CE}) is measured. V_{CE} and I_{CE} data is extracted from the positive slope portion of the curve, and a linear least-squares (LLSQ) line is fit to the data. The inverse of the slope of this line gives R_{CSAT}. The device is in the common-emitter configuration.

For high-speed or microwave bipolar devices, best results are obtained by starting at the maximum current and sweeping the current to a lower value.

An incorrect value of β can result in a large excursion of V_{CE}, which may break down the device. Because of this, the collector voltage is limited to 16 V.

To measure $R_{C\,SAT}$ correctly, make sure the transistor is saturated. You can do this by entering a smaller than actual value for β (overdriving the base).

Make sure the collector current range selected is not near the bend in the I_B-V_{CE} curve (knee region of the curve). If operated within this region, the `rcsat` subroutine may return negative or unpredictable results.

Two delays are incorporated into the `rcsat` subroutine; these delays calculate the time required for a stable forcing of base-emitter current (I_{BE}) with a 3 V emitter voltage limit and I_{CE} with a 16 V voltage limit.

If a zero or negative substrate pin is specified, the substrate is left floating. If the pin number is greater than 0 and V_{SUB} is less than 0.9 mV, the substrate is grounded. In all other cases, it is connected and forced.

Typical value for the `beta` parameter is 10.

V/I polarities

NPN $+I_{CE}$ and $-V_{SUB}$

PNP $-I_{CE}$ and $-V_{SUB}$

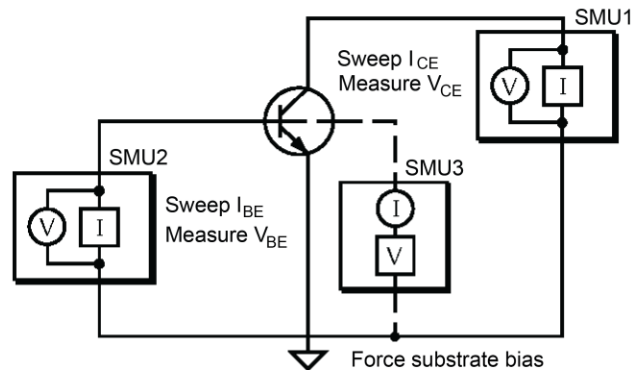
Source-measure units (SMUs)

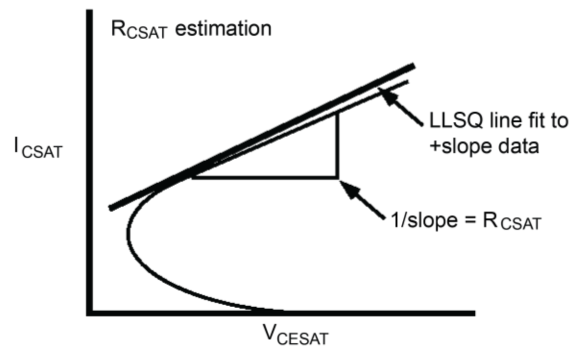
- SMU1: Sweeps I_{CE}, 16 V voltage limit, measures V_{CESAT}
- SMU2: Sweeps I_{CE}/β, 3 V voltage limit
- SMU3: Forces `vsub`, default current limit

Example

```
result = rcsat(e, b, c, sub, ice1, ice2, beta, vsub, npts, &r, &iflag)
```

Schematic





re

This subroutine estimates emitter resistance (R_E).

Usage

```
double re(int e, int b, int c, int sub, double ib1, double ib2, double vsub, int npts,
    int *iflag, double *r)
```

e	Input	The emitter pin of the device
b	Input	The base pin of the device
c	Input	The collector pin of the device
sub	Input	The substrate pin of the device
ib1	Input	The start of the base-emitter current (I_{BE}) sweep, in amperes
ib2	Input	The end of the I_{BE} sweep, in amperes
vsub	Input	Substrate bias, in volts
npts	Input	The number of points in the sweep
iflag	Output	The status flag: ■ 0 = Normal completion ■ 1 = Insufficient points for LLSQ analysis ■ 2 = Calculated LLSQ slope is 0.0 ■ 3 = Bad range specified in the call to the `logstp` subroutine
r	Output	The correlation coefficient
Returns	Output	The estimated emitter resistance

Details

This subroutine uses Getreu's method (Ian Getreu, *Modeling the Bipolar Transistor*, Tektronix, 1976) to estimate the emitter resistance modeling parameter. This routine should be used with caution because the value returned may include some parasitic values. The technique sweeps base current and measures the open (floating) developed collector voltage.

The subroutine assumes the device is in saturation. For a device in saturation, with the collector open, Getreu gives:

$V_{CE}=kT/q\ln(1/\alpha_R)+I_B R_E$

Where:

- α_R = Reverse emitter efficiency
- kT/q = 25.96 mV at 300 K
- I_B = The base current
- R_E = The emitter current
- V_{CE} = The collector-emitter voltage

Plotting I_{BE} versus V_{CE} gives the slope as R_E. The sample plot shows the typical V_{CE} - I_{BE} characteristic. To calculate R_E, the V_{CE} - I_{BE} data is analyzed for positive slope data, and a linear least-squares (LLSQ) line is fit to the extracted data. The emitter resistance is then the inverse of the calculated slope. The result is returned in ohms.

The I_B versus V_{CE} curve has a flyback region where V_{CE} decreases as I_B increases (the curve has a negative slope). The `re` subroutine drops all points with a negative slope in its calculation of the emitter resistance. An error code, "IFLAG=1" is generated if there are too few remaining points to continue the calculation of `re`.

A delay is incorporated into the `re` subroutine; this delay is the calculated time required for stable forcing of I_{BE} with a 30 V voltage limit.

If a zero or negative substrate pin is specified, the substrate is left floating. If the pin number is greater than 0 and V_{SUB} is less than 0.9 mV, the substrate is grounded. In all other cases, it is connected and forced.

V/I polarities

NPN $+I_{BE}$ and $-V_{SUB}$

PNP $-I_{BE}$ and $-V_{SUB}$

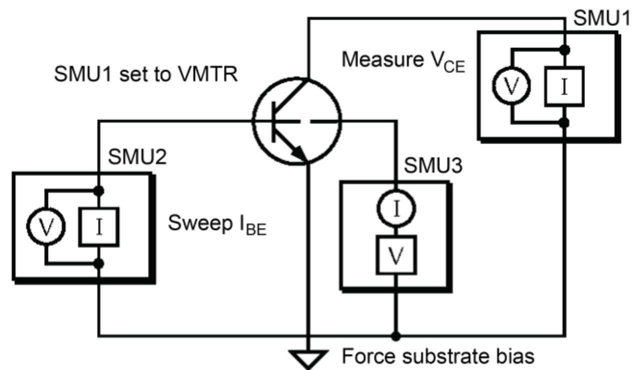
Source-measure units (SMUs)

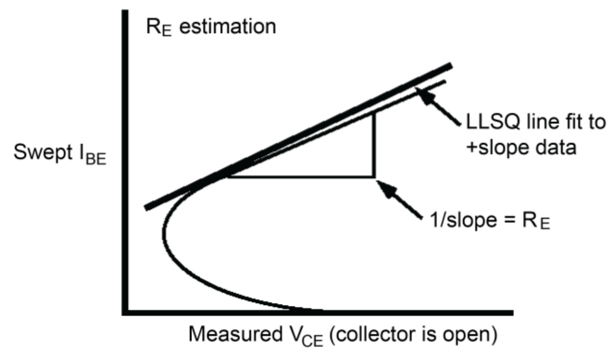
- SMU1: Set to VMTR, measures V_{CE}
- SMU2: Sweeps I_{BE}, 3 V voltage limit
- SMU3: Forces *`vsub`*, default current limit

Example

```
result = re(e, b, c, sub, ib1, ib2, vsub, npts, &iflag, &r)
```

Schematic





res

This subroutine calculates the resistance of a two-terminal resistor (force I, measure V).

Usage

`double res(int hi, int lo, int sub, double itest)`

hi	Input	The HI pin of the device
lo	Input	The LO pin of the device
sub	Input	The substrate pin of the device
itest	Input	The forced current, in amperes
Returns	Output	The calculated resistance: ▪ 0.0 = if the `itest` parameter is 0.0 ▪ 2.0E+21 = Measured voltage is within 98% of the default voltage limit

Details

This subroutine calculates the resistance of a two-terminal resistor by forcing a current and measuring the voltage. The voltage is limited to 30 V.

If a positive substrate pin is specified, the substrate is grounded. If a positive substrate pin is not specified, the substrate is left floating.

A delay is incorporated into the `res` subroutine; this delay is the calculated time required for stable forcing of *`itest`* with a 30 V voltage limit.

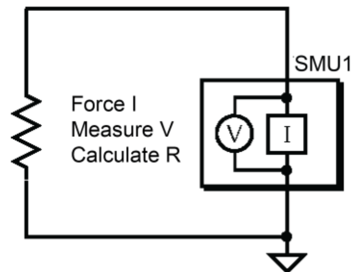
Source-measure units (SMUs)

- SMU1: Forces *`itest`*, 30 V voltage limit, measures voltage

Example

```
result = res(hi, lo, sub, itest)
```

Schematic



res2

This subroutine measures two-terminal resistance with a voltage limit.

Usage

```
double res2(int hi, int lo, int sub, double itest, double vlim)
```

hi	Input	The HI pin of the device
lo	Input	The LO pin of the device
sub	Input	The substrate pin of the device
itest	Input	The forced current, in amperes
vlim	Input	The voltage limit, in volts
Returns	Output	The calculated resistance: ▪ 0.0 = Measured voltage is < 0.002 V or *itest* = 0.0 ▪ 2.0E+21 = Measured voltage is within 98% of the voltage limit

Details

This subroutine measures the resistance of a two-terminal resistor by forcing a current and measuring the voltage.

If a positive substrate pin is specified, the substrate is grounded. If a positive substrate pin is not specified, the substrate is left floating.

A delay is incorporated into the `res2` subroutine; this delay is the calculated time required for stable forcing of *`itest`* with *`vlim`* voltage limit.

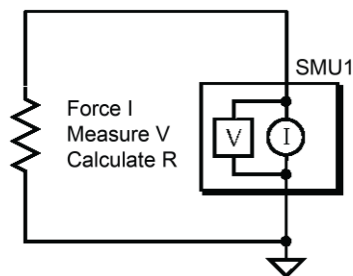
Source-measure units (SMUs)

- SMU1: Forces *`itest`*, programmable voltage limit, measures voltage

Example

```
result = res2(hi, lo, sub, itest, vlim)
```

Schematic



res4

This subroutine measures the resistance of a four-terminal resistor.

Usage

```
double res4(int his, int him, int los, int lom, int sub, double itest)
```

his	Input	The high source pin of the device
him	Input	The high measure pin of the device
los	Input	The low source pin of the device (ground)
lom	Input	The low measure pin of the device
sub	Input	The substrate pin of the device
itest	Input	The forced current, in amperes
Returns	Output	The calculated resistance: ▪ 0.0 = Measured voltage is < 0.002 V ▪ 2.0E+21 = Measured voltage is within 98% or the 40 V voltage limit

Details

This subroutine calculates the resistance of a four-terminal resistor (usually a van der Pauw structure) by forcing current and measuring the voltage. All device pins must be unique.

If a positive substrate pin is specified, the substrate is grounded. If a positive substrate pin is not specified, the substrate is left floating.

A delay is incorporated into the `res4` subroutine; this delay is the calculated time required for stable forcing of *itest* with a 40 V voltage limit.

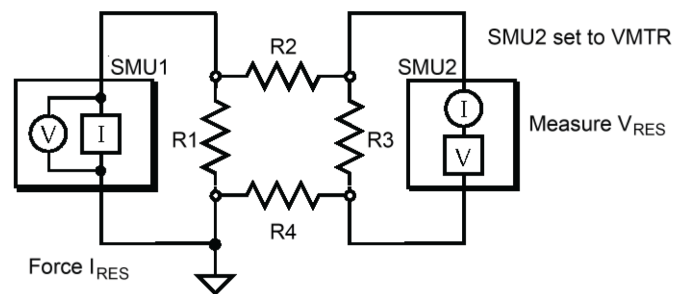
Source-measure units (SMUs)

- SMU1: Forces *itest*, 40 V voltage limit
- SMU2: Set to VMTR, measures voltage

Example

```
result = res4(his, him, los, lom, sub, itest)
```

Schematic



resv

This subroutine measures two-terminal resistance (force V, measure I).

Usage

```
double resv(int hi, int lo, int sub, double v)
```

hi	Input	The HI pin of the device
lo	Input	The LO pin of the device
sub	Input	The substrate pin of the device
v	Input	The forced voltage, in volts
Returns	Output	The calculated resistance: ▪ 1.0E+20 = Measured current is < 10 pA ▪ 4.0E+21 = Measured current is within 98% of the 200 mA current limit

Details

This subroutine calculates the resistance of a two-terminal resistor by forcing a specified voltage and measuring the resulting current.

If a positive substrate pin is specified, the substrate is grounded. If a positive substrate pin is not specified, the substrate is left floating.

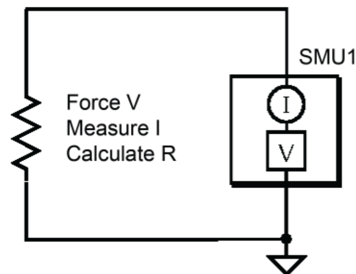
Source-measure units (SMUs)

SMU1: Forces V, maximum current limit, measures I

Example

```
result = resv(hi, lo, sub, v)
```

Schematic



rvdp

This subroutine makes a four-terminal van der Pauw measurement.

Usage

```
double rvdp(int pin1, int pin2, int pin3, int pin4, int sub, double itest, double *ratio)
```

pin1	Input	First pin on the device
pin2	Input	Second pin on the device
pin3	Input	Third pin on the device
pin4	Input	Fourth pin on the device
sub	Input	The substrate pin of the device
itest	Input	The forced current, in amperes
ratio	Output	The ratio of resistances (R_S)
Returns	Output	The estimated sheet resistance: ■ 0.0 = Measured voltage is < 0.002 V or `itest` = 0.0 ■ 2.0E+21 = Measured voltage is within 98% of the voltage limit

Details

This subroutine estimates the sheet resistance of a four-terminal sample using the standard technique of forcing current through two adjacent pins and measuring the voltage developed across the two remaining pins. The device connections are then shifted 90 degrees and the measurements are repeated.

The sheet resistance is calculated as the average of the two resistances. The difference between the two orientations is returned in the `ratio` variable. See the schematic for the correct pin orientation on the sample.

If a positive substrate pin is specified, the substrate is grounded. If a positive substrate pin is not specified, the substrate is left floating.

A delay is incorporated into the `rvdp` subroutine; this delay is the calculated time required for a stable forcing of `itest` with a 30 V voltage limit.

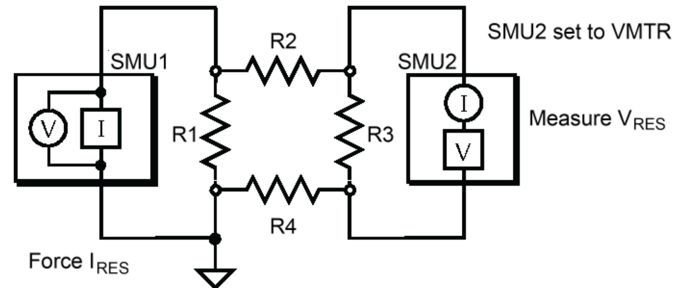
Source-measure units (SMUs)

- SMU1: Forces `itest`, default voltage limit
- SMU2: Set to VMTR, measures voltage

Example

```
result = rvdp(pin1, pin2, pin3, pin4, sub, itest, &ratio)
```

Schematic



tdelay

This subroutine calculates the delay time, in seconds, for the number of pins, current, and voltage specified as input parameters.

Usage

```
double tdelay(int npin, double i, double v)
```

npin	Input	The number of pins connected to the charging node
i	Input	The current, in amperes
v	Input	The voltage, in volts
Returns	Output	The calculated delay time

Details

This subroutine calculates the delay based on system capacitance, leakage currents, and the number of pins connected to the source.

The `tdelay` subroutine differs from the `kdelay` subroutine because `kdelay` calculates and provides a delay, but `tdelay` simply returns a value that can be passed into LPTLib calls such as `sweepX` and `searchX` to provide an appropriate delay. See the discussion in the kdelay (on page 3-56) subroutine for more information.

Example

```
delay_time = tdelay(npin, i, v)
```

tox

This subroutine calculates the thickness of an oxide layer from the capacitance and the area of a metal-oxide semiconductor (MOS) capacitor.

Usage

```
double tox(int hi, int lo, int sub, double vbias, double area)
```

hi	Input	The HI pin of the device
lo	Input	The LO pin of the device
sub	Input	The substrate pin of the device
vbias	Input	The voltage bias on the device, in volts
area	Input	The area of the capacitor, in cm^2
Returns	Output	The calculated oxide thickness: 4.0E+21 = Preliminary leakage test fails

Details

This subroutine makes a capacitance and a conductance measurement, corrects the capacitance measurement, and then calculates the oxide thickness. The common equation below is used to estimate oxide thickness. The oxide thickness is returned in angstroms. The area should be specified in cm^2.

$T_{OX} = E_{OX} A / C_{CORRECTED}$

Where:

$E_{OX} = 34.52^{-14}$ farads per cm and is the oxide dielectric constant

Calculations assume that CMTR1 is a Keithley Instruments Model 9125 1 MHz capacitance meter.

Before T_{OX} is calculated, voltage bias (V_{BIAS}) is forced with a current limit of 1 µA, and the resulting current is measured. If the current is within 98% of the limit, the capacitor is considered too leaky and the function returns a value of 4.0E+21.

NOTE

This subroutine can be modified to accommodate other dielectric materials by adding the dielectric constant into the argument list.

Example

```
result = tox(hi, lo, sub, vbias, area)
```

vbes

This subroutine measures base-emitter voltage of a bipolar transistor.

Usage

```
double vbes(int e, int b, int c, int sub, double ipgm, char type)
```

e	Input	The emitter pin of the device
b	Input	The base pin of the device
c	Input	The collector pin of the device
sub	Input	The substrate pin of the device
ipgm	Input	The forced current, in amperes
type	Input	Type of transistor: `'N'` or `'P'`
Returns	Output	-1.0 = Type not specified as 'N' or 'P' +2.0E+21 = Voltage limit reached; measured voltage is within 98% of the 3 V limit

Details

For a PNP transistor, this subroutine measures the base-emitter voltage at a specified emitter current with the base and collector terminals tied to ground.

For an NPN transistor, this subroutine measures the emitter-base voltage at a specified base current with the emitter and collector terminals tied to ground.

If a positive substrate pin is specified, the substrate is grounded. If a positive substrate pin is not specified, the substrate is left floating.

A delay is incorporated into the `vbes` subroutine; this delay is calculated time required for stable forcing of *ipgm* with a 3 V voltage limit.

V/I polarities

The polarity of *ipgm* is determined by device type.

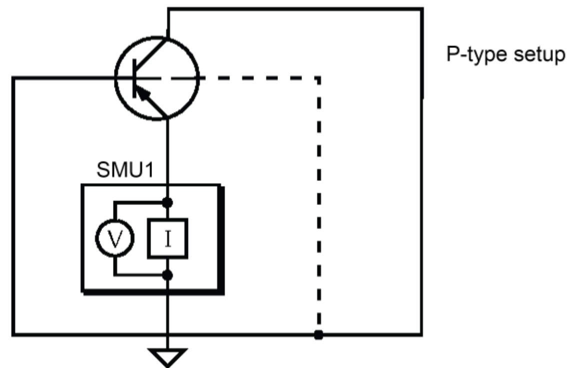
Source-measure units (SMUs)

- SMU1: Forces *ipgm*, 3 V voltage limit, measures `vbes`

Example

```
result = vbes(e, b, c, sub, ipgm, type)
```

Schematic



vf

This subroutine measures the forward-biased junction voltage of a diode when a current is forced.

Usage

`double vf(int hi, int lo, int sub, double itest)`

hi	Input	The HI pin of the device (anode)
lo	Input	The LO pin of the device (cathode)
sub	Input	The substrate pin of the device
itest	Input	The forced current, in amperes
Returns	Output	Measured voltage: +2.0E+21 = Measured voltage is within 98% of the 3 V voltage limit

Details

If a positive substrate pin is specified, the substrate is grounded. If a positive substrate pin is not specified, the substrate is left floating.

A delay is incorporated into the `vf` subroutine; this delay is the calculated time required for stable forcing of *itest* with a 3 V voltage limit.

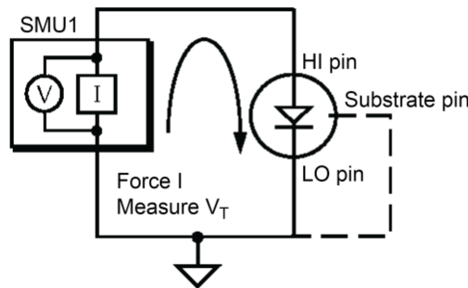
Source-measure units (SMUs)

- SMU1: Forces *itest*, 3 V voltage limit, measures voltage

Example

```
result = vf(hi, lo, sub, itest)
```

Schematic



vg2

This subroutine measures gate-source voltage (V_{GS}) at a specified drain current (I_{DS}), drain voltage (V_{DS}), and substrate bias (V_{BS}).

Usage

```
double vg2(int d, int g, int s, int sub, char type, double idspec, double errpct, double
    vds, double vbs, double vglo, double vghi, int maxitr, double *idmeas, int *istat)
```

d	Input	The drain pin of the device
g	Input	The gate pin of the device
s	Input	The source pin of the device
sub	Input	The substrate pin of the device
type	Input	Type of transistor: `'N'` or `'P'`
idspec	Input	Target value of I_{DS}, in amperes
errpct	Input	Maximum percent error in drain current
vds	Input	The forced drain voltage, in volts
vbs	Input	The forced substrate bias, in volts
vglo	Input	Start of the gate-source voltage (V_{GS}) search, in volts
vghi	Input	End of the V_{GS} search, in volts
maxitr	Input	Maximum number of iterations
idmeas	Output	Final measured I_{DS}, in amperes
istat	Output	Return status code: ■ > 0 = Success, `istat` is the number of iterations ■ -1 = *type* not 'N' or 'P' ■ -2 = *vglo* is ≥ *vghi* ■ -3 = Maximum iteration count reached ■ -4 = I_{DS} window too small ■ -5 = *maxitr* < 0
Returns	Output	Measured gate-source voltage, or 0.0 if `istat` is < 0

Details

Drain voltage is forced and a binary search is done on V_{GS}, starting with the two input values of V_{GS} (*vglo* and *vghi*). The binary search is controlled by two parameters: The error estimate (*errpct*) and the maximum number of iterations (*maxitr*). Error codes are returned based on the results of the search.

If a zero or negative substrate pin is specified, the substrate is left floating. If the pin number is greater than 0 and V_{BS} is less than 0.9 mV, the substrate is grounded. In all other cases, it is connected and forced.

V/I polarities

The polarities of V_{GS}, V_{DS}, and V_{BS} are determined by device type.

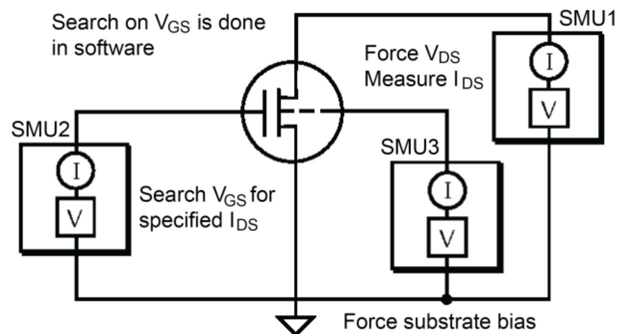
Source-measure units (SMUs)

- SMU1: Forces *vds*, default current limit, measures I_{DS}
- SMU2: Searches V_{GS}, current limit set to (1.25 * *idspec*)
- SMU3: Forces *vbs*, default current limit

Example

```
result = vg2(d, g, s, sub, type, idspec, errpct, vds, vbs, vglo, vghi, maxitr,
      &idmeas, &istat)
```

Schematic



vgsat

This subroutine measures saturated threshold voltage (V_{GSAT}) of a field-effect transistor (FET) at a specified drain-source current (I_{DS}).

Usage

```
double vgsat(int d, int g, int s, int sub, double ipgm, double vlim, double vsub)
```

d	Input	The drain pin of the device
g	Input	The gate pin of the device
s	Input	The source pin of the device
sub	Input	The substrate pin of the device
ipgm	Input	The forced drain current, in amperes
vlim	Input	The drain voltage limit, in volts
vsub	Input	Substrate bias, in volts
Returns	Output	Measured gate-source voltage (V_{GS}): ▪ 2.0E+21 = Measured voltage (V_{GSAT}) is within 98% of the specified voltage limit (*vlim*)

Details

This subroutine forces gate-source current (I_{GS}) and measures V_{GS} with the drain shorted to the gate.

If a zero or negative substrate pin is specified, the substrate is left floating. If the pin number is greater than 0 and VBS is less than 0.9 mV, the substrate is grounded. In all other cases, it is connected and forced.

A delay is incorporated into the `vgsat` subroutine; this delay is the calculated time required for stable forcing of *ipgm* within the *vlim* voltage limit.

V/I polarities

N-channel +Ipgm, $-V_{BS}$

P-channel -Ipgm, $+V_{BS}$

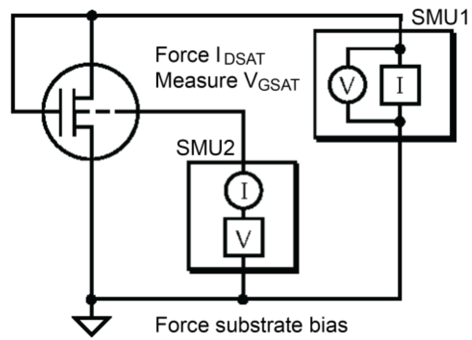
Source-measure units (SMUs)

- SMU1: Forces *ipgm*, programmed voltage limit, measures `vgsat`
- SMU2: Forces V_{BS}, default current limit

Example

```
result = vgsat(d, g, s, sub, ipgm, vlim, vsub)
```

Schematic



vp

This subroutine estimates the voltage at which the current flow between the source and drain is blocked (pinched-off) for a metal-semiconductor field-effect transistor (MESFET) at a specified drain voltage and fraction of saturated drain current.

Usage

```
double vp(int d, int g, int s, int sub, double vdss, double idlim, double factor, double
    v1, double v2, double *idss, double *ip, int *iflag)
```

d	Input	The drain pin of the device
g	Input	The gate pin of the device
s	Input	The source pin of the device
sub	Input	The substrate pin of the device
vdss	Input	The forced drain voltage, in volts
idlim	Input	Drain current limit, in amperes
factor	Input	Fraction of saturated drain current (I_{DSS})
v1	Input	Start of the gate-source voltage (V_{GS}) search, in volts
v2	Input	End of the V_{GS} search, in volts
idss	Output	Measured I_{DSS}, in amperes
ip	Output	Targeted pinch-off current, in amperes
iflag	Output	Return status: ▪ 0 = Normal completion ▪ 1 = Device did not trigger ▪ 2 = IDSS is within 98% of drain current limit ▪ 3 = <not used> ▪ 4 = The factor parameter is ≤ 0.0 or > 1 ▪ 5 = Device triggered on starting voltage ▪ 6 = Device triggered on ending voltage
Returns	Output	The voltage at which the current flow between the source and drain is blocked (pinch-off voltage)

Details

This subroutine estimates the pinch-off voltage for a MESFET at a specified drain voltage and fraction of I_{DSS}. First, it measures I_{DSS}, and then searches for a gate voltage that achieves a targeted pinch-off current (`ip`). The `ip` output parameter is normally described as a fraction of I_{DSS} (usually 0.02 of I_{DSS}). The trigger and search routines are used to find the V_{GS} that forces the targeted drain-source current (I_{DS}) value (`ip`).

If a positive substrate pin is specified, the substrate is grounded. If a positive substrate pin is not specified, the substrate is left floating.

The factor for I_{DSS} is normally 0.02.

This subroutine does not call the `idss` subroutine.

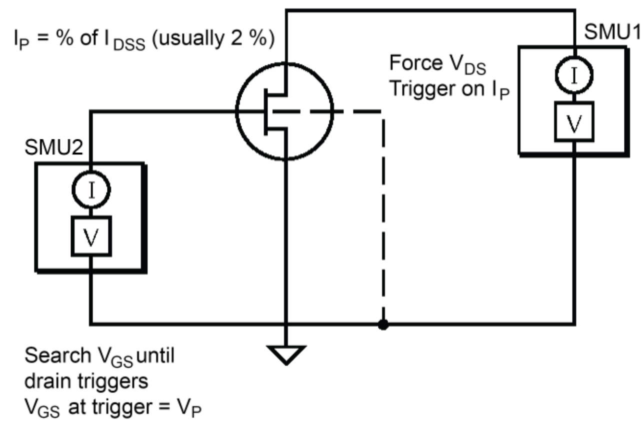
Source-measure units (SMUs)

- SMU1: Forces V_{DS}, programmable current limit, sets trigger on I_P ($I_P = I_{DSS}$ * *factor*)
- SMU2: Searches V_{GS}, default current limit

Example

```
result = vp(d, g, s, sub, vdss, idlim, factor, v1, v2, &idss, &ip, &iflag)
```

Schematic



vp1

This subroutine estimates the voltage at which the current flow between the source and drain is blocked (pinched-off) for a metal-semiconductor field-effect transistor (MESFET) at a specified pinch-off current and drain-source voltage (V_{DS}).

Usage

```
void vp1(int d, int g, int s, int sub, double ids, double vdlim, double vg1, double vg2,
    double iglim, double *iflag, double *vp)
```

d	Input	The drain pin of the device
g	Input	The gate pin of the device
s	Input	The source pin of the device
sub	Input	The substrate pin of the device
ids	Input	The forced drain current (I_P), in amperes
vdlim	Input	The drain voltage target, in volts
vg1	Input	Start of the gate-source voltage (V_{GS}) search, in volts
vg2	Input	End of the V_{GS} search, in volts
iglim	Input	Gate current limit, in amperes
iflag	Output	Return status flag: 0 = Normal completion 1 = Pinch-off voltage (V_P) is 0.0 2 = Device triggered on starting voltage 3 = Device triggered on ending voltage
vp	Output	Measured V_P

Details

This subroutine is a variant of the `vp` subroutine. The trigger is set to the specified V_{DS} and pinch-off current (I_P) is forced on the drain. The gate voltage is then searched until the V_{DS} value is reached. At 0 V gate-source voltage (V_{GS}), the drain voltage is below the specified V_{DS}. As V_{GS} is increased, V_{DS} increases as the device approaches the voltage at which the current flow between the source and drain is blocked.

If a positive substrate pin is specified, the substrate is grounded. If a positive substrate pin is not specified, the substrate is left floating.

A delay is incorporated into the `vp1` subroutine; this delay is the calculated time required for stable forcing of drain-source current (I_{DS}) with the V_{DS} voltage limit.

V/I polarities

vg1 and *vg2* ensure that polarity forward biases the gate-source diode. The starting voltage, *vg1*, must allow the drain SMU to supply pinch-off current without exceeding the limit.

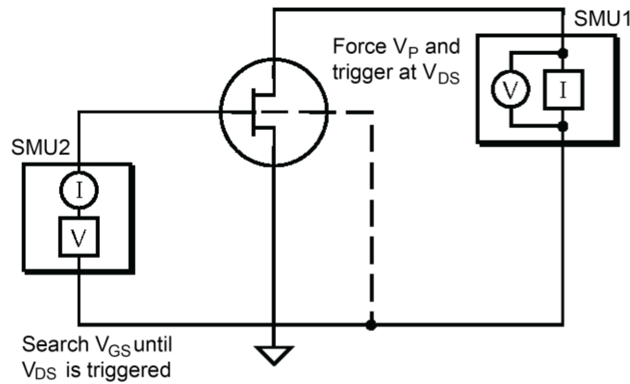
Source-measure units (SMUs)

- SMU1: Forces drain-source current (*ids*), programmable voltage limit, trigger set to V_{DS}
- SMU2: Searches V_{GS}, programmable current limit

Example

```
vp1(d, g, s, sub, ids, vdlim, vg1, vg2, iglim, &iflag, &vp)
```

Schematic



vt14

This subroutine estimates the extrapolated threshold voltage (V_T) of a metal-oxide field-effect transistor (MOSFET) using a simple two-point technique.

Usage

```
double vt14(int d, int g, int s, int sub, double vlow, double vhigh, double vds, double
    vbs, double ithr, double niter)
```

d	Input	The drain pin of the device
g	Input	The gate pin of the device
s	Input	The source pin of the device
sub	Input	The substrate pin of the device
vlow	Input	The start of the gate-source voltage (V_{GS}) binary search, in volts
vhigh	Input	The end of the V_{GS} binary search, in volts
vds	Input	The forced drain voltage, in volts
vbs	Input	Substrate bias, in volts
ithr	Input	The targeted drain current (I_{DS}), in amperes
niter	Input	The number of iterations in the search
Returns	Output	The extrapolated threshold voltage

Details

This subroutine does a binary search on V_{GS} to locate the target threshold current using the vtati (on page 3-80) subroutine. This current is I_{D1}. I_{D4} is then calculated as 4 * I_{D1}. A binary search is done again on V_{GS} to find I_{D4}. A linear least-squares (LLSQ) line is fit between these two points, and the V_T parameter is estimated.

A typical value for `niter` is 10 iterations. If `niter` is less than 2, a value of 2 is used. If it is greater than 16, a value of 16 is used.

V/I polarities

N-channel $+V_{DS}$, $+V_G$, $-V_{BS}$, $+I_{THR}$

P-channel $-V_{DS}$, $-V_G$, $+V_{BS}$, $-I_{THR}$

Source-measure units (SMUs)

See the vtati (on page 3-80) subroutine.

Example

```
result = vt14(d, g, s, sub, vlow, vhigh, vds, vbs, ithr, niter)
```

vtati

This subroutine returns the value of the threshold voltage (V_T) needed to produce a specified drain current (I_{DS}).

Usage

```
double vtati(int d, int g, int s, int sub, double vlow, double vhigh, double vds, double
    vbs, double ithr, int niter)
```

d	Input	The drain pin of the device
g	Input	The gate pin of the device
s	Input	The source pin of the device
sub	Input	The substrate pin of the device
vlow	Input	The start of the gate voltage (V_{GS}) sweep
vhigh	Input	The end of the V_{GS} sweep
vds	Input	Drain voltage, in volts
vbs	Input	Substrate bias, in volts
ithr	Input	The targeted drain current (I_{DS}), in amperes
niter	Input	The number of iterations in the search
Returns	Output	Calculated V_T: ■ 1.0E+21 = Device triggered on starting voltage ■ 2.0E+21 = Device triggered on end voltage ■ 4.0E+21 = Measured gate current is within 98% of the 10 µA current limit

Details

This subroutine executes a binary search on gate-source voltage (V_{GS}) to find I_{DS} when drain-source voltage (V_{DS}) and substrate bias voltage (V_{BS}) are fixed. The number of iterations is programmable.

If a zero or negative substrate pin is specified, the substrate is left floating. If the pin number is greater than 0 and V_{BS} is less than 0.9 mV, the substrate is grounded. In all other cases, it is connected and forced.

A typical value for `niter` is 10 iterations. If `niter` is less than 2, a value of 2 is used. If it is greater than 16, a value of 16 is used.

V/I polarities

N-channel $+V_{DS}$, +Vlow, +Vhigh, $-V_{BS}$, $+I_{THR}$

P-channel $-V_{DS}$, -Vlow, -Vhigh, $+V_{BS}$, $-I_{THR}$

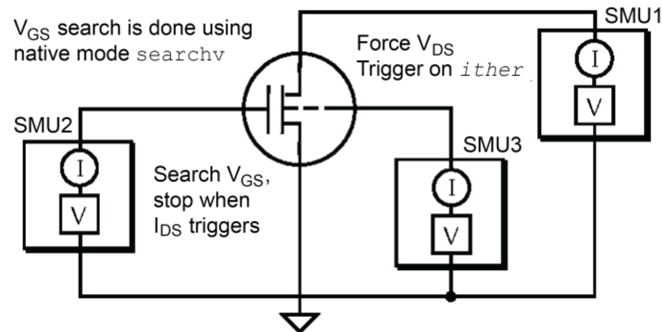
Source-measure units (SMUs)

- SMU1: Force *vds*, trigger on *ithr*, default current limit
- SMU2: Search V_{GS}, 10.0 µA current limit
- SMU3: Force *vbs*, default current limit

Example

```
result = vtati(d, g, s, sub, vlow, vhigh, vds, vbs, ithr, niter)
```

Schematic



vtext

This subroutine estimates the extrapolated gate-source threshold voltage of a metal-oxide field-effect transistor (MOSFET).

Usage

```
double vtext(int d, int g, int s, int sub, char type, double vlow, double vhigh, double
    vds, double vbs, double ithr, double vstep, int nmax, double *slope, int *kflag)
```

d	Input	The drain pin of the device
g	Input	The gate pin of the device
s	Input	The source pin of the device
sub	Input	The substrate pin of the device
type	Input	Type of transistor: `'N'` or `'P'`
vlow	Input	The start of the gate-source voltage (V_{GS}) binary search, in volts
vhigh	Input	The end of the V_{GS} binary search, in volts
vds	Input	The forced drain voltage, in volts
vbs	Input	Substrate bias, in volts
ithr	Input	Drain-source trigger current (I_{DS}), in amperes
vstep	Input	V_{GS} step size, in volts
nmax	Input	The maximum number of steps
slope	Output	The calculated slope
kflag	Output	Return status flag: ■ 0 = Normal operation ■ 1 = The *ithr* parameter is too high; indicates that the *vlow* parameter is above the voltage threshold (V_T) and the slope is constantly decreasing ■ 2 = Did not find peak slope; indicates that the *vhigh* parameter was below V_T (maximum slope was the last value) ■ 3 = Binary search on V_{GS} failed; may indicate that the *vlow* and *vhigh* parameters were below V_T and the device never turned on ■ 4 = The *type* parameter was not specified as 'N' or 'P' ■ 5 = V_{GS} step size is 0.0
Returns	Output	Gate-source voltage threshold, in volts.

Details

This subroutine estimates the extrapolated threshold voltage of a MOSFET using the maximum slope method. Maximum slope refers to the common technique of numerically differentiating the I_{DS} versus V_{GS} curve. Slope refers to the FET transconductance (g_m).

This subroutine uses a two-step method of finding V_T. First, a binary search is done on the V_{GS} to find a drain current (I_{DS}) that is within 0.25 of the estimated threshold current (for most enhancement devices this value is 1 µA). If the measured I_{DS} value is within tolerance, the routine continues with a sliding five-point linear least-squares (LLSQ) analysis of the I_{DS}-V_{GS} curve.

The steps used by this technique to find the maximum slope or maximum g_m (where V_{GS} step size is a user-input variable):

1. From the last point below `ithr` (threshold I_{DS}), measure four points (I_{DS}, V_{GS}). V_{GS} is stepped, and I_{DS} is measured.
2. Calculate the first slope. This value is now the MAX SLOPE.
3. Step V_{GS}, measure I_{DS}.
4. Delete the first point in the five-point LLSQ array, and add the most recent point (shift left one point).
5. Calculate the new slope.
6. Compare the new slope to MAX SLOPE.
7. Repeat steps 3 through 6 until the peak slope is crossed.
8. At peak slope, evaluate the V_{GS} intercept (V_G).
9. Calculate V_T.
10. Return V_T as `vtext` result and `slope`.

This subroutine is very sensitive to the V_{GS} step size (`vstep`) and the start and stop points for the binary search (`vlow`, `vhigh`), but is generally the most accurate way of evaluating the extrapolated threshold voltage. The `vtext2` subroutine is a further variation on this technique. The major differences are that `vtext2` uses the `trigger` and `sweep` calls to collect data, and it uses some refinements in the data analysis. It also runs faster than the `vtext` subroutine.

If a zero or negative substrate pin is specified, the substrate is left floating. If the pin number is greater than 0 and V_{BS} is less than 0.9 mV, the substrate is grounded. In all other cases, it is connected and forced.

The `nmax` parameter defaults to 21 steps if a smaller value is entered.

V/I polarities

The polarities of V_{DS} and I_{THR} are determined by the device type.

The `vstep` parameter is 10 mV to 100 mV. This depends on the `vlow` and `vhigh` parameters. For an N-channel, if `vlow` = 0.0 and `vhigh` = 2.0, `vstep` should be positive.

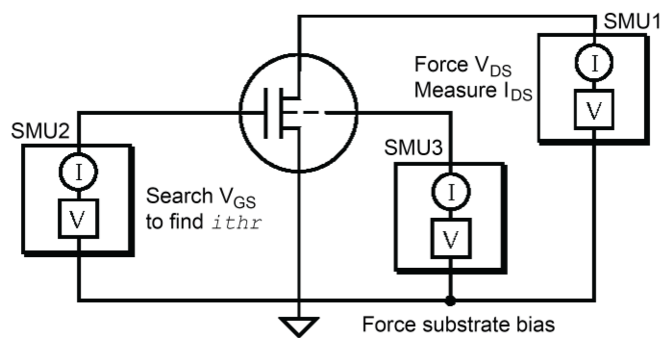
Source-measure units (SMUs)

- SMU1: Forces `vds`, default current limit, measures I_{DS}
- SMU2: Searches V_{GS}, default current limit
- SMU3: Forces `vbs`, default current limit

Example

```
result = vtext(d, g, s, sub, type, vlow, vhigh, vds, vbs, ithr, vstep, nmax,
      &slope, &kflag)
```

Schematic



vtext2

This subroutine estimates the extrapolated gate-source threshold voltage of a metal-oxide field-effect transistor (MOSFET) using a modified version of the `vtext` subroutine method.

Usage

```
double vtext2(int d, int g, int s, int sub, double vlow, double vhigh, double vds, double
    vbs, double ithr, double vstep, int npts, double *slope, int *kflag)
```

d	Input	The drain pin of the device
g	Input	The gate pin of the device
s	Input	The source pin of the device
sub	Input	The substrate pin of the device
vlow	Input	The start of the gate-source voltage (V_{GS}) binary search, in volts
vhigh	Input	The end of the V_{GS} binary search, in volts
vds	Input	Drain voltage, in volts
vbs	Input	Substrate bias, in volts
ithr	Input	Drain-source trigger current (I_{DS}), in amperes
vstep	Input	V_{GS} step size, in volts
npts	Input	The number of points in the sweep
slope	Output	The calculated transconductance (g_m)
kflag	Output	Return status flag: ■ 0 = Normal operation ■ 1 = The *ithr* parameter is too high; indicates that the *vlow* parameter is above the voltage threshold (V_T) and the slope is constantly decreasing ■ 2 = Did not find peak slope; indicates that the *vhigh* parameter was below V_T (maximum slope was the last value) ■ 3 = Binary search on V_{GS} failed; may indicate that the *vlow* and *vhigh* parameters were below V_T and the device never turned on ■ 4 = V_{GS} step size is 0.0
Returns	Output	The estimated threshold voltage

Details

This subroutine is a modified version of the vtext (on page 3-82) subroutine. The differences are:

- All setup parameters must have the correct sign (polarity)
- An initial binary search is done with LPTLib `trigi` and `searchv` subroutines
- The I_{DS}-V_{GS} data is measured at one time (LPTLib `sweepv`, `smeasi`)
- The *vlow* parameter must be algebraically smaller than the *vhigh* parameter

The procedural differences are:

- A binary search on V_{GS} is done to find I_{DS} (`vstart`)
- Calculate sweep limits: The *`vlow`* parameter = `vstart`
- The *`vhigh`* parameter = `vstart` + *`npts`* * *`vstep`*
- Sweep the I_{DS}-V_{GS} data set
- Perform a sliding five-point linear least-squares (LLSQ) analysis (as in the `vtext` subroutine)

If a zero or negative substrate pin is specified, the substrate is left floating. If the pin number is greater than 0 and V_{BS} is less than 0.9 mV, the substrate is grounded. In all other cases, it is connected and forced.

The *`npts`* parameter must be greater than 5. If a value less than 5 is used, the subroutine uses 5 points by default.

V/I polarities

The polarities of V_{DS} and I_{THR} are determined by the device type.

The *`vstep`* parameter is 10 mV to 100 mV. This depends on the *`vlow`* and *`vhigh`* parameters. For an N-channel, if *`vlow`* = 0.0 and *`vhigh`* = 2.0, *`vstep`* should be positive.

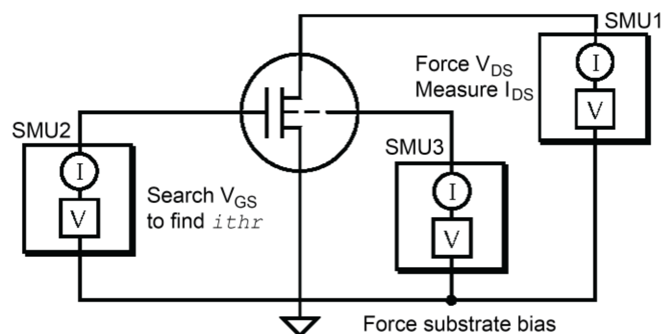
Source-measure units (SMUs)

- SMU1: Forces *`vds`*, default current limit, measures I_{DS}
- SMU2: Searches V_{GS}, default current limit
- SMU3: Forces *`vbs`*, default current limit

Example

```
result = vtext2(d, g, s, sub, vlow, vhigh, vds, vbs, ithr, vstep, npts,
     &slope, &kflag)
```

Schematic



vtext3

This subroutine estimates the extrapolated gate-source threshold voltage of a metal-oxide field-effect transistor (MOSFET) using a condensed version of the `vtext` and `vtext2` subroutine method.

Usage

```
void vtext3(int d, int g, int s, int sub, double vg1, double vg2, double vds, double
    vbs, int npts, double *slope, double *vt, int *flag);
```

d	Input	The drain pin of the device
g	Input	The gate pin of the device
s	Input	The source pin of the device
sub	Input	The substrate pin of the device
vg1	Input	Start of the gate-source voltage (V_{GS}) search, in volts
vg2	Input	End of the V_{GS} search, in volts
vds	Input	Drain voltage, in volts
vbs	Input	Substrate bias, in volts
npts	Input	The number of points in the sweep
slope	Output	The calculated inductance
vt	Output	Threshold voltage
flag	Output	Return status: 0 = Normal operation 1 = Bad data collected 2 = Calculated linear least-squares (LLSQ) slope = 0.0, bad data

Details

This subroutine is the most condensed form of the basic maximum slope techniques to find threshold voltage (V_T).

This subroutine does the following to estimate V_T:

1. Sweeps a drain-source current (I_{DS}), gate-source voltage (V_{GS}) array (using the `idvsvg` subroutine).
2. Differentiates the data set using dy/dx notation (also known as Leibniz's notation).
3. Finds the maximum slope.
4. Finds the index of the maximum slope in the slope array.
5. Returns the gate-source voltage (V_{GS}) intercept and slope.

V/I polarities

N-channel $+V_{DS}$, $+V_G$, $-V_{BS}$

P-channel $-V_{DS}$, $-V_G$, $-V_{BS}$

Source-measure units (SMUs)

See the idvsvg (on page 3-52) subroutine.

Example

```
vtext3(d, g, s, sub, vg1, vg2, vds, vbs, npts, &slope, &vt, &flag);
```

Section 4

HVLib command reference

In this section:

Introduction 4-1
How to use the library reference 4-1
High-Voltage Library commands 4-4

Introduction

The Keithley High-Voltage Library (HVLib) is a set of commands you can use to make measurements on an S540 Power Semiconductor Test System using the Keithley Test Environment (KTE) software.

You can use these commands in two- and three-terminal measurement applications to measure capacitance-voltage (C-V), calculate compensation constants, do open, load, and short compensation, and do breakdown voltage tests.

NOTE

The commands in this library are only supported for the S540 Power Semiconductor Test System, which has an HVM1212A Matrix.

For more detailed information about making high-voltage capacitance-voltage (C-V) measurements, see "High-voltage C-V measurements" in the *S540 Reference Manual* (part number S540-901-01).

How to use the library reference

The commands in the Keithley Test Environment (KTE) High-Voltage Library (HVLib) are in the C programming language. Each command is presented in a standard format that follows the pattern below:

- **Purpose statement:** The first line of text under the command heading contains a brief explanation of what the command does.

Figure 13: Example purpose statement

hvcv_comp

This command does complex mathematical calculations to implement specified impedance compensation models.

- **Usage:** A line of code representing the prototype of the command, followed by a table listing the input and output parameters for the command.

  Parameters that you specify are shown in *monospace italic* font. Parameters preceded by an asterisk (*) are character parameters that are passed into the function (input) or pointers to information that is returned (output).

  Each parameter is preceded by one of the following declarations that specifies the data type for the parameter: `int` (integer), `double` (double-precision floating-point), and `char` (a character string).

Figure 14: Example syntax and parameter description

Usage

```
int hvcv_comp(char *label, char *comp_mode, double Freq, double *CpComp, double
    *GpComp, double *DComp)
```

label	Input	Device name (label), for example, DUT1 or p1_3
comp_mode	Input	Compensation type: • CompNone (use this if you do not want to run any compensation or if the *dut* parameter is set to anything other than dut) • CompOpen • CompLoad • CompShort • CompOpenLoad • CompShortEx • CompShortOpen • CompShortOpenLoad
Freq	Input	Frequency (1e4 to 2e6)
CpComp	Output	Compensated capacitance (*Cp*) value after correction
GpComp	Output	Compensated conductance (G*p*) value after correction
DComp	Output	Compensated dissipation factor after correction
cal	Output	Value of calibration constants

- **Details:** Additional information about using the command.

Figure 15: Example details

Details

This command does the following:

- Verifies input conditions
- Gets capacitance-voltage (*Cp*, *Gp*) data for the specified device label and all specified device types (dut, open, short, load)
- Runs compensation model specified by the *comp_mode* parameter
- Reports corrected *Cp* and *Gp* values

This command does separate CompOpen, CompShort, and CompLoad compensation or any combination of these modes (for example, CompOpenLoad, CompShortOpen, CompShortOpenLoad). If compensation data is not already stored in the data pool when device testing is done or incorrect labels are used, an error is returned.

- **Example:** Lines of code showing what a call to the command might look like in actual use.

Figure 16: Example command call

Example

```
stat = hvcv_comp("pin1_pin2", "CompShortOpen", 1e5, CpComp, GpComp, DComp, Cal)
```

Does ShortOpen compensation on the device named `pin1_pin2` and returns the Cp, Gp, and D values after compensation.

- **Also see:** Cross-references to other related commands and topics, where applicable.

Figure 17: Example cross-references

Also see

hvcv_measure (on page 3-14)
hvcv_comp (on page 3-7)

High-Voltage Library commands

The Keithley Test Environment (KTE) High-Voltage Library (HVLib) commands are described in detail in the following topics.

gate_charge

This command measures the gate charge required to switch on the power transistor.

Models supported

S540

Usage

```
int gate_charge(int gate, int drain, int source, double Vds, double drainLimitI, double
    gateCurrent, double gateMaxV, double timeOut, int measDrain, double *timeArray, int
    timeArraySize, double *VgArray, int VgArraySize, double *VgCharge, int VgChargeSize,
    double *VdArray, int VdArraySize, double *Slope, int SlopeSize, double Coffset,
    double *Ceff, double *Vpl, double *T1, double *T2, double *Qgs, double *Qgd)
```

gate	Input	The gate pin
drain	Input	The drain pin
source	Input	The source pin
Vds	Input	Drain voltage
drainLimitI	Input	Current limit for the drain instrument; 1 A maximum
gateCurrent	Input	Amount of current to force into the gate
gateMaxV	Input	Voltage compliance limit for the gate; 200 V maximum
timeOut	Input	Timeout value for the test; 200 seconds maximum
measDrain	Input	Flag to enable (1) or disable (0) drain voltage measurement
timeArray	Output	Array to store timestamps
timeArraySize	Input	Size of the *timeArray* parameter
VgArray	Output	Array to store gate voltage
VgArraySize	Input	Size of the *VgArray* parameter
VgCharge	Output	Array to store gate charge
VgChargeSize	Input	Size of the *VgCharge* parameter
VdArray	Output	Array to store drain voltage
VdArraySize	Input	Size of the *VdArray* parameter
Slope	Output	Array to store slope (gate voltage (Vg) versus time) values
SlopeSize	Input	Size of the *Slope* parameter array
Coffset	Input	Parasitic capacitance of the gate cable
Ceff	Output	Ratio of total gate charge to maximum gate voltage
Vpl	Output	Gate voltage of the plateau area
T1	Output	Timestamp where the plateau area begins
T2	Output	Timestamp where the plateau area ends

Qgs	Output	Gate to source charge; calculate according to the JEDEC standard JESD 24-2
Qgd	Output	Gate to drain charge; calculate according to the JEDEC standard JESD 24-2

Details

This test does the following:

- Verifies input conditions
- Sets up a gate SMU and a drain SMU (Model 2636s):
  - Sets gate SMU voltage limit to *gateMaxV*
  - Sets gate SMU current range to the fixed 10**gateCurrent* range
  - Sets drain SMU current compliance and range to *drainLimitI*
- Makes connections
- Forces *GateCurrent* into the gate
- Measures gate voltage as a function of time
- Optionally (depending on the value of the *measDrain* parameter, `0` or `1`) measures drain voltage
- Determines the two points of inflection on the curve of Vg = Vg(Time) and reports it
- Adjusts values of the gate charge using the specified parasitic capacitance (*Coffset*)
- Returns effective capacitance, defined as gateMaxV/total-gate-charge
- Calculates and returns the gate charges (*Qgs* and *Qgd*) values according to the JEDEC standard JESD 24-2
- Returns the test status

This command returns a status:

- 1 Success
- –1 Gate voltage compliance limit exceeds 200 V
- –2 Drain current limit exceeds 1 A
- –3 *VgArraySize* is not equal to *timeArraySize*
- –4 *VdArraySize* is not equal to *timeArraySize*
- –5 Timeout exceeds maximum: 200 s
- –6 Test time exceeds timeout value
- –7 Number of measurements exceeds maximum allowed (10000)
- –8 *SlpSize* is not equal to *timeArraySize*
- –9 Compliance or test error
- –10 Error calculating S1, correlation factor < 0.9
- –11 Error calculating S2, correlation factor < 0.9

- –12 Power limit (current should be less than 0.1 A if voltage is greater than 20 V) exceeded

Example

```
pts = 600;
stats = gate_charge(3, 4, 5, 20, 0.1, 3e-8, 10, 10, 1, Time, pts, Vg, pts,
   Vq, pts, Vd, pts, Slp, pts, 3e-10, Coff, Vpl, T1, T2, Qgs, Qgd)
```

This test implements the JEDEC standard JESD 24-2.

Also see

None

hv_bvsweep

This command does a breakdown voltage sweep.

Models supported

S540

Usage

```
int hv_bvsweep(int high1, int high2, int high3, int low1, int low2, int low3, double
   vStart, double vStop, double vStep, double stepDelay, double trigCurrent, double
   compl, double ratio, double *bV, double *bVR, double *LeakR, double *Vbias, int
   VbiasPts, double *Imeas, int ImeasPts)
```

high1	Input	High pin 1
high2	Input	High pin 2
high3	Input	High pin 3
low1	Input	Low pin 1
low2	Input	Low pin 2
low3	Input	Low pin 3
vStart	Input	The start voltage of the sweep
vStop	Input	The stop voltage of the sweep
vStep	Input	The voltage step size for the sweep
stepDelay	Input	The step delay, in seconds
trigCurrent	Input	The current trigger level
compl	Input	The current limit
ratio	Input	The ratio of voltage to breakdown voltage at which voltage and current are reported
bV	Output	Breakdown voltage
bVR	Output	Voltage at a specified percent (`ratio`) of the breakdown voltage
LeakR	Output	The current level at `bVR`
Vbias	Output	The forced voltage bias
VbiasPts	Input	The number of voltage bias points
Imeas	Output	The measured current
ImeasPts	Input	The number of current measure points

Details

This command does the following:

- Verifies input conditions
- Configures source-measure unit (SMU) and trigger levels
- Sweeps voltage from `vStart` to `vStop`
- Reports breakdown voltage (bV) at the trigger point
- Reports leakage at a specified ratio of breakdown voltage

The `VbiasPts` and `ImeasPts` parameters are the number of points to use; they should be equal to or greater than (Vstop – Vstart)/Vstep + 1.

This command returns a status:

- `1` = Success
- `-1` = Invalid high pins
- `-2` = Invalid low pins
- `-3` = Invalid `Vstart` and `Vstop` values
- `-4` = Invalid ratio; valid range is 0.01 to 0.99
- `-5` = Invalid `trigCurrent;` valid range is 1e-9 A to 0.001 A
- `-6` = Invalid `vStep;` valid range is 0.1 V to 20 V
- `-7` = Invalid `stepDelay;` valid range is 0.001 s to 0.5 s
- `-8` = Wrong number of points; should be equal to or larger than (Vstop – Vstart)/Vstep + 1
- `-9` = Low-voltage pins are used for high-voltage test
- `-10` = Test is too fast; slow it down or increase the `trigCurrent` level

Example

```
startV = 0
stopV = 2500.0
status = hv_bvsweep(pin1, -1, -1, pin2, -1, -1,startV, stopV, 5, 0.1,1e-6,1e-5,
     0.85, BV, BVR, Leak, Vbias, 501, Imeas, 501)
```

Measures breakdown voltage by sweeping 0 V to 2500 V in 5 V steps.

Also see

None

hvcv_3term

This command measures output capacitance (C_{oss}), input capacitance (C_{iss}), or short-circuit reverse transfer capacitance (C_{rss}) of three-terminal devices.

Models supported

S540

Usage

```
int hvcv_3term(int drain, int gate, int source, double gateV, double startV, double
    stopV, char *mode, char *dut, char *comp_mode, double Freq, int doComp, int doRetest,
    double *drainV, int drainVPts, double *drainI, int drainIPts, double *Cp, int CpPts,
    double *D, int DPts, double *Gp, int GpPts)
```

drain	Input	Drain pin
gate	Input	Gate pin
source	Input	Source pin
gateV	Input	Gate voltage
startV	Input	Start voltage of drain voltage sweep
stopV	Input	Stop voltage of drain voltage sweep
mode	Input	Bias connections mode (`Ciss`, `Coss`, or `Crss`)
dut	Input	Device under test; valid options: ■ `dut` = Test the DUT itself with the high-voltage capacitance meter (CMTR) ■ `open` = Characterize the open device using the high-voltage CMTR; this can be done with the chuck down (pins not in contact with the device) ■ `short` = Characterize the short device using the high-voltage CMTR ■ `load` = Characterize the load device using the high-voltage CMTR ■ `shortEx` = Characterize the short device using the low-voltage capacitance meter (CMTR); this data is used to do `CompShort` compensation of the *`loadEx`* device ■ `loadEx` = Measure the load device using the low-voltage CMTR to get the expected value of the `loadEx` device ■ `openEx` = Characterize the open device using the low-voltage CMTR; this data is used to do `CompOpen` compensation of the `loadEx` device

comp_mode	Input	Compensation type: ▪ CompNone (use this if you do not want to run any compensation or if the *dut* parameter is set to anything other than dut) ▪ CompOpen ▪ CompShort ▪ CompLoad ▪ CompOpenLoad ▪ CompShortOpen ▪ CompShortLoad ▪ CompShortOpenLoad
Freq	Input	Frequency; recommended frequency is 1e5 Hz
doComp	Input	Specifies whether to do system-level compensation: ▪ 0 = Do not do system-level compensation ▪ 1 = Do system-level compensation
doRetest	Input	Specifies whether to remeasure compensation data: ▪ 0 = Do not remeasure compensation data ▪ 1 = Remeasure compensation data once, then reuse that measurement in any additional calls See **Details** for more information
drainV	Output	Drain voltage array
drainVPts	Input	Number of drain voltage points in the array
drainI	Output	Drain current array
drainIPts	Input	Number of drain current points in the array
Cp	Output	Capacitance
CpPts	Input	Compensated capacitance points
D	Output	Dissipation factor
DPts	Input	Compensated dissipation factor points
Gp	Output	Compensated conductance
GpPts	Input	Compensated conductance points

Details

This command can also do open compensation of the device under test (DUT), defined by the *comp_mode* parameter. This includes separate CompOpen, CompShort, and CompLoad compensation or any combination of these modes (for example, CompOpenLoad, CompShortOpen, CompShortOpenLoad). If compensation data (open, short, load, loadEx, openEx, shortEx) is not available before device testing, an error is returned.

For best results measuring C_{rss}, suppress the AC signal at the source terminal by connecting the high-voltage ground (HV GND) terminal to the source. In a system configured with a high-voltage matrix and long high-voltage cables, passive AC guarding (GND) provides superior performance over AC guarding using bias tees.

This command also collects compensation data for `open`, `short`, `load`, `loadEx`, `openEx`, and `shortEx`. Compensation data for `openEx` and `loadEx` is collected using the low-voltage CMTR, bypassing the bias-tees.

The *`doComp`* parameter provides a switch that enables or disables system-level compensation. To do ShortOpenLoad compensation using a system-level compensation file that is stored on the system (`cvCALsystem.ini`), set this parameter to `1`. To do ShortOpenLoad compensation using a user-generated compensation file (`cvCAL.ini`), set this parameter to `2`.

The *`doRetest`* parameter provides a switch that enables or disables remeasurement of the compensation data.

This command returns a status:

- `-1` = Arrays have a different number of output points; all arrays must have the same number of points
- `-2` = Gate, drain, or source pins are not defined
- `-3` = Invalid *`dut`* parameter name; valid names are `dut`, `open`, `load`, `loadEx`, `openEx`, and `shortEx`
- `-4` = Invalid compensation mode (*`comp_mode`*) name; valid names are `CompNone`, `CompOpen`, `CompShort`, `CompLoad`, `CompOpenLoad`, `CompShortOpen`, `CompShortEx`, and `CompShortOpenLoad`
- `-5` = Error when moving chuck down
- `-6` = Invalid *`mode`* parameter name; valid names are `Crss`, `Coss`, and `Ciss`
- `-7` = Low voltage pin is used for high-voltage test

Example

```
status = hvcv_3term(drain, gate, source, 0, 0, 10, "Ciss", "dut", "CompOpen", 1e5,
    1, 1, drainV, 11, drainI, 11, Cp, 11, D, 11, Gp, 11)
```

Measures C_{iss} of a three-terminal device.

Also see

hvcv_3term_basic (on page 4-11)

hvcv_3term_basic

This command measures output capacitance (C_{oss}), input capacitance (C_{iss}), or short-circuit reverse transfer capacitance (C_{rss}) of three-terminal devices.

Models supported

S540

Usage

```
int hvcv_3term_basic(int drain, int gate, int source, double gateV, double startV,
    double stopV, char *mode, double Freq, double *drainV, int drainVPts, double *drainI,
    int drainIPts, double *Cp, int CpPts, double *D, int DPts, double *Gp, int GpPts)
```

drain	Input	Drain pin
gate	Input	Gate pin
source	Input	Source pin
gateV	Input	Gate voltage
startV	Input	Start voltage of drain voltage sweep
stopV	Input	Stop voltage of drain voltage sweep
mode	Input	Bias connections mode (`Ciss`, `Coss`, or `Crss`)
Freq	Input	Frequency; recommended frequency is `1e5` Hz
drainV	Output	Drain voltage array
drainVPts	Input	Number of drain voltage points in the array
drainI	Output	Drain current array
drainIPts	Input	Number of drain current points in the array
Cp	Output	Capacitance
CpPts	Input	Compensated capacitance points
D	Output	Dissipation factor
DPts	Input	Compensated dissipation factor points
Gp	Output	Compensated conductance
GpPts	Input	Compensated conductance points

Details

For best results measuring C_{rss}, suppress the AC signal at the source terminal by connecting the high-voltage ground (HV GND) terminal to the source. In a system configured with a high-voltage matrix and long high-voltage cables, passive AC guarding (GND) provides superior performance over AC guarding using bias tees.

This command returns a status:

- `0` = Skip system-level compensation
- `-1` = Arrays have a different number of output points; all arrays must have the same number of points
- `-2` = Gate, drain, or source pins are not defined
- `-3` = Invalid *`dut`* parameter name; valid names are `dut`, `open`, `load`, `loadEx`, `openEx`, and `shortEx`
- `-5` = Error when moving chuck down
- `-6` = Invalid *`mode`* parameter name; valid names are `Crss`, `Coss`, and `Ciss`
- `-7` = Low voltage pin is used for high-voltage test

Example

```
status = hvcv_3term(drain, gate, source, 0, 0, 10, "Ciss", 1e5, drainV, 11, drainI,
    11, Cp, 11, D, 11, Gp, 11)
```

Measures C_{iss} of a three-terminal device.

Also see

hvcv_3term (on page 4-8)

hvcv_comp

This command does complex mathematical calculations to implement specified impedance compensation models.

Models supported

S540

Usage

```
int hvcv_comp(char *label, char *comp_mode, double Freq, double *CpComp, double *GpComp,
    double *DComp)
```

label	Input	Device name (label), for example, `DUT1` or `p1_3`
comp_mode	Input	Compensation type: ■ `CompNone` (use this if you do not want to run any compensation or if the *dut* parameter is set to anything other than `dut`) ■ `CompOpen` ■ `CompShort` ■ `CompLoad` ■ `CompOpenLoad` ■ `CompShortOpen` ■ `CompShortLoad` ■ `CompShortOpenLoad`
Freq	Input	Frequency (`1e4` to `2e6`)
CpComp	Output	Compensated capacitance (*Cp*) value after correction
GpComp	Output	Compensated conductance (G*p*) value after correction
DComp	Output	Compensated dissipation factor after correction
cal	Output	Value of calibration constants

Details

This command does the following:

- Verifies input conditions
- Gets capacitance-voltage (*Cp*, *Gp*) data for the specified device label and all specified device types (`dut`, `open`, `short`, `load`)
- Runs compensation model specified by the *comp_mode* parameter
- Reports corrected *Cp* and *Gp* values

This command does separate `CompOpen`, `CompShort`, and `CompLoad` compensation or any combination of these modes (for example, `CompOpenLoad`, `CompShortOpen`, `CompShortOpenLoad`). If compensation data is not already stored in the data pool when device testing is done or incorrect labels are used, an error is returned.

This command returns a status:

- `1` = Success
- `-1` = Device label is NULL
- `-2` = Device label is less than 2 characters or more than 64
- `-3` = Invalid compensation mode; valid modes are `CompNone`, `CompOpen`, `CompLoad`, `CompShort`, `CompOpenLoad`, `CompShortEx`, `CompShortOpen`, or `CompShortOpenLoad`
- `-4` = Frequency is out of valid range (`1e4` to `2e6`)
- `-5` = Failed on data retrieval for DUT
- `-6` = Failed on data retrieval for open
- `-7` = Failed on data retrieval for short
- `-8` = Failed on data retrieval for load
- `-9` = Failed on data retrieval for loadEx

Example

```
stat = hvcv_comp("pin1_pin2", "CompShortOpen", 1e5, CpComp, GpComp, DComp, Cal)
```

Does ShortOpen compensation on the device named `pin1_pin2` and returns the Cp, Gp, and D values after compensation.

Also see

None

hvcv_genCompData

This command generates correction factors for system-level high-voltage capacitance-voltage (C-V) compensation.

Models supported

S540

Usage

```
int hvcv_genCompData(int hpin, int lpin)
```

hpin	Input	Pin for the capacitance meter (CMTR) high signal
lpin	Input	Pin for CMTR low signal

Details

The correction factors generated by this command are saved as calibration constants in `/opt/kiS530/cvCAL.ini`. These calibration constants are used by the `hvcv_intgcg` command.

`CompOpen`, `CompShort`, and `CompLoad` devices must be connected to run this procedure. Select a `CompLoad` device with a value close to the capacitance you are measuring. If you are configuring three-terminal capacitance measurements, use a 1 nF to 2 nF capacitor.

This command uses the low-voltage CMTR as a calibration tool to provide load values for high-voltage CMTR characterization.

This command returns a status:

- 1 = Success
- -1 = Low or high pins are not defined
- -2 = Open measurement failed
- -3 = Open correction canceled
- -4 = Short measurement failed
- -5 = Short correction canceled
- -6 = Load measurement failed
- -7 = Load correction canceled
- -8 = DUT/load measurement failed
- -9 = Failed to open `/opt/kiS530/cvCAL.ini` file
- -10 = Failed running compensation calculations

Example

```
status= hvcv_genCompData(pin1, pin2)
```

Generates compensation factors for pin 1 and pin 2.

Also see

None

hvcv_genCompFreq

This command generates compensation factors for system-level capacitance compensation for a single specified frequency.

Models supported

S540

Usage

```
int hvcv_genCompFreq(int hpin, int lpin, int epin, int CMTRs, double Freq, double Cp,
    double Gp, double *CpCalc, double *GpCalc)
```

hpin	Input	Pin for the capacitance meter (CMTR) high signal
lpin	Input	Pin for CMTR low signal
epin	Input	Extra pin
CMTRs	Input	Number of CMTRs to use to do capacitance-voltage compensation: 1 = Use only high-voltage CMTR for compensation measurements; for load values, use the *CP* and *GP* parameters 2 = Use both high-voltage and low-voltage CMTRs to obtain *Cp* and *Gp* values (user-specified values using the *Cp* and *Gp* parameters are ignored); the low-voltage CMTR obtains the *Cp* and the high-voltage CMTR obtains the *Gp*.
Freq	Input	Frequency
Cp	Input	Expected or known value for compensated capacitance; use when low-voltage CMTR cannot be used to collect *Cp* value
Gp	Input	Expected or known value for compensated conductance; use when low-voltage CMTR cannot be used to collect *Gp* value
CpCalc	Output	Corrected value for compensated capacitance; should be close to the expected, known, and measured values on low-voltage CMTR
GpCalc	Output	Corrected value for compensated conductance; should be close to the expected, known, and measured values on low-voltage CMTR

Details

This command can be used in two different CMTR configurations, as specified by the *CMTRs* parameter:

- 1 = Using only a high-voltage CMTR connected through bias tees; you must provide values for the load device, compensated capacitance, and compensated conductance
- 2 = Using a low-voltage CMTR and a high-voltage CMTR; the low-voltage CMTR bypasses the bias tees and provides data for the load device (user-specified values using the *Cp* and *Gp* parameters are ignored)

When the command runs successfully, correction factors are displayed on the computer. You can then add these values to the `/opt/kiS530/cvCAL.ini` file.

NOTE

The correction factors are not automatically added to the `/opt/kiS530/cvCAL.ini` file; you must add them.

This command returns a status:

- 1 = Success
- -1 = Low or high pins are not defined
- -2 = Open measurement failed
- -3 = Open correction canceled
- -4 = Short measurement failed
- -5 = Short correction canceled
- -6 = Load measurement failed
- -7 = Load correction canceled
- -8 = DUT or load measurement failed
- -9 = Low-voltage measurement of DUT failed
- -10 = Storing expected value failed
- -11 = CompShortOpenLoad compensation routine failed
- -12 = Correction data does not match expected data

Example

```
Freq = 1e5
Cp = 1.23e-9
Gp = 4.5e-6
CMTRs = 2
status = hvcv_genCompFreq(pin1, pin2, -1, CMTRs, Freq, Cp, Gp, CpCalc, GpCalc);
```

Collects the compensation factor for one frequency.

Also see

None

hvcv_getData

This command gets compensated capacitance (Cp) and compensated conductance (Gp) data from the data pool.

Models supported

S540

Usage

```
int hvcv_getData(char *label, char *dut, double Freq, double *Cp, double *Gp)
```

label	Input	Device name (label), for example, `DUT1` or `p1_3`
dut	Input	Device type (`dut`, `open`, `short`, `load`, `loadEx`, `shortEx`, or `openEx`)
Freq	Input	Frequency (`1e4` to `2e6`)
Cp	Output	Compensated capacitance value
Gp	Output	Compensated conductance value

Details

This command gets *Cp* and *Gp* data from the data pool using a keyword that specifies which device to get the data from. The keyword is derived by combining the device name (label), device type, and frequency. For example, the keyword `trans1_dut_10000` identifies the device named `trans1`, with a device type of `dut`, at a frequency of 1e+4 Hz.

This command returns a status:

- `1` = Success
- `-1` = Device label is NULL
- `-2` = Device label is less than 2 characters or more than 64
- `-3` = DUT is not one of the following: `dut`, `open`, `short`, `load`, `loadEx`, `shortEx`, or `openEx`
- `-4` = Frequency is out of valid range (1e4 to 2e6)
- `-5` = Failed to read values from data pool

Example

```
status = hvcv_getData("pin1_pin2", "dut", 1e5, Cp, Gp)
```

Gets capacitance-voltage (C-V) data with the label `pin1_pin2` from a data pool.

Also see

None

hvcv_intgcg

This command measures capacitance and does system-level ShortOpenLoad compensation on the high-voltage capacitance meter (CMTR).

Models supported

S540

Usage

```
void hvcv_intgcg(int instr, int doComp, double Freq, double *Cp, double *Gp)
```

instr	Input	High-voltage CMTR (`CMTR2`)
doComp	Input	Specifies whether to do ShortOpenLoad compensation: ▪ `0` = Do not do ShortOpenLoad compensation ▪ `1` = Do ShortOpenLoad compensation using a system-level file installed on the system (`cvCALsystem.ini`) ▪ 2 = Do ShortOpenLoad compensation using a user-created file (`cvCAL.ini`)
Freq	Input	Frequency
Cp	Input	Capacitance value, according to the parallel capacitor model
Gp	Input	Conductance value, according to the parallel capacitor model

Details

This command does the following:

- Reads compensation `CompOpen`, `CompShort`, and `gain` correction parameters from `/opt/kiS530/cvCAL.ini`
- Makes a standard capacitance-voltage (C-V) measurement using the `intgcv` LPT command
- Does `CompOpen`, `CompShort`, and `CompLoad` compensation on the C-V measurements

Use this command instead of the `intgcg` Linear Parametric Test (LPT) command when you need to compensate for connections through bias tees.

The `hvcv_intgcg` command measures capacitance like the standard `intgcg` command, but it also does system-level compensation using a single set of constants stored in the `/opt/kiS530/cvCAL.ini` file. These constants are created using the `hvcv_genCompData` and `hvcv_genCompFreq` commands.

The instrument specified by the `instr` parameter must be a high-voltage CMTR (`CMTR2`).

Example

```
hvcv_intgcg(CMTR2, 1, 1e5, Cp, Gp)
```

Measures capacitance and does system-level ShortOpenLoad compensation on the high-voltage capacitance meter.

Also see

hvcv_genCompData (on page 4-14)
hvcv_genCompFreq (on page 4-16)

hvcv_measure

This command measures and stores compensated capacitance (Cp) and compensated conductance (Gp) values.

Models supported

S540

Usage

```
int hvcv_measure(int instr, char *label, char *dut, double Freq, double ACV, double PLC,
    int doComp, double *Cp, double *Gp, double *D)
```

instr	Input	Capacitance meter (CMTR) instrument ID
label	Input	Device name (label), for example, `DUT1` or `p1_3`
dut	Input	Device under test; valid options: ■ `dut` = Test the DUT itself with the high-voltage capacitance meter (CMTR) ■ `open` = Characterize the open device using the high-voltage CMTR; this can be done with the chuck down (pins not in contact with the device) ■ `short` = Characterize the short device using the high-voltage CMTR ■ `load` = Characterize the load device using the high-voltage CMTR ■ `shortEx` = Characterize the short device using the low-voltage capacitance meter (CMTR); this data is used to do `CompShort` compensation of the *`loadEx`* device ■ `loadEx` = Measure the load device using the low-voltage CMTR to get the expected value of the `loadEx` device ■ `openEx` = Characterize the open device using the low-voltage CMTR; this data is used to do `CompOpen` compensation of the `loadEx` device
Freq	Input	Frequency (`1e4` to `2e6`)
ACV	Input	AC amplitude level
PLC	Input	Power line integration time (recommend `1` to `3` PLC)
doComp	Input	Specifies whether to do system-level compensation: ■ `0` = Do not do ShortOpenLoad compensation ■ `1` = Do ShortOpenLoad compensation using `cvCALsystem.ini` ■ 2 = Do ShortOpenLoad compensation using a user-created file (`cvCAL.ini`)
Cp	Output	Compensated capacitance value
Gp	Output	Compensated conductance value
D	Output	Dissipation factor

Details

This command does the following:

- Verifies input conditions
- Configures the CMTR with the specified *ACV*, *Freq*, and *PLC*
- Disables all compensation operations on the CMTR
- Configures a parallel measurement model (CpGp)
- If the CMTR is high-voltage and requires system-level compensation, this command calls the `hvcv_intgcg` command, which does system-level ShortOpenLoad compensation; if the CMTR is low-voltage, this command calls the `intgcg` LPT command, which does not do system-level compensation
- Makes capacitance measurements
- Using the `hvcv_storeData` command, stores Cp and Gp data with the specified *label*, *dut*, and *freq* parameters in the data pool

This command returns a status:

- `1` = Success
- `-1` = Device label is NULL
- `-2` = Device label is less than 2 characters or more than 64
- `-3` = The *dut* parameter is not one of the following: `dut`, `open`, `short`, `load`, `loadEx`, `shortEx`, or `openEx`
- `-4` = Frequency is out of the valid range (`1e4` to `2e6`)
- `-5` = The *ACV* parameter value exceeds 0.1 V or less than 0.01 V
- `-6` = The *PLC* parameter value is out of range (`0.1` to `30`)

Example

```
ACV = 0.1
doComp = 1
status = hvcv_measure(CMTR1, "pin1_pin2", "dut", 1e5, ACV, 1, doComp, Cp, Gp, D)
```

Measures and stores Cp and Gp values from CMTR1 to the data pool under the label `pin1_pin2`.

Also see

hvcv_intgcg (on page 4-19)

hvcv_storeData

This command stores compensated capacitance (Cp) and compensated conductance (Gp) data in the data pool.

Models supported

S540

Usage

```
int hvcv_storeData(char *label, char *dut, double Freq, double Cp, double Gp)
```

label	Input	Device name (label), for example, `DUT1` or `p1_3`
dut	Input	Device type (`dut`, `open`, `short`, `load`, `loadEx`, `shortEx`, or `openEx`)
Freq	Input	Frequency (`1e4` to `2e6`)
Cp	Input	Compensated capacitance value
Gp	Input	Compensated conductance value

Details

This command stores *Cp* and *Gp* data in the data pool under a keyword that identifies a specific device. The keyword is derived by combining the device name (label), device type, and frequency. For example, the keyword `trans1_dut_10000` identifies the device named `trans1`, with a device type of `dut`, at a frequency of 1e+4 Hz.

This command returns a status:

- `1` = Success
- `-1` = Device label is NULL
- `-2` = Device name is less than two characters or more than 64
- `-3` = The *dut* parameter is not one of the following values: `dut`, `open`, `short`, `load`, `loadEx`, or `openEx`
- `-4` = Frequency is out of valid range (`1e4` to `2e6`)

Example

```
status = hvcv_storeData("pin1_pin2", "dut", 1e5,12.2e-12,1.56e-8)
```

Stores capacitance-voltage (C-V) data in the data pool with the label `pin1_pin2`.

Also see

None

hvcv_sweep

This command does a high-voltage capacitance-voltage (C-V) sweep.

Models supported

S540

Usage

```
int hvcv_sweep(int high_pin1, int high_pin2, int high_pin3, int low_pin1, int low_pin2,
    int low_pin3, char *dut, char *comp_mode, int doComp, char forceSide, int doRetest,
    double Freq, double startV, double stopV, double *Vbias, int Vpts, double *Ileak,
    int Ipts, double *Cp, int CpPts, double *D, int Dpts, double *Gp, int GpPts)
```

high_pin1	Input	First pin to connect to high-voltage capacitance meter (CMTR) high side
high_pin2	Input	Second pin to connect to high-voltage CMTR high side
high_pin3	Input	Third pin to connect to high-voltage CMTR high side
low_pin1	Input	First pin to connect to high-voltage CMTR low side
low_pin2	Input	Second pin to connect to high-voltage CMTR low side
low_pin3	Input	Third pin to connect to high-voltage CMTR low side
dut	Input	Device under test; valid options: ■ `dut` = Test the DUT itself with the high-voltage capacitance meter (CMTR) ■ `open` = Characterize the open device using the high-voltage CMTR; this can be done with the chuck down (pins not in contact with the device) ■ `short` = Characterize the short device using the high-voltage CMTR ■ `load` = Characterize the load device using the high-voltage CMTR ■ `shortEx` = Characterize the short device using the low-voltage capacitance meter (CMTR); this data is used to do `CompShort` compensation of the *`loadEx`* device ■ `loadEx` = Measure the load device using the low-voltage CMTR to get the expected value of the `loadEx` device ■ `openEx` = Characterize the open device using the low-voltage CMTR; this data is used to do `CompOpen` compensation of the `loadEx` device

comp_mode	Input	Compensation type: ▪ CompNone (use this if you do not want to run any compensation or if the *dut* parameter is set to anything other than dut) ▪ CompOpen ▪ CompShort ▪ CompLoad ▪ CompOpenLoad ▪ CompShortOpen ▪ CompShortLoad ▪ CompShortOpenLoad
doComp	Input	Specifies whether to do ShortOpenLoad compensation: ▪ 0 = Do not do ShortOpenLoad compensation ▪ 1 = Do ShortOpenLoad compensation using a system-level file installed on the system (cvCALsystem.ini) ▪ 2 = Do ShortOpenLoad compensation using a user-created file (cvCAL.ini)
forceSide	Input	Side used to force DC bias voltage: "H" = High (CMTR1H, CMTR2H) "L" = Low (CMTR1L, CMTR2L)
doRetest	Input	Specifies whether to remeasure compensation data: ▪ 0 = Do not remeasure compensation data ▪ 1 = Remeasure compensation data once, then reuse that measurement in any additional calls See **Details** for more information
Freq	Input	Frequency (1e4 to 2e6)
startV	Input	Sweep start bias
stopV	Input	Sweep stop bias
Vbias	Output	Sweep of voltage bias points
Vpts	Input	Sweep size; must be same as *Ipts*, *CpPts*, *Dpts*, *GpPts*
Ileak	Output	Leakage current
Ipts	Input	Sweep size; must be same as *Vpts*, *CpPts*, *Dpts*, *GpPts*
Cp	Output	Compensated capacitance value
CpPts	Input	Sweep size; must be same as *Vpts*, *Ipts*, *Dpts*, *GpPts*
D	Output	Compensated dissipation factor
Dpts	Input	Sweep size; must be same as *Vpts*, *Ipts*, *CpPts*, *GpPts*
Gp	Output	Compensated conductance value
GpPts	Input	Sweep size; must be same as *Vpts*, *Ipts*, *CpPts*, *Dpts*

Details

This command does the following:

- Verifies input conditions and checks pins
- Checks whether the compensation mode (*comp_mode*) is valid
- Makes connections to CMTR1 and CMTR2
- Uses the high-voltage CMTR (CTMR2) for the *dut* parameter options dut, open, short, and load; uses the low-voltage CMTR (CMTR1) for *dut* parameter options loadEx, openEx, and shortEx
- Forces sweep voltage and measures current
- Calls the hvcv_measure command to measure *Cp* and *Gp*
- When the *dut* parameter is set to open or openEx, the routine moves the chuck down, measures, and moves the chuck up again
- Runs compensation according to the compensation mode (*comp_mode*)

This command returns a status:

- 1 = Success
- -1 = No valid pins
- -2 = Wrong number of points
- -3 = No valid compensation mode is specified; valid options are CompNone, CompOpen, CompLoad, CompShort, CompShortOpen, CompShortEx, CompShortOpenLoad
- -4 = No valid DUT is specified; valid options are dut, open, and short
- -5 = Frequency is out of valid range (1e4 to 2e6)
- -6 = Error with prober chuck moving down
- -7 = Error in the compensation procedure
- -8 = Low-voltage pin is used for high-voltage test

Use the *doRetest* parameter to save time when you are characterizing a compensation device (open, short, or load) in an automated setting where test macros are repeated multiple times on a wafer or group of wafers. When this parameter is set to 1, the compensation device is retested once the first time the test macro is encountered. Any further calls to the test macro during the test plan run automatically use the value from the retest.

Example

```
status = hvcv_sweep(pin1, -1, -1, pin2, -1, -1, "dut", "CompNone", 1, "H", 1,
         1e5, 0, 10, Vbias, 11, Ileak, 11, Cp, 11, D, 11, Gp, 11)
```

Performs an 11-point high-voltage C-V sweep.

Also see

hvcv_comp (on page 4-13)
hvcv_measure (on page 4-21)
hvcv_sweep_basic (on page 4-27)

hvcv_sweep_basic

This command does a high-voltage capacitance-voltage (C-V) sweep.

Models supported

S540

Usage

```
int hvcv_sweep(int high_pin1, int high_pin2, int high_pin3, int low_pin1, int low_pin2,
    int low_pin3, char forceSide, double Freq, double startV, double stopV, double
    *Vbias, int Vpts, double *Ileak, int Ipts, double *Cp, int CpPts, double *D, int Dpts,
    double *Gp, int GpPts)
```

high_pin1	Input	First pin to connect to high-voltage capacitance meter (CMTR) high side
high_pin2	Input	Second pin to connect to high-voltage CMTR high side
high_pin3	Input	Third pin to connect to high-voltage CMTR high side
low_pin1	Input	First pin to connect to high-voltage CMTR low side
low_pin2	Input	Second pin to connect to high-voltage CMTR low side
low_pin3	Input	Third pin to connect to high-voltage CMTR low side
forceSide	Input	Side used to force DC bias voltage: "H" = High (CMTR1H, CMTR2H) "L" = Low (CMTR1L, CMTR2L)
Freq	Input	Frequency (1e4 to 2e6)
startV	Input	Sweep start bias
stopV	Input	Sweep stop bias
Vbias	Output	Sweep of voltage bias points
Vpts	Input	Sweep size; must be same as *Ipts*, *CpPts*, *Dpts*, *GpPts*
Ileak	Output	Leakage current
Ipts	Input	Sweep size; must be same as *Vpts*, *CpPts*, *Dpts*, *GpPts*
Cp	Output	Compensated capacitance value
CpPts	Input	Sweep size; must be same as *Vpts*, *Ipts*, *Dpts*, *GpPts*
D	Output	Compensated dissipation factor
Dpts	Input	Sweep size; must be same as *Vpts*, *Ipts*, *CpPts*, *GpPts*
Gp	Output	Compensated conductance value
GpPts	Input	Sweep size; must be same as *Vpts*, *Ipts*, *CpPts*, *Dpts*

Details

This command does the following:

- Verifies input conditions and checks pins
- Forces sweep voltage and measures current
- Calls the `hvcv_measure` command to measure *Cp* and *Gp*

This command returns a status:

- `1` = Success
- `-1` = No valid pins
- `-2` = Wrong number of points
- `-5` = Frequency is out of valid range (`1e4` to `2e6`)
- `-6` = Error with prober chuck moving down
- `-8` = Low-voltage pin is used for high-voltage test

Example

```
status = hvcv_sweep(pin1, -1, -1, pin2, -1, -1, "H", 1e5, 0,
        10, Vbias, 11, Ileak, 11, Cp, 11, D, 11, Gp, 11)
```

Performs an 11-point high-voltage C-V sweep.

Also see

hvcv_measure (on page 4-21)
hvcv_sweep (on page 4-24)

hvcv_test

This command makes a high-voltage capacitance-voltage (C-V) measurement at a single frequency.

Models supported

S540

Usage

```
int hvcv_test(int high_pin1, int high_pin2, int high_pin3, int low_pin1, int low_pin2,
    int low_pin3, char *dut, char *comp_mode, int doComp, int doRetest, double Freq,
    double biasV, double *Cp, double *Gp, double *D, double *iCurr)
```

high_pin1	Input	First pin to connect to high-voltage capacitance meter (CMTR) high side
high_pin2	Input	Second pin to connect to high-voltage CMTR high side
high_pin3	Input	Third pin to connect to high-voltage CMTR high side
low_pin1	Input	First pin to connect to high-voltage CMTR low side
low_pin2	Input	Second pin to connect to high-voltage CMTR low side
low_pin3	Input	Third pin to connect to high-voltage CMTR low side
dut	Input	Device under test; valid options: ■ `dut` = Test the DUT itself with the high-voltage capacitance meter (CMTR) ■ `open` = Characterize the open device using the high-voltage CMTR; this can be done with the chuck down (pins not in contact with the device) ■ `short` = Characterize the short device using the high-voltage CMTR ■ `load` = Characterize the load device using the high-voltage CMTR ■ `shortEx` = Characterize the short device using the low-voltage capacitance meter (CMTR); this data is used to do `CompShort` compensation of the *loadEx* device ■ `loadEx` = Measure the load device using the low-voltage CMTR to get the expected value of the `loadEx` device ■ `openEx` = Characterize the open device using the low-voltage CMTR; this data is used to do `CompOpen` compensation of the `loadEx` device

comp_mode	Input	Compensation type: ■ CompNone (use this if you do not want to run any compensation or if the *dut* parameter is set to anything other than dut) ■ CompOpen ■ CompShort ■ CompLoad ■ CompOpenLoad ■ CompShortOpen ■ CompShortLoad ■ CompShortOpenLoad
doComp	Input	Specifies whether to do system-level compensation: ■ 0 = Do not do ShortOpenLoad compensation ■ 1 = Do ShortOpenLoad compensation using cvCALsystem.ini ■ 2 = Do ShortOpenLoad compensation using a user-created file (cvCAL.ini)
doRetest	Input	Specifies whether to remeasure compensation data: ■ 0 = Do not remeasure compensation data ■ 1 = Remeasure compensation data once, then reuse that measurement in any additional calls See **Details** for more information
Freq	Input	Frequency (1e3 to 3e6)
biasV	Input	Voltage bias
Cp	Output	Compensated capacitance value
Gp	Output	Compensated conductance value
D	Output	Compensated dissipation factor
iCurr	Output	Leakage current at the biasV voltage

Details

This command does the following:

- Verifies input conditions and checks pins
- Checks whether the compensation mode (*comp_mode*) is valid
- Makes connections to CMTR1 and CMTR2
- Uses the high-voltage CMTR (CTMR2) for the *dut* parameter options dut, open, short, and load; uses the low-voltage CMTR (CMTR1) for *dut* parameter options loadEx, openEx, and shortEx
- Forces *biasV*
- Measures Cp and Gp by calling the hvcv_measure command
- When the *dut* parameter is set to open or openEx, the routine moves the chuck down, measures, and moves the chuck up again

- Runs compensation as specified by the *comp_mode* parameter

This command returns a status:

- 0 = Skip system-level compensation
- 1 = Success
- -1 = No valid pins
- -2 = No valid compensation mode is specified; valid options are CompNone, CompOpen, CompLoad, CompShort, CompShortOpen, CompShortEx, CompShortOpenLoad
- -3 = No valid *dut* parameter is specified; valid options are dut, open, or short
- -4 = Frequency is out of valid range (1e3 to 3e6)
- -5 = Error with PrChuck
- -6 = Error in the compensation procedure
- -7 = Low-voltage pin is used for high-voltage test

If compensation data (open, short, load, loadEx, openEx, shortEx) is not available before DUT testing, an error is generated.

This command collects dut, open, short, or load data with a high-voltage CMTR on the dut, open, short, or load device.

This command collects openEx, loadEx, or shortEx data with a low-voltage CMTR on an open, load, or short structure.

The *doComp* parameter provides a switch that enables or disables system-level compensation. To do ShortOpenLoad compensation using a system-level compensation file that is stored on the system (cvCALsystem.ini), set this parameter to 1. To do ShortOpenLoad compensation using a user-generated compensation file (cvCAL.ini), set this parameter to 2.

Example

```
doRetest = 1
doComp = 1
status = hvcv_test(pin1, -1, -1, pin2, -1, -1, "open", "CompNone", doComp,
        doRetest, 1e5, 0.0, Cp, Gp, D, ICurr)
```

Makes a single-point C-V measurement.

Also see

hvcv_comp (on page 4-13)
hvcv_measure (on page 4-21)
hvcv_test_basic (on page 4-32)

hvcv_test_basic

This command makes a high-voltage capacitance-voltage (C-V) measurement at a single frequency.

Models supported

S540

Usage

```
int hvcv_test(int high_pin1, int high_pin2, int high_pin3, int low_pin1, int low_pin2,
    int low_pin3, double Freq, double biasV, double *Cp, double *Gp, double *D, double
    *iCurr)
```

high_pin1	Input	First pin to connect to high-voltage capacitance meter (CMTR) high side
high_pin2	Input	Second pin to connect to high-voltage CMTR high side
high_pin3	Input	Third pin to connect to high-voltage CMTR high side
low_pin1	Input	First pin to connect to high-voltage CMTR low side
low_pin2	Input	Second pin to connect to high-voltage CMTR low side
low_pin3	Input	Third pin to connect to high-voltage CMTR low side
Freq	Input	Frequency (1e3 to 3e6)
biasV	Input	Voltage bias
Cp	Output	Compensated capacitance value
Gp	Output	Compensated conductance value
D	Output	Compensated dissipation factor
iCurr	Output	Leakage current at the *biasV* voltage

Details

This command does the following:

- Verifies input conditions and checks pins
- Makes connections to CMTR2
- Forces biasV
- Measures Cp and Gp by calling the `hvcv_measure` command

This command returns a status:

- 1 = Success
- -1 = No valid pins
- -4 = Frequency is out of valid range (1e3 to 3e6)
- -5 = Error with PrChuck
- -7 = Low-voltage pin is used for high-voltage test

Example

```
status = hvcv_test(pin1, -1, -1, pin2, -1, -1, 1e5, 0.0, Cp, Gp, D, ICurr)
```

Makes a single-point C-V measurement.

Also see

hvcv_measure (on page 4-21)
hvcv_test (on page 4-29)

Section 5

KI_MultiSite command reference

In this section:

Introduction 5-1
multi_site_clear_mapping() 5-1
multi_site_mapping() 5-3

Introduction

The `KI_MultiSite` user library (usrlib) is included with every Model S535 system for dual-site support. This user library manages the dual-site `conpin` mirroring using the `site_mapping` LPTLib function. The following topics describe the commands in this user library.

NOTE

The commands in this library are only supported for the S535 Wafer Acceptance Test System.

For information about the LPTLib commands you can use for multisite testing, see the LPTLib command reference (on page 2-1). For more information about dual-site testing, see the *S535 Reference Manual* (part number S535-901-01).

multi_site_clear_mapping()

This function clears all dual-site mappings from SITE0 to SITE1.

Models supported

S535

Usage

```
void multi_site_clear_mapping(void);
```

Details

Use this function to avoid MX_ILLSITE errors with the `conpin` command when completely remapping site information.

Call this command after the first call to the `devint` command.

Example

```
/* Arrays are defined in a global data file as: */
/* 1,2,3 - PinArray0 */
/* 4,5,6 - PinArray1 */
/* 1,2,3 - PinArray1a */
/* 4,5,6 - PinArray0a */

/* Map SITE0 sites. */
site_mapping(0, PinArray0, 3)

/* Map SITE1 site. */
site_mapping(1, PinArray1, 3)

conpin(SMU1,1, 0)
conpin(SMU2,2, 0)
conpin(SMU5,3, 0)
delay(1000)

/* Reset active instruments to default state and clear all site mappings. */
devint()
multi_site_clear_mapping()

/* Set up new site mappings. */
site_mapping(0, PinArray0a, 3)
site_mapping(1, PinArray1a, 3)

conpin(SMU1,1, 0)
conpin(SMU2,2, 0)
conpin(SMU5,3, 0)
delay(1000)
```

This example maps dual sites and connects dual pins, then completely changes the dual-site mappings and connects the pins. The `multi_site_clear_mapping()` command is called before setting up the new site mappings.

multi_site_mapping()

This function calls the `site_mapping()` LPTLib command for both `SITE0` anchor pins and `SITE1` mirror pins to facilitate switching between different probe card and wafer layouts in dual-site mode.

Models supported

S535

Usage

```
multi_site_mapping();
```

Details

For easy switching between different probe card and wafer layouts during dual-site operation, add the `multi_site_mapping()` command to the `UAP_CASSETTE_LOAD` user access point (UAP).

NOTE

If you have multiple pin mappings per cassette, you must update the mapping in the global data file (`*.gdf`) and then manually call the `multi_site_mapping()` function each time the mapping has changed.

If you are using the Keithley Test Execution Engine (KTXE), you do not need to call this command again after you have added it to the `UAP_CASSETTE_LOAD` UAP.

If you are using the Keithley Interactive Test Tool (KITT), you must use the `multi_site_mapping()` command before calling any multisite `conpin` commands.

ATTENTION

When defining pin mappings for dual-site operation, you must append `_site1` to the variable names of the mirror-site pins in the global data file (`.gdf`). If a pin is not defined in the global data file, the Keithley Test Environment (KTE) will not close a dual pin. If you do not want to use the `KI_MultiSite` library, you can manually generate the pin mapping and call the `site_mapping` LPTLib command directly.

NOTE

If you have a chuck connected in an S535 system that is in dual-site mode, the pin the chuck is on cannot be mirrored.

Example

```
/* Global data file variables: */
source        2
sense         8
gate          3,4,6
source_site1  5
sense_site1   9
gate_site1    18,19,21

/* KULT code: */
conpin(SMU1, 2, 0);
/* The conpin command also connects the mirrored SMU to pin 5.*/
conpin(SMU2, 3, 4, 5, 0):
/* The conpin command also connects the mirrored SMU to pins 18, 19, 21.*/
```

Section 6

Prober and prober driver command reference

In this section:

Introduction 6-1

Introduction

The prober and prober driver commands are described in detail in this section.

NOTE

Not all commands are supported by all probers. Refer to the documentation for your prober or contact your field service engineer (FSE) for information about prober compatibility.

PrAbsMove

This command tells the prober to move an absolute number of millimeters relative to the wafer origin.

Usage

```
status = PrAbsMove(double x_value, double Y_value);
```

x_value	The number of millimeters to move along the x-axis relative to the wafer origin
y_value	The number of millimeters to move along the y-axis relative to the wafer origin

Details

The `PrRelMove` command, by contrast, tells the prober to move a relative number of millimeters relative to the present position.

PrAdjustZHeight

This command tells the prober to move the Z-stage (chuck surface) a relative number of units (as defined on the prober) up/contact (+) or down/separate (–).

Usage

```
status = PrAdjustZHeight(double z_height);
```

z_height	The number of units to move the chuck from its current position; a valid Z height is defined for each prober

Details

None

PrAutoAlign

This command automatically aligns the prober based on the optical pattern specified during the wafer setup on the prober.

Usage

```
status = PrAutoAlign();
```

Details

Before using this command, an auto alignment must be stored on the prober.

The KTXE execution engine only runs this command if the `PrCheckOptions` command has a return value that indicates that the prober has all the features and options required for this command to execute.

PrCassetteMap

This command returns a map of the slot status for the specified cassette to an integer array.

Usage

```
status = PrCassetteMap (int cassette_number, int *cassette_map);
```

cassette_number	The cassette from which to return the map of the slot status; valid values are `1` to `PROBER_n_MAX_CASSETTE`
cassette_map	An array representing the status of slots in the cassette; `0` to `PROBER_n_MAX_SLOT - 1`

Details

The *cassette_number* parameter identifies the cassette (in a single or multi-cassette prober) for which a map will be returned. The value of the *cassette_number* parameter ranges from `1` to `PROBER_n_MAX_CASSETTE`. `PROBER_n_MAX_CASSETTE` is a constant in the `KIDAT/prbcnfg_XXXX.dat` file that represents the total number of cassettes contained in the prober. This file may also be referenced by its environmental variable, `KI_PRB_CONFIG`. The *n* in `PROBER_n_MAX_CASSETTE` is substituted by the test station number in the actual file.

The *cassette_map* array is a map of the slot statuses of the cassette. The size of the array, which corresponds to the number of slots in the cassette, is given by the constant, `PROBER_n_MAX_SLOT`; this constant is in the `KIDAT/prbcnfg_XXXX.dat` file. The array is indexed from `0` to (`PROBER_n_MAX_SLOT - 1`). Each element of the array represents a single slot and is assigned one of the statuses listed below. Each status has a corresponding constant integral value; these values may be in the `prb.h` file:

```
PR_SLOT_STATUS_UNMAPPED
PR_SLOT_STATUS_IN PROCESS
PR_SLOT_STATUS_EMPTY
PR_SLOT_STATUS_UNPROBED
PR_SLOT_STATUS_PROBED
PR_SLOT_STATUS_PROBLEM
PR_SLOT_STATUS_UNSCHEDULED
DEFAULT_SLOT_STATUS
```

The KTXE execution engine only runs this command if the `PrCheckOptions` command has a return value that indicates that the prober has all the features and options required for this command to execute.

PrCassetteMask

This command allows you to select which slots in the specified cassette are available for probing. This command must be used with the `PrSetSlotStatus` command.

Usage

```
status = PrCassetteMask (int cassette);
```

cassette	The number of millimeters to move along the x-axis relative to the wafer origin

Details

This command is only valid on P8XL model probers.

A series of calls to the `PrSetSlotStatus` command are followed by a call to the `PrCassetteMask` command. This specifies which wafers on the prober are probed.

PrCheckOptions

This command returns which optional features are present on the prober.

Usage

```
status = PrCheckOptions(int *OcrPresent, int *AutoAlnPresent, int *ProfilerPresent, int
    *HotchuckPresent, int *HandlerPresent, int *Probe2PadPresent);
```

OcrPresent	The status of the OCR subsystem: `1` = OCR present `0` = No OCR
AutoAlnPresent	The status of the autoalign subsystem: `1` = Autoalign enabled `0` = Autoalign disabled
ProfilerPresent	The status of the profiler subsytem: `1` = Profiler present and enabled `0` = Profiler disabled
HotchuckPresent	The status hot chuck: `1` = Hot chuck present `0` = Option disabled or not installed
HandlerPresent	The status of the random access handler: `1` = Random access system present `0` = Random access system disabled
Probe2PadPresent	The status of the automatic probe-to-pad system: `1` = Probe-to-pad enabled `0` = Probe-to-pad disabled

Details

The KTXE execution engine uses this command.

PrChuck

This command raises or lowers the chuck.

Usage

```
status = PrChuck(int chuck_position);
```

chuck_position	The position to move the chuck to: `PR_CHUCK_DOWN` = Lower chuck `PR_CHUCK_UP` = Raise chuck

Details

The KTXE execution engine uses this command.

PrClearAll

This command returns all wafers to the cassette and terminates the lot.

Usage

```
status = PrClearAll();
```

Details

This command is valid only with a prober equipped with SMIF technology.

PrClearPipeline

This command clears the pipeline of all wafers.

Usage

```
status = PrClearPipeline();
```

Details

This command unloads all wafers on the quick-loader and realigner and returns them to their respective slots in their cassettes of origin.

PrError

This command obtains information about a prober error.

Usage

```
status = PrError();
```

Details

This command returns an encoded prober error number that must be decoded before it can be compared with the numbers listed in the error documentation for your prober. Look up the encoded error number in the documentation supplied by the prober manufacturer.

The decoding equation is as follows:

prober manufacturer’s error number = –(return value + 1500)

The number returned may be an error number or a return status code.

The KTXE execution engine uses this command.

PrGetNxtWafer

This command loads the next unprobed wafer from the specified cassette to the chuck.

Usage

```
status = PrGetNxtWafer(int cassette_number);
```

cassette_number	The number of the source cassette from which to load a wafer

Details

The *cassette_number* parameter specifies a particular source cassette. The value of the cassette number can be from `1` to a defined constant. The constant is defined in the file pointed to by `KI_PRB_CONFIG` (`PROBER_n_MAX_CASSETTE`, where *n* = test station number). See the example in the `KIDAT/prbcnfg_XXXX.dat` file.

If a wafer is presently on the chuck, it is unloaded to its origin slot and cassette. When called as a function, this command returns the slot number of the chosen wafer. If there are no unprobed wafers in the specified cassette, the command generates an error message and number.

PrGetProduct

This command instructs the prober to return the currently loaded product file.

Usage

```
status = PrGetProduct(char *file_name);
```

file_name	Variable containing the product file name currently loaded on the prober (pointer to `char` array)

Details

This command is not used by the KTXE execution engine.

PrGetWafer

This command loads a wafer from the specified cassette and slot to the chuck.

Usage

```
status = PrGetWafer(int cassette_number, int slot_number);
```

cassette_number	The cassette containing the wafer to load
slot_number	The location of the wafer to load

Details

This command loads a wafer from a specified source cassette. The value of the cassette number can be from `1` to a defined constant. The constant is defined in the file pointed to by `KI_PRB_CONFIG` (`PROBER_n_MAX_CASSETTE`, where *n* = test station number). See the example in the `KIDAT/prbcnfg_XXXX.dat` file.

If a wafer is presently on the chuck, it is unloaded to its original slot and cassette.

The parameters for this command are integers.

PrInit

This command initializes the prober with the following information: Probing mode, die size, first coordinates, and units.

Usage

```
status = PrInit(int mode, double x_die_size, double y_die_size, int x_start_coordinate,
    int y_start_coordinate, int units, int sub_type);
```

mode	The probing mode to use:
	`PR_MODE_MANUAL`
	`PR_MODE_EXTERNAL`
	`PR_MODE_EDGE`

x_die_size	A double floating-point value defining the X dimension of the die
y_die_size	A double floating-point value defining the Y dimension of the die
x_start_coordinate	The X coordinate for prober alignment
y_start_coordinate	The Y coordinate for prober alignment
units	The unit of measure for numerical data: `PR_ENGLISH` = Use Imperial units of measure `PR_METRIC` = Use metric units of measure
sub_type	This deprecated parameter is always set to `0`; it is included here for compatibility with older systems

Details

The KTXE execution engine uses this command.

The selected probing mode must support the specific prober model you have. The mode options are defined in the `prb.h` file.

The *x_die_size* and *y_die_size* parameters define the dimensions of the die. The values entered for these parameters depend on the units chosen to represent the data. The units selected (millimeters or mils) are specified in the *units* parameter.

The *x_start_coordinate* and *y_start_coordinate* parameters assign the X and Y coordinates to the location of the prober at alignment. This assignment defines the origin of the coordinate system for the wafer, providing a point of reference from which other locations may be identified.

The argument options for the *units* parameter are defined in the `prb.h` file.

PrLoad

This command unloads the wafer currently on the chuck and loads, profiles, and aligns the next wafer to be tested.

Usage

```
status = PrLoad();
```

Details

The KTXE execution engine uses this command.

PrLoadProduct

This command instructs the prober to load a product file from the specified drive.

Usage

```
status = PrLoadProduct (char *file_name, char *drive_name);
```

file_name	An array that contains the name of the product file to be loaded
drive_name	The location in memory that contains the drive from which to load the product file

Details

The *drive_name* parameter points to the location in memory that contains the identifying letter of the drive from which to load the product file. Use `NULL` for the value of this parameter to specify the default drive (usually a hard drive).

NOTE

TEL P8 probers do not use the *drive_name* parameter; the external interface of the prober does not allow you to select a drive.

The KTXE execution engine only runs this command if the `PrCheckOptions` command has a return value that indicates that the prober has all the features and options required for this command to execute.

PrLowerBoat

This command lowers the indexer and maps the specified cassette.

Usage

status = PrLowerBoat(int *i_pod_number*);

i_pod_number	The number of the cassette to map

Details

This command is valid only with a prober equipped with SMIF technology.

PrMove

This command tells the prober to move to the specified x, y location when the prober is running in external mode.

Usage

status = PrMove(int *x_location*, int *y_location*, int *ink_number*);

x_location	The X location of the die to which the prober should move after inking the present die
y_location	The Y location of the die to which the prober should move after inking the present die
ink_number	The inker to fire

Details

The KTXE execution engine uses this command.

PrNeedleClean

This command directs the prober to clean the needles on the probe card using internal prober cleaning methods.

Usage

status = PrNeedleClean(int *clean_function*);

clean_function	The cleaning method to use: `PR_CLEAN_NONE` = Do not perform needle cleaning `PR_CLEAN_BRUSH` = Use prober brush `PR_CLEAN_FIBER` = Use prober fiber pad `PR_CLEAN_CERAMIC` = Use ceramic pad `PR_CLEAN_METAL` =Use metal pad `PR_CLEAN_STICKY` = Use sticky pad `PR_CLEAN_BLOW` = Use air to clean `PR_CLEAN_GENERIC` = Other unspecified cleaning station

Details

Each prober supports different needle cleaning types; the availability of each cleaning function is determined by the specific prober.

Dependencies: `PRBCOM`

PrProfile

This command forces the profiling of a wafer.

Usage

```
status = PrProfile();
```

Details

The KTXE execution engine only runs this command if the `PrCheckOptions` command has a return value that indicates that the prober has all the features and options required for this command to execute.

PrPutNxtSlot

This command unloads the wafer presently on the chuck, puts it in the next empty slot of the specified cassette, and loads the next specified wafer onto the chuck for testing.

Usage

```
status = PrPutNxtSlot(int cassette_number, int reason_code);
```

cassette_number	The destination cassette

reason_code	The reason for unloading the wafer:
	`PR_NORMAL_UNLOAD`
	`PR_PROFILE_FAIL`
	`PR_ALIGN_FAIL`

The cassette number can be from `1` to a defined constant. The constant is defined in the file pointed to by `KI_PRB_CONFIG` (`PROBER_`*n*`_MAX_CASSETTE` where *n* = test station number). See the example in the `KIDAT/prbcnfg_XXXX.dat` file.

When called as a function, the return value for a successful execution of this command corresponds to one of two possible constants that are defined in the `prb.h` file, shown in the following table.

Constants	Meaning
`PR_WAFERCOMPLETE`	New wafers successfully loaded
`PR_CASSETTECOMPLETE`	End of lot, no more wafers
If unsuccessful, the return value corresponds to a specific error number; `<0` means various errors.	

If no slots are available in the specified cassette when this command is called as a function, it returns an error message and number. If successful, it returns the number of the destination slot. The parameters are integers.

PrPutWafer

This command unloads the wafer presently on the chuck to the specified cassette and slot.

Usage

```
status = PrPutWafer(int cassette_number, int slot_number, int reason_code);
```

cassette_number	The destination cassette
slot_number	The slot in the destination cassette to place the wafer

reason_code	The reason for unloading the wafer:
	`PR_NORMAL_UNLOAD`
	`PR_PROFILE_FAIL`
	`PR_ALIGN_FAIL`

Details

If the specified slot is unavailable, this command returns an error message and number. The parameters are integers.

The value of the cassette number can be from `1` to a defined constant. The constant is defined in the file pointed to by `KI_PRB_CONFIG` (`PROBER_`*`n`*`_MAX_CASSETTE` where *n* = test station number). See the example in the `KIDAT/prbcnfg_XXXX.dat` file.

Each reason code has a corresponding numerical code which is defined in the `prb.h` file. The parameters are integers.

The KTXE execution engine only runs this command if the `PrCheckOptions` command has a return value that indicates that the prober has all the features and options required for this command to execute.

PrQueryChuckTemp

Queries chuck temperature

Usage

```
status = PrQueryChuckTemp (double *chuck_temp);
```

chuck_temp	The memory location in which to place the temperature of the chuck

PrQueryZHeight

This command returns the current Z-stage chuck height in units defined on the prober.

Usage

```
status = PrQueryZHeight(double *z_height);
```

z_height	The present height of the Z-stage chuck

Details

None

PrReadId

This command reads wafer ID information from the prober and places it in an 80-character user buffer.

Usage

```
status = PrReadId(char *user_buf);
```

user_buf	The name of the buffer

Details

The only parameter used with this command is the name of an 80-character buffer that contains the string returned from the prober by the prober request wafer ID command.

If ID is not entered in the prober, a null string is returned. Otherwise the length of the null terminated ID and its pointer is returned.

The KTXE execution engine only runs this command if the `PrCheckOptions` command has a return value that indicates that the prober has all the features and options required for this command to execute.

PrRelMove

This command moves the prober a specified amount within a single site relative to its existing position in the site.

Usage

```
status = PrRelMove(double x_value, double y_value);
```

x_value	The number of millimeters or mils to move for the X position
y_value	The number of millimeters or mils to move for the Y position

Details

This command moves the prober a specified number of millimeters or mils in the X and Y directions on the present site relative to the present location of the probe pins. This command is used for intrasite prober moves.

The KTXE execution engine uses this command, but the command will not execute if the microprobing utilities of the prober are being used to perform the subsite probing.

PrRelReturn

This command returns the prober to the location it occupied before any subsite moves were executed.

Usage

```
status = PrRelReturn();
```

Details

After any number of `PrRelMove` commands, the `PrRelReturn` command must be executed before the prober can be moved to another site.

The KTXE execution engine uses this command.

PrSerialPoll

This command does a serial poll of the prober GPIB interface.

Usage

```
status = PrSerialPoll();
```

Details

This command is available but not used by the KTXE execution engine.

PrSetChuckTemp

This command sets the temperature of the chuck.

Usage

```
status = PrSetChuckTemp(double chuck_temp);
```

chuck_temp	The temperature to set, in °C

Details

Also see PrQueryChuckTemp (on page 6-11).

PrSetDiam

This command provides the wafer diameter to the prober.

Usage

```
status = PrSetDiam(int diameter);
```

diameter	The diameter of the wafer; expressed as an integer number of millimeters or inches

PrSetDieSize

This command sets the X, Y die size of a wafer.

Usage

```
status = PrSetDieSize(double x_die_size, double y_die_size);
```

x_die_size	The size of the die in relationship to the X coordinate; expressed in millimeters or mils
y_die_size	The size of the die in relationship to the Y coordinate; expressed in millimeters or mils

PrSetFlat

This command tells the prober in which direction to place the flat of the wafer.

Usage

```
status = PrSetFlat(int flat_number);
```

flat_number	The integer number direction of the flat `PrSetMode`

The integral values for the *flat_number* parameter are defined in the `KI_KULT_PATH XXXX.h` file (where `XXXX` is the abbreviation for your prober model).

PrSetMode

This command changes the probing mode from the one set with the `PrInit` command.

Usage

```
status = PrSetMode(int mode);
```

mode	The probing mode to use: 1 = PR_MODE_MANUAL 2 = PR_MODE_EXTERNAL 3 = PR_MODE_EDGE

Details

None

PrSetPipeline

This command enables and disables pipelining on the prober.

Usage

```
status = PrSetPipeline(int on_off);
```

on_off	The state in which to set pipelining on the prober: `PR_ENABLE_PIPELINE` = Enable pipelining `PR_DISABLE_PIPELINE` = Disable pipelining

Details

`PR_ENABLE_PIPELINE` and `PR_DISABLE_PIPELINE` are constants defined in the `prb.h` file.

Pipelining is a technique used to speed up the probing process. When enabled, the prober prepares the next wafer to be loaded on the chuck. For example:

The wafer is taken from the cassette and placed on the pre-aligner (the pre-aligner finds the flat/notch and orients the wafer correctly). Once pre-aligned, the prober places the wafer on the quick loader (the wafer on the quick loader is the next wafer probed after the wafer presently on the chuck is unloaded). This is how the second and subsequent wafers are "pipelined."

The first wafer goes directly to the chuck.

Also see PrClearPipeline (on page 6-5).

PrSetQuadrant

This command sets the directions in which X and Y coordinates will increase.

Usage

```
status = PrSetQuadrant(int quad_number);
```

quad_number	The value direction in which to move the X, Y coordinates; an integral value between `1` and `4`

Details

The `quad_number` parameter is expressed as an integral value between `1` and `4`, inclusive, where each value represents a unique directions-of-increase arrangement for the X, Y axes.

PrSetRefDie

This command assigns an X, Y coordinate to the target or reference die of the prober.

Usage

```
status = PrSetRefDie(int x_start_position, int y_start_position);
```

x_start_position	The X coordinate to assign to the initial reference die (integer)
y_start_position	The Y coordinate to assign to the initial reference die (integer)

PrSetSlotStatus

This command programs the cassette map of the prober to probe a specified slot, skip a slot, or mark the slot as unprobed.

Usage

```
status = PrSetSlotStatus(int Cassette, int Slot, int StatusCode);
```

Cassette	The cassette number of the map to change
Slot	The number of the slot to change
StatusCode	The status of the slot: PR_SLOT_PROBED (1) PR_SLOT_UNPROBED (2) PR_SLOT_SKIPPED (3)

Details

The cassette size is from 1 to a defined constant. The constant is defined in the file pointed to by KI_PRB_CONFIG (PROB_*n*_MAX_CASSETTE, where *n* = test station number). See the example in the KIDAT/prbcnfg_XXXX.dat file.

The array size for the *Slot* parameter is from 1 to a defined constant. The constant is defined in the file pointed to by KI_PRB_CONFIG (PROBER_*n*_MAX_SLOT, where *n* = test station number). See the example in the KIDAT/prbcnfg_XXXX.dat file.

The *StatusCode* parameter is the value that the status of the slot changes to after the PrSlotStatus command. One of the following values is passed to this parameter:

- PR_SLOT_PROBED indicates that the wafer in the slot has already been probed
- PR_SLOT_UNPROBED indicates that the wafer in the slot is designated to be probed but has not yet been probed
- PR_SLOT_SKIPPED indicates that the wafer in the slot will not be probed

This command informs the prober whether a particular slot is to be probed, skipped, or marked as unprobed. In general, only unprobed slots will be tested.

The following is a list of the statuses to which a slot may have been set during the execution of the PrCassetteMap command; these constants are defined in the prb.h file.

Constants set in the PrCassetteMap command:

- DEFAULT_SLOT_STATUS
- PR_SLOTSTATUS_UNMAPPED
- PR_SLOTSTATUS_INPROCESS
- PR_SLOTSTATUS_EMPTY
- PR_SLOTSTATUS_UNPROBED
- PR_SLOTSTATUS_PROBED

- `PR_SLOTSTATUS_PROBLEM`
- `PR_SLOTSTATUS_UNSCHEDULED`

PrSetTime

This command changes the system default timeout value for prober commands.

Usage

```
status = PrSetTime(int new_time);
```

new_time	The new timeout value, in seconds

Details

The timeout value represents the integral number of seconds the prober will be given to respond to commands before the system returns a timeout error.

This command changes the system default timeout value of 120 seconds on prober commands. The system will use this new value until the timeout is again changed by another `PrSetTime` command. Timeout errors occur when the prober takes longer than the default timeout value to respond to commands.

PrSetUnits

This command sets the prober to metric or Imperial units of measure.

Usage

```
status = PrSetUnits(int Units);
```

units	The unit of measure for numerical data: `PR_ENGLISH` = Use Imperial units of measure `PR_METRIC` = Use metric units of measure

Details

None

PrSetZHeight

This command tells the prober to set the present Z-stage chuck height as the new contact position.

Usage

```
status = PrSetZHeight();
```

Details

Constants associated with this command are defined in the `prb.h` file for your system.

PrSmifClamp

This command causes the SMIF (FOUP) pod to be engaged (clamped) or disengaged (unclamped).

Usage

```
status = PrSmifClamp(int i_pod_number, int i_clamp_state);
```

i_pod_number	The pod to change to the specified state
i_clamp_state	The state in which to set the pod: `SMIF_CLAMP_STATUS_UNLATCH` = Unclamp `SMIF_CLAMP_STATUS_LATCH` = Clamp

Details

The `SMIF_CLAMP_STATUS_UNLATCH` and `SMIF_CLAMP_STATUS_LATCH` constants are defined in the `prb.h` file.

A clamped cassette cannot be removed from the indexer. An unclamped cassette can be removed from the indexer.

This command is valid only with a prober equipped with SMIF technology.

PrSmifLock

This command causes the SMIF (FOUP) pod to be engaged (locked) or disengaged (unlocked).

Usage

```
status = PrSmifLock(int i_pod_number, int i_lock_state);
```

i_pod_number	The pod to lock or unlock
i_lock_state	The state in which to set the pod: `SMIF_LATCH_STATUS_UNLATCH` = Unlocked `SMIF_LATCH_STATUS_LATCH` = Locked

Details

The `SMIF_LATCH_STATUS_UNLATCH` and `SMIF_LATCH_STATUS_LATCH` constants are defined in the `prb.h` file.

If locked, the cassette cannot be lowered; if unlocked, the cassette can be lowered.

This command is valid only with a prober equipped with SMIF technology.

PrSmifStatus

This command returns the operator mode, latch and lock status, cassette home status, pod present status, and clamp status.

Usage

```
status = int PrSmifStatus(int i_pod_number, int *i_status_array, int
   i_status_array_size);
```

i_pod_number	The pod for which to return information
i_status_array	The name of the array in which the information is placed
i_status_array_size	The size of the array defined by `MAX_SMIF_STATUS_ITEMS`

Parameters

The `i_status_array` parameter represents an array that contains information about the status of the pod and prober. Following is a list of constants that represent data entries to this array; the names of constants meaningfully convey the prober features and their corresponding statuses:

- `SMIF_OPER_MODE_OFFSET`
- `SMIF_OPER_MODE_OFFSET` (operator mode)
- `SMIF_NORMAL_OPER_MODE`
- `SMIF_GEM_OPER_MODE`
- `SMIF_EXTERNAL_OPER_MODE`
- `SMIF_SORTLINK_OPER_MOD`
- `SMIF_LATCH_STATUS_OFFSET`
- `SMIF_LATCH_STATUS_OFFSET` (latch and lock status)
- `SMIF_LATCH_STATUS_UNKNOWN`
- `SMIF_LATCH_STATUS_UNLATCH`
- `SMIF_CASS_HOME_OFFSET`
- `SMIF_CASS_HOME_OFFSET` (indexer position)
- `SMIF_CASSETTE_HOME`
- `SMIF_POD_PRESENT_OFFSET`
- `SMIF_CLAMP_STATUS_OFFSET`
- `SMIF_POD_PRESENT_OFFSET` (pod status)
- `SMIF_CASSETTE_PRESENT`
- `SMIF_CLAMP_STATUS_OFFSET` (clamp status)

- `SMIF_CLAMP_STATUS_UNCLAMPED`
- `SMIF_CLAMP_STATUS_CLAMPED`

Details

This command is valid only with a prober equipped with SMIF technology.

The term "clamp" is defined as the availability of the pod to be removed from the indexer. If clamped, the pod may not be removed.

The terms "lock" and "latch" are used interchangeably, and are defined as the availability of the cassette or indexer to be lowered or mapped. If locked or latched, the cassette may not be mapped.

PrStart

This command starts the prober.

Usage

```
status = PrStart();
```

PrStatus

This command gets status information about the following prober features: Ready-for-probing, prober location in relationship to the wafer, chuck position, and probing mode.

Usage

```
status = PrStatus (int *ready, int *x_location, int *y_location,
    int *chuck_position, int *prober_mode);
```

ready	The status of the chuck: `1` = The chuck is up and touching the prober pins are touching the wafer `0` = The chuck and prober pins are not in the ready position
x_location	The X coordinate of the present probe site
y_location	The Y coordinate of the present probe site
chuck_position	The position of the chuck: `1` = The chuck is up `0` = The chuck is down
prober_mode	The prober mode of operation: `PR_MODE_MANUAL` = Manual mode `PR_MODE_EXTERNAL` = External mode `PR_MODE_EDGE` = Edge mode

Details

The *x_location* and *y_location* parameters specify the xy-coordinates of the present probe site. These coordinates are given with respect to the coordinate system that was established by setting the target die coordinates using the `PrInit` command.

The `PR_MODE_MANUAL`, `PR_MODE_EXTERNAL`, and `PR_MODE_EDGE` constants are defined in the `prb.h` file.

NOTE

Some probers do not support all prober modes. Refer to the documentation for your prober for modes supported by your prober.

PrStop

This command stops the prober.

Usage

```
status = PrStop();
```

PrUnLoad

This command forces the wafer presently on the chuck to unload to its original cassette and slot.

Usage

```
status = PrUnLoad();
```

Details

This command is not used by the KTXE execution engine.

PrWriteRead

NOTE

This command is for use with serial probers only; see the PrWriteReadSRQ (on page 6-22) command for GPIB probers.

This command allows you to create a string command and define the response of the serial prober to the newly created command.

Usage

```
status = PrWriteRead (char *input_buf, int input_buf_len, char *output_buf, int
    output_buf, int terminator, int terminator_cnt);
```

input_buf	A pointer to an array whose content is a string representing the user-created command; the string's maximum length is 80 characters
input_buf_len	An integer that represents the actual number of characters in the name of the newly-defined command
output_buf	A pointer to an array that contains a string representing the response of the prober to the newly created command found in the `input_buf` parameter
output_buf_len	An integer that defines the maximum length of the output buffer; use to avoid overflowing the user output buffer
terminator	The ASCII values of the characters used to terminate the `input_buf` file
terminator_cnt	The number of terminators that will be used to terminate the `input_buf` name

Details

Prober responses must be chosen from a list of available prober functions, provided in the documentation for your prober.

The value of the *terminator_cnt* parameter is defined by the constant NUM_TERMINATORS in the header file for your model of prober (for example, KI_KULT_PATH PrXXXX.h)

When using this command, you must know which low-level commands are available for your model of prober and whether or not a reply is expected from the prober. Refer to the documentation for your prober for information on compatible low-level prober commands.

PrWriteReadSRQ

NOTE

This command is for use with GPIB probers only; see the PrWriteRead (on page 6-21) command for serial probers.

This command allows you to create a string command and define the response of the GPIB prober to the newly created command.

Usage

```
status = PrWriteReadSRQ (char *input_buf, int input_buf_len, char *output_buf, int
    output_buf_len, int timeout, int *i_srq);
```

input_buf	A pointer to an array whose content is a string representing the user-created command; the string's maximum length is 80 characters
input_buf_len	An integer that represents the actual number of characters in the name of the newly-defined command
output_buf	A pointer to an array that contains a string representing the response of the prober to the newly created command found in the input_buf parameter
output_buf_len	An integer that defines the maximum length of the output buffer; use to avoid overflowing the user output buffer
timeout	The number of seconds the PrWriteReadSRQ function should wait for a prober response before timing out
i_srq	A pointer to the SRQ bit; the value is the SRQ byte received from the prober (integer)

Details

Zero and positive integer values returned for the *i_srq* parameter indicate OK. Values less than (but not equal to) zero indicate errors. Use the return value from the function to indicate the status of the command (success or failure). For example:

```
x=PrWriteReadSRQ(...);
                 if (x==PR_OK);
                        /* all is well */
                        •
                        •
                        •
                        /* look at the i_srq integer */
                        •
                        •
                        •
                 else;
                        /* generate error message to the user */
                        •
                        •
                        •
```

The prober responses must be chosen from a list of available prober functions, provided in the documentation for your prober.

This command automatically polls for an SRQ.

When using this command, determine which low-level commands are available on the prober and whether or not a reply is expected from the prober. Refer to the documentation for your prober for information on compatible low-level prober commands.

Lower-level functions use the string length values to determine if a read or a write will be performed. If a string length is less than or equal to zero, the associated action is not performed. Also, the lower-level functions use the *i_srq* parameter value to determine whether or not to serial poll. If the *i_srq* parameter value is a decimal value greater than or equal to `64`, a serial poll is performed. If the *i_srq* parameter value is a positive value greater than `0` but less than `64`, a serial poll is not performed. Because of this:

- Set the *output_buf_len* parameter to `0` if you are sending a command to the prober without returning a response in the *output_buf* parameter (if you want to write but not read).
- Set the *input_buf_len* parameter to `0` if you are returning a response from the prober without sending the command line contained in the *input_buf* parameter to the prober (if you want to read but not write).

Initialize the integer pointed to by the *i_srq* parameter to a decimal value greater than or equal to `64` if you are returning an SRQ from the prober.

Initialize the integer pointed to by the *i_srq* parameter to a decimal value equal to `0` if you are returning an SRQ from the prober. Combinations of the string length values may also be used because each action is independent. For example:

Set the *output_buf_len* and *input_buf_len* parameters to a value greater than `0`, and if the *i_srq* parameter is greater than or equal to `64`, the prober will write to the bus, poll until an SRQ of greater than or equal to `64` (or timeout) is received, and then read from the bus.

PrZParams

This command sets one of the following height (Z-axis) parameters for the prober: Overtravel, clearance, up limit, down limit, or align height.

Usage

```
status = PrZParams(int function, int value);
```

function	The Z-axis parameter to set: `PR_Z_TRAVEL` = Overtravel `PR_Z_CLEARANCE` = Clearance `PR_Z_LIMIT` = Z up limit `PR_Z_DOWN_LIMIT` = Z down limit `PR_Z_ALIGN_HEIGHT` = Z align height
value	An integer that is the value of the Z-mode selected by the first parameter function

Details

The constants for the `function` parameter are defined in the `prb.h` file.

PrZTravel

This command sets the Z travel mode from the tester.

Usage

```
status = PrZTravel(int number);
```

number	The type of travel used for Z motion, as limited by edge sensor, profiler, or limit-to-limit setting: `PR_Z_LIMIT_MODE` = Enables limits mode (limit-to-limit: Z up and Z down) `PR_Z_EDGE_MODE` = Enables the edge sensor `PR_Z_PROFILE_MODE` = Enables the Z stage to be guided by the profiler

Details

Refer to `KI_KULT_PATH/PrXXXX.h` for definitions of the number parameter constants.

Section 7

Keithley data files (KDF) library command reference

In this section:

Overview 7-1
Data logging routines 7-1
Update comment routines 7-15
Update limits routines... 7-16
Structure handling routines 7-17

Overview

The Keithley Data Files (KDF) library is a set of routines to organize and save parametric test data into simple ASCII data files.

This section contains detailed information about the commands in the Keithley data files (KDF) library. The command descriptions are organized into the following categories:

- Data logging routines (on page 7-1)
- Update comment routines (on page 7-15)
- Update limits routines (on page 7-16)
- Structure handling routines (on page 7-17)

For additional information about using KDF, see "Software" in the reference manual for your system.

Data logging routines

Descriptions of the data logging routines are in the following topics.

PutLot

This routine will log the header information to the DB or FF. Lotadd is used to either append (Lotadd = APPENDLOT), create new (Lotadd = CREATELOT), or to replace an existing lot (Lotadd = CREATELOT). Opens a file, writes to it, and then closes it.

Usage

Status = PutLot(*LotStruct, Lotadd)

LOT *LotStruct	The lot where data is logged
int Lotadd	Can be set to either CREATELOT or APPENDLOT; determines what will happen if a lot file already exists

Details

You cannot use the * or the ? characters in the Lot structure fields because they are the wildcard characters. Use of these characters will result in an error.

The Lot ID will be used as the lot filename, with the .kdf extension added.

PutLot logs all the data up to and including the <EOH> marker.

If a lot file already exists, it will be renamed from a .kdf extension to a .kd% extension.

Example

```
strcpy (testlot->id, "test1");
strcpy (testlot->testname, "Voltage 1");
GetStartTime(testlot->starttime);

/* Start the logging of the new lot "test1". The lot has 3 wafers and 10
sites to be logged.
*/
status = PutLot(testlot, CREATELOT);
if (status < 0)
return(status);
```

Full code can be found in sample program 1.

PutWafer

This routine will log the information in the Wafer Structure to the DB or FF for a specific instance of a lot. Opens a file, writes to it, and then closes it. Must be followed at some point by EndWafer.

Usage

```
Status = PutWafer(*LotStruct, *WafStruct)
```

LOT *LotStruct	The lot where data is logged
WAFER *WafStruct	The wafer to log to the lot

Details

After a call to PutWafer, you must make a call to EndWafer before calling PutWafer again.

You cannot use the * or the ? characters in the wafer structure fields because they are the wildcard characters. Use of these characters will result in an error.

Example

```
for (waferloop = 1;waferloop <= 3;waferloop++)
{
sprintf(testwafer->id,"%i",waferloop);
status = PutWafer(testlot,testwafer);
if (status < 0)
return(status);
```

Full code can be found in sample program 1.

PutSite

This routine will log the information contained in the Site Structure to the DB or FF for a specific instance of a lot and wafer. Opens a file, writes to it, and then leaves it open for parameter data. Must be followed at some point by an EndSite.

Usage

```
status = PutSite(*LotStruct, *WafStruct, *SiteStruct)
```

LOT *LotStruct	The lot where data is logged
WAFER *WafStruct	The wafer to log to the lot
SITE *SiteStruct	The site to log to the lot

Details

After a call to PutSite, you must make a call to EndSite before calling PutSite again.

You cannot use the * or the ? characters in the site structure fields because they are the wildcard characters. Use of these characters will result in an error.

Example

```
for (siteloop = 1; siteloop <= 10; siteloop++)
{
sprintf(testsite->id,"%i",siteloop);
status = PutSite(testlot,testwafer,testsite);
if (status < 0)
return(status);
```

Full code can be found in sample program 1.

PutParam

This routine will log the information in the Param Structure to the DB or FF for a specific instance of a lot, wafer, and site. Writes to the already open file.

Usage

```
Status = PutParam(*LotStruct, *WafStruct, *SiteStruct,*ParamStruct)
```

LOT *LotStruct	The lot where data is logged
WAFER *WafStruct	The wafer to log to the lot
SITE *SiteStruct	The site to log to the lot
PARAM *ParamStruct	The parameter to log to the lot

Details

You cannot use the `*` or the `?` characters in the `ParamStructure` fields because they are the wildcard characters. Use of these characters will result in an error.

`EndSite` should be called after the last parameter for the current site is logged.

Example

```
strcpy(testparam->id, "beta1");
/* calculate the beta*/
testparam->value = beta1(1, 4, 7, -1, 0.5e-3, 2.0, ’N’);
status = PutParam(testlot,testwafer,testsite,testparam);
if (status < 0)
return(status);
```

Full code can be found in sample program 1.

PutParamList

This routine will log a list of `param` Structures to the `DB` or `FF` for a specific instance of a lot, wafer, and site. The next pointer set to NULL will signify the end of the list to be logged.

Usage

```
Status = PutParamList(*LotStruct, *WafStruct, *SiteStruct, *Param)
```

`LOT *LotStruct`	The lot where data is logged
`WAFER *WafStruct`	The wafer to log to the lot
`SITE *SiteStruct`	The site to log to the lot
`PARAM *ParamStruct`	The linked list of parameters to log to the lot

Details

`PutParamList` makes a series of calls to `PutParam` as it traverses the linked list, so it has the same rules as `PutParam`.

After calling `PutParamList`, remember to free up the memory from the list (use a while loop with `RemoveParam`).

Example

```
testparam=CreateNewParam();
strcpy(testparam->id, "Volts 1e-2");
/* voltagetest is an example test routine that would return a voltage */
testparam->value = voltagetest(1e-2);

new=CreateNewParam();

strcpy(new->id, "Volts 1e-1");
/* voltagetest is an example test routine that would return a voltage */
testparam->value = voltagetest(1e-1);
/* Add current into the list following testparam */
AddNewParam(testparam,new);
testparam = FindNextParam(testparam);
new = CreateNewParam();

strcpy(new->id, "Volts 1");
/* voltagetest is an example test routine that would return a voltage */
testparam->value = voltagetest(1);
/* Add current into the list following testparam */
AddNewParam(testparam,new);

/* Go to the first param in the list to get ready for PutParamList */
testparam=FindFirstParam(testparam);
PutParamList(testlot,testwafer,testsite,testparam);
```

Full code can be found in sample program 2.

EndLot

This routine ends the logging of the current lot. It must be called before another lot can be logged.

Usage

```
status = EndLot()
```

Details

`EndLot` must be called before `PutLot` can be called again, otherwise an error is generated.

The end of the lot is signified by the end of the lot file.

EndWafer

This routine ends the logging of the current wafer. It must be called after a call to `PutWafer`.

Usage

```
status = EndWafer()
```

Details

`EndWafer` must be called before `PutWafer` can be called again, otherwise an error is generated.

`EndWafer` writes the `<EOW>` marker to the file.

EndSite

This routine ends the logging of the current site. It must be called after a call to `PutSite`.

Usage

```
status = EndSite()
```

Details

`EndSite` must be called before `PutSite` can be called again, otherwise an error is generated.

`EndSite` writes the `<EOS>` marker to the file.

GetLot

This routine returns a NULL terminated list of lots, starting with `LotStructGot`, that match the criteria specified in `LotStructWanted`. The `LotStructGot` pointer should already point to a structure when the routine is called (for example, `LotStructGot = CreateNewLot`). Wildcards are supported in the wanted structure. Wildcards cannot be entered in the integer fields (a value of zero in an integer position is the same as `*`).

Usage

```
Status = GetLot(*LotStructWanted,*LotStructGot)
```

`LOT *LotStructWanted`	Lot structure containing the information on the lot to be retrieved. It can be very general (using wildcards) or very specific
`LOT *LotStructGot`	Lot structure to which the found data is returned. It must be an allocated structure when it is sent to `GetLot`

Details

NOTE

The Lot ID is the only required field.

Using wildcards may cause a noticeable decrease in performance in a directory with many files.

If the status returned is greater than or equal to zero, then it is the number of lots found. If it is less than zero, it is an error code.

Example

```
gotlot = CreateNewLot();

strcpy(testlot->id, "test1");
/* Retrieve all the data that was just logged */
GetLot(testlot, gotlot);
```

Full code can be found in sample program 1.

GetWafer

This routine returns a NULL terminated list of wafers, starting with `WaferStructGot`, that match the criteria specified in `WafStructWanted` for the specific (single) `LotStruct`. Wildcards are supported in the wanted structure. The `WafStructGot` pointer should already point to a structure when the routine is called (for example, `WafStructGot = CreateNewWafer`). Wildcards are not supported for the integer fields (a value of zero in an integer position is the same as `*`). If an empty (NULL) wanted structure is passed in, the routine will return all wafers in the specified lot.

Usage

```
Status = GetWafer(*LotStruct, *WafStructWanted,*WaferStructGot)
```

`LOT *LotStruct`	The specific lot in which the wafer should be found
`WAFER *WaferStructWanted`	Wafer structure containing the information on the wafer to be retrieved. It can be very general (using wildcards) or very specific
`WAFER *WaferStructGot`	Wafer structure to which the found data is returned. It must be an allocated structure when it is sent to `GetWafer`

Details

If the status returned is greater than or equal to zero, then it is the number of wafers found. If it is less than zero, it is an error code. The wanted structure must contain a valid string or wildcard for the `"id"` and `"split"` string items. A null string in either of these string items will not return a match.

Example

```
gotwafer = CreateNewWafer();

/* Get all the wafers in the lot */
strcpy(testwafer->id, "*");
GetWafer(gotlot, testwafer, gotwafer);
```

Full code can be found in sample program 1.

GetSite

This routine returns a NULL terminated list of sites, starting with `*SiteStructGot`, that match the criteria specified in `SiteStructWanted` for the specific (single) `LotStruct` and `WafStruct`. The `SiteStructGot` pointer should already point to a structure when the routine is called (for example, `SiteStructGot = CreateNewSite`). Wildcards are supported in the wanted structure. Wildcards are not supported for the integer fields (a value of zero in an integer position is the same as `*`).

Usage

```
Status = GetSite(*LotStruct, *WafStruct, *SiteStructWanted, *SiteStructGot)
```

`LOT *LotStruct`	The specific lot in which the wafer should be found
`WAFER *WaferStruct`	The specific wafer in which the site should be found
`SITE *SiteStructWanted`	Site structure containing the information on the site to be retrieved. It can be very general (using wildcards) or very specific
`SITE *SiteStructGot`	Site structure to which the found data is returned. It must be an allocated structure when it is sent to `GetSite`

Details

If the status returned is greater than or equal to zero, then it is the number of sites found. If it is less than zero, it is an error code.

Example

```
gotsite = CreateNewSite();

/* Get all the sites in this wafer */
strcpy(testsite->id, "*");
GetSite(gotlot, gotwafer, testsite, gotsite);
```

Full code can be found in sample program 1.

GetParam

This routine returns a NULL terminated list of parameters, starting with the `*ParamStructGot`, that match the criteria specified in `ParamStructWanted` for the specific (single) `LotStruct`, `WafStruct`, and `SiteStruct`. The `ParamStructGot` pointer should already point to a structure when the routine is called (for example, `ParamStructGot = CreateNewParam`). Wildcards are supported in the wanted structure. Wildcards are not supported for the integer fields (a value of zero in an integer position is the same as `*`).

Usage

```
Status = GetParam(*LotStruct, *WafStruct, *SiteStruct, *ParamStructWanted,
    *ParamStructGot)
```

`LOT *LotStruct`	The specific lot in which the wafer should be found
`WAFER *WaferStruct`	The specific wafer in which the site should be found
`SITE *SiteStruct`	The specific site in which the param should be found
`PARAM *ParamStructWanted`	Parameter structure containing information on the parameter to be retrieved; it can be very general (using wildcards) or very specific
`PARAM *ParamStructGot`	Parameter structure to which the found data is returned. It must be an allocated structure when it is sent to `GetParam`

Details

If the status returned is greater than or equal to zero, then it is the number of parameters found. If it is less than zero, it is an error code.

Example

```
gotparam = CreateNewParam();

/* Get all the parameters in this site */
strcpy(testparam->id, "*");
GetParam(gotlot, gotwafer, gotsite, testparam, gotparam);
```

Full code can be found in sample program 1.

GetParamList

This routine returns a NULL terminated list of parameters to *ParamStruct that are included in the *ParamStructList list. Wildcards are supported at the parameter level.

Usage

```
Status = GetParamList(*LotStruct, *WaferStruct, *SiteStruct, *ParamStructList,
    *ParamStruct)
```

LOT *LotStruct	The specific lot in which the wafer should be found
WAFER *WaferStruct	The specific wafer in which the site should be found
SITE *SiteStruct	The specific site in which the param should be found
PARAM *ParamStructList	List of parameter structures to be retrieved. Each parameter in the list can be very general (using wildcards) or very specific
PARAM *ParamStruct	Head of the list of parameter structures to which the found data is returned. It must be an allocated structure when it is sent to GetParamList

Details

If the status returned is greater than or equal to zero, then it is the number of parameters found. If it is less than zero, it is an error code.

GetLotData

This routine returns a tree structure of all the wafers, sites, and parameters in LotWanted. The returned list begins with the next field of the lot structure sent to the function.

Usage

```
Status = GetLotData (*LotWanted)
```

LOT *LotWanted	Put the specific lot data that you want to find in this structure and the found lot will be returned in the LotWanted->next field

Details

The data is returned in LotWanted->next. Then the data follows a tree structure from there.

LotWanted->wafers points to the first wafer in the lot.

LotWanted->wafers->sites points to the first site in the first wafer.

LotWanted->wafers->sites->params points to the first parameter in the first site of the first wafer.

Wildcards are not supported.

Example

```
gotlot = CreateNewLot();

strcpy(gotlot->id, "test1");
/* Retrieve all the data that was just logged */
GetLotData(gotlot);
```

Full code can be found in sample program 1.

MatchParam2Limit

This routine takes the list of parameters and matches them to the corresponding limit codes. Each parameter points to its corresponding limit code and each limit points back to the parameter. If no match is found for a parameter, the pointer is set to NULL.

Usage

```
Status = MatchParam2Limit(*ParamList, *LimitList)
```

PARAM *ParamList	List of parameters
LIMIT *LimitList	List of limits to match to the above parameters

FileExist

Checks for the existence of a file in the current data directory (where the lot files are being stored). Returns TRUE (1) if the file is in the directory, FALSE (0) if the file is not in the directory.

Usage

```
Status = FileExist(filename)
```

char filename[]	Name of the file to find

Details

The present data directory is determined by the value in the `kth.ini` file after "`Datapath=`".

LotExist

Checks for the existence of a lot file in the current data directory. Returns TRUE (1) if the lot is in the directory, FALSE (0) if the lot is not in the directory.

Usage

```
Status = LotExist(*LotStruct)
```

LOT *LotStruct	The lot structure containing the information to be found

Details

The present data directory is determined by the value in the `kth.ini` file after "`Datapath=`".

GetStartTime

Returns a time and date string in the format "DD-MM-YYYY hh:mm" where DD=day, MM=month, YYYY=year, hh=hour, and mm=minutes.

Usage

```
GetStartTime(timestring)
```

char timestring[]	String to which the current time is returned; must be at least 20 characters

DeleteLot

This routine will delete all lot-associated data for the specified lot structure. All Lots in the NULL terminated linked list will be deleted. The DeleteLot routine is responsible for ensuring referential integrity.

Usage

```
Status = DeleteLot(*LotStruct)
```

LOT *LotStruct	The lot or linked list of lots to be deleted

Details

Wildcard deletes are not allowed.

When a lot is deleted, the lot is renamed from the .kdf extension to a .kd% extension.

DeleteWafer

This routine will delete all wafer information for the specified lot and wafer. All wafers for the NULL terminated linked list wafers will be deleted. Wildcards are not allowed in either structure.

Usage

```
Status = DeleteWafer(*LotStruct, *WafStruct)
```

LOT *LotStruct	The lot that contains the wafer to be deleted
WAFER *WafStruct	The wafer or linked list of wafers to be deleted

Details

Wildcard deletes are not allowed.

When a wafer is deleted, the lot is renamed from the .kdf extension to a .kd% extension.

DeleteSite

This routine will delete all site information for the specified lot, wafer, and site. All sites for the NULL terminated linked list of sites will be deleted. Wildcards are not allowed in any of the structures.

Usage

```
Status = DeleteSite(*LotStruct, *WafStruct, *SiteStruct)
```

`LOT *LotStruct`	The lot that contains the wafer to be deleted
`WAFER *WafStruct`	The wafer that contains the site to be deleted
`SITE *SiteStruct`	The site or linked list of sites to be deleted

Details

Wildcard deletes are not allowed.

When a site is deleted, the lot is renamed from the `.kdf` extension to a `.kd%` extension.

DeleteParam

This routine will delete the parameter information for the specified lot, wafer, site, and parameter. All parameters for the NULL terminated linked list of parameters will be deleted. Wildcards are not allowed in any of the structures.

Usage

```
Status = DeleteParam(*LotStruct, *WafStruct, *SiteStruct, *ParamStruct)
```

`LOT *LotStruct`	The lot that contains the wafer to be deleted
`WAFER *WafStruct`	The wafer that contains the site to be deleted
`SITE *SiteStruct`	The site that contains the parameter to be deleted
`PARAM *ParamStruct`	The parameter or linked list of parameters to be deleted

Details

Wildcard deletes are not allowed.

When a parameter is deleted, the lot is renamed from the `.kdf` extension to a `.kd%` extension.

DeleteLimitCode

This routine is used to delete entire sets of limits defined by a limit code. All limits specified in the NULL terminated list of limit codes will be deleted.

Usage

```
Status = DeleteLimitCode(*LimitcodeStruct)
```

LIMITCODE *LimitcodeStruct	The limit code or linked list of limit codes to be deleted

Details

Wildcard deletes are not allowed.

Limit code information is used to generate the limits filename.

When a limit code is deleted, the lot is renamed from the `.klf` extension to a `.kl%` extension.

DeleteLimit

This routine is used to delete limit records from the DB. All limits specified in the NULL terminated list of limits will be deleted.

Usage

```
Status = DeleteLimit(*LimitcodeStruct, *LimitStruct)
```

LIMITCODE *LimitcodeStruct	The limit code containing the limit to be deleted
LIMIT *LimitStruct	The limit or linked list of limits to be deleted

Update comment routines

The following topics describe the update comment routines.

GetComment

This routine will fetch the comment from the `.kdf` file for a specific lot occurrence.

Usage

```
Status = GetComment(*LotStruct, comment)
```

LOT *LotStruct	The lot to retrieve the comment from
char comment[]	The string where the comment is returned

PutComment

This routine will overwrite the comment in the `.kdf` file for a specific lot occurrence.

Usage

```
Status = PutComment(*LotStruct, comment[])
```

`LOT *LotStruct`	The lot where the comment is to be changed
`char comment[]`	The string sent to be logged as the new comment

Update limits routines

The following topics describe the update limits routines.

GetLimitCode

This routine will fetch a list of limit codes that match the criteria specified in the wanted structure.

Usage

```
Status = GetLimitCode(*LimitcodeStructwanted, *LimitcodeStructlist)
```

`LIMITCODE *LimitcodeStructwanted`	The limit code information to search for in the logging directory
`LIMITCODE *LimitcodeStructlist`	The returned list of limit codes that were found in the search

Details

Limit code information is used to generate the limits filename.

Wildcards can be used in the `*LimitcodeStructwanted` structure.

The return value is the number of limit codes found or an error value if it is less than zero.

GetLimit

This routine will fetch a NULL terminated linked list of limit structures with the specified limit code and limit information. The head of the linked list of limits is returned in both the `*LimitStruct` and in the `LimitcodeStruct->limits` fields.

Usage

```
Status = GetLimit(*LimitcodeStruct, *LimitStruct)
```

`LIMITCODE *LimitcodeStruct`	The limit code to retrieve the limits from
`LIMIT *LimitStruct`	The returned list of limits that were found in the limit code

Details

Limit code information is used to generate the limits filename.

The return value is the number of limits found or an error value if it is less than zero.

PutLimit

This routine will write a list of limits to the DB or FF for the limit code specified. If the limit code already exists, the new limits will overwrite and append.

Usage

```
Status = PutLimit(*LimitcodeStruct, *LimitStruct)
```

LIMITCODE *LimitcodeStruct	The limit code to log the limits to
LIMIT *LimitStruct	The list of limits to log to the limit code

Details

If the limit code already exists, it will be renamed from a .klf extension to a .kl% extension.

Structure handling routines

There is a version of each of the structure handling routines for every structure (LOT, WAFER, SITE, PARAM, and LIMITCODE). These routines are described in the following topics.

AddNew[STRUCTURE]

This routine adds new to the list following current. There is a version of each of the structure handling routines for every structure (LOT, WAFER, SITE, PARAM, and LIMITCODE).

Usage

```
AddNewLimitCode(*current, *new)
AddNewLot(*current, *new)
AddNewWafer(*current, *new)
AddNewSite(*current, *new)
AddNewParam(*current, *new)
```

LIMITCODE *current	Pointer to the current limit code
LOT *current	Pointer to the current lot
WAFER *current	Pointer to the current wafer
SITE *current	Pointer to the current site
PARAM *current	Pointer to the current parameter
LIMITCODE *new	The new limit code to be added
LOT *new	The new lot to be added
WAFER *new	The new wafer to be added
SITE *new	The new site to be added
PARAM *new	The new param to be added

Example

```
LIMITCODE *current, *new;
AddNewLimitCode(current, new);
```

CreateNew[STRUCTURE]

This routine allocates the memory for and returns a pointer to the new LIMIT CODE, LOT, WAFER, SITE, or PARAM.

Usage

```
LimitCodePtr = CreateNewLmtCode()
LotPtr = CreateNewLot()
WaferPtr = CreateNewWafer()
SitePtr = CreateNewSite()
ParamPtr = CreateNewParam()
```

`LIMITCODE *LimitCodePtr`	Pointer to a limit code structure
`LOT *LotPtr`	Pointer to a lot structure
`WAFER *WaferPtr`	Pointer to a wafer structure
`SITE *SitePtr`	Pointer to a site structure
`PARAM *ParamPtr`	Pointer to a parameter structure

Details

This routine must be called before performing operations with `LimitCodePtr`, `LotPtr`, `WaferPtr`, `SitePtr`, or `ParamPtr`.

Example

```
LIMITCODE *LimitCodePtr;
LimitCodePtr = CreateNewLmtCode();
```

FindFirst[STRUCTURE]

This routine returns the first LIMIT CODE, LOT, WAFER, SITE, or PARAM that `current` points to in the list. NULL is returned if `current` is NULL.

Usage

```
LimitCodePtr = FindFirstLmtCode(*current)
LotPtr = FindFirstLot(*current)
WaferPtr = FindFirstWafer(*current)
SitePtr = FindFirstSite(*current)
ParamPtr = FindFirstParam(*current)
```

`LIMITCODE *LimitCodePtr`	Pointer to a limit code structure
`LOT *LotPtr`	Pointer to a lot structure
`WAFER *WaferPtr`	Pointer to a wafer structure
`SITE *SitePtr`	Pointer to a site structure
`PARAM *ParamPtr`	Pointer to a parameter structure
`LIMITCODE *current`	Pointer to the current limit code
`LOT *current`	Pointer to the current lot
`WAFER *current`	Pointer to the current wafer
`SITE *current`	Pointer to the current site
`PARAM *current`	Pointer to the current parameter

Example

```
LIMIT *LimitCodePtr, *current;
LimitCodePtr = FindFirstLmtCode(current);
```

FindLast[STRUCTURE]

This routine returns the last LIMIT CODE, LOT, WAFER, SITE, or PARAM that `current` points to in the list. It returns NULL if `current` is NULL.

Usage

```
LimitCodePtr = FindLastLimitCode(*current)
LotPtr = FindLastLot(*current)
WaferPtr = FindLastWafer(*current)
SitePtr = FindLastSite(*current)
ParamPtr = FindLastParam(*current)
```

`LIMITCODE *LimitCodePtr`	Pointer to a limit code structure
`LOT *LotPtr`	Pointer to a lot structure
`WAFER *WaferPtr`	Pointer to a wafer structure
`SITE *SitePtr`	Pointer to a site structure

`PARAM *ParamPtr`	Pointer to a parameter structure
`LIMITCODE *current`	Pointer to the current limit code
`LOT *current`	Pointer to the current lot
`WAFER *current`	Pointer to the current wafer
`SITE *current`	Pointer to the current site
`PARAM *current`	Pointer to the current parameter

Example

```
LIMITCODE *LimitCodePtr, *current;
LimitCodePtr = FindLastLimitCode(current);
```

FindNext[STRUCTURE]

This routine finds the next LIMIT CODE, LOT, WAFER, SITE, or PARAM after the position that `current` points to in the list. NULL is returned if `current` is at the end of the list.

Usage

```
LimitCodePtr = FindNextLimitCode(*current)
LotPtr = FindNextLot(*current)
WaferPtr = FindNextWafer(*current)
SitePtr = FindNextSite(*current)
ParamPtr = FindNextParam(*current)
```

`LIMITCODE *LimitCodePtr`	Pointer to a limit code structure
`LOT *LotPtr`	Pointer to a lot structure
`WAFER *WaferPtr`	Pointer to a wafer structure
`SITE *SitePtr`	Pointer to a site structure
`PARAM *ParamPtr`	Pointer to a param structure
`LIMITCODE *current`	Pointer to the current limit code
`LOT *current`	Pointer to the current lot
`WAFER *current`	Pointer to the current wafer
`SITE *current`	Pointer to the current site
`PARAM *current`	Pointer to the current parameter

Example

```
LIMITCODE *LimitCodePtr, *current;
LimitCodePtr = FindNextLimitCode(current);
```

FindPrev[STRUCTURE]

This routine finds the previous LIMIT CODE, LOT, WAFER, SITE, or PARAM before the position `current` points to in the list. NULL is returned if `current` is at the beginning of the list.

Usage

```
LimitCodePtr = FindPrevLimitCode(*current)
LotPtr = FindPrevLot(*current)
WaferPtr = FindPrevWafer(*current)
SitePtr = FindPrevSite(*current)
ParamPtr = FindPrevParam(*current)
```

`LIMITCODE *LimitCodePtr`	Pointer to a limit code structure
`LOT *LotPtr`	Pointer to a lot structure
`WAFER *WaferPtr`	Pointer to a wafer structure
`SITE *SitePtr`	Pointer to a site structure
`PARAM *ParamPtr`	Pointer to a param structure
`LIMITCODE *current`	Pointer to the current limit code
`LOT *current`	Pointer to the current lot
`WAFER *current`	Pointer to the current wafer
`SITE *current`	Pointer to the current site
`PARAM *current`	Pointer to the current parameter

Example

```
LIMITCODE *LimitCodePtr, *current;
LimitCodePtr = FindPrevLimitCode(current);
```

InsertNew[STRUCTURE]

This routine adds `new` into the LIMIT CODE LOT, WAFER, SITE, or PARAM list before `current`.

Usage

```
InsertNewLmtCode(*current, *new)
InsertNewLot(*current, *new)
InsertNewWafer(*current, *new)
InsertNewSite(*current, *new)
InsertNewParam(*current, *new)
```

`LIMITCODE *current`	Pointer to the current limit code
`LOT *current`	Pointer to the current lot
`WAFER *current`	Pointer to the current wafer
`SITE *current`	Pointer to the current site
`PARAM *current`	Pointer to the current param
`LIMITCODE *new`	The new limit code to be inserted
`LOT *new`	The new lot to be inserted
`WAFER *new`	The new wafer to be inserted
`SITE *new`	The new site to be inserted
`PARAM *new`	The new parameter to be inserted

Example

```
LIMITCODE *current, *new;
InsertNewLmtCode(current,new);
```

Remove[STRUCTURE]

This routine removes the LIMIT CODE, LOT, WAFER, SITE, or PARAM pointed to by `current` and returns a pointer to the next limit code, lot, wafer, site, or parameter in the list. If the next pointer is NULL, the previous limit code, lot, wafer, site, or parameter is returned. If the previous limit code, lot, wafer, site, or parameter is also NULL, then NULL is returned.

Usage

```
LimitCodePtr = RemoveLimitCode(*current)
LotPtr = RemoveLot(*current)
WaferPtr = RemoveWafer(*current)
SitePtr = RemoveSite(*current)
ParamPtr = RemoveParam(*current)
```

`LIMITCODE *LimitCodePtr`	Pointer to a limit code structure
`LOT *LotPtr`	Pointer to a lot structure
`WAFER *WaferPtr`	Pointer to a wafer structure
`SITE *SitePtr`	Pointer to a site structure
`PARAM *ParamPtr`	Pointer to a parameter structure
`LIMITCODE *current`	Pointer to the current limit code
`LOT *current`	Pointer to the current lot
`WAFER *current`	Pointer to the current wafer
`SITE *current`	Pointer to the current site
`PARAM *current`	Pointer to the current parameter

Example

```
LIMITCODE *LimitCodePtr, *current;
LimitCodePtr = RemoveLimitCode(current);
```

LimitExist

This routine tests for the existence of a limit file in the current `Limit File` data directory based on the limit code "`limitin`." TRUE (`1`) is returned if the limit files exist, and FALSE (`0`) is returned otherwise. This routine is only for the `LIMITCODE` structure.

Usage

```
Status = LimitExist(*limitin)
```

`int Status`	The result value of the call
`LIMITCODE *limitin`	The limit code to search for

Example

```
int Status;
LIMITCODE *limitin;
Status = LimitExist (limitin);
```

KTXE_RP zone-based testing command reference

In this section:

Introduction 8-1

Introduction

Zone-based testing is a test method where the sites to be tested are generated at run time. The sites to be tested are selected randomly from predefined zones, or patterns, contained in the wafer definition file.

The KTXE_RP user library contains commands you can use at user access points (UAPs) in your tests. Descriptions of the commands follow.

KTXE_RP_CleanUpWDF

This command renames the randomly generated wafer description file (`.wdf`) to `lot_XXX_wdfname`.

Usage

```
KTXE_RP_CleanUpWDF
```

Details

This command preserves the `.wdf` file used in the test for a future adaptive retest. This routine is used if a lot execution is suspended.

This function should be called at `UAP_LOT_END`.

KTXE_RP_CreateRandomWDF

This function creates a wafer file based on the maximum frequency and number of wafers.

Usage

```
KTXE_RP_CreateRandomWDF
```

Details

The name of the wafer description file is saved in the `cpf_info` structure. This function uses the `rand_pat` utility to generate random patterns. For information on the `rand_pat` utility, refer to "KTE support utilities" in the reference manual for your system.

This function should be called at `UAP_WRITE_LOT_INFO`.

KTXE_RP_CreateWPF

This function creates a new wafer plan file to use the appropriate probe pattern name.

Usage

```
KTXE_RP_CreateWPF
```

Details

The new wafer plan file is created by duplicating the base file and modifying the probe pattern name.

This function should be called at `UAP_WAFER_PREPARE`.

KTXE_RP_GetUsrArgs

This function checks the command-line arguments used to invoke the Keithley Test Execution Engine (KTXE) to determine the test mode.

Usage

```
KTXE_RP_GetUsrArgs
```

Details

This overrides the global data entry, `KTXE_RP_test_mode`, if used.

This function should be called at `UAP_PROG_ARGS`.

KTXE_RP_RemoveWPF

This function deletes wafer patterns that were generated at run time.

Usage

```
KTXE_RP_RemoveWPF
```

Details

This function should be called at `UAP_WAFER_END`.

KTXE_AT result-based testing command reference

In this section:

Introduction 9-1

Introduction

The Keithley Result-Based Testing User Library (KTXE_AT) contains commands that you can use for result-based testing. For more information about result-based testing, see the "Result-Based Testing" topic in the reference manual for your system.

KTXE_AT commands can be used at user access points (UAPs) in your tests. Descriptions of these commands follow.

KTXE_AT_alternate_site_site_end()

This function is used to determine, at completion of site testing, if an alternate site should be tested.

Usage

```
KTXE_AT_alternate_site_site_end()
```

Details

Use at `UAP_SITE_END`.

KTXE_AT_alternate_site_test_end()

This function is used to determine if any test results failed during the testing of a Keithley test module (KTM).

Usage

```
KTXE_AT_alternate_site_test_end()
```

Details

Use this function at `UAP_TEST_END`.

KTXE_AT_AlterWWP()

This function is used to alter the working wafer plan (WWP) list to add all Keithley test modules (KTMs) that have failed on the current site to be executed at an alternate site.

Usage

```
int KTXE_AT_AlterWWP(long *current_wwp_list,float altx, alty)
```

Details

This function is used internally.

KTXE_AT_CheckResWithLimits()

This routine should be called from another routine or from a user access point (UAP) to check the value of a result with the limits specified in a limit list or a limit sublist.

Usage

```
int KTXE_AT_CheckResWithLimits(char *Result_Name, double Result_Value, int Limit_Code,
    char *Limit_List)
```

Result_Name	Character string specifying the name of the result; the name should exist in the limit list
Result_Value	Double value of the result
Limit_Code	An integer value between 1 and 4; the result is checked against Valid, Spec, Ctrl or Engr limit based on the code ■ 1 = Check against Valid limit ■ 2 = Check against Spec limit ■ 3 = Check against Ctrl limits ■ 4 = Check against Engr limits
Limit_List	The name of the limit list to use for looking up the result; you can pass the default `limit_list` or pass in a limit sublist name for this parameter
Return value	If the result is within the limits `TRUE` is returned; if the result is not within the limits, `FALSE` is returned

Details

This function is used internally.

This routine can be run at any UAP if the result and the limit list exist in the data pool at that UAP. This routine can also be called from other routines (for example, `KI_SubsiteTest`).

This routine returns a `TRUE` or a `FALSE`. The return value should be checked after this routine is called and action should be taken based on the returned value.

To see the errors generated by this macro when running in the Keithley Test Execution (KTXE), set the `KI_KTXE_ERROR_LOG` environment variable.

KTXE_AT_cleanup_site()

This function is used to reset counts and clean up data structures.

Usage

```
KTXE_AT_cleanup_site()
```

Details

This function is used internally.

KTXE_AT_debug_print()

This function is used to print the internal flags and counters used in `KTXE_AT`.

Usage

```
KTXE_AT_debug_print()
```

Details

Use this function at `UAP_SITE_END`.

KTXE_AT_demo_data_func()

This function generates demo data.

Usage

```
double KTXE_AT_demo_data_func(double x)
```

Details

This function is called internally by `KTXE_AT_generate_val`.

KTXE_AT_enable_kdf()

This function enables Keithley data file (KDF) logging, if previously disabled.

Usage

```
KTXE_AT_enable_kdf()
```

Details

Use this function at `UAP_TEST_DATA_LOG`.

KTXE_AT_FindAltSite()

This function is used to find the alternate site coordinates for the current site.

Usage

```
KTXE_AT_FindAltSite()
```

Details

This function is used internally.

KTXE_AT_generate_val()

This demo data generation function returns a value for each variable tested based on *highV* and *lowV*, which are upper and lower bounds, and a specified *demo_type*.

Usage

```
double KTXE_AT_generate_val(int demo_type, double highV, double lowV)
```

demo_type	The demo type (int); a number from 0 to 12 that defines result failures based on site location on the wafer ■ 0 = Random high/low fail ■ 1 = Inside fails high ■ 2 = Outside fails high ■ 3 = Top fails high ■ 4 = Bottom fails high ■ 5 = Right fails high ■ 6 = Left fails high ■ 7 = Inside fails low ■ 8 = Outside fails low ■ 9 = Top fails low ■ 10 = Bottom fails low ■ 11 = Right fails low ■ 12 = Left fails low
highV	*demo_high_lim* = Upper bound of passing results (double)
lowV	*demo_low_lim* = Lower bound of passing results (double)
Result	The test result (double)

Details

Call this function from a Keithley test module (KTM) as follows:

```
result = KTXE_AT_generate_val(demo_type, demo_high_lim, demo_low_lim)
```

Define *demo_type*, *demo_high_lim*, and *demo_low_lim* as global data for ease of use.

KTXE_AT_LogResultList()

This function is used to log saved result lists to a Keithley data file (KDF).

Usage

```
KTXE_AT_LogResultList(char *result_list_name)
```

Details

The function is used internally.

KTXE_AT_more_sites_cur_wafer_site_end()

This function is used to determine, at completion of site testing, if more sites should be tested on the present wafer.

Usage

```
KTXE_AT_more_sites_cur_wafer_site_end()
```

Details

Use this function at `UAP_SITE_END`.

KTXE_AT_more_tests_curr_wafer_site_end()

This function is used to determine, at completion of site testing, if more tests should be executed on the present wafer.

Usage

```
KTXE_AT_more_sites_cur_wafer_site_end()
```

Details

Use this function at `UAP_SITE_END`.

KTXE_AT_more_tests_curr_wafer_wafer_begin()

This function is used to add results in the alternate test class to the list of results that are disabled during initial execution.

Usage

```
KTXE_AT_more_tests_cur_wafer_wafer_begin()
```

Details

Use this function at `UAP_WAFER_BEGIN`.

KTXE_AT_more_tests_next_wafer_site_end()

This function is used to change wafer plan to be used for the next wafer that has the same wafer plan name as the current wafer plan.

Usage

```
KTXE_AT_more_tests_next_wafer_site_end()
```

Details

Use this function at `UAP_SITE_END`.

KTXE_AT_wafer_begin()

Library initialization module.

Usage

```
KTXE_AT_wafer_begin()
```

Details

Use this function at `UAP_WAFER_BEGIN`.

Section 10

Keithley User Interface Library command reference

In this section:

Introduction 10-1

Introduction

The Keithley User Interface (KUI) Library contains commands that provide the program developer with a basic set of user interfaces for operator data entry and program status monitoring program specifically for test programs.

Descriptions of the KUI commands follow.

GetProgramArgs

This command gets program command-line arguments.

Usage

```
void GetProgramArgs(int argc, char *argv[], int *debug, int *err_report_mode, char
   **err_log_fname, int *gui_look, LOT **lot, char **sum_report_options, char
   **kwf_fname, char *user_arg)
```

Details

The `GetProgramArgs` command allows values for specific test program global variables to be passed in using the command line.

The `GetProgramArgs` command parses the command-line arguments to match switches assigned to the test program variables. When a match is found, the `GetProgramArgs` command will then try to parse in its argument, converting and bounding it per the variable type, as required.

If an error occurs in interpreting the command-line arguments, the `GetProgramArgs` command will print an error message explaining the reason and the allowable command-line arguments to `stderr`, and exit the test program.

The allowable command-line arguments can be displayed by typing `-h` at the command line following the test program's name.

Valid switches are shown in the following table.

Switch	Description
`-c`	Text lot information comment field
`-d`	Text lot information device field
`-e` *`n`* `[`*`fname`*`]`	Error reporting mode Where *n* = 0 to 3: ■ 0 - None ■ 1 - Display Error Messages ■ 2 - Log Error Messages ■ 3 - Display and Log Error Messages [*fname*] - Error log file path and name
`-g` *`n`*	GUI look and feel operation Where *n* = 0 to 4: ■ 0 - Motif ■ 1 - OpenLook ■ 2 - Microsoft Windows
`-h`	■ Display command line options
`-i`	■ ID lot information lot id field
`-k`	*n* `text` Lot search key Where *n* = 1 to 3 fields
`-l id`	Lot information limit ID
`-o text`	Lot information operator field
`-p text`	Lot information process field
`-r "options"`	Lot summary report options
`-s text`	Lot information system field
`-t` *`n`*	Test station Where *n* = 1 to 4
`-w` *`fname`*	Wafer description filename
`-u text`	User argument
`-x` *`n`*	Debug flag Where *n* = -32767 to 32768 *fname* - Valid filename and path *options* - Options must be enclosed in quotes *text* - Command-line arguments will be concatenated to the switch argument until the next switch (dash-letter) occurs

Example

```
/* KI_Strncpy guarantees the member will be xxx_LENGTH & null terminated */
KI_Strncpy(lot->limitcode, "tutorial_limits", LIMITCODE_LENGTH);
KI_Strncpy(lot->id, "tutorial_lot", LOT_ID_LENGTH);
lot->teststation = 1;
GetStartTime(lot->starttime);

if(getenv("USER")!=NULL)
KI_Strncpy(lot->operator, getenv("USER"), LOT_OPERATOR_LENGTH);

if(getenv("HOST")!=NULL)
KI_Strncpy(lot->system, getenv("HOST"), LOT_SYSTEM_LENGTH);

/*>> set default Keithley wafer (description) file name */
kwf_fname = "sample.wdf";

/*>> default lot summary report options */
sum_report_options = "-s";

/*>> set/override program run time values and flags w/ command line args */
GetProgramArgs(argc, argv, &debug, &err_report_mode, &err_log_fname, &gui_look,
   &lot, &sum_report_options, &kwf_fname, user_arg);
```

This example illustrates how test program global variables can be set with default values within the program and optionally overridden from the command line.

InitUINew

This command initializes the Keithley User Interface (KUI) Library so that dialogs can be displayed and become operable. It must be called before any dialogs are called. In addition, it allows specifying the look and feel of the displayed dialogs as provided in the `DLG_LOOK_XXXX` constants and starts the control thread of execution.

Usage

```
InitUINew(int look, void *Main () );
```

Details

The KUI system is now threaded for better response to operator input. As a result, the control thread process name is specified in the `InitUINew` command. The `look` program variable is defined in `kui_proto.h`. It is used with the `GetProgramArgs` command to set the look using the test program command line.

InputMsgDlg

The `InputMsgDlg` command displays a modal dialog window containing the passed message string and provides the user with a text edit widget to enter information into the program.

Usage

```
int InputMsgDlg( char *msgstr, char *inputstr)
```

Details

The program remains in the dialog until the user presses one of the buttons. The text edit widget text is then placed in the `inputstr` buffer. The dialog then returns either `DLG_OK` or `DLG_EXIT` per the OK or CANCEL buttons. The input string pointer cannot contain a null value. It should point to some allocated memory space to place the entered string. If you do not want to use the string, pass `0` as the argument.

LotDlg

This command displays a modal dialog window in order to collect lot information from the operator. It has as arguments a pointer to a lot structure, a field enable array, and a bound for the maximum test station that can be selected.

Usage

```
LotDlg(LOT *lot, char lot_dlg_fields[NUM_LOT_FIELDS], int max_teststation)
```

Details

Default entries for the lot dialog fields can be passed through this structure prior to calling `LotDlg` as follows:

(`KI_Strncpy` guarantees the member will be `xxx_LENGTH` and null terminated)

```
LOT *lot;
KI_Strncpy(lot->limitcode,"tutorial_limits", LIMITCODE_LENGTH);
KI_Strncpy(lot->id, "tutorial_lot", LOT_ID_LENGTH);
lot->teststation = 1;
```

The `lot_dlg_fields` array is an array passed to the lot information dialog to indicate the fields that can be altered by the user. The `lot_fields` enum corresponds to indexes within that array and should be used with `FIELD_ENABLED` and `FIELD_DISABLED` constants to set array elements before calling `LotDlg` as follows:

```
lot_dlg_fields[SYSTEM_ID] = FIELD_DISABLED;
lot_dlg_fields[TEST_NAME] = FIELD_DISABLED;
LotDlg(lot, lot_dlg_fields, max_teststation);
```

All fields are enabled by default.

The `LotDlg` command returns either `DLG_OK` or `DLG_ABORT` per the OK and ABORT buttons. If exited with OK, the dialog fields are updated and returned in the lot structure. The return value can then be used to determine further program execution.

NOTE

`InitUI` must be called before `LotDlg`.

OkCancelAbortMsgDlg

This command displays a modal dialog window containing the passed message string and requires the user acknowledge it by pushing either OK or CANCEL before the program can continue.

Usage

```
int OkCancelAbortMsgDlg(char *msgstr)
```

Details

The message should be phrased to state the test program can be aborted by pressing CANCEL. If OK is pressed, the dialog returns `DLG_OK`. If CANCEL is pressed, the user is then prompted through another modal dialog to verify aborting the test program. The dialog will then return either `DLG_ABORT` if OK was pressed, or `DLG_NO` if CANCEL was pressed in the verification dialog.

OkCancelMsgDlgDialog

This command displays a modal dialog window containing the passed message string and requires the user acknowledge it by pushing either OK or CANCEL before the program can continue.

Usage

```
int OkCancelMsgDlg( char *msgstr )
```

Details

Returns either `DLG_OK` or `DLG_EXIT`.

OkMsgDlg

This command displays a modal dialog window containing the passed message string and requires the user acknowledge it by pushing OK before the program can continue.

Usage

```
void OkMsgDlg(char *msgstr)
```

Details

The `OkMsgDlg` command does not return a value. It is used to pause the test program and require the operator to acknowledge the message before the program can continue.

QuitUI

This command removes any remaining displayed dialogs and performs clean-up actions required for using the dialogs in the User Interface library.

Usage

```
int QuitUI()
```

Details

The `QuitUI` command should be called prior to program exit.

ScrollMsgDlg

This command sets a label of a dialog window.

Usage

```
void ScrollMsgDlg(char *label)
```

Details

Messages accumulate and can be scrolled through in the visible area until it is cleared or the `QuitUI` command is called. The `ScrollMsgDlg` command is called with a label to display at the top of the dialog window.

Refer to UpdateModelessDlgs - Update Modeless dialogs (on page 10-8) and UpdateStatusDlg — Update Status Dialog (on page 10-8) for issues regarding the responsiveness of the `ScrollMsgDlg` command.

ScrollMsgDlgClr

This command is used to clear all messages from the scrolling message dialog.

Usage

```
void ScrollMsgDlgClr()
```

Details

The scrolling message dialog will remain displayed and the first message sent will be placed at the top of the scrolling area.

ScrollMsgDlgMsg

This command is used to post a message to the scrolling message dialog.

Usage

```
void ScrollMsgDlgMsg(char *msgstr)
```

Details

The passed string should make use of `\n` as needed. If the `ScrollMsgDlg` command was not called previous to the `ScrollMsgDlgMsg` command, it will be called from within it, and the dialog label set as "Test Program Messages." The buffer size is limited by the `maxScrollLines` variable, which defaults to ~500 lines. When this buffer size is exceeded, the oldest 2/3 of the buffer is thrown away and the buffer will continue to grow. The maximum value can be adjusted using the data pool. Refer to "Data pool" in the reference manual for your system for an example.

StatusDlg

This command provides a modeless dialog window that is displayed from when `StatusDlg` is first called until `QuitUI` releases the user interface on program exit. Besides providing display fields for key test program variables, it also provides a single text display line on which a program-specific status message can be displayed as well as a level of program execution control through the PAUSE, CONTINUE, and ABORT buttons.

Usage

```
void StatusDlg(LOT **lot, WAFER **wafer, SITE **site, SUBSITE **subsite, int
   *total_wafers, int *wafers_tested, int *total_sites, int *sites_tested,
   kui_support_t **KUI_Support, kui_user_t **KUI_User);
```

Details

The `StatusDlg` command was designed in conjunction with guide test program data. In order to minimize the argument list required for `UpdateStatusDlg`, `StatusDlg` establishes pointers to the test program variables from which the dialog fields will obtain their display information for the remainder of the test program. The status dialog treats all these variables as read-only. The lot, wafer, site, and subsite pointers point to structures with members corresponding to status dialog display fields. Since the addresses passed for these structures also point to the present structure in their respective linked list, the status dialog automatically tracks and displays the correct information. The status dialog is called as follows:

```
StatusDlg(&lot, &wafer, &site, &subsite,
   &total_wafers, &wafers_tested,
   &total_sites, &sites_tested,
   &KUI_Support, &KUI_User );
```

The status dialog contains a Total Time field. This field displays a running timer indicating elapsed time for test plan execution. Two user fields are also available for custom use.

There is a pointer to the `KUI_User` structure in the data pool. Populating this structure will enable or disable the user fields. These fields must be initialized at `UAP_LOT_INFO` or earlier for proper operation. Once the fields are initialized, changing the values and calling the `KTXEUpdateStatusAbort()` command will cause an immediate update. If the user fields are left uninitialized, they will not be present on the Status dialog window.

The following figure shows a sample status dialog window.

Figure 18: Status dialog window

UpdateModelessDlgs

This command is used to update and display the scroll message dialog when their contents change.

Usage

```
void UpdateModelessDlgs()
```

UpdateStatusDlg

This command is provided to update and display the status dialog fields when their contents change in the test program, as well as pass a status message appropriate for that point in the program.

Usage

```
int UpdateStatusDlg(char *user_msg)
```

Details

Although the status dialog remains displayed throughout the test program, even while other dialogs are displayed, the fields are only updated and buttons checked when `UpdateStatusDlg` is called as in the following example:

```
UpdateStatusDlg("Loading Wafer ... ");
```

When `UpdateStatusDlg` is called, it first checks for differences between the test program variables and their display fields, updating them if necessary. The status message is then placed in its field. The PAUSE, CONTINUE, and ABORT buttons are then checked.

The pause-continue-abort `pca_flag` reflects the state of program execution (by default, it is set to `KI_CONTINUE`). When the status dialog buttons are checked by `UpdateStatusDlg`, if a button is pressed, it will set the flag appropriately.

Just before the `UpdateStatusDlg` command returns, it checks the `pca_flag`. If paused, the test program will remain within the `UpdateStatusDlg` call until either CONTINUE or ABORT is pressed. The `pca_flag` will be set appropriately and allow the function to return with either `DLG_OK` or `DLG_ABORT`, respectively. The return value can be used to determine further program execution.

The responsiveness of updating the status dialog and the PAUSE, CONTINUE, and ABORT buttons is directly a result of where and how often `UpdateStatusDlg` appears in the test program. For example, if a number of tests that require significant time appear in the test program, it may be necessary to intersperse `UpdateStatusDlg` between them.

NOTE

The `InitUI` command must be called before the `StatusDlg` command. The `UpdateStatusDlg` command may appear in the test program before the `StatusDlg` command, but will be ignored as the status dialog pointers and the window itself have not been established.

VarMsgDlg

This command creates a window with a scrolling text region and multiple push buttons. In addition, each push button has a user-defined label.

Usage

```
int VarMsgDlg(*VarMsgDlgDataPtr) ;
```

Details

This routine is passed a structure containing configuration data for the window. The structure definition is shown below:

```
typedef struct _VarMsgDlgData
{
int no_buttons;
int no_lines;
char **button_labels;
char *win_label;
char *ted_string;
}
VarMsgDlgDataRec, *VarMsgDlgDataPtr;
```

- **no_buttons:** The number of pushbuttons displayed. There is no limit to the number of buttons, but screen space and size will impose a practical limit.
- **no_lines:** How many lines of text will be displayed in the scrolling region.
- **button_labels:** An array of text for labels.

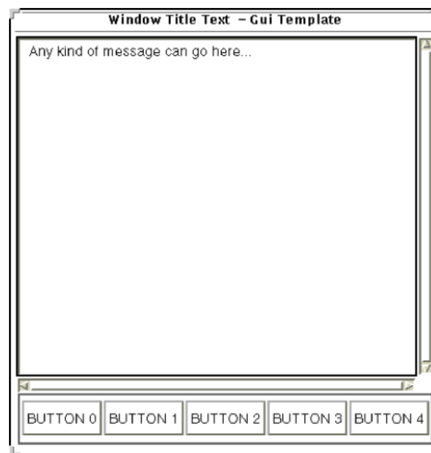
- **win_label:** The text used for the title bar on the window.
- **ted_string:** The message to be displayed in the scrolling window.

This command will return the index number of the button that was pressed by the user. The first button is index 0.

There is an example program located in the distribution showing sample Keithley User Interface (KUI) Library calls and their uses. This file is located in `$KIHOME/src/gui_template.c`.

The following figure shows an example of a dialog window created using this routine.

Figure 19: Variable message window



WfrIdsDlgDialog

NOTE

This window is not supported in the KTE Integrated Display Service (KIDS) graphical user interface (GUI). The standard Keithley User Interface (KUI) Classic window will be used.

This command displays a modal dialog window in order to collect information from the operator for multiple wafers to be tested by the test program. It facilitates automated testing as the wafer id will not have to be prompted as each wafer is tested.

Usage

```
int WfrIdsDlg(WAFER **wafer_ptr, int max_cassette, int *total_ptr)
```

Details

The `WfrIdsDlg` command has as arguments a pointer to a wafer structure pointer, a bound for the highest cassette that can be selected, and a pointer to the variable that will contain the total number of wafers to be tested.

The multiple wafer information dialog allows you to enter from `1` to `max_cassette` lists of wafer IDs and split entries corresponding to 25 slots on each cassette. While the dialog is displayed, you can switch between cassettes and can enter IDs in any order. A split entry can only be made if the slot has a corresponding ID entry.

The `WfrIdsDlg` command is passed a pointer to the test program's present WAFER structure pointer. A pointer to the pointer is required to allow the `WfrIdsDlg` command to modify the test program's present WAFER structure pointer that is passed in and return a different pointer value.

For example, the wafer structure pointer may point to NULL when the `WfrIdsDlg` command is called, but points to an allocated wafer structure in the wafer linked list upon return.

As you enter wafer information, the `WfrIdsDlg` command will allocate elements and maintain a linked list of wafer structures. When you exit the multiple wafer information dialog, the entered wafers will be in cassette-slot order in the linked list. The test program variable pointed to will contain the number of wafers in the linked list.

The `WfrIdsDlg` command returns either `DLG_OK` or `DLG_ABORT` per the OK and ABORT buttons. The return value can then be used to determine further program execution.

In most cases, the pointer to the wafer structure pointer will point to NULL when the dialog is called as the number of wafers, their position and IDs will most likely be determined at program run time. The default entries for the ID and Split fields will be empty. However, the ability to pass in a pointer to an established linked list of wafers is reserved for possible future development. It is intended for possible use to accommodate situations where wafer IDs might remain the same from test run to test run, such as program development, where generic wafer IDs are suitable, or for validation or editing of IDs obtained through some other means such as optical character recognition or bar coding.

For this reason, the linked list of wafers passed to the dialog will reflect changes even if the dialog is exited with ABORT as no tracking of which wafers were passed in will be maintained by the dialog.

The multiple wafer information dialog is called as in the following example:

```
WfrIdsDlg(&wafer, max_cassette, &total_wafers);
```

NOTE

The `InitUI` command must be called before the `WfrIdsDlg` command.

WfrIdDlg

NOTE

This window is not supported in the KTE Integrated Display Service (KIDS) graphical user interface (GUI). The standard Keithley User Interface (KUI) Classic window will be used.

This command displays a modal dialog window in order to collect information from the operator for single wafers to be tested by the test program. It is intended primarily for situations where an operator is prompted to enter information on a wafer-by-wafer basis, such as when manual wafer loading occurs. It has as arguments a pointer to a wafer structure pointer, and a bound for the highest cassette which can be selected.

Usage

```
int WfrIdDlg(WAFER **wafer_ptr, int max_cassette)
```

Details

The single wafer information dialog allows you to enter the cassette number bounded from `1` to `max_cassette`, the slot from `1` to `25`, the wafer ID and split information. A split entry can only be made if the slot has an ID entry.

A pointer to a WAFER structure pointer is passed to the `WfrIdDlg` command. A pointer to the pointer is used only to maintain similar arguments and reduce confusion with the multiple wafer information dialog `WfrIdsDlg` command, which requires a pointer to a pointer. The passed pointer value will not be modified by the `WfrIdDlg` command.

Unlike the multiple wafer information dialog, `WfrIdsDlg`, the single wafer information dialog will not allocate a wafer structure if needed within the dialog call and expects to be passed a pointer to one.

Since it is passed a pointer to an allocated wafer structure, the dialog fields can be initialized based on the values of the members of the passed structure.

If you exit the dialog with OK, the structure members are updated with the dialog field entries and the `WfrIdDlg` returns `DLG_OK`. If ABORT is used to exit the dialog, the structure members are not updated and `DLG_ABORT` is returned. The return value can then be used to determine further program execution.

The single wafer information dialog is called as in the following example, which allocates a wafer structure and initializes the fields based on the present entry in the wafer linked list, and adds it to the existing wafer linked list if the dialog is exited with OK:

```
next_wafer = CreateNewWafer();
KI_Strncpy(next_wafer->split, wafer->split, WAFER_SPLIT_LENGTH);
next_wafer->boat = wafer->boat;
next_wafer->slot = wafer->slot;

if(next_wafer->slot < MAX_SLOT)
next_wafer->slot++;
switch( WfrIdDlg( &next_wafer, max_cassette) )
{
case DLG_ABORT:
EXIT_PRGM
case DLG_OK:
AddNewWafer( wafer, next_wafer );
wafer = next_wafer;
wafers_tested++;
total_wafers++;
```

NOTE

The `InitUI` command must be called before the `WfrIdDlg` command.

YesNoAbortMsgDlg

This command displays a modal dialog window containing the passed message string and requires the user acknowledge it by pushing either YES, NO, or ABORT before the program can continue.

Usage

```
int YesNoAbortMsgDlg(char *msgstr)
```

Details

The message should be phrased to state the test program can be aborted by pressing ABORT. If YES is pressed, the dialog returns `DLG_YES`. If NO is pressed, the dialog returns `DLG_NO`. If ABORT is pressed, you are then prompted through another modal dialog to verify aborting the test program. The dialog will then return either `DLG_ABORT` if OK was pressed, or `DLG_NO` if CANCEL was pressed in the verification dialog.

YesNoCancelMsgDlg

This command displays a modal dialog window containing the passed message string and requires the user acknowledge it by pushing either YES, NO, or CANCEL before the program can continue.

Usage

```
int YesNoCancelMsgDlg(char *msgstr)
```

Details

Returns either `DLG_YES`, `DLG_NO`, or `DLG_EXIT`.

ContSkipAbortDlg

This command displays a message in a dialog box with the Continue, Skip, and Abort choices.

Usage

```
int ContSkipAbortDlg (char *msg)
```

Details

Continue returns `DLG_YES`, Skip returns `DLG_SKIP`, and Abort returns `DLG_ABORT`.

LBoxDlg

NOTE

This window is not supported in the KTE Integrated Display Service (KIDS) graphical user interface (GUI). The standard Keithley User Interface (KUI) Classic window will be used.

This command opens a window containing a list of items and allows selection of one or more of these items.

Usage

```
int LBoxDlg(char *text_label, char *windowTitle, LBOXDLG_ListPtr **listPtr, int
   MultipleSelectionEnabled) ;
```

Details

`text_label` = Text label over list of items

`windowTitle` = Title text for window frame

`listPtr` = Pointer to a linked list of `LBOXDLG_ListPtr` items

```
typedef struct _lboxDlg
{
   char *label ;
   int selected ;
   struct _lboxDlg *next ;
} LBOXDLG_ListPtr ;
```

`MultipleSelectionEnabled` = Single or Multiple Selection flag. Possible values: `LBOXDLG_SINGLE_SELECT` or `LBOXDLG_MULTI_SELECT`.

Return values:

`DLG_OK` = OK button pressed

`DLG_EXIT` = Cancel button pressed

`DLG_ABORT` = Abort button pressed

In addition, the `listPtr` list of items has been modified to reflect any selection changes. These changes only take effect if the OK button is pressed. Pressing the CANCEL or ABORT buttons leaves the list in its original state.

Example

```
int retVal, i ;
LBOXDLG_ListPtr *lboxList, *new, *last ;
/* Create a list of items to display
 */
new = ( LBOXDLG_ListPtr *)malloc( sizeof( LBOXDLG_ListPtr ) ) ;
new->label = strdup( "Item 1" ) ;
new->selected = 1 ;
new->next = NULL ;
last = lboxList = new ;
for ( i = 2 ; i < 10 ; i++ )
{
   char label[ 64 ] ;
   sprintf( label, "Item %d", i ) ;
   new = ( LBOXDLG_ListPtr *)malloc( sizeof( LBOXDLG_ListPtr ) ) ;
   new->label = strdup( label ) ;
   new->selected = 0 ;
   new->next = NULL ;

   last->next = new ;
   last = new ;
}
/* Show list and get selection. ( Single select mode )
 */
retVal = LBoxDlg( "Test lbox text",
      "Lbox Title",
      &lboxList,
      LBOXDLG_SINGLE_SELECT ) ;
/* free the list. Display the data while we are here
 */
while( lboxList != NULL )
{
   LBOXDLG_ListPtr *tmp ;
   if ( 1 == lboxList->selected )
      printf( "%s was selected!\n", lboxList->label ) ;

   free( lboxList->label ) ;
   tmp = lboxList->next ;
   free( lboxList ) ;
   lboxList = tmp ;
}
```

Section 11

Data pool command reference

In this section:

Introduction 11-1

Introduction

The data pool holds global data while the Keithley Test Execution Engine (KTXE) is running. When KTXE starts, the variables declared in the global data files and probe card file are copied into the data pool.

The data pool is also used to look up variables for parameters that are passed to KULT-generated modules when they are run at the UAPs.

Descriptions of the data pool commands follow.

dpAddData

This command adds new non-pointer type data to the data pool.

Usage

```
int dpAddData(char *name, int type, ...);
```

name	Input	Character string containing the name of the data
type	Input	Non-pointer type of data (`INT`, `FLOAT`, `LONG`, `DOUBLE`, `CHAR`)
value	Input	Absolute data value; the value is interpreted internally based on the *type* parameter
Return value	Output	Returns one of the following statuses: ■ `OK_dpAdd`: The add was successful ■ `FAILED_dpAdd`: There was a problem allocating memory for the data node; the add was unsuccessful

Details

If data of the same name already exists in the data pool, the type and the value is overwritten with the new data.

Examples

```
status = dpAddData("MyIntData", INT, 10);
```

```
status = dpAddData("MyFloatData", FLOAT, 10.11);
```

dpAddPointer

This function is used to add pointer-type data to the data pool. If data of the same name already exists in the data pool, the type and the value is overwritten with the new data.

Usage

```
int dpAddPointer(char *name, int type, void *valuep);
```

name	Input	Character string containing the name of the data
type	Input	Pointer type of data (INT_P, FLOAT_P, LONG_P, DOUBLE_P, CHAR_P)
valuep	Input	Value pointer; see Details
Return value	Output	Returns one of the following statuses: ■ OK_dpAdd: The add was successful ■ FAILED_dpAdd: There was a problem allocating memory for the data node. The add was unsuccessful

Details

When using a dpAddPointer call, the data value being added to the data pool must be either static or malloc'd. Do not use automatic pointers.

Example

```
status = dpAddPointer("MyPointerData", FLOAT_P, floatptr);
```

dpAddArray

This function is used to add arrays to the data pool. If data of the same name already exists in the data pool, the type and the value is overwritten with the new data.

Usage

```
int dpAddArray(char *name, int type, void *valuep, int elements);
```

name	Input	Character string containing the name of the data
type	Input	Array type of data (`INT_ARRAY`, `FLOAT_ARRAY`, `DOUBLE_ARRAY`)
valuep	Input	Pointer to the first element of the array
elements		Number of elements in the array
Return value	Output	Returns one of the following statuses: ■ `OK_dpAdd`: The add was successful ■ `FAILED_dpAdd`: There was a problem allocating memory for the data node; the add was unsuccessful

Example

```
status = dpAddArray("MyArrayData", INT_ARRAY, arrayptr, 10);
```

*dpGetDataPtr

This function is used to get a pointer to a value of non-pointer type data (for example, `INT`, `FLOAT`, `DOUBLE`, `LONG`, `CHAR`) in the data pool. This function returns a void pointer. It is the user's responsibility to typecast the value to the appropriate data type.

Usage

```
void *dpGetDataPtr(char *name, int type);
```

name	Input	Character string containing the name of the data to be extracted from the data pool
type	Input	The data type of the variable
Return value	Output	■ If the data of a specified name and type is found in the data pool, a void pointer to the value is returned; it is the user's responsibility to typecast the data to the appropriate data type (for example, `int *`, `float *`, `double *`, `long *`, `char *`) ■ If the data is not found in the data pool, `NULL` is returned

Example

```
valueptr = (int *)dpGetDataPtr("MyInt Data", INT);
```

*dpGetPointer

This function is used to get the pointer value of pointer type data (for example, `INT_P`, `FLOAT_P`, `DOUBLE_P`, `LONG_P`, `CHAR_P`) in the data pool. This function returns a void pointer. It is the user's responsibility to typecast the value to the appropriate type.

Usage

```
void *dpGetDataPtr(char *name, int type)
```

name	Input	Character string containing the name of the data to be extracted from the data pool
type	Input	The data type of the variable
Return value	Output	■ If the data of a specified name and type is found in the data pool, a void pointer to the value is returned; it is the user's responsibility to typecast the data to the appropriate data type (for example, `int *`, `float *`, `double *`, `long *`, `char *`) ■ If the data is not found in the data pool, `NULL` is returned

Example

```
valueptr = (float *)dpGetPointer("MyPointer Data", FLOAT_P);
```

*dpGetArrayElement

This function is used to get a specific element of an array from the data pool.

Usage

```
*dpGetArrayElement(char *arrname, int type, int element)
```

arrname	Input	Character string containing the name of the array
type	Input	The data type of the variable (for example, `INT_ARRAY`, `FLOAT_ARRAY`, `DOUBLE_ARRAY`)
element		Integer value specifying the index of the array
Return value	Output	Returns a pointer to the specific element in the array. ■ If the specified array name is found in the data pool, a void pointer to the specified element in the array is returned; it is the user's responsibility to typecast the data to the appropriate type `NULL` is returned in one of the following situations: ■ If the array name is not found in the data pool ■ If the data requested is not of an array type (for example, `INT_ARRAY`, `FLOAT_ARRAY`, `DOUBLE_ARRAY`) ■ If the element requested is less than 0 or larger than the number of elements in the array

Example

```
valueptr = (int *)dpGetArrayElement("MyArray Data", INT_ARRAY, 7);
```

dpRemoveData

This function is used to remove specific data from the data pool. It will free up the allocated memory for the node in the data pool and the memory for the value of any of the non-pointer type data (for example, `INT`, `FLOAT`, `LONG`, `DOUBLE`, `CHAR`). The user is responsible for reallocating memory for all of the pointer type data and the arrays.

> **CAUTION**
> **Do not remove any variables put into the data pool by KTXE. This may cause fatal errors.**

Usage

```
void dpRemoveData(char *name, int type);
```

name	Input	Character string containing the name of the data to be removed from the data pool
type	Input	The data type of the variable

Details

This function does not return anything.

Examples

```
dpRemoveData("MyArrayData", INT_ARRAY);
```

```
dpRemoveData("MyPointerData", FLOAT_P);
```

dpPrintData

This function allows the user to print a variable and its value in the data pool by passing the name and type of data.

> **NOTE**
> All the `dpPrint` routines print to the location specified in the `KI_KTXE_DEBUG_LOG` environment variable. To print to the screen, set this environment variable to "`/dev/tty`".

Usage

```
void *dpGetDataPtr(char *name, int type);
```

name	Input	Character string containing the name of the data to be printed
type	Input	The data type of the variable

Details

This function does not return anything.

Example

```
dpPrintData("MyIntData", INT);
```

dpPrintAllData

This function allows you to print all the data in the data pool.

NOTE

All the `dpPrint` routines print to the location specified in the `KI_KTXE_DEBUG_LOG` environment variable. To print to the screen, set this environment variable to "`/dev/tty`".

Usage

```
void dpPrintAllData(void)
```

Details

No parameters are required for this function.

This function does not return anything.

Examples

```
dpPrintAllData();
```

Index

*

*dpGetArrayElement • 11-4

*dpGetDataPtr • 11-3

*dpGetPointer • 11-4

1

100 MX_INVLDCNT • 2-19

101 MX_NOPIN • 2-19

102 MX_MULTICON • 2-20

109 MX_ILLGLTSN • 2-20

113 MX_NOSWITCH • 2-20

114 MX_ILLGLCON • 2-20

122 UT_INVLDPRM • 2-20

126 UT_NOURAM • 2-20

129 UT_TMRIVLD • 2-20

137 UT_INVLDVAL • 2-21

152 CB_BADFUNC • 2-21

156 CB_NOFILE • 2-21

157 CB_FORMAT • 2-21

162 CB_INVLDERROR • 2-21

163 CB_INVLDEVENT • 2-21

166 CB_INSNOTREC • 2-21

173 CB_MULTITIMER • 2-22

194 MX_INVLDTRM • 2-22

2

20 LPT_PREVERR • 2-19

21 LPT_FATAL • 2-19

22 LPT_FATALINTEST • 2-19

233 FM_NOCON • 2-22

24 LPT_TOMANYARGS • 2-19

3

3 LPT_NOCOMCHAN • 2-18

4

455 ECP_PROTOVER • 2-22

5

5 SYS_MEM_ALLOC_ERR • 2-19

6

601 SYS_INTERNAL_ERR • 2-22

610 SYS_SPAWN_ERR • 2-22

611 SYS_NETWORK_ERR • 2-22

612 SYS_PROTOCOL_ERR • 2-22

650 TAPI_BADCHANNEL • 2-23

651 TAPI_BADTESTER • 2-23

652 TAPI_NOTFOUND • 2-23

653 TAPI_REFUSED • 2-23

656 TAPI_CHANLIMIT • 2-23

657 TAPI_BUFOFLOW • 2-23

A

addcon • 2-24

AddNew • 7-17

AddNew[STRUCTURE] • 7-17

adelay • 2-25

arrays • 3-34, 3-58

asweepX • 2-26

Averaged measurements • 2-6

avgX • 2-28

B

beta • 3-6, 3-8, 3-9, 3-11, 3-13

beta1 • 3-6

beta2 • 3-8

beta2a • 3-9

beta3a • 3-11

bice • 3-13

bipolar

bipolar subroutines • 3-4, 3-13, 3-15, 3-17, 3-18, 3-19, 3-21, 3-26, 3-42, 3-43, 3-44, 3-45, 3-59, 3-61, 3-70

Bipolar subroutines • 3-4

bkdn • 3-14

bmeasX • 2-30

body effect • 3-37

breakdown voltage

collector-base • 3-15, 3-17

collector-emitter • 3-18, 3-19, 3-21, 3-22

drain-source • 3-24, 3-25

emitter-base • 3-26

voltage • 3-14

bsweepX • 2-32

bvcbo • 3-15

bvcbo1 • 3-17

bvceo • 3-18

bvceo2 • 3-19

bvces • 3-21

bvces1 • 3-22

bvdss • 3-24

bvdss1 • 3-25

bvebo • 3-26

C

Calling the getlpterr function • 2-17

cap • 3-28

capacitance measurements • 3-28

capacitors • 3-6

Categorized command lists • 2-1

Categorized subroutine lists • 3-4

clrcon • 2-33

clrscn • 2-34

clrtrg • 2-35

Combination commands • 2-2

Commands for USB instruments not supported by systems drivers • 2-15

Commands supported by all drivers • 2-12

Commands supported for CVUs • 2-13

Commands supported for DMMs • 2-14

Commands supported for PGUs • 2-14

Commands supported for scope cards • 2-14

Commands supported for SMUs • 2-13

Commands supported for switch mainframes • 2-15

Commands supported for systems • 2-14

Commands supported for the RSA306B spectrum analyzer • 2-13

Commands that support timer functions • 2-15

comment

update comment routines • 7-15

conpin • 2-37

conpth • 2-39

Contact information • 1-1

ContSkipAbortDlg • 10-13

ContSkipAbortDlg - Continue Skip Abort Message Dialog • 10-13

Conventions used in this manual • 1-2

CreateNew • 7-18

CreateNew[STRUCTURE] • 7-18

current

collector-base • 3-43

drain • 3-46

leakage • 3-44, 3-45, 3-53, 3-57

substrate • 3-55

D

data logging

data logging routines • 7-2

Data logging routines • 7-2

data pool

data pool functions • 11-1

Data pool command reference • 11-1

delay • 2-40, 3-56, 3-68

delcon • 2-41

delete

DeleteLimit • 7-15

DeleteLimitCode • 7-15

DeleteLot • 7-13

DeleteParam • 7-14

DeleteSite • 7-14

DeleteWafer • 7-13

DeleteLimit • 7-15

DeleteLimitCode • 7-15

DeleteLot • 7-13

DeleteParam • 7-14

DeleteSite • 7-14

DeleteWafer • 7-13

deltl1 • 3-29

deltw1 • 3-30

devclr • 2-42

devint • 2-42

dialog

ContSkipAbortDlg - Continue Skip Abort Message Dialog • 10-13

InputMsgDlg - Input Message Dialog • 10-4

LBoxDlg - List Box Message Dialog • 10-14

LotDlg - Lot Information Dialog • 10-4

OkCancelAbortMsgDlg - Ok Cancel Abort Message Dialog • 10-5

OkCancelMsgDlg - Ok Cancel Message Dialog • 10-5

OkMsgDlg - Ok Message Dialog • 10-5

ScrollMsgDlg - Scrollable Message Dialog • 10-6

ScrollMsgDlgClr - Scrollable Message Dialog Clear • 10-6

ScrollMsgDlgMsg - Scrollable Message Dialog Message • 10-7

StatusDlg - Status Dialog • 10-7

UpdateModelessDlgs - Update Modeless Dialogs • 10-8

UpdateStatusDlg - Update Status Dialog • 10-8

VarMsgDlg - Variable Message Dialog • 10-9

WfrIdDlg - Single Wafer Information Dialog • 10-11

WfrIdsDlg - Multiple Wafer Information Dialog • 10-10

YesNoAbortMsgDlg - Yes No Abort Message Dialog • 10-13

YesNoCancelMsgDlg - Yes No Cancel Message Dialog • 10-13

diode • 3-6, 3-57, 3-71

disable • 2-44

dpAddArray • 11-3

dpAddData • 11-1

dpAddPointer • 11-2

dpPrintAllData • 11-6

dpPrintData • 11-5

dpRemoveData • 11-5

drain

drain conductance • 3-39

drain current • 3-46, 3-48, 3-49

Dual-site commands • 2-2

E

enable • 2-44

end

EndLot • 7-6

EndSite • 7-6

EndWafer • 7-6

EndLot • 7-6

EndSite • 7-6

EndWafer • 7-6

Error handling • 2-16

Error messages • 2-17, 2-18

ev • 3-31

F

FET

FET and JFET subroutines • 3-4, 3-40, 3-49, 3-76, 3-78

FET and JFET subroutines • 3-4

FileExist • 7-12

fimv • 3-33

FindFirst • 7-19

FindFirst[STRUCTURE] • 7-19

FindLast • 7-19

FindLast[STRUCTURE] • 7-19

FindNext • 7-20

FindNext[STRUCTURE] • 7-20

FindPrev • 7-21

FindPrev[STRUCTURE] • 7-21

Fixed range versus autorange measurements • 2-16

Fixed ranging • 2-9

Fix-range trigger instruments • 2-16

fnddat • 3-34

fndtrg • 3-35

forceX • 2-45

fvmi • 3-36

G

gamma • 3-37

gamma1 • 3-37

gate_charge • 4-4

gd • 3-39

General commands • 2-2

GetComment • 7-15

GetLimit • 7-16

GetLimitCode • 7-16

GetLot • 7-7

GetLotData • 7-11

getlpterr • 2-46

GetParam • 7-10

GetParamList • 7-11

GetProgramArgs • 10-1

GetProgramArgs - Get Program Command Line Arguments • 10-1

GetSite • 7-9

GetStartTime • 7-13

getstatus • 2-47

GetWafer • 7-8

gm • 3-40

GPIB • 2-12

GPIB commands • 2-3

H

High-Voltage Library commands • 4-4

How to use the library reference • 3-2, 4-1

hv_bvsweep • 4-6

hvcv_3term • 4-8

hvcv_3term_basic • 4-11

hvcv_comp • 4-13

hvcv_genCompData • 4-14

hvcv_genCompFreq • 4-16

hvcv_getData • 4-18

hvcv_intgcg • 4-19

hvcv_measure • 4-21

hvcv_storeData • 4-23

hvcv_sweep • 4-24

hvcv_sweep_basic • 4-27

hvcv_test • 4-29

hvcv_test_basic • 4-32

HVLib command reference • 4-1

I

ibic1 • 3-42

icbo • 3-43

iceo • 3-44

ices • 3-45

id1 • 3-46

idsat • 3-48

idss • 3-49

idvsvd • 3-51

idvsvg • 3-52

iebo • 3-53

imeast • 2-48

InitUINew • 10-3

InitUINew - Initialize User Interface Library • 10-3

InputMsgDlg • 10-4

InputMsgDlg - Input Message Dialog • 10-4

insbind • 2-49

InsertNew • 7-22

InsertNew[STRUCTURE] • 7-22

Instrument and terminal IDs • 2-15

Instruments and instrument drivers • 2-12

Integrated measurements • 2-6

intgX • 2-50

Introduction • 1-1, 2-1, 3-1, 4-1, 5-1, 6-1, 8-1, 9-1, 10-1, 11-1

isubmx • 3-55

K

kdelay • 3-56

Keithley data files (KDF) library command reference • 7-1

Keithley User Interface (KUI) library

ContSkipAbortDlg - Continue Skip Abort Message Dialog • 10-13

GetProgramArgs - Get Program Command Line Arguments • 10-1

InitUINew - Initialize User Interface Library • 10-3

InputMsgDlg - Input Message Dialog • 10-4

LBoxDlg - List Box Message Dialog • 10-14

LotDlg - Lot Information Dialog • 10-4

OkCancelAbortMsgDlg - Ok Cancel Abort Message Dialog • 10-5

OkCancelMsgDlg - Ok Cancel Message Dialog • 10-5

OkMsgDlg - Ok Message Dialog • 10-5

QuitUI - Quit User Interface • 10-6

ScrollMsgDlg - Scrollable Message Dialog • 10-6

ScrollMsgDlgClr - Scrollable Message Dialog Clear • 10-6

ScrollMsgDlgMsg - Scrollable Message Dialog Message • 10-7

StatusDlg - Status Dialog • 10-7

UpdateModelessDlgs - Update Modeless Dialogs • 10-8

UpdateStatusDlg - Update Status Dialog • 10-8

VarMsgDlg - Variable Message Dialog • 10-9

WfrIdDlg - Single Wafer Information Dialog • 10-11

YesNoAbortMsgDlg - Yes No Abort Message Dialog • 10-13

YesNoCancelMsgDlg - Yes No Cancel Message Dialog • 10-13

Keithley User Interface Library command reference • 10-1

KI_MultiSite command reference • 5-1

kibdefclr • 2-52

kibdefint • 2-53

kibrcv • 2-54

kibsnd • 2-55

kibspl • 2-56

kibsplw • 2-57

KTXE_AT result-based testing command reference • 9-1
KTXE_AT_alternate_site_site_end() • 9-1
KTXE_AT_alternate_site_test_end() • 9-1
KTXE_AT_AlterWWP() • 9-2
KTXE_AT_CheckResWithLimits() • 9-2
KTXE_AT_cleanup_site() • 9-3
KTXE_AT_debug_print() • 9-3
KTXE_AT_demo_data_func() • 9-3
KTXE_AT_enable_kdf() • 9-3
KTXE_AT_FindAltSite() • 9-4
KTXE_AT_generate_val() • 9-4
KTXE_AT_LogResultList() • 9-5
KTXE_AT_more_sites_cur_wafer_site_end() • 9-5
KTXE_AT_more_tests_curr_wafer_site_end() • 9-5
KTXE_AT_more_tests_curr_wafer_wafer_begin() • 9-5
KTXE_AT_more_tests_next_wafer_site_end() • 9-6
KTXE_AT_wafer_begin() • 9-6
KTXE_RP zone-based testing command reference • 8-1
KTXE_RP_CleanUpWDF • 8-1
KTXE_RP_CreateRandomWDF • 8-1
KTXE_RP_CreateWPF • 8-2
KTXE_RP_GetUsrArgs • 8-2
KTXE_RP_RemoveWPF • 8-2

L

LBoxDlg • 10-14
LBoxDlg - List Box Message Dialog • 10-14
leak • 3-57
leakage current • 3-44, 3-53, 3-57
limit code
  DeleteLimit • 7-15
  DeleteLimitCode • 7-15
  GetLimitCode • 7-16
  LimitExist • 7-24
  PutLimit • 7-17
LimitExist • 7-24
limits
  update limits routines • 7-16
limitX • 2-57
logstp • 3-58
lorangeX • 2-59
lot
  DeleteLot • 7-13
  EndLot • 7-6
  GetLot • 7-7
  LotExist • 7-12
  PutLot • 7-2
LotDlg • 10-4
LotDlg - Lot Information Dialog • 10-4
LotExist • 7-12
LPTLib command descriptions • 2-24
LPTLib command reference • 2-1

M

MatchParam2Limit • 7-12
math
  math subroutines • 3-5, 3-34, 3-35, 3-56, 3-58, 3-68
Math and support subroutines • 3-5
Matrix commands • 2-3
Matrix operations • 2-9
Measure commands • 2-3
Measuring • 2-6
measX • 2-61
MESFET • 3-40, 3-49, 3-76, 3-78
MOSFET
  drain conductance • 3-39
  gate length reduction • 3-29

gate width reduction • 3-30

MOSFET subroutines • 3-5, 3-24, 3-25, 3-29, 3-30, 3-37, 3-39, 3-46, 3-48, 3-55, 3-72, 3-74, 3-79, 3-80, 3-82, 3-85

threshold voltage • 3-79, 3-82, 3-85

MOSFET subroutines • 3-5

mpulse • 2-63

multi_site_clear_mapping() • 5-1

multi_site_mapping() • 5-3

N

new

AddNew • 7-17

CreateNew • 7-18

O

OkCancelAbortMsgDlg • 10-5

OkCancelAbortMsgDlg - Ok Cancel Abort Message Dialog • 10-5

OkCancelMsgDlg - Ok Cancel Message Dialog • 10-5

OkCancelMsgDlgDialog • 10-5

OkMsgDlg • 10-5

OkMsgDlg - Ok Message Dialog • 10-5

Optimizing test sequences • 2-16

Ordinary measurements • 2-6

Overview • 1-2, 7-1

oxide thickness • 3-69

P

parameter

DeleteParam • 7-14

GetParamList • 7-11

PutParam • 7-4

PutParamList • 7-5

PARLib command reference • 3-1

pgu_current_limit • 2-64

pgu_delay • 2-64

pgu_fall • 2-65

pgu_halt • 2-66

pgu_height • 2-66

pgu_init • 2-67

pgu_load • 2-67

pgu_mode • 2-68

pgu_offset • 2-69

pgu_period • 2-69

pgu_range • 2-70

pgu_rise • 2-71

pgu_select • 2-71

pgu_trig • 2-72

pgu_trig_burst • 2-72

pgu_trig_unit • 2-73

pgu_width • 2-74

PrAbsMove • 6-1

PrAdjustZHeight • 6-2

PrAutoAlign • 6-2

PrCassetteMap • 6-2

PrCassetteMask • 6-3

PrCheckOptions • 6-4

PrChuck • 6-4

PrClearAll • 6-5

PrClearPipeline • 6-5

PrError • 6-5

PrGetNxtWafer • 6-6

PrGetProduct • 6-6

PrGetWafer • 6-7

PrInit • 6-7

PrLoad • 6-8

PrLoadProduct • 6-8

PrLowerBoat • 6-9

PrMove • 6-9

PrNeedleClean • 6-9

Prober and prober driver command reference • 6-1
PrProfile • 6-10
PrPutNxtSlot • 6-10
PrPutWafer • 6-11
PrQueryChuckTemp • 6-11
PrQueryZHeight • 6-12
PrReadId • 6-12
PrRelMove • 6-12
PrRelReturn • 6-13
PrSerialPoll • 6-13
PrSetChuckTemp • 6-13
PrSetDiam • 6-14
PrSetDieSize • 6-14
PrSetFlat • 6-14
PrSetMode • 6-14
PrSetPipeline • 6-15
PrSetQuadrant • 6-15
PrSetRefDie • 6-15
PrSetSlotStatus • 6-16
PrSetTime • 6-17
PrSetUnits • 6-17
PrSetZHeight • 6-17
PrSmifClamp • 6-18
PrSmifLock • 6-18
PrSmifStatus • 6-19
PrStart • 6-20
PrStatus • 6-20
PrStop • 6-21
PrUnLoad • 6-21
PrWriteRead • 6-21
PrWriteReadSRQ • 6-22
PrZParams • 6-24
PrZTravel • 6-24
Pulse generator commands • 2-4
pulseX • 2-75
PutComment • 7-16
PutLimit • 7-17
PutLot • 7-2
PutParam • 7-4
PutParamList • 7-5
PutSite • 7-3
PutWafer • 7-3

Q

QuitUI • 10-6
QuitUI - Quit User Interface • 10-6

R

Range commands • 2-4
Range limits • 2-9
rangeX • 2-77
Ranging • 2-8
rcsat • 3-59
rdelay • 2-79
re • 3-61
refctrl • 2-79
remove • 7-23
remove data points • 3-34
Remove[STRUCTURE] • 7-23
res • 3-63
res2 • 3-64
res4 • 3-65
resistance
    measurements • 3-61, 3-63, 3-64, 3-65, 3-66, 3-67
resistors • 3-6, 3-63, 3-64, 3-65, 3-66
Resistors, diodes, capacitors, and special structure subroutines • 3-6
Result values indicating an error • 2-18
resv • 3-66
reverse bias • 3-43, 3-53

rsa_close • 2-80
rsa_detect_peaks • 2-81
rsa_init • 2-83
rsa_measure • 2-83
rsa_measure_next • 2-84
rsa_selftest • 2-85
rsa_setup • 2-86
rtfary • 2-88
rttrigary • 2-88
rvdp • 3-67

S

savgX • 2-89
Scope card commands • 2-5
scp_close • 2-90
scp_detect_peaks • 2-91
scp_init • 2-93
scp_measure • 2-93
scp_measure_next • 2-94
scp_selftest • 2-95
scp_setup • 2-96
ScrollMsgDlg • 10-6
ScrollMsgDlg - Scrollable Message Dialog • 10-6
ScrollMsgDlgClr • 10-6
ScrollMsgDlgClr - Scrollable Message Dialog Clear • 10-6
ScrollMsgDlgMsg • 10-7
ScrollMsgDlgMsg - Scrollable Message Dialog Message • 10-7
searchX • 2-97
setauto • 2-100
setmode • 2-101
setmode modifier tables • 2-102
Settling time • 2-8
setXmtr • 2-108
sintgX • 2-109
site
  DeleteSite • 7-14
  EndSite • 7-6
  GetSite • 7-9
  PutSite • 7-3
site_disable • 2-110
site_enable • 2-111
site_mapping • 2-112
site_status • 2-113
Smart ranging • 2-8
smeasX • 2-114
Source commands • 2-5
Sourcing and limits • 2-7
Special error values returned • 2-17
Spectrum analyzer commands • 2-5
ssmeasX • 2-116
StatusDlg • 10-7
StatusDlg - Status Dialog • 10-7
Sticky ranging • 2-8
structure
  structure handling routines • 7-17
Structure handling routines • 7-17
Subroutine descriptions • 3-6
substrate • 3-55
Sweeping • 2-10
sweepX • 2-118
Systems documentation • 1-1

T

tdelay • 3-68
Timing commands • 2-6
tox • 3-69
transconductance • 3-40
transistor • 3-59, 3-70, 3-74, 3-76
trigger mode

native • 3-35

Triggers • 2-11

trigXg, trigXl • 2-121

tstsel • 2-124

U

update

update comment routines • 7-15

update limits routines • 7-16

Update comment routines • 7-15

Update limits routines • 7-16

UpdateModelessDlgs • 10-8

UpdateModelessDlgs - Update Modeless Dialogs • 10-8

UpdateStatusDlg • 10-8

UpdateStatusDlg - Update Status Dialog • 10-8

Use combination commands • 2-16

User interface library functions • 10-1

User interface library variables • 10-4

Using the LPTLib • 2-6

V

VarMsgDlg • 10-9

VarMsgDlg - Variable Message Dialog • 10-9

vbes • 3-70

vf • 3-71

vg2 • 3-72

vgsat • 3-74

voltage

base-emitter • 3-70

early • 3-31

forward biased junction • 3-71

gate-source • 3-72, 3-74, 3-82, 3-85

pinch-off • 3-76, 3-78

threshold • 3-74, 3-79, 3-80, 3-82

vp • 3-76

vp1 • 3-78

vt14 • 3-79

vtati • 3-80

vtext • 3-82

vtext2 • 3-85

vtext3 • 3-87

W

WfrIdDlg • 10-11

WfrIdDlg - Single Wafer Information Dialog • 10-11

WfrIdsDlgDialog • 10-10

Y

YesNoAbortMsgDlg • 10-13

YesNoAbortMsgDlg - Yes No Abort Message Dialog • 10-13

YesNoCancelMsgDlg • 10-13

YesNoCancelMsgDlg - Yes No Cancel Message Dialog • 10-13

Specifications are subject to change without notice.
All Keithley trademarks and trade names are the property of Keithley Instruments.
All other trademarks and trade names are the property of their respective companies.

Keithley Instruments
Corporate Headquarters • 28775 Aurora Road • Cleveland, Ohio 44139 • 440-248-0400 • Fax: 440-248-6168 • 1-800-935-5595 • www.tek.com/keithley