tek.com/keithley

Model 4200A-SCS LPT Library

Programming

4200A-LPT-907-01 Rev. C March 2023

4200A-LPT-907-01C

KEITHLEY
A Tektronix Company

Model 4200A-SCS

LPT Library

Programming

Document number: 4200A-LPT-907-01 Rev. C March 2023

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.

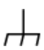
If you are using a test fixture, keep the lid closed while power is applied to the device under test. Safe operation requires the use of a lid interlock.

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 2018.

Table of contents

Section 1

Introduction

In this section:

LPT library reference

The Keithley Instruments Linear Parametric Test Library (LPT library) is a high-speed data acquisition and instrument control software library. It is the programmer's lowest level of command interface to the system instrumentation. You can use the library commands to configure the relay matrix and instrumentation for parametric tests.

This section lists the commands included in the LPT library and describes how to use them. The descriptions include:

- A brief description of the command.
- Usage, which shows how the command should be organized and descriptions of each parameter. The parameters that you need to supply are shown in italics. For example, for the command `int delay(long n);`, replace *n* with the duration of the delay.
- Detailed information about the command.
- Examples that show a typical use of the command in a test sequence.

The following conventions are used when explaining the commands:

- All LPT library commands are case-sensitive and must be entered as lower case when writing program code.
- 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 specific command.

- A capital letter *X* in a command name indicates that you must select from a list of replacement suffixes. For example, in `forceX`, replace the *X* with either a `v` for voltage or `i` for current. The following is a table of possible suffixes, the parameter each represents, and the units used throughout the LPT library for that parameter.

Suffix	Parameter	Unit
`i`	Current	Amperes
`t`	Time	Seconds
`v`	Voltage	Volts
`f`	Frequency	Hertz

Lists of LPT library commands

These topics list the LPT library commands that are available in the 4200A-SCS. A brief description and links to full descriptions of each command are provided.

The LPT library commands are grouped as follows:

- General operation commands (on page 1-2)
- Math operation commands (on page 1-4)
- SMU commands (on page 1-4)
- PGU (pulse only) and PMU (pulse and measure) commands (on page 1-5)
- CVU commands (on page 1-7)
- Switch commands (on page 1-8)

General operation commands

General operation commands include commands to control timing, execution, communications, and test status.

Command	Description
clrscn (on page 2-2)	Clears the measurement scan tables associated with a sweep.
clrtrg (on page 2-3)	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.
delay (on page 2-5)	Provides a user-programmable delay in a test sequence.
devint (on page 2-6)	Resets all active instruments in the system to their default states.
disable (on page 2-8)	Stops the timer and sets the time value to zero (0).
enable (on page 2-8)	Provides correlation of real time to measurements of voltage, current, conductance, and capacitance.
execut (on page 2-8)	Causes the system to wait for the preceding test sequence to be executed.
getinstattr (on page 2-9)	Returns configured instrument attributes.
getinstid (on page 2-10)	Returns the instrument identifier (ID) from the instrument name string.
getinstname (on page 2-10)	Returns the instrument name string from the instrument identifier (ID).
GetKiteCycle (on page 2-11)	Returns the present Clarius cycle number.
GetKiteDevice (on page 2-11)	Returns the device that Clarius is presently testing.

Command	Description
GetKiteSite (on page 2-11)	Returns the site number for the site that Clarius is presently testing.
GetKiteSubsite (on page 2-12)	Returns the subsite number for the site that Clarius is presently testing.
GetKiteTest (on page 2-12)	Returns the test that Clarius is presently testing.
getlpterr (on page 2-13)	Returns the first LPT library error since the last `devint` command.
imeast (on page 2-13)	Forces a reading of the timer and returns the result.
inshld (on page 2-13)	Provided for compatibility with Model S400 LPT library.
kibcmd (on page 2-14)	Enables universal, addressed, and unaddressed GPIB bus commands to be sent through the GPIB interface.
kibdefclr (on page 2-15)	Defines the device-dependent command sent to an instrument connected to the GPIB interface.
kibdefdelete (on page 2-16)	Deletes all command definitions previously made with the `kibdefclr` (Keithley GPIB define device clear) and `kibdefint` (Keithley GPIB define device initialize) commands.
kibdefint (on page 2-16)	Defines a device-dependent command sent to an instrument connected to the GPIB interface.
kibrcv (on page 2-17)	Reads a device-dependent string from an instrument connected to the GPIB interface.
kibsnd (on page 2-18)	Sends a device-dependent command to an instrument connected to the GPIB interface.
kibspl (on page 2-19)	Serial polls an instrument connected to the GPIB interface.
kibsplw (on page 2-20)	Synchronously serial polls an instrument connected to the GPIB interface.
kspcfg (on page 2-20)	Configures and allocates a serial port for RS-232 communications.
kspdefclr (on page 2-21)	Defines a device-dependent character string sent to an instrument connected to a serial port.
kspdefdelete (on page 2-22)	Deletes all command definitions previously made with the `kspdefclr` (Keithley Serial Define Device Clear) and `kspdefint` (Keithley Serial Define Device Initialize) commands.
kspdefint (on page 2-22)	Defines a device-dependent character string sent to an instrument connected to a serial port.
ksprcv (on page 2-23)	Reads data from an instrument connected to a serial port.
kspsnd (on page 2-23)	Sends a device-dependent command to an instrument attached to a RS-232 serial port.
PostDataDouble (on page 2-24)	Posts double-precision floating-point data from memory into the Clarius Analyze sheet.
PostDataDoubleBuffer (on page 6-11)	Posts PMU data retrieved from the buffer into the Clarius Analyze sheet (large data sets).
PostDataInt (on page 2-25)	Posts an integer-type point from memory to the Clarius Analyze sheet in the user test module and plots it on the graph.
PostDataString (on page 2-26)	Transfers a string from memory into the Clarius Analyze sheet in the user test module and plots it on the graph.
rdelay (on page 2-26)	Sets a user-programmable delay.
rtfary (on page 2-27)	Returns the force array determined by the instrument action.
savgX (on page 2-27)	Makes an averaging measurement for every point in a sweep.
scnmeas (on page 2-29)	Makes a single measurement on multiple instruments at the same time.
searchX (on page 2-29)	Used to determine the voltage or current required to get a current or voltage.
setmode (on page 2-32)	Sets instrument-specific operating mode parameters.
sintgX (on page 2-34)	Makes an integrated measurement for every point in a sweep.
smeasX (on page 2-35)	Allows a number of measurements to be made by a specified instrument during a `sweep`*X* command. The results of the measurements are stored in the defined array.

Command	Description
trigcomp (on page 2-36)	Causes a trigger when an instrument goes in or out of compliance.
trigXg, trigXl (on page 2-37)	Monitors for a predetermined level of voltage, current, or time.
tstdsl (on page 2-40)	Deselects a test station.
tstsel (on page 2-40)	Enables or disables a test station.

Math operation commands

Command	Description
kfpabs (on page 3-1)	Takes a user-specified positive or negative value and converts it into a positive value that is returned to a specified variable.
kfpadd (on page 3-2)	Adds two real numbers and stores the result in a specified variable.
kfpdiv (on page 3-2)	Divides two real numbers and stores the result in a specified variable.
kfpexp (on page 3-3)	Supplies the base of natural logarithms (e) raised to a specified power and stores the result as a variable.
kfplog (on page 3-4)	Returns the natural logarithm of a real number to the specified variable.
kfpmul (on page 3-4)	Multiplies two real numbers and stores the result as a specified variable.
kfpneg (on page 3-5)	Changes the sign of a value and stores the result as a specified variable.
kfppwr (on page 3-6)	Raises a real number to a specified power and assigns the result to a specified variable.
kfpsqrt (on page 3-7)	Performs a square root operation on a real number and returns the result to the specified variable.
kfpsub (on page 3-8)	Subtracts two real numbers and stores their difference in a specified variable.

SMU commands

Command	Description
adelay (on page 4-1)	Specifies an array of delay points to use with `asweepX` command calls.
asweepX (on page 4-2)	Generates a waveform based on a user-defined forcing array (logarithmic sweep or other custom forcing commands).
avgX (on page 4-4)	Makes a series of measurements and averages the results.
bmeasX (on page 4-5)	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).
bsweepX (on page 4-7)	Supplies a series of ascending or descending voltages or currents and shuts down the source when a trigger condition is encountered.
devclr (on page 4-9)	Sets all sources to a zero state.
devint (on page 2-6)	Resets all active instruments in the system to their default states.
forceX (on page 4-11)	Programs a sourcing instrument to generate a voltage or current at a specific level.
getstatus (on page 4-12)	Returns the operating state of a specified instrument.
intgX (on page 4-14)	Performs voltage or current measurements averaged over a user-defined period (usually one ac line cycle).
limitX (on page 4-16)	Allows the programmer to specify a current or voltage limit other than the default limit of the instrument.
lorangeX (on page 4-17)	Defines the bottom autorange limit.
measX (on page 4-18)	Allows the measurement of voltage, current, or time.
mpulse (on page 4-20)	Uses a source-measure unit (SMU) to force a voltage pulse and measure both the voltage and current for exact device loading.

Command	Description
adelay (on page 4-1)	Specifies an array of delay points to use with `asweepX` command calls.
pulseX (on page 4-21)	Directs a SMU to force a voltage or current at a specific level for a predetermined length of time.
rangeX (on page 4-23)	Selects a range and prevents the selected instrument from autoranging.
rtfary (on page 2-27)	Returns the array of force values used during the subsequent voltage or frequency sweep.
segment_sweepX_list (on page 4-26)	Creates and returns up to a 4-segment linear sweep force table based on user-defined start, stop, and step values.
setauto (on page 4-27)	Re-enables autoranging and cancels any previous `rangeX` command for the specified instrument.
ssmeasX (on page 4-28)	Makes a series of readings until the change (delta) between readings is within a specified percentage.
sweepX (on page 4-29)	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.

PGU (pulse only) and PMU (pulse and measure) commands

In the LPT commands, the pulse-only module (4220-PGU) is referred to as VPU1, VPU2, and so on. The pulse-measure module (4225-PMU) is referred to as PMU1, PMU2, and so on. The 4210-CVU or 4215-CVU is referred to as CVU1, CVU2, and so on.

Note that the 4225-PMU and 4220-PGU support the PG2 commands.

Command	Description
arb_array (on page 6-3)	Used to define a full-arb waveform and name the file.
arb_file (on page 6-4)	Loads a waveform from an existing full-arb waveform file.
dev_abort (on page 6-4)	PGU, PMU. Programmatically ends (aborts) a test from within the user module that was started with the `pulse_exec` command.
devclr (on page 4-9)	Sets all sources to a zero state.
devint (on page 2-6)	Resets all active instruments in the system to their default states.
getstatus (on page 4-12)	Returns the operating state of a specified instrument.
pg2_init (on page 6-10)	Resets the pulse card to the specified pulse mode (standard, full arb, or Segment Arb) and its default conditions.
pmu_offset_current_comp (on page 6-11)	PMU. Collects offsets current constants from the 4225-PMU for offset compensation measurements.
PostDataDoubleBuffer (on page 6-11)	PMU. Posts PMU data retrieved from the buffer into the Clarius Analyze sheet (large data sets).
pulse_burst_count (on page 6-14)	For the burst mode, this command sets the number of pulses to output during a burst sequence.
pulse_chan_status (on page 6-15)	PMU. Used to determine how many readings are stored in the data buffer.
pulse_conncomp (on page 6-16)	PMU. Enables or disables connection compensation.
pulse_current_limit (on page 6-17)	Channel number of the pulse card: `1` or `2`
pulse_dc_output (on page 6-18)	Selects the dc output mode and sets the voltage level.
pulse_delay (on page 6-19)	Sets the delay time from the trigger to when the pulse output starts.
pulse_exec (on page 6-20)	PGU, PMU. Used to validate the test configuration and start test execution.
pulse_exec_status (on page 6-21)	PGU, PMU. Used to determine if a test is running or completed.
pulse_fall (on page 6-22)	Sets the fall transition time for the pulse output.
pulse_fetch (on page 6-24)	PMU. Retrieves enabled test data and temporarily stores it in the data buffer.

Command	Description
pulse_float (on page 6-28)	PMU. Sets the state of the floating relay for the given pulse instrument
pulse_halt (on page 6-29)	Stops all pulse output from the pulse card.
pulse_init (on page 6-30)	Resets the pulse card to the default settings for the pulse mode that is presently selected.
pulse_limits (on page 6-31)	PMU. Sets measured voltage and current thresholds at the DUT and sets the power threshold for each channel.
pulse_load (on page 6-32)	Sets the output impedance for the load (DUT).
pulse_meas_sm (on page 6-33)	PMU. Configures spot mean measurements.
pulse_meas_timing (on page 6-34)	PMU. Sets the measurement windows.
pulse_meas_wfm (on page 6-36)	PMU. Configures waveform measurements.
pulse_measrt (on page 6-37)	PMU. Returns pulse source and measure data in pseudo real time.
pulse_output (on page 6-38)	Sets the pulse output of a pulse card channel on or off.
pulse_output_mode (on page 6-39)	Sets the pulse output mode of a pulse card channel.
pulse_period (on page 6-40)	Sets the period for pulse output.
pulse_range (on page 6-41)	Sets a pulse card channel for low voltage (fast speed) or high voltage (slow speed).
pulse_ranges (on page 6-42)	PGU, PMU. Sets the voltage pulse range and voltage/current measure ranges.
pulse_remove (on page 6-44)	PGU, PMU. Removes a pulse channel from the test.
pulse_rise (on page 6-45)	Sets the rise transition time for the pulse card pulse output.
pulse_sample_rate (on page 6-46)	PMU. Sets the measurement sample rate.
pulse_source_timing (on page 6-47)	PGU, PMU. Sets the pulse period, pulse width, rise time, fall time, and delay time.
pulse_ssrc (on page 6-48)	Controls the high-endurance output relay (HEOR) for each output channel of the PGU.
pulse_step_linear (on page 6-50)	PGU, PMU. Configures the pulse stepping type.
pulse_sweep_linear (on page 6-53)	PGU, PMU. Configures the pulse sweeping type.
pulse_train (on page 6-56)	PGU, PMU. Configures the pulse card to output a pulse train using fixed voltage values.
pulse_trig (on page 6-57)	Selects the trigger mode (continuous, burst, or trigger burst) and initiates the start of pulse output or arms the pulse card.
pulse_trig_output (on page 6-58)	Sets the output trigger on or off.
pulse_trig_polarity (on page 6-59)	Sets the polarity (positive or negative) of the pulse card output trigger.
pulse_trig_source (on page 6-60)	Sets the trigger source.
pulse_vhigh (on page 6-62)	Sets the pulse voltage high level.
pulse_vlow (on page 6-63)	Sets the pulse voltage low value.
pulse_width (on page 6-65)	Sets the pulse width for pulse output.
rpm_config (on page 6-66)	PMU with 4225-RPM. Sends switching commands to the 4225-RPM.
seg_arb_define (on page 6-67)	Defines the parameters for a Segment Arb® waveform.
seg_arb_file (on page 6-69)	Used to load a waveform from an existing Segment Arb® waveform file.
seg_arb_sequence (on page 6-70)	PGU, PMU. Defines the parameters for a Segment Arb waveform pulse-measure sequence.
seg_arb_waveform (on page 6-73)	PGU, PMU. Creates a voltage segment waveform.
setmode (on page 6-74)	PMU. Sets the number of iterations for load-line effect compensation (LLEC) for the PMU. Also enables or disables offset current compensation.

CVU commands

Command	Description
adelay (on page 4-1)	Specifies an array of delay points to use with asweepX command calls.
asweepv (on page 5-3)	Does a dc voltage sweep using an array of voltage values.
bsweepX (on page 4-7)	Supplies a series of ascending or descending voltages or currents and shuts down the source when a trigger condition is encountered.
cvu_custom_cable_comp (on page 5-6)	Determines the delays needed to accommodate custom cable lengths.
devclr (on page 4-9)	Sets all sources to a zero state.
devint (on page 2-6)	Resets all active instruments in the system to their default states.
dsweepf (on page 5-8)	Performs a dual frequency sweep.
dsweepv (on page 5-10)	Performs a dual linear staircase voltage sweep.
forcev (on page 5-11)	Sets the dc bias voltage level.
getstatus (on page 5-12)	Returns parameters that describe the state of the 4210-CVU or 4215-CVU.
measf (on page 5-13)	Returns the frequency sourced during a single measurement.
meass (on page 5-14)	Returns the status referenced to a single measurement.
meast (on page 5-15)	Returns a timestamp referenced to a measurement or a system timer.
measv (on page 5-15)	Returns the dc bias voltage sourced during a single measurement.
measz (on page 5-16)	Makes an impedance measurement.
rangei (on page 5-17)	Selects an impedance measurement range.
rtfary (on page 4-25)	Returns the array of force values used during the subsequent voltage or frequency sweep.
setauto (on page 5-19)	Selects the automatic measurement range.
setfreq (on page 5-19)	Sets the frequency for the ac drive.
setlevel (on page 5-20)	Sets the voltage level of the ac drive.
setmode (on page 5-21)	Sets operating modes specific to the 4210-CVU or 4215-CVU.
smeasf (on page 5-23)	Returns the frequencies used for a sweep.
smeasfRT (on page 5-23)	Returns the sourced frequencies (in real time) for a sweep.
smeass (on page 5-24)	Returns the measurement status values for every point in a sweep.
smeast (on page 5-25)	Returns timestamps referenced to sweep measurements or a system timer.
smeastRT (on page 5-26)	Returns timestamps (in real time) referenced to sweep measurements or a system timer.
smeasv (on page 5-26)	Returns the dc bias voltages used for a sweep.
smeasvRT (on page 5-27)	Returns the sourced dc bias voltages (in real time) for a sweep.
smeasz (on page 5-28)	Performs impedance measurements for a sweep.
smeaszRT (on page 5-29)	Makes and returns impedance measurements for a voltage or frequency sweep in real time.
sweepf (on page 5-30)	Performs a frequency sweep.
sweepf_log (on page 5-32)	Performs a logarithmic frequency sweep using a 4215-CVU instrument. This is not available for the 4210-CVU.
sweepv (on page 5-33)	Performs a linear staircase dc voltage sweep.

Switch commands

Command	Description
addcon (on page 7-1)	Adds connections without clearing existing connections.
clrcon (on page 7-2)	Opens or de-energizes all device under test (DUT) pins and instrument matrix relays, disconnecting all crosspoint connections.
conpin (on page 7-2)	Connects pins and instruments.
conpth (on page 7-3)	Connects pins and instruments using a specific pathway.
cviv_config (on page 7-4)	Sends switching commands to the 4200A-CVIV Multi-Switch.
cviv_display_config (on page 7-5)	Configures the LCD display on the 4200A-CVIV Multi-Switch.
cviv_display_power (on page 7-6)	Sets the display state of the LCD display on the 4200A-CVIV.
delcon (on page 7-6)	Removes specific matrix connections.
devint (on page 2-6)	Resets all active instruments in the system to their default states.

LPT Library Status and Error codes

Error codes are displayed whenever an invalid parameter or configuration occurs. The messages associated with the error codes describe the error condition to help the user module programmer or user determine how to address the error. Once an error occurs, the response of the user module to the error depends on how the user module is programmed. If a user module does not have any error handling, an initial error could cause additional errors on following LPT commands.

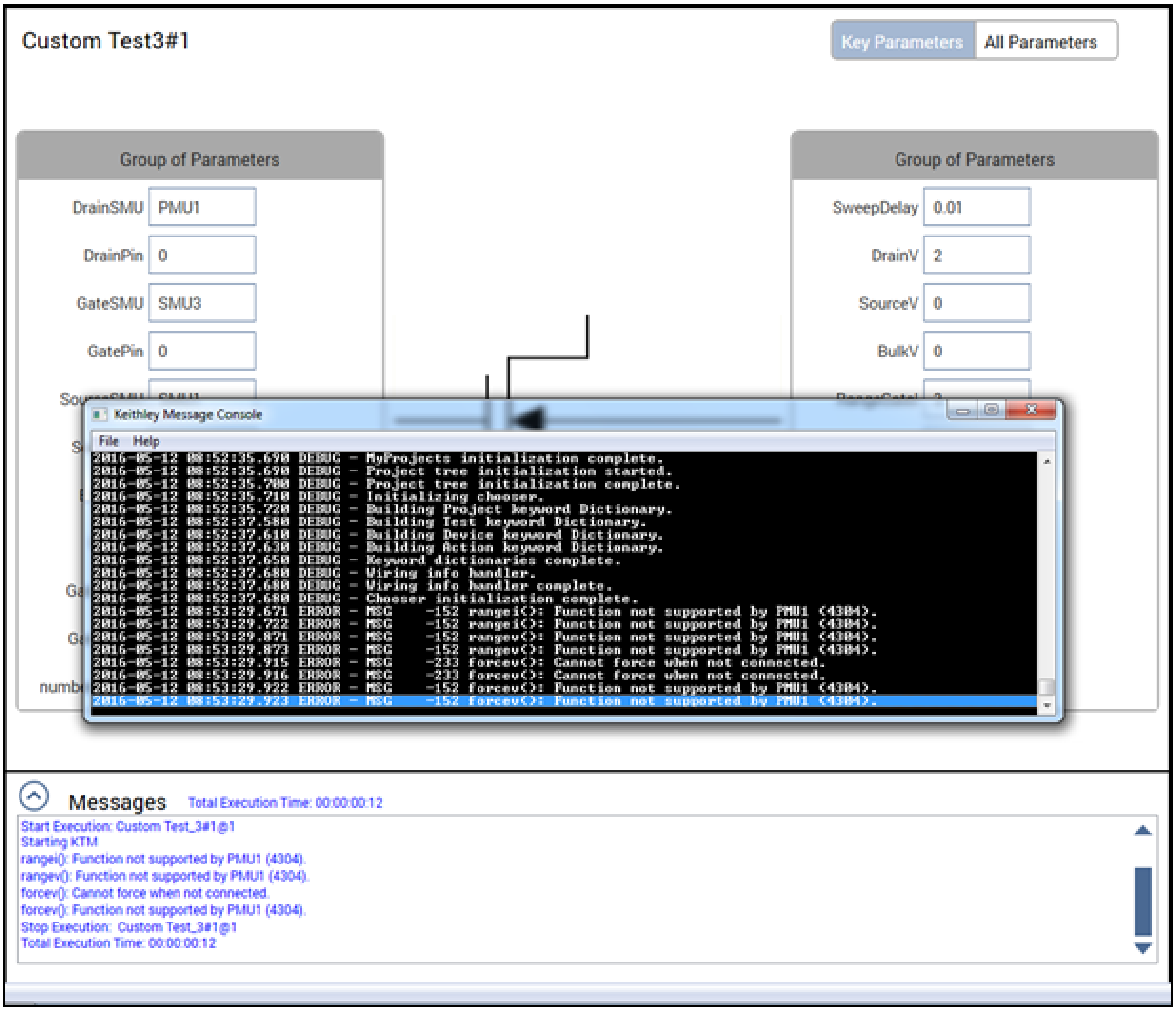
Library status and error codes are reported in Clarius in the message area.

Codes with positive values are statuses or updates. Codes with negative values are errors and warnings.

Each error code number is associated with a brief text explanation. However, many of the error texts are customized with specific information, such as a particular SMU or ID number. See Customized error texts (on page 1-9) for an explanation of the type of customized data.

In addition to error codes, some conditions may prevent a valid measurement condition. In these cases, the reported measurement value reports a condition. This is usually a large number with an exponent of 10^{22} or 10^{23}. See Large number reported readings and explanations (on page 1-15) for the conditions associated with these large numbers.

Figure 1: LPT error codes in the Clarius message areas



Customized error texts

Key	Explanation
%d	Signed decimal number; may be a parameter index or GPIB address
%g	Double value
%i	Signed decimal number
%s	String, such as "SMU1" or other test resource
%u	Unsigned integer
%04x	Hexadecimal number, 4 places
%08x	Hexadecimal number, 8 places

Code status or error titles

Code	Status or error titles
2802 to 2807	RPM: Invalid Configuration Requested
2801	RPM: Returned ID Error Response
2800	RPM: Command Response Timeout
2702	PMU: Temperature Within Normal Range
2701	PMU: High Temperature Limit Exceeded
1905	PMU: Measure Program Error
1904	PMU: Source Program Error
1902	PMU: Transmission to analog from digital error
1901	PMU: Handshake from analog to digital error
1900	PMU: DA Communication Timeout
400 to 402	PMU: Invalid Attributes in SW Command
100	LPTLib is executing function %s on instrument ID %d.
55	%s is no longer in thermal shutdown.
54	%s VXIBus device busy (command ID %04x). Timed out after %g seconds.
53	%s VXIbus transaction recovered after %u timeouts.
52	%s VXIbus transaction (command ID %04x) timed out after %g seconds.
51	Interlock reset.
50	Interlock tripped.
40	%s
24	Config %d-%d complete for %s (%d).
23	Config %d-%d starting for %s (%d).
22	Binding %s (%d) to driver %s.
21	Loading driver %s.
20	Preloading model code %08x (%s).
15	Executor started.
14	%s channel closed.
13	%s channel starting.
12	TAPI services shutting down.
11	Starting TAPI services.
9	System configuration complete.
8	System configuration starting.
4	System initialization complete.
1	The call was successful (no error).
0	The call was successful (no error).
−4	Too many instruments in configuration file %s.
−5	Memory allocation failure.
−6	Memory allocation error during configuration with configuration file %s.
−20	Command not executed because a previous error was encountered.
−21	Tester is in a fatal error state.
−22	Fatal condition detected while in testing state.
−23	Execution aborted by user.
−24	Too many arguments.
−25	%s is unavailable because it is in use by another test station.
−40	%s.

Code	Status or error titles
−87	Can not load library %s.
−88	Invalid configuration file %s.
−89	Duplicate IDs.
−90	Duplicate instrument addresses in configuration file %s.
−91	Duplicate instrument slots in configuration file %s.
−93	Unrecognized/missing interface for %s in configuration file %s.
−94	Unrecognized/missing PCI slot number for %s in configuration file %s.
−95	Unrecognized/missing GPIB address for %s in configuration file %s.
−96	GPIB Address out of range for %s was %i in configuration file %s.
−97	PCI slot number out of range for %s was %i in configuration file %s.
−98	Error attempting to load driver for model %s in configuration file %s.
−99	Unrecognized/missing instrument ID in configuration file %s.
−100	Invalid connection count, number of connections passed was %d.
−101	Argument #%d is not a pin in the current configuration.
−102	Multiple connections on %s.
−103	Dangerous connection using %s.
−104	Unrecognized instrument or terminal not connected to matrix, argument #%d.
−105	No pathway assigned to argument #%d.
−106	Path %d previously allocated.
−107	Not enough pathways to complete connection.
−108	Argument #%d is not defined by configuration.
−109	Illegal test station: %d.
−110	A ground connection MUST be made.
−111	Instrument low connection MUST be made.
−113	There are no switching instruments in the system configuration.
−114	Illegal connection.
−115	Operation not allowed on a connected pin: %d.
−116	No physical bias path from %s to %s.
−117	Connection cannot be made because a required bus is in use.
−118	Cannot switch to high current mode while sources are active.
−119	Pin %d in use.
−120	Illegal connection between %s and GNDU.
−121	Too many calls were made to trigXX.
−122	Illegal value for parameter #%d.
−124	Sweep/Scan measure table overflow.
−126	Insufficient user RAM for dynamic allocation.
−129	Timer not enabled.
−137	Invalid value for modifier.
−138	Too many points specified in array.
−139	An error was encountered while accessing the file %s.
−140	%s unavailable while slaved to %s.
−141	Timestamp not available because no measurement was made.
−142	Cannot bind, instruments are incompatible.
−143	Cannot bind, services unavailable or in use.
−152	Function not supported by %s (%d).
−153	Instrument with ID %d is not in the current configuration.
−154	Unknown instrument name %s.

Code	Status or error titles
−155	Unknown instrument ID %i.
−158	VXI device in slot %d failed selftest (mfr ID: %04x, model number: %04x).
−159	VME device with logical address %d is either non-VXI or non-functional.
−160	Measurement cannot be performed because the source is not operational.
−161	Instrument in slot %d has non-functional dual-port RAM.
−164	VXI device in slot %d statically addressed at reserved address %d.
−165	Service not supported by %s (%d).
−166	Instrument with model code %08x is not recognized.
−167	Invalid instrument attribute %s.
−169	Instrument %s is not in the current configuration.
−190	Ill-formed connection.
−191	Mode conflict.
−192	Instrument sense connection MUST be made.
−200	Force value too big for highest range %g.
−202	I-limit value %g too small for specified range.
−203	I-limit value %g too large for specified range.
−204	I-range value %g too large for specified range.
−206	V-limit value %g too large for specified range.
−207	V-range value %g too large for specified range.
−213	Value too big for range selection, %g.
−218	Safe operating area for device exceeded.
−221	Thermal shutdown has occurred on device %s.
−224	Limit value %g too large for specified range.
−230	V-limit value %g too small for specified range.
−231	Range too small for force value.
−233	Cannot force when not connected.
−235	C-range value %g too large for specified range.
−236	G-range value %g too large for specified range.
−237	No bias source.
−238	VMTR not allocated to make the measurement.
−239	Timeout occurred attempting measurement.
−240	Power Limited to 20 W. Check voltage and current range settings.
−250	IEEE-488 time out during data transfer for addr %d.
−252	No IEEE-488 interface in configuration.
−253	IEEE-488 secondary address %d invalid for device.
−254	IEEE-488 invalid primary address: %d.
−255	IEEE-488 receive buffer overflow for address %d.
−261	No SMU found, kelvin connection test not performed.
−262	SRU not responding.
−263	DMM not connected to SRU.
−264	GPIB communication problem.
−265	SRU not mechanically calibrated.
−266	Invalid SRU command.
−267	SRU hardware problem.
−268	SRU kelvin connection problem.
−269	SRU general error.
−270	Floating point divide by zero.

Code	Status or error titles
−271	Floating point log of zero or negative number.
−272	Floating point square root of negative number.
−273	Floating point pwr of negative number.
−280	Label #%d not defined.
−281	Label #%d redefined.
−282	Invalid label ID #%d.
−301	PCI ID read back on send error, slot.
−455	Protocol version mismatch.
−510	No command byte available (read) or SRQ not asserted.
−511	CAC conflict.
−512	Not CAC.
−513	Not SAC.
−514	IFC abort.
−515	GPIB timed out.
−516	Invalid function number.
−517	TCT timeout.
−518	No listeners on bus.
−519	Driver problem.
−520	Bad slot number.
−521	No listen address.
−522	No talk address.
−523	IBUP Software configuration error.
−524	No utility function.
−550	EEPROM checksum error in %s: %s.
−551	EEPROM read error in %s: %s.
−552	EEPROM write error in %s: %s.
−553	%s returned unexpected error code %d.
−601	System software internal error; contact the factory.
−602	Module load error: %s.
−603	Module format error: %s.
−604	Module not found: %s.
−610	Could not start %s.
−611	Network error.
−612	Protocol error.
−620	Driver load error. Could not load %s.
−621	Driver configuration function not found. Driver is %s.
−640	%s serial number %s failed diagnostic test %d.
−641	%s serial number %s failed diagnostic test %d with a fatal fault.
−650	Request to open unknown channel type %08x.
−660	Invalid group ID %d.
−661	Invalid test ID %d.
−662	Ill-formed list.
−663	Executor is busy.
−664	Invalid unit ID %d.
−701	Error configuring serial port %s.
−702	Error opening serial port %s.

Code	Status or error titles
−703	Call kspcfg before using kspsnd or ksprcv.
−704	Error reading serial port.
−705	Timeout reading serial port.
−706	Terminator not received before read buffer filled.
−707	Error closing serial port %s.
−801	Exception code %d reported from VPU in slot %d, channel %d.
−802	VPU in slot %d has reached thermal limit.
−803	Start and stop values for defined segmented arb violate minimum slew rate.
−804	Function not valid in the present pulse mode.
−805	Too many points specified in array.
−806	Not enough points specified in array.
−807	Function not supported by 4200-VPU.
−808	Solid state relay control values ignored for 4200-VPU.
−809	Time Per Point must be between %g and %g.
−810	Attempts to control VPU trigger output are ignored by the 4200-VPU.
−811	Measure range not valid for %s.
−812	WARNING: Sequence %d, segment %d. Cannot measure with PGUs/VPUs.
−820	PMU segment start value %gV at index %d does not match previous segment stop value of %gV.
−821	PMU segment stop time (%g) greater than segment duration (%g)
−822	PMU sequence error for entry %d. Start value %gV does not match previous stop value of %gV.
−823	Start and stop window was specified for PMU segment %d, but no measurement type was set.
−824	Measurement type was specified for PMU segment %d, but start and stop window is invalid.
−825	%s set to post to column %s. Cannot fetch data that was registered as real-time.
−826	Cannot execute PMU test. No channels defined.
−827	Invalid pulse timing parameters in PMU Pulse IV test.
−828	Maximum number of segments per PMU channel exceeded (%d).
−829	The sum of base and amplitude voltages (%gV) exceeds maximum (%gV) for present range.
−830	Pulse waveform configuration exceeded output limits. Increase pulse period or reduce amplitude or total time of pulsing.
−831	Maximum number of samples per channel (%d) exceeded for PMU%d-CH%d.
−832	Pulse slew rate is too low. Increase pulse amplitude or reduce pulse rise and fall time.
−833	Invalid trigger source for PIV test.
−834	Invalid pulse timing parameters.
−835	Using the specified sample rate of %g samples/s, the time (%g) for sequence %d is too short for a measurement.
−836	WARNING: Sequence %d, segment %d is attempting to measure while solid state relay is open. Disabling measurement.
−837	No RPM connected to channel %d of PMU in slot %d.
−838	Timing parameters specify a pulse that is too short for a measurement using %g samples/s.
−839	Timing parameters contain measurement segments that are too short to measure using %g samples/s.
−840	SSR cannot be opened when using RPM ranges. Please change SSR array to enable relay or select PMU measure range.
−841	WARNING: SSR is open on segment immediately preceding sequence %d. Measurement will be invalid for 25 μs while relay settles.
−842	This test has exceeded the system power limit by %g watts.

Code	Status or error titles
−843	Step size of %g is not evenly divisible by 10 ns.
−844	Invalid combination of start %g1, stop %g2 and step %g3.
−845	No pulse sweeper was configured - Test will not run.
−846	Maximum Source Voltage Reached: Requested voltage across DUT resistance exceeds maximum voltage available.
−847	Output was not configured - Test will not run.
−848	Sweep step count mismatch for the sweeping channels. All sweeping channels must have same # of steps.
−849	ILimit command is not supported for RPM in slot %d, channel %d.
−850	Sample Rate mismatch. All channels in test must have the sample rate.
−851	Invalid PxU stepper/sweeper configuration.
−900	Environment variable KI_PRB_CONFIG is not set. The prober drivers will be inaccessible.
−901	Environment variable KI_PRB_CONFIG contains an invalid path. The prober drivers will be inaccessible.
−902	Prober configuration file not found. File was %s. The prober drivers will be inaccessible.
−903	Unable to copy the prober configuration %s to %s. The prober driver many not be available.
−10000 to −20000	User Module (UTM) error codes. Refer to user module description (help) for details.

Large number reported readings and explanations

Measurement value	Condition
1.0000E+22 or 10.0000E+21	SMU is in range compliance, where the reading is at the maximum of a fixed range. See “Compliance limits” in the *Model 4200A-SCS Source-Measure Unit (SMU) User's Manual* for details.
5.0000E+22	SMU is in range compliance while autoranging, where the reading is not at the maximum voltage or current range.
7.0000E+22	SMU in real compliance. See “Compliance limits” in the *Model 4200A-SCS Source-Measure Unit (SMU) User's Manual* for details.
1.0000E+23	Measurement aborted or cannot be performed, such as when using an LPT command to make a measurement if the SMU output is not enabled.

LPT library and Clarius interaction when using UTMs

ITMs and UTMs are typically independent. However, an ITM and a UTM are not independent if the UTM occurs before an ITM in the project and the UTM configures a switch matrix. Under these conditions, the following occur:

- Clarius assumes that the ITM depends on the UTM-created switch configuration.
- Clarius maintains the UTM-created switch configuration during execution of the ITM.

Clarius actions affected by ITM and UTM sequence

Test sequence in the project	Clarius action
A UTM precedes an ITM	Before the ITM executes, the `devint` command initializes all devices, except for the switch matrix (the switch configuration is preserved to run the subsequent ITM).
A UTM precedes a UTM	No initialization operations occur.
An ITM precedes an ITM	No LPT library calls occur.
An ITM precedes a UTM	Before the UTM executes, the `devint` command initializes all devices, including the switch matrix.

Section 2

LPT commands for general operations

In this section:

LPT commands for general operations

General operation commands include commands to control timing, execution, communications, and test status.

clrscn

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

Usage

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

Details

When a single `sweepX` command is used in a test sequence, there is no need to program a `clrscn` command because the `execut` command clears the table.

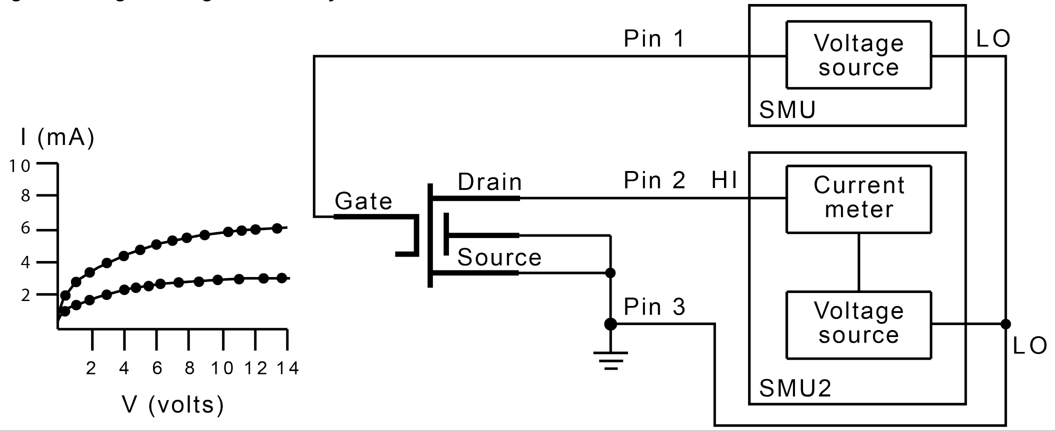
The `clrscn` command is only required when multiple sweeps and multiple sweep measurements are used in a single test sequence.

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 sweep*X* 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

execut (on page 2-8)
sweepX (on page 4-29)

clrtrg

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

Usage

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

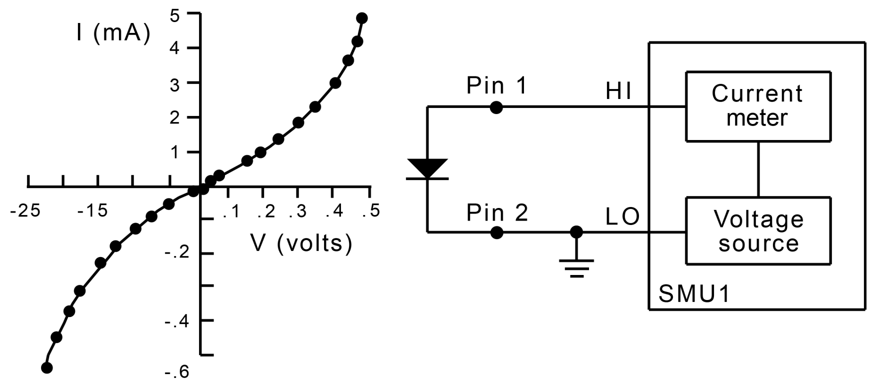
Details

The search*X*, sweep*X*, asweep*X*, or bsweep*X* command, each with different voltage or current levels, may be used repeatedly within a command if each is separated by a clrtrg command.

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 4-2)
bsweepX (on page 4-7)
searchX (on page 2-29)
sweepX (on page 4-29)
trigXg, trigXl (on page 2-37)

delay

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

Usage

```
int delay(long n);
```

n	The duration of the delay in milliseconds

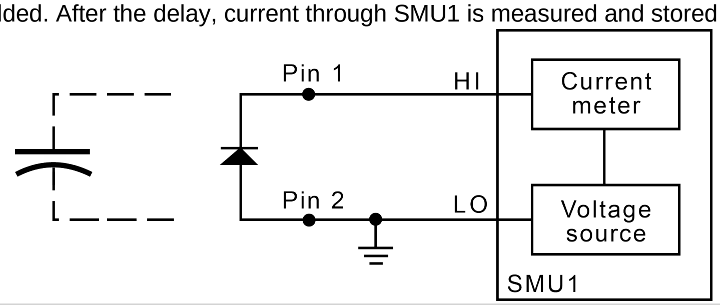
Details

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 2-26)

devint

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

Usage

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

Details

Resets all active instruments, including the 4200A-CVIV, 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.

The `devint` command is implicitly called by the `execut` and `tstdsl` commands.

To abort a running `pulse_exec` pulse test, see `dev_abort`.

`devint` does the following:

1. Clears all sources by calling `devclr`.
2. Clears the matrix crosspoints by calling `clrcon`.
3. Clears the trigger tables by calling `clrtrg`.
4. Clears the sweep tables by calling `clrscn`.
5. Resets GPIB instruments by sending the string defined with `kibdefint`.
6. Resets the active instrument cards.

Instrument cards are reset in the following order:

1. SMU instrument cards
2. CVU instrument cards
3. Pulse instrument cards (4225-PMU or 4220-PGU)

The SMUs return to the following states:

- 100 µA and 10 V ranges
- Autorange on
- Voltage source
- 0 V dc bias

The 4210-CVU or 4215-CVU returns to the following states:

- 30 mV_{RMS} ac signal
- 0 V dc bias
- 100 kHz frequency
- Autorange on
- Cable length compensation set to 0 m
- Open/Short/Load compensation disabled

The 4225-PMU or 4220-PGU returns to the following states:

- The pulse mode is maintained. For example, if the pulse card is in Segment Arb mode, it is still in Segment Arb mode after the `devint` process is complete.
- 5 V and 10 mA ranges
- If in pulse mode:
  - Period of 1 µs
  - Transition times (rise and fall) of 100 ns
  - Width of 500 ns
  - Voltage high and low of 0 V
  - Load of 50 Ω
- If in segmented arb mode, Start Voltage is 0 V
- If in arbitrary waveform mode, Table Length is 100

Also see

clrcon (on page 7-2)
clrscn (on page 2-2)
clrtrg (on page 2-3)
dev_abort (on page 6-4)
devclr (on page 4-9)
kibdefint (on page 2-16)

disable

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

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 2-8)

enable

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

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.

Also see

disable (on page 2-8)

execut

This command causes the system to wait for the preceding test sequence to be executed.

Usage

```
int execut(void);
```

Details

This command waits for all previous LPT library commands to complete and then sends the `devint` command.

Also see

devint (on page 2-6)

getinstattr

This command returns configured instrument attributes.

Usage

```
int getinstattr(int instr_id, char *attrstr, char *attrvalstr);
```

instr_id	The instrument identification code of the LPT library instrument
attrstr	The instrument attribute name string
attrvalstr	The value string of the requested attribute; see **Details**

Details

All instruments in the system configuration have specific attributes. GPIB address is an example of an attribute. The values of these attributes change as the system configuration is changed. Therefore, by getting the values of key attributes at run time, user modules can be developed in a configuration-independent manner. Given an instrument identification code and an attribute name string, this module returns the specified attribute value string.

If the attribute value string exists, the returned string will match one of the values shown in the Attribute value string column of the following table. If the requested attribute does not exist, the `attrvalstr` parameter is set to a null string.

Possible values for the `getinstattr` parameters are listed in the following table.

`getinstattr` parameter values

Instrument identification code	Attribute name string	Attribute value string
GPI*x*	GPIBADDR	1 to 30
	MODELNUM	GPI 2-terminal GPI 4-terminal
CMTR*x*	GPIBADDR	1 to 30
	MODELNUM	KI82 KI590 KI595 KI4284 KI4294
PGU*x*	GPIBADDR	1 to 30
	MODELNUM	KI3401 KI3402 HP8110 HP81110
SMU*x*	MODELNUM	KI4200 KI4210
MTRX1	MODELNUM	KI707 KI708
TF1	MODELNUM	KI8006 KI8007
	NUMOFPINS	12 72

`getinstattr` parameter values

Instrument identification code	Attribute name string	Attribute value string
PRBR1	NUMOFPINS	2 to 72
	MODELNUM	FAKE CC12K CM500 MANL MM40 PA200 MPI
CVU*x*	MODELNUM	KICVU4210
VPU*x* VPU*x*CH1 VPU*x*CH2	MODELNUM	KIVPU4220
PMU*x* PMU*x*CH1 PMU*x*CH2	MODELNUM	KIPMU4225
CVIV*x*	MODELNUM	KICVIV
GNDU	MODELNUM	GNDU

Also see

None

getinstid

This command returns the instrument identifier (ID) from the instrument name string.

Usage

```
int getinstid(char *instr_name, int *instr_id);
```

instr_name	The instrument name string
instr_id	The instrument identification code

Also see

None

getinstname

This command returns the instrument name string from the instrument identifier (ID).

Usage

```
int getinstname(int *instr_id, char *inst_name);
```

instr_id	The instrument identification code
inst_name	The returned instrument name string

Also see

None

GetKiteCycle

This command returns the present Clarius cycle number.

Usage

```
int GetKiteCycle(void);
```

Details

If no cycling is active, `GetKiteCycle` returns `1`.

Also see

None

GetKiteDevice

This command returns the device that Clarius is presently testing.

Usage

```
int GetKiteDevice(void);
```

Example

```
char strVal[25];
GetKiteSubsite(strVal, 25);
printf("KiteSubsite = %s\n", strVal);
GetKiteDevice(strVal, 25);
printf("KiteDevice = %s\n", strVal);
GetKiteTest(strVal, 25);
printf("KiteTest = %s\n", strVal
```

A user test module (UTM) that returns the present subsite, device, and test.

Also see

None

GetKiteSite

This command returns the site number for the site that Clarius is presently testing.

Usage

```
int GetKiteSite(void);
```

Details

The site number is an integer that designates the relative numerical position of the presently tested site in the prober site-visit sequence. However, users normally correlate Clarius site numbers with prober site coordinates. `GetKiteSite` does not return prober site coordinates.

For more information about Clarius site numbers, refer to “Configure sites” in the *Model 4200A-SCS Clarius User’s Manual*.

Also see

None

GetKiteSubsite

This command returns the subsite number for the site that Clarius is presently testing.

Usage

```
int GetKiteSubsite(void);
```

Example

```
char strVal[25];
GetKiteSubsite(strVal, 25);
printf("KiteSubsite = %s\n", strVal);
GetKiteDevice(strVal, 25);
printf("KiteDevice = %s\n", strVal);
GetKiteTest(strVal, 25);
printf("KiteTest = %s\n", strVal
```

A user test module (UTM) that returns the present subsite, device, and test.

Also see

None

GetKiteTest

This command returns the test that Clarius is presently testing.

Usage

```
int GetKiteTest(void);
```

Example

```
char strVal[25];
GetKiteSubsite(strVal, 25);
printf("KiteSubsite = %s\n", strVal);
GetKiteDevice(strVal, 25);
printf("KiteDevice = %s\n", strVal);
GetKiteTest(strVal, 25);
printf("KiteTest = %s\n", strVal
```

A user test module (UTM) that returns the present subsite, device, and test.

Also see

None

getlpterr

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

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 2-6)

imeast

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

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.

Also see

None

inshld

Provided for compatibility with Model S400 LPT library.

Usage

```
int inshld(void);
```

Also see

None

kibcmd

This command enables universal, addressed, and unaddressed GPIB bus commands to be sent through the GPIB interface.

Usage

```
int kibcmd(unsigned int timeout, unsigned int numbytes, char* cmdbuffer);
```

timeout	The timeout for transfer in 100 ms units (for example, a timeout of 40 = 4.0 s)
numbytes	The number of bytes in *cmdbuffer* to send with the ATN line asserted
cmdbuffer	The array that contains the bytes to transfer over the GPIB interface

Details

These commands can consist of any command that is valid with the ATN line asserted, such as `DCL`, `SDC`, and `GET`. The following table lists these GPIB commands.

`kibcmd` does the following:

1. Asserts attention (ATN).
2. Sends byte string (command buffer).
3. De-asserts ATN.

GPIB command list

GPIB command	Data byte (Hex)	Comments
Universal		
LLO (local lockout)	11	Locks out front-panel controls.
DCL (device clear)	14	Returns instrument to default conditions.
SPE (serial poll enable)	18	Enables serial polling.
SPD (serial poll disable)	19	Disables serial polling.
Addressed		
SDC (selective device clear)	04	Returns instrument to default conditions.
GTL (go to local)	01	Sends go to local.
GET (group execute trigger)	08	Triggers instrument for reading.
Unaddressed		
UNL (unlisten)	3F	Removes all listeners from GPIB bus.
UNT (untalk)	5F	Removes any talkers from GPIB bus.
LAG (listen address group)	20 to 3E	Place instrument at this primary address (0 through 30) in listen mode.
TAG (talk address group)	40 to 5E	Place instrument at this primary address (0 through 30) in talk mode.
SCG (secondary command group)	60 to 7E	Place instrument at this secondary address (0 through 30) in listen mode.

Example

```
int status;
char GPIBtrigger[5] = {0x3F, 0x2F, 0x08, 0x3F, 0x00};
/* Unlisten = 3F (UNL) */
/* Listen address = 32 + 15 = 2F */
/* Group Execute Trigger (GET) = 08 */
/* UNL */
/* Terminate string with NULL */
.
.
.
status = kibcmd(30, strlen(GPIBtrigger),GPIBtrigger);
/* Use 3s timeout */
```

This example illustrates how the `kibcmd` command could be used to issue a GPIB bus trigger command to a GPIB instrument located at address 15.

Also see

None

kibdefclr

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

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, a timeout of 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 `kibdefclr` copies parameters into a data structure within the tester memory. These data structures are allocated dynamically. After the execution of the command buffer using `execut`, these tables are cleared. 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. Once defined, these strings remain in effect until the `execut` statement is processed.

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

bsweepX (on page 4-7)
devclr (on page 4-9)
devint (on page 2-6)
execut (on page 2-8)
kibdefint (on page 2-16)

kibdefdelete

This command deletes all command definitions previously made with the `kibdefclr` (Keithley GPIB define device clear) and `kibdefint` (Keithley GPIB define device initialize) commands.

Usage

```
int kibdefdelete(void);
```

Details

Once this command is issued, any previous definitions made using `kibdefclr` or `kibdefint` will no longer occur at `devint` or `devclr` time.

You can override this command by re-issuing the `kibdefint` and `kibdefclr` commands.

Also see

None

kibdefint

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

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 (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
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 `kibdefclr` copies parameters into a data structure within the tester memory. These data structures are allocated dynamically. After the execution of the command buffer using `execut`, these tables are cleared. 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. Once defined, these strings remain in effect until the `execut` statement is processed.

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

bsweepX (on page 4-7)
devclr (on page 4-9)
devint (on page 2-6)
execut (on page 2-8)
kibdefclr (on page 2-15)

kibrcv

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

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 size of the buffer that receives data; 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.

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 (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)
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.

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 (1 to 30; 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 serial poll status byte returned by the device presently being polled

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), else *serial_poll_byte* = 0 (indicating error; device not SRQing).
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 2-20)

kibsplw

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

Usage

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

pri_addr	The primary address of the instrument (2 to 31)
sec_addr	The secondary address of the instrument (1 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 2-19)

kspcfg

This command configures and allocates a serial port for RS-232 communications.

Usage

```
int kspcfg(int port, int baud, int databits, int parity, int stopbits, int
    flowctl);
```

port	The RS-232 port to be used; only port 1 is supported
baud	The transmission rate to be used; valid rates are 2400, 4800, 9600, 14400, and 19200 baud
databits	The number of data bits to be used; valid inputs are 7 or 8 bits
parity	Determines whether or not parity bits will be transmitted; valid inputs are: `0` (no parity), `1` (odd parity), or `2` (even parity)
stopbits	Sets the number of stop bits to be transmitted; `1` or `2`
flowctl	Determines the type of flow control to be used: `0` (no flow control), `1` (XON/XOFF flow control), or `2` (hardware)

Details

Port 1 must not be allocated to another program or utility when using the ksp (Keithley Serial Port) commands.

- The databits, parity, stopbits, and flowctl settings must match those on the instrument or device that you wish to control.
- Using a flow control setting of 0 may result in buffer overruns if the device or instrument that you are controlling has a high data rate.
- If you use a flow-control setting of 2 (hardware), you must make sure that the RS-232 cable has enough wires to handle the RTS/CTS signals.

Example

```
int status;
.
.
.
status = kspcfg(1, 19200, 8, 1, 1, 1);/* port 1, 19200 baud,
   8 bits, odd parity,
   1 stop bit, and
   xon-xoff flow ctl */
```

This example uses `kspcfg` to set port 1 to 19200 baud, 8 data bits, odd parity, 1 stop bit, and XON/XOFF flow control.

Also see

None

kspdefclr

This command defines a device-dependent character string sent to an instrument connected to a serial port.

Usage

`int kspdefclr(int port, double timeout, double delay, int buffsize, char *buffer);`

port	The RS-232 port to be used; only port 1 is supported; this port must have been configured for communications with the `kspcfg` command
timeout	The serial communications timeout (0 s to 600 s)
delay	The amount of time to delay after sending the string to the serial device (0 s to 600 s)
buffsize	The length of the string to send to the serial device
buffer	A character string that contains the data to send to the serial device

Details

This string is sent during the normal tester `devclr` process. It ensures that if the tester is calling `devclr` internally, any device connected to the configured serial port will be cleared with the given string.

Before issuing this command, you must configure the serial port using the `kspcfg` command.

- The commands sent to the serial device are issued in the order in which they were defined using the `kspdefclr` command.
- The `kspdefdelete` command can be used to delete any previous definitions.
- The `kspdefclr` and `kspdefint` command strings are sent before normal (for example, a SMU) instrument `devclr` and `devint` execution.

Also see

kspcfg (on page 2-20)

kspdefdelete

This command deletes all command definitions previously made with the `kspdefclr` (Keithley Serial Define Device Clear) and `kspdefint` (Keithley Serial Define Device Initialize) commands.

Usage

```
int kspdefdelete( void );
```

Details

Once this command is issued, any previous definitions made using `kspdefclr` or `kspdefint` will no longer occur at `devint` or `devclr` time.

You can override this command by re-issuing the original `kspdefint` and `kspdefclr` commands.

Also see

None

kspdefint

This command defines a device-dependent character string sent to an instrument connected to a serial port.

Usage

```
int kspdefint(int port, double timeout, double delay, int buffsize, char *buffer);
```

port	The RS-232 port to be used; only port 1 is supported; this port must have been configured for communications with the `kspcfg` command
timeout	The serial communications timeout (0 s to 600 s)
delay	The amount of time to delay after sending the string to the serial device (0 s to 600 s)
buffsize	The length of the string to send to the serial device
buffer	A character string that contains the data to send to the serial device

Details

This string is sent during the normal tester `devint` process. It ensures that if the tester is calling `devint` internally, any device connected to the configured serial port will be cleared with the given string.

Before issuing this command, you must configure the serial port using the `kspcfg` command.

- The commands sent to the serial device are issued in the order in which they were defined using the `kspdefclr` command.
- The `kspdefdelete` command can be used to delete any previous definitions.
- The `kspdefclr` and `kspdefint` command strings are sent before normal (for example, a SMU) instrument `devclr` and `devint` execution.

Also see

kspcfg (on page 2-20)

ksprcv

This command reads data from an instrument connected to a serial port.

Usage

```
int ksprcv(int port, char terminator, double timeout, int
   rcvsize, int *rcv_len, char *rcv_buffer);
```

port	The RS-232 port to be used; only port 1 is supported; this port must have been configured for communications with the `kspcfg` command
terminator	The ASCII terminator for the received data; this character is used to terminate the read
timeout	The serial communications timeout: 0 s to 600 s
rcvsize	The physical buffer size; this is used to control the maximum number of characters that can be read from the device
rcv_len	The actual number of characters read from the device; this value is returned to the `ksprcv` command by the software
rcv_buffer	A character array in which to store the data returned from the serial device

Also see

kspcfg (on page 2-20)

kspsnd

This command sends a device-dependent command to an instrument attached to a RS-232 serial port.

Usage

```
int kspsnd( int port, double timeout, int cmdlen, char *cmd);
```

port	The RS-232 port to be used; only port 1 is supported; this port must have been configured for communications with the `kspcfg` command
timeout	The serial communications timeout: 0 s to 600 s
cmdlen	The number of characters that you are sending out the serial port
cmd	The character array containing the data that you want sent out of the serial port

Also see

None

PostDataDouble

This command posts double-precision floating-point data from memory into the Clarius Analyze sheet.

Usage

`int PostDataDouble(char *ColName, double *array);`

ColName	Column name for the data array in the Clarius Analyze sheet
array	An array of data values for the Clarius Analyze sheet

Pulsers

4225-PMU

Pulse mode

Standard and Segment Arb

Details

You can use the `PostDataDouble` command to post double-precision floating-point data into the Clarius Analyze sheet. Up to 65,535 points (rows) of data can be posted into the Analyze sheet. These commands are used after one measurement is finished and a data value is assigned to the corresponding output variable.

NOTE

If you do not need to analyze or manipulate the test data before posting it into the Analyze sheet, you can use a `smeasXRT` command for CVUs or `pulse_measrt` for PMUs.

Example

```
// Code to configure the PMU test here
// Start the test (no analysis)
pulse_exec(0);
// While loop (continues while test is still running), with delay
// (30 ms)
while(pulse_exec_status(&elapsedt) == 1)
{
   Sleep(30);
}
// Retrieve V and I data (no timestamp or status)
status = pulse_fetch(PMU1, 1, 0, 100, Vmeas, Imeas, NULL, NULL);
// Separate V & I measurements for high (amplitude) and
// low (base)
for (i = 0; i<100; i++)
{
   VmeasHi_sheet[i] = Vmeas[2*i];
   ImeasHi_sheet[i] = Imeas[2*i];
   VmeasLo_sheet[i] = Vmeas[2*i+1];
   ImeasLo_sheet[i] = Imeas[2*i+1];
   PostDataDouble("DrainVmeas", VmeasHi_sheet[i]);
   PostDataDouble("DrainImeas", ImeasHi_sheet[i]);
}
```

Posts spot mean measurement data into the Clarius Analyze sheet. This example assumes that a PMU spot mean test is configured to perform 100 (or more) voltage and current measurements for pulse high and low. Use `pulse_meas_sm` to configure the spot mean test. The code: ■ Starts the configured test. ■ Uses a while loop to allow the spot mean test to finish. ■ Retrieves voltage and current readings (100 points) from the buffer. ■ Separates the voltage and current readings for high (amplitude) and low (base). ■ Posts the high measurement data into the Clarius Analyze sheet. Low measurement data is not posted into the sheet.

Also see

"Enabling real-time plotting for UTMs" in *Model 4200A-SCS KULT and KULT Extension Programming*
PostDataDoubleBuffer (on page 6-11)
pulse_fetch (on page 6-24)
pulse_meas_sm (on page 6-33)
pulse_measrt (on page 6-37)
smeasfRT (on page 5-23)
smeastRT (on page 5-26)
smeasvRT (on page 5-27)
smeaszRT (on page 5-29)

PostDataInt

This command posts an integer-type point from memory to the Clarius Analyze sheet in the user test module and plots it on the graph.

Usage

```
PostDataInt(char *variableName, int *variableValue);
```

variableName	The variable name
variableValue	The value of the variable to be transferred

Details

The first parameter is the variable name, defined as `char *`. For example, if the output variable name is `DrainI`, then `DrainI` (with quotes) is first parameter.

The second parameter is the value of the variable to be transferred. For example, if `DrainI[10]` is transferred, then you call `PostDataInt("DrainI", DrainI[10])`.

Also see

None

PostDataString

This command transfers a string from memory into the Clarius Analyze sheet in the user test module and plots it on the graph.

Usage

`PostDataString(char *variableName, char *variableValue);`

variableName	The variable name
variableValue	The value of the variable to be transferred

Details

The first parameter is the variable name. For example, if the output variable name is `DrainI`, then `DrainI` (with quotes) is first parameter.

The second parameter is the value of the variable to be transferred. For example, if `DrainI[10]` is transferred, then you call `PostDataString("DrainI", DrainI[10])`.

Also see

None

rdelay

This command sets a user-programmable delay.

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 2-5)

rtfary

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

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 used with the `bsweepX`, `sweepX`, or `searchX` commands, 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.

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

Example

Refer to the examples for the `smeasX` and `sweepX` commands.

Also see

smeasX (on page 2-35)
sweepX (on page 4-29)

savgX

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

Usage

```
int savgi(int instr_id, double *result, long count, double delay);
int savgv(int instr_id, double *result, long count, double delay);
```

instr_id	The instrument identification code of the measuring instrument
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.

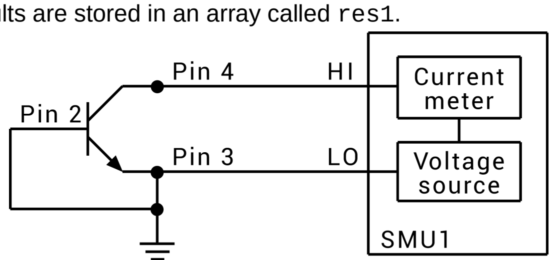
The `savgX` command sets up the new scan table entry to make an averaging measurement. The measurement results are stored in the array specified by the *`result`* parameter. Each time a measurement scan is made, a new measurement result is stored at the next location in the *`result`* array. If the scan table is not cleared, performing multiple sweeps will continue adding new measurement results to the end of the array. Make sure the *`result`* array is large enough to hold all measurements made before the scan table is cleared. The scan table is cleared by an explicit call to the `clrscn` command or implicitly when the `devint` or `execut` 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.

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 2-2)
devint (on page 2-6)

scnmeas

This command makes a single measurement on multiple instruments at the same time.

Usage

```
int scnmeas(void);
```

Details

This command behaves like a single point sweep. It makes a single measurement on multiple instruments at the same time. Any forcing or delaying must be done before calling `scnmeas`.

`smeasX`, `sintgX`, or `savgX` must be used to set up result arrays just as is done for a sweep call. Each call to `scanmeas` adds one element to the end of each array.

Calls to `scnmeas` may be mixed with calls to `sweepX`, and all results are appended to the result arrays in the same way multiple `sweepX` calls behave.

Also see

savgX (on page 2-27)
sintgX (on page 2-34)
smeasX (on page 2-35)

searchX

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

Usage

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

instr_id	The instrument identification code of the sourcing instrument
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

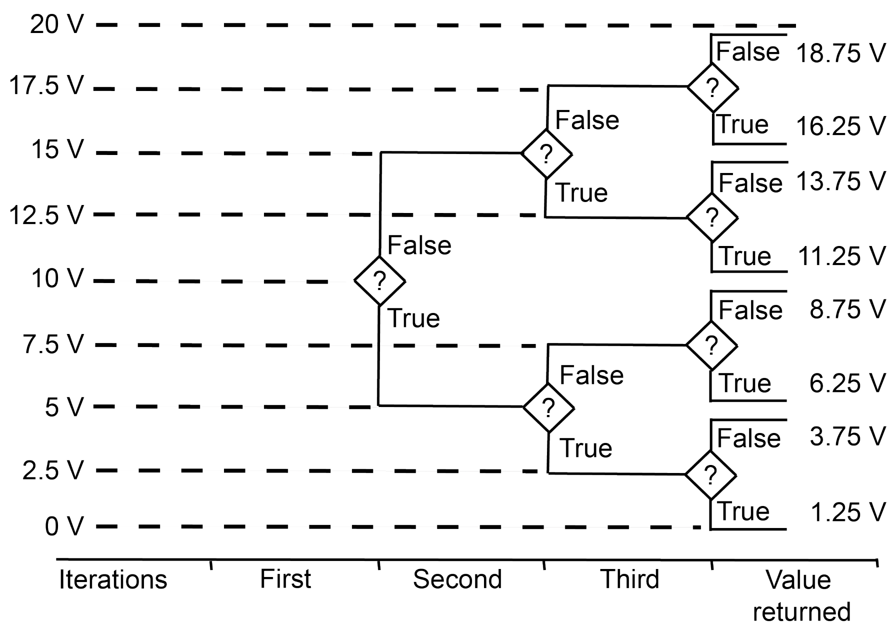
The `trigXg` or `trigXl` command must be used with the `searchX` command. Triggers and the `searchX` command together initiate a search operation consisting of a series of steps referred to as iterations. During each iteration, the following events occur:

- 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.



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:

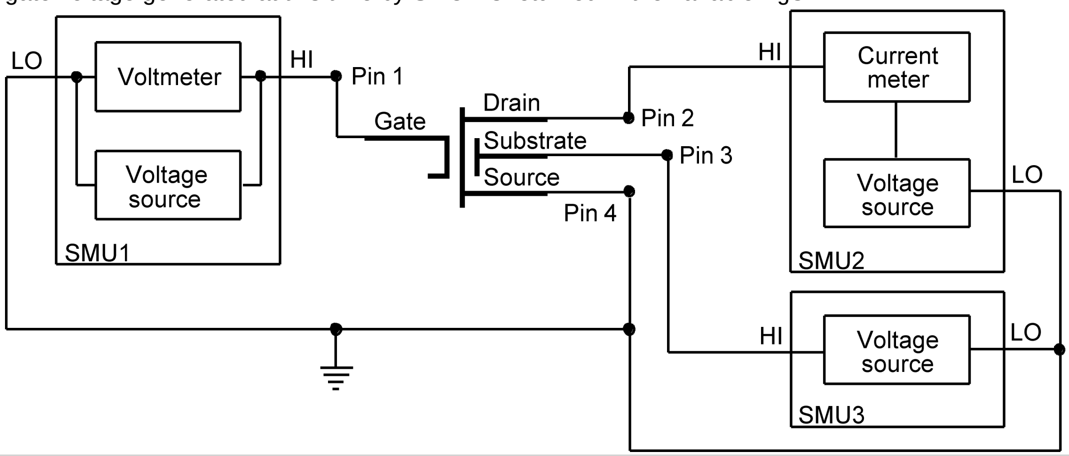
$$\frac{20}{2^{(16+1)}} = 1.2\text{ mV}$$

Changing the source mode of the SMU can modify the measure range. If the sourcing mode is changed from voltage to current sourcing (or from current to voltage sourcing), the measure range may be changed to minimize variations in the SMU output level. See rangeX (on page 4-23) for recommended command order.

Example

```
double ssbiasv, vgs1, vds1;
.
conpin(SMU1, 1, 0);
conpin(SMU2, 2, 0);
conpin(SMU3, 3, 0);
conpin(GND, 4, 0);
trigig(SMU2, +l.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

setmode

This command sets instrument-specific operating mode parameters.

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; see **Details**
value	The specified value of the operating parameter

Details

The `setmode` command allows you to control certain instrument-specific operating characteristics.

A special instrument ID named `KI_SYSTEM` is used to set operating characteristics of the system.

The following table describes `setmode` *`modifier`* parameters that are supported for `KI_SYSTEM`.

modifier	***value***	**Comment**

`KI_TRIGMODE`	`KI_MEASX`
	`KI_INTEGRATE`
	`KI_AVERAGE`
	`KI_ABSOLUTE`
	`KI_NORMAL`
		Redefines all existing triggers to use a new method of measurement.

`KI_AVGNUMBER`	<*value*>	Number of readings to make when `KI_TRIGMODE` is set to `KI_AVERAGE`.
`KI_AVGTIME`	<*value*> (in units of seconds)	Time between readings when `KI_TRIGMODE` is set to `KI_AVERAGE`.

The following `KI_SYSTEM` *modifier* parameters are accepted, but do no operations in the 4200A-SCS. They are included for compatibility so that existing S530 or S600 programs that use `setmode` can be ported to the 4200A-SCS without generating errors.

- `KI_MX_DEFMODE`
- `KI_HICURRENT`
- `KI_CC_AUTO`
- `KI_CC_SRC_DLY`
- `KI_CC_COMP_DLY`
- `KI_CC_MEAS_DLY`

The following `setmode` *modifier* parameters are supported for SMU instruments.

modifier	***value***	**Comment**
KI_INTGPLC	<*value*> (in units of line cycles)	Specifies the integration time the SMU will use for the `intg`*X* and `sintg`*X* commands. The default `devint` value is 1.0. The valid range is 0.01 to 10.0.
KI_AVGMODE	KI_MEASX KI_INTEGRATE	Controls what kind of readings are taken for `avgX` calls. The devint default value is `KI_MEASX`. When `KI_INTEGRATE` is specified, the integration time used is that specified by the `KI_INTGPLC` `setmode` call.
KI_DELAY_FACTOR	<*value*>	This factor scales the internal delay times used by the SMU. A value larger than one increases the delays; a value less than one decreases the delays. A minimum delay is enforced by the SMU. This command should not be used when setting the SMU speed to FAST, NORMAL, or QUIET modes; the delay factor is set internally by these modes, so changing the value while using one of the predefined modes corrupts the speed settings or the delay factor.
KI_LIM_INDCTR	Any	Controls the measure value that is returned if the SMU is at its programmed limit. The `devint` default is SOURCE_LIMIT (7.0e22). NOTE: The SMU always returns `INST_OVERRANGE` (1.0e22) if it is on a fixed range that is too low for the signal being measured.
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` is to return an indicator value.
KI_OUTP_RELAY_STATE	KI_OUTP_HIZ KI_OUTP_NORM	Only available if there are no preamplifiers. `KI_OUTP_HIZ` sets the state to high impedance (open). `KI_OUTP_NORM` sets the state to normal (closed, force V 0).

The following SMU *`modifier`* parameters are accepted but do no operations in the 4200A-SCS. They are included for compatibility so that existing S530 or S600 programs that use `setmode` can be ported to the 4200A-SCS without generating errors.

- `KI_IMTR`
- `KI_VMTR`

Also see

None

sintgX

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

Usage

```
int sintgi(int instr_id, double *result);
int sintgv(int instr_id, double *result);
```

`instr_id`	The instrument identification code of the measuring instrument
`result`	The floating-point array where the results are stored

Details

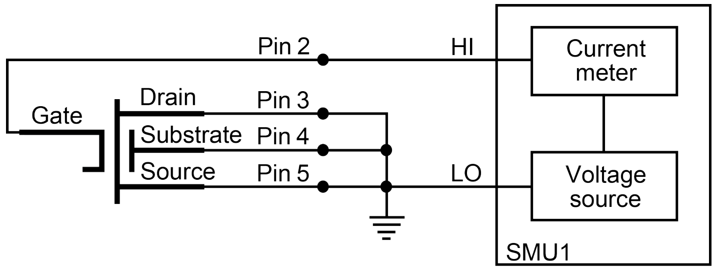
Use this command to create an entry in the measurement scan table. During any of the sweeping commands, a measurement scan is performed for every force point in the sweep. During each scan, a measurement is made for every entry in the scan table. The measurements are made in the same order in which the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

The `sintgX` command sets up the new scan table entry to make an integrated measurement. The measurement results are stored in the array, specified by the *`result`* parameter. Each time a measurement scan is made, a new measurement result is stored at the next location in the results array. If the scan table is not cleared, making multiple sweeps will continue to add new measurement results to the end of the array. Care must be taken that the results array is large enough to hold all measurements that are made before the scan table is cleared. The scan table is cleared by an explicit call to the `clrscn` command or implicitly when the `devint` or `execut` command is called.

Example

```
double 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 2-2)
devint (on page 2-6)
execut (on page 2-8)
sweepX (on page 4-29)

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.

Usage

```
int smeasi(int instr_id, double *result);
int smeast(int instr_id, double *result);
int smeasv(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

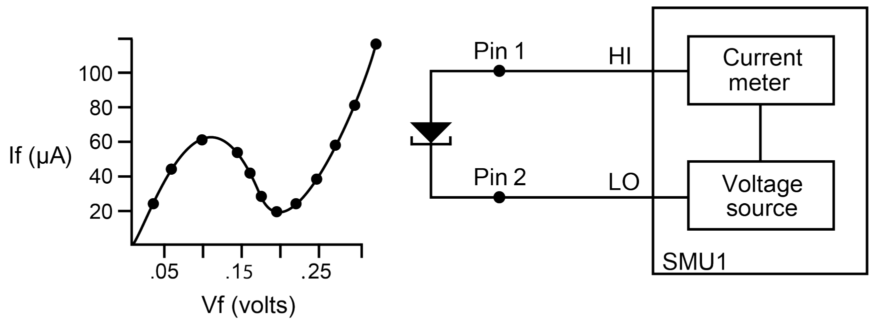
This command creates an entry in the measurement scan table. During any of the sweep functions, a measurement scan is done for every force point in the sweep. During each scan, a measurement is made for every entry in the scan table. The measurements are made in the same order in which the entries were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

The `smeasX` command sets up the new scan table entry to make an ordinary measurement. The measurement results are stored in the array specified by the *result* parameter. Each time a measurement scan is made, a new measurement result is stored at the next location in the *result* array. If the scan table is not cleared, doing multiple sweeps continues adding new measurement results to the end of the array. Care must be taken that the results array is large enough to hold all measurements that are made before the scan table is cleared. The scan table is cleared by an explicit call to the `clrscn` command or implicitly when the `devint` or `execut` command is called.

Example

```
double resi[13]; /* Defines array. */
double vf[13];
.
.
.
conpin(SMU1, l, 0);
conpin(GND, 2, 0);
rtfary(vf); /* Return the voltage force array*/
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 V to */
 /* 0.3 V. */
```

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 2-2)
devint (on page 2-6)
execut (on page 2-8)
sweepX (on page 4-29)

trigcomp

This command causes a trigger when an instrument goes in or out of compliance.

Usage

`int trigcomp(int instr_id, int mode);`

instr_id	The instrument identification code the trigger is set to
mode	Specifies whether to trigger when an instrument is in or out of compliance: ■ `1`: Trigger when in compliance ■ `0`: Trigger when out of compliance

Details

This command monitors the given instrument for compliance. A trigger can be set when the instrument is either in compliance or out of compliance, based on the specified mode.

Also see

None

trigXg, trigXl

This command monitors for a predetermined level of voltage, current, or time.

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, 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`.

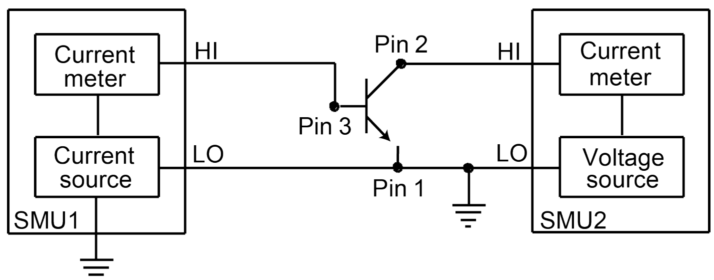
- The `trigXg` or `trigXl` command provides the `sweepX` command the digital feedback to allow for the increase or decrease in sourcing values.
- The `trigXl` and `trigXg` commands must be located before any associated `searchX` commands.
- Triggers are not automatically reset by the `searchX` or `sweepX` 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`, `execut`, or `devint` command is executed.

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`.

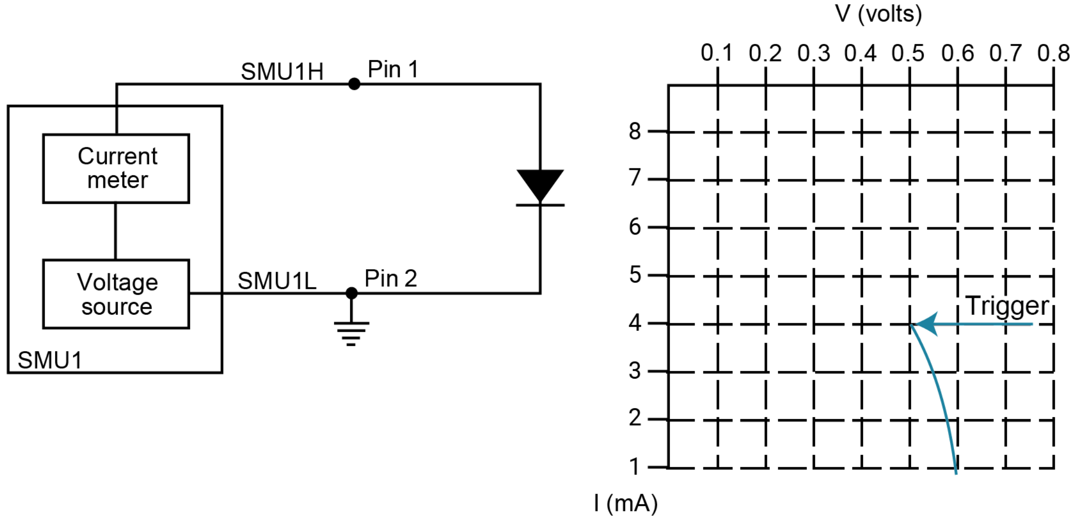


Example 2

```
double res1[20];
.
.
conpin(SMU1, 1, 0);
conpin(GND,2, 0);
trigil(SMU1, +4.0E-3); /* If less than +4 mA, */
 /* stop ramping. */

smeasi(SMU1, res1); /* Measure current at each of */
 /* the 19 levels; return */
 /* results to the res1 array. */
sweepv(SMU1, 0.0, 0.6, 18, 1.00E-3); /* Generate */
     /* 0.0 V to 0.6 V */
 /* in 18 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 2-27)
searchX (on page 2-29)
sintgX (on page 2-34)
smeasX (on page 2-35)
sweepX (on page 4-29)

tstdsl

This command deselects a test station.

Usage

```
tstdsl(void);
```

Details

To relinquish control of an individual test station, a new test station must be selected using `tstsel` before any subsequent test control commands are run.

The `tstdsl` command has the same effect as the `tstsel(0)` command.

NOTE

`tstdsl` is not required for use in a user test module (UTM).

Example

```
tstdsl( );    /* Disables test station.*/
```

Also see

tstsel (on page 2-40)

tstsel

This command enables or disables a test station.

Usage

```
tstsel(int x);
```

x	The test station number: `0` or `1`

Details

`tstsel` is normally called at the beginning of a test program.

`tstsel(1)` selects the first test station and loads the instrumentation configuration.

NOTE

The `tstsel` command is not required for use in a user test module (UTM).

Also see

tstdsl (on page 2-40)

LPT commands for math operations

In this section:

LPT commands for math operations

The following commands provide math operations.

kfpabs

This command takes a user-specified positive or negative value and converts it into a positive value that is returned to a specified variable.

Usage

```
int kfpabs(double *x, double *z);
```

x	Pointer to the variable to be converted to an absolute value
z	Pointer to the variable where the result is stored

Example

```
double ares2, vb1;
.
.
forcev(SMU1, vb1);/* Output vb1 from SMU1. */
measi(SMU1, &ares2);/* Measure SMU1 current; */
/* store in ares2. */
kfpabs(&ares2, &ares2);/* Convert ares2 to absolute */
/* value; return result to ares2*/
```

This example takes the absolute value of a current reading. `forcev` outputs vb1 volts from SMU1. This current is measured with `measi`, and the result is stored in location `ares2`. The absolute value of `ares2` is then calculated and stored as `ares2`.

Also see

None

kfpadd

This command adds two real numbers and stores the result in a specified variable.

Usage

`int kfpadd(double *x, double *y, double *z);`

x	The first of two values to add
y	The second of two values to add
z	A variable in which the sum *x* + *y* is stored

Details

The values referenced by *x* and *y* are summed and the result is stored in the location pointed to by *z*. If an overflow occurs, the result is `±Inf`. If an underflow occurs, the result is zero (0).

Example

```
double res1, res2, resia;
.
.
measv(SMU1, &res1);/* Measure SMU1 voltage; store */
/* in res1. */
measi(SMU2, &res2);/* Measure SMU2 current; store */
/* in res2. */
kfpadd(&res1, &res2, &resia);/* Adds res1 and res2; return */
/* result to resia. */
.
.
```

This example adds the data in `res1` to the data in `res2`. The result is stored in the `resia` variable.

Also see

None

kfpdiv

This command divides two real numbers and stores the result in a specified variable.

Usage

`int kfpdiv(double *x, double *y, double *z);`

x	The dividend
y	The divisor
z	A variable where the result of *x*/*y* is stored

Details

The value referenced by *x* is divided by the value referenced by *y*. The result is stored in the location pointed to by *z*. If an overflow occurs, the result is `±Inf`. If an underflow occurs, the result is zero (0).

Example

```
double res1, res2, resia;
.
.
measv(SMU1, &res1);/* Measure SMU1 voltage; store */
/* in res1. */
measi(SMU2, &res2);/* Measure SMU2 current; store */
/* in res2. */
kfpdiv(&res1, &res2, &resia);/* Divide res1 by res2; return */
/* result to resia. */
.
.
```

This example divides the data in `res1` by the data in `res2`. The result is stored in the `resia` variable.

Also see

None

kfpexp

This command supplies the base of natural logarithms (e) raised to a specified power and stores the result as a variable.

Usage

`int kfpexp(double *x, double *z);`

x	The exponent
z	The variable where the result of e^x is stored

Details

e raised to the power of the value referenced by *x* is stored in the location pointed to by *z*. If an overflow occurs, the result is `±Inf`. If an underflow occurs, the result is zero (0).

Example

```
double res4, res4e;
.
.
measv(SMU1, &res4);/* Raise the base of natural */
/* logarithms e to the power */
/* res4; */
kfpexp(&res4, &res4e;/* return the result to res4e. */
.
.
```

In this example, `kfpexp` raises the base of natural logarithms to the power specified by the exponent `res4`. The result is stored in `res4e`.

Also see

None

kfplog

This command returns the natural logarithm of a real number to the specified variable.

Usage

`int kfplog(double *x, double *z);`

x	A variable containing a floating-point number
z	A variable where the result of ln (*x*) is stored

Details

This command returns a natural logarithm, not a common logarithm. The natural logarithm of the value referenced by *x* is stored in the location pointed to by *z*.

If a negative value or zero (0) is supplied for *x*, a log of negative value or zero (0) error is generated and the result is `NaN` (not a number).

Example

```
double res1, logres;
.
.
measv(SMU1, &res1);/* Measure SMU1; store in res1. */
kfplog(&res1, &logres);/* Convert res1 to a natural */
/* LOG and store in logres. */
.
```

This example calculates the natural logarithm of a real number (`res1`). The result is stored in `logres`.

Also see

None

kfpmul

This command multiplies two real numbers and stores the result as a specified variable.

Usage

`int kfpmul(double *x, double *y, double *z);`

x	A variable containing the multiplicand
y	A variable containing the multiplier
z	The variable where the result of *x***y* is stored

Details

The value referenced by *x* is multiplied by the value referenced by *y*, and the result is stored in the location pointed to by *z*. If an overflow occurs, the result is `±Inf`. If an underflow occurs, the result is zero (0).

Example

```
double res1, res2, pwr2;
.
.
measi(SMU1, &res1);/* Measure SMU1 current; */
/* store in res1. */
measv(SMU1, &res2);/* Measure SMU1 voltage; */
/* store in res2. */
kfpmul(&res1, &res2, &pwr2);/* Multiply res1 by res2; */
/* return result to pwr2. */
.
.
```

This example multiplies variables `res1` and `res2`. The result is stored in the variable `pwr2`.

Also see

None

kfpneg

This command changes the sign of a value and stores the result as a specified variable.

Usage

`int kfpneg(double *x, double *z);`

x	A variable containing the number to be converted
z	A variable where the result of −*x* is stored

Details

If the value is positive, it is converted to a negative. If the value is negative, it is converted to a positive.

Example

```
double res4;
.
.
forcev(SMU1, 10.0);/* Output 10 V from SMU1. */
measi(SMU1, &res4);/* Measure SMU1 current; store */
/* in res4. */
kfpneg(&res4, &res4);/* Convert sign of res4; */
./* return results to res4. */
.
```

This example changes the sign of a positive voltage reading. `forcev` outputs a positive 10 V from SMU1. The current is measured with `measi` and the result is stored as `res4`. The `kfpneg` command reads `res4` and converts the data to a negative value. `res4` is then overwritten with the converted value.

Also see

None

kfppwr

This command raises a real number to a specified power and assigns the result to a specified variable.

Usage

```
int kfppwr(double *x, double *y, double *z);
```

x	A variable that contains a floating-point number
y	A variable that contains the exponent
z	A variable where the result of x^y is stored

Details

The value referenced by *x* is raised to the power of the value referenced by *y*, and the result is stored in the location pointed to by *z*. If an overflow occurs, the result is `±Inf`. If an underflow occurs, the result is zero (0).

If *x* points to a negative number, a power of a negative number error is generated, and the result returned is `-Inf`.

If *x* points to a value of zero (0) and y points to a negative number, a divide by zero (0) error is generated, and the result returned is `+Inf`.

If *x* points to a value of 1.0, the result is 1.0, regardless of the exponent.

Example

```
double res2, pwres2, power=3.0;
.
.
measv(SMU1, &res2);/* Measure SMU1; store */
/* result in res2. */
kfppwr(&res2, &power,
    &pwres2);/* res2 to the third power; */
/* return result to pwres2. */
.
```

Raises the variable `res2` by the power of three. The result is stored in `pwres2`.

Also see

None

kfpsqrt

This command performs a square root operation on a real number and returns the result to the specified variable.

Usage

```
int kfpsqrt(double *x, double *z);
```

x	A variable that contains a floating-point number
z	A variable where the result, the square root of x, is stored

Details

The square root of the value referenced by *x* is stored in the location pointed to by *z*.

If *x* points to a negative number, a square root of negative number error is generated, and the result is `NaN` (not a number).

Example

```
double res1, sqres2;
.
.
measv(SMU1, &res1);/* Measure SMU1; store result */
./* in res1. */
kfpsqrt(&res1, &sqres2);/* Find square root of res1; */
/* return result to sqres2. */
.
```

This example converts a real number (`res1`) into its square root. The result is stored in `sqres2`.

Also see

None

kfpsub

This command subtracts two real numbers and stores their difference in a specified variable.

Usage

`int kfpsub(double *x, double *y, double *z);`

x	A variable containing the minuend
y	A variable containing the subtrahend
z	The variable where the result of *x - y* is stored

Details

The value referenced by *y* is subtracted from the value referenced by *x*. The result is stored in the location pointed to by *z*. If an overflow occurs, the result is `±Inf`. If an underflow occurs, the result is zero (0).

Example

```
double res1, res2, diff2;
.
.
measv(SMU1, &res1);/* Measure SMU1; store result */
/* in res1. */
measv(SMU2, &res2);/* Measure SMU2; store result */
/* in res2. */
kfpsub(&res1, &res2, &diff2);/* Subtract res2 from res1; */
./* return the place with */
/* result to diff2. */
```

This example subtracts `res2` from `res1`. The result is returned to `diff2`.

Also see

None

Section 4

LPT commands for SMUs

In this section:

LPT commands for SMUs

The following information explains the commands in the LPT library for the SMUs.

adelay

This command specifies an array of delay points to use with `asweep`*`X`* command calls.

Usage

`int adelay(long` *`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.

Also see

asweepX (on page 4-2)

asweepX

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

Usage

```
int asweepi(int instr_id, long num_points, double delay_time, double *force_array);
int asweepv(int instr_id, long 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 `asweepX` command is used with the `smeasX`, `sintgX`, or `savgX` commands.

The `trigXl` or `trigXg` command can also be used with the `asweepX` 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 `asweepX` command. Data resulting from each step is stored in an array, as noted above, with `smeasX`. 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` command for additional information.

The maximum number of times data is measured (using the `smeasX`, `sintgX`, or `savgX` command) is determined by the `num_points` argument in the `asweepX` 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.

When multiple calls to the `asweepX` command are executed in the same test sequence, the `smeasX`, `sintgX`, or `savgX` arrays are loaded sequentially. This appends the measurements from the second `asweepX` command to the previous results. If the arrays are not dimensioned correctly, access violations occur. The measurement table remains intact until the `devint` or `clrscn` command is executed.

Defining new test sequences using the `smeasX`, `sintgX`, or `savgX` command appends the command to the active measure list. Previous measures are still defined and will be used. The `clrscn` command is used to eliminate previous buffers for the second sweep. Using the `smeasX`, `sintgX`, and `savgX` commands after calling the `clrscn` or `execut` command causes the appropriate new measures to be defined and used.

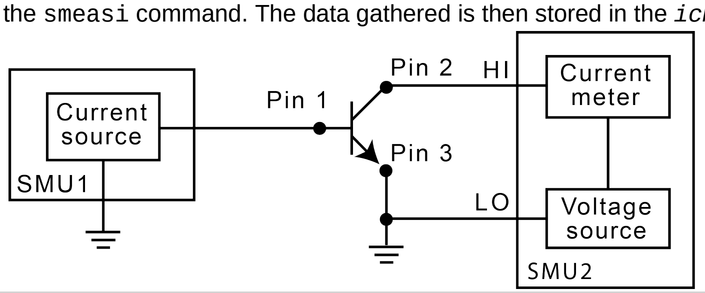
Changing the source mode of the SMU can modify the measure range. If the sourcing mode is changed from voltage to current sourcing (or from current to voltage sourcing), the measure range may be changed to minimize variations in the SMU output level. See `rangeX` for the recommended command order.

If `adelay` is called before `asweepX`, each `adelay` value is added to the `asweepX` *`delay_time`*. This sum is compared to the maximum delay for the configured instrument card and if any value is larger, an error occurs. The SMU maximum delay is 2,147.483 s. The CVU maximum is 999 s.

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 A to 1.0E-1 A 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 4-16)
rangeX (on page 4-23)
savgX (on page 2-27)
sintgX (on page 2-34)
smeasX (on page 2-35)
trigXg, trigXl (on page 2-37)

avgX

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

Usage

```
int avgi(int instr_id, double *result, long stepno, double steptime);
int avgv(int instr_id, double *result, long stepno, double steptime);
```

instr_id	The instrument identification code of the measuring instrument
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

Details

The `avg`*X* 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 `meas`*X* 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.

The `range`*X* command directly affects the operation of the `avg`*X* command. The use of the `range`*X* command prevents the addressed instrument from automatically changing ranges. This can result in an overrange condition similar to what would occur when measuring 10.0 V on a 4.0 V range. An overrange condition returns the value 1.0e+22 as the result of the measurement.

If the `range`*X* command is not in the test sequence before the `avg`*X* call, the measurements performed automatically select the optimum range.

A compliance limit setting goes into effect when the SMU is on a measure range that can accommodate the limit value. For manual ranging, the `range`*X* command is used to select the range. For autoranging, the `avgi` or `avgv` commands triggers a needed range change before the measurement is made. See “Compliance limits” in the *Model 4200A-SCS Source-Measure Unit (SMU) User's Manual* for details.

Example

```
double leakage;

.
.
limiti(SMU1, 1.0e-06); /* Limit the maximum current */
 /* to 1 uA */
forcev(SMU1, 10.0); /* Force 10 V across the DUT */
delay(100); /* Delay 100 ms to allow for */
 /* device settling */
avgi(SMU1, &leakage, 5, 0.01); /* Average 5 readings, delay */
 /* 10 ms per measurement */
```

This example illustrates how to use the `avgX` command to make five current readings and return the average of the measurements to the variable *`leakage`*.

Also see

measX (on page 4-18)
rangeX (on page 4-23)

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).

Usage

```
int bmeasi(int instr_id, double *result, long numrdg,

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

instr_id	The instrument identification code of the measuring instrument
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 *`timerdata`* array.

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 `bmeas`*X* 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 `bmeas`*X* 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.

Usage

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

instr_id	The instrument identification code of the sourcing instrument
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 32,767)
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

`bsweepi` is only available for SMUs.

The `bsweep`*X* command is used with the `trig`*X*`g`, `trig`*X*`l`, or `trigcomp` 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 `bsweep`*X* command terminates the sweep and clears all sources by executing a `devclr` command internally. The standard `sweep`*X* 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 `bsweep`*X* command can also be used with the `smeas`*X*, `sintg`*X*, `savg`*X*, 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 `smeas`*X*, `sintg`*X*, `savg`*X*, or `clrscn` command. As multiple calls to `sweep`*X* 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 `bsweep`*X* command step.

When multiple calls to the `bsweep`*X* command are executed in the same test sequence, the arrays defined by calls to the `smeas`*X*, `sintg`*X*, or `savg`*X* command are all loaded sequentially. The results from the second call to the `bsweep`*X* command are appended to the results of the previous `bsweep`*X* command call. This can cause access violation errors if the arrays were not dimensioned for the absolute total. The measurement scan table remains intact until a `devint`, `execut`, or `clrscn` command completes.

Defining new test sequences using the `smeas`*X*, `sintg`*X*, or `savg`*X* command adds the command to the active measure list. The previous measurements are still defined and used; however, previous measurements for the second sweep can be eliminated by calling the `clrscn` command. New measurements are defined and used by calling the `smeasX`, `sintgX`, or `savgX` command after a `clrscn` command.

Note that changing the source mode of the SMU can modify the measure range. If the sourcing mode is changed from voltage to current sourcing (or from current to voltage sourcing), the measure range may be changed to minimize variations in the SMU output level. See `rangeX` for recommended command order.

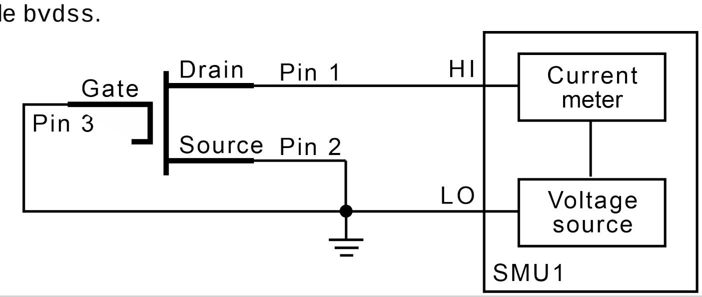
NOTE

It is recommended that you do not use GPIB instruments when doing sweeps with the `bsweepX` command. Refer to `kibdefint` for additional information.

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 2-2)
devclr (on page 4-9)
execut (on page 2-8)
kibdefint (on page 2-16)
rangeX (on page 4-23)
rtfary (on page 2-27)
savgX (on page 2-27)
sintgX (on page 2-34)
smeasX (on page 2-35)
sweepX (on page 4-29)
trigXg, trigXl (on page 2-37)
trigcomp (on page 2-36)

devclr

This command sets all sources to a zero state.

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. `devclr` is implicitly called by `clrcon`, `devint`, `execut`, and `tstdsl`.

For C-V testing, this command turns off the dc bias voltage.

Also see

clrcon (on page 7-2)
devint (on page 2-6)
execut (on page 2-8)
kibdefclr (on page 2-15)
tstdsl (on page 2-40)

devint

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

Usage

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

Details

Resets all active instruments, including the 4200A-CVIV, 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.

The `devint` command is implicitly called by the `execut` and `tstdsl` commands.

To abort a running `pulse_exec` pulse test, see `dev_abort`.

`devint` does the following:

1. Clears all sources by calling `devclr`.
2. Clears the matrix crosspoints by calling `clrcon`.
3. Clears the trigger tables by calling `clrtrg`.
4. Clears the sweep tables by calling `clrscn`.
5. Resets GPIB instruments by sending the string defined with `kibdefint`.
6. Resets the active instrument cards.

Instrument cards are reset in the following order:

1. SMU instrument cards
2. CVU instrument cards
3. Pulse instrument cards (4225-PMU or 4220-PGU)

The SMUs return to the following states:

- 100 µA and 10 V ranges
- Autorange on
- Voltage source
- 0 V dc bias

The 4210-CVU or 4215-CVU returns to the following states:

- 30 mV_{RMS} ac signal
- 0 V dc bias
- 100 kHz frequency
- Autorange on
- Cable length compensation set to 0 m
- Open/Short/Load compensation disabled

The 4225-PMU or 4220-PGU returns to the following states:

- The pulse mode is maintained. For example, if the pulse card is in Segment Arb mode, it is still in Segment Arb mode after the `devint` process is complete.
- 5 V and 10 mA ranges
- If in pulse mode:
  - Period of 1 µs
  - Transition times (rise and fall) of 100 ns
  - Width of 500 ns
  - Voltage high and low of 0 V
  - Load of 50 Ω
- If in segmented arb mode, Start Voltage is 0 V
- If in arbitrary waveform mode, Table Length is 100

Also see

clrcon (on page 7-2)
clrscn (on page 2-2)
clrtrg (on page 2-3)
dev_abort (on page 6-4)
devclr (on page 4-9)
kibdefint (on page 2-16)

forceX

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

Usage

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

instr_id	The instrument identification code
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 `limit`*X*, `range`*X*, and `force`*X* commands on the same source at the same time in a test sequence, call the `limit`*X* and `range`*X* commands before the `force`*X* command. See "Compliance limits" in the *Model 4200A-SCS Source-Measure Unit (SMU) User's Manual* for details.

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.

To force zero current with a higher voltage limit than the 20 V default, include one of the following calls ahead of the `forcei` call:

- A `measv` call, which causes the 4200A-SCS to autorange to a higher voltage limit.
- A `rangev` call to an appropriate fixed voltage, which results in a fixed voltage limit.

To force zero volts with a higher current limit than the 10 mA default, include one of the following calls ahead of the `forcev` call:

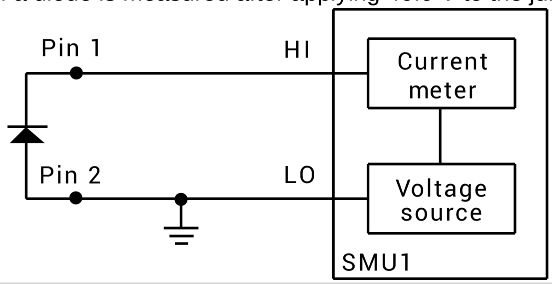
- A `measi` call, which causes the 4200A-SCS to autorange to a higher current limit.
- A `rangei` call to an appropriate fixed current, which results in a fixed current limit.

If you change the source mode of the source-measure unit (SMU), it can modify the measure range. If the source mode is changed from voltage to current source (or from current to voltage source), the measure range may be changed to minimize variations in the SMU output level. See `range`*X* for the recommended command order.

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

rangeX (on page 4-23)

getstatus

This command returns the operating state of a specified instrument.

Usage

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

instr_id	The instrument identification code
parameter	The parameter of query; see **Details**
result	The data returned from the instrument; the `getstatus` command returns one item

Details

NOTE

If the `UT_INVLDPRM` invalid parameter error is 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) and a pulse card (VPU) are provided in the following tables.

No status values are provided for measurement-specific conditions.

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_IARANGE	The presently active range	Current range (full-scale range value)
KI_VARANGE		Voltage range (full-scale range value)
KI_COMPLNC	Compliance status of last reading	Bitmapped values: 2 = LIMIT (at the compliance limit set by `limit`*X*) 4 = RANGE (at the top of the range set by `range`*X*)
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_RANGE_COMPLIANCE	Range compliance status of last reading	Returns 1 if in range compliance

Supported pulse card getstatus query parameters

Parameter	Returns	Comment
General parameters		
KI_VPU_PERIOD	Pulse period	Pulse period value in seconds
KI_VPU_TRIG_POLARITY	Trigger polarity	Rising or falling edge
KI_VPU_CARD_STATUS	Card status	Card level status
KI_VPU_TRIG_SOURCE	Trigger source	Trigger source value
Channel-based parameters		
KI_VPU_CH1_RANGE	Source range	Channel 1 range value in volts (5.0 or 20.0)
KI_VPU_CH2_RANGE	Source range	Channel 2 range value in volts (5.0 or 20.0)
KI_VPU_CH1_RISE	Rise time	Channel 1 rise time value in seconds
KI_VPU_CH2_RISE	Rise time	Channel 2 rise time value in seconds
KI_VPU_CH1_FALL	Fall time	Channel 1 fall time value in seconds
KI_VPU_CH2_FALL	Fall time	Channel 2 fall time value in seconds
KI_VPU_CH1_WIDTH	Pulse width	Channel 1 pulse width value in seconds
KI_VPU_CH2_WIDTH	Pulse width	Channel 2 pulse width value in seconds
KI_VPU_CH1_VHIGH	Pulse high	Channel 1 pulse high level value in volts
KI_VPU_CH2_VHIGH	Pulse high	Channel 2 pulse high level value in volts
KI_VPU_CH1_VLOW	Pulse low	Channel 1 pulse low level value in volts
KI_VPU_CH2_VLOW	Pulse low	Channel 2 pulse low level value in volts
KI_VPU_CH1_DELAY	Pulse delay	Channel 1 pulse delay from trigger value in seconds
KI_VPU_CH2_DELAY	Pulse delay	Channel 2 pulse delay from trigger value in seconds
KI_VPU_CH1_ILIMIT	Current limit	Channel 1 current Limit value in amps

Supported pulse card getstatus query parameters

Parameter	Returns	Comment
`KI_VPU_CH2_ILIMIT`	Current limit	Channel 2 current Limit value in amps
`KI_VPU_CH1_BURST_COUNT`	Burst count	Channel 1 burst count value
`KI_VPU_CH2_BURST_COUNT`	Burst count	Channel 2 burst count value
`KI_VPU_CH1_TEST_STATUS`	Status	Channel 1 test status
`KI_VPU_CH2_TEST_STATUS`	Status	Channel 2 test status
`KI_VPU_CH1_DC_OUTPUT`	DC output	Channel 1 dc output value
`KI_VPU_CH2_DC_OUTPUT`	DC output	Channel 2 dc output value
`KI_VPU_CH1_LOAD`	Pulse load	Channel 1 pulse load value
`KI_VPU_CH2_LOAD`	Pulse load	Channel 2 pulse load value

Also see

getinstid (on page 2-10)

intgX

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

Usage

```
int intgi(int instr_id, double *result);
int intgv(int instr_id, double *result);
```

instr_id	The instrument identification code of the measuring instrument, such as SMU1
result	The variable assigned to the result of the 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.

For a measurement conversion, the signal is sampled for a specific period of time. This sampling time for measurement is called the integration time. For the `intgX` command, the default integration time is set to 1 PLC. For 60 Hz line power, 1 PLC = 16.67 ms (1 PLC/60 Hz). For 50 Hz line power, 1 PLC = 20 ms (1 PLC/50 Hz).

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.

NOTE

The only difference between `measX` and `intgX` is the integration time. For `measX`, the integration time is fixed at 0.01 PLC. For `intgX`, the default integration time is 1 PLC but can set to any PLC value between 0.01 and 10.0 by using the `setmode` command.

`rangeX` directly affects the operation of `intgX`. The use of `rangeX` 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 4.0 V range. An overrange condition returns the value 1.0E+22 as the measurement result.

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

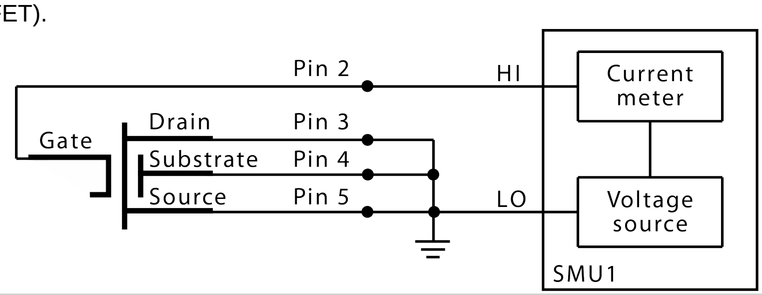
In general, measurement commands that return multiple results are more efficient than sending multiple measurement commands.

A compliance limit setting goes into effect when the SMU is on a measure range that can accommodate the limit value. For manual ranging, the `rangeX` command is used to select the range. For autoranging, `intgi` or `intgv` triggers a needed range change before the measurement is made. See “Compliance limits” in the *Model 4200A-SCS Source-Measure Unit (SMU) User's Manual* for details.

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 2-6)
measX (on page 4-18)
rangeX (on page 4-23)
setmode (on page 2-32)

limitX

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

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
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 temporarily limits to full scale of range.

This command must be called in the test sequence before the associated `forceX`, `pulseX`, `bsweepX`, `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.

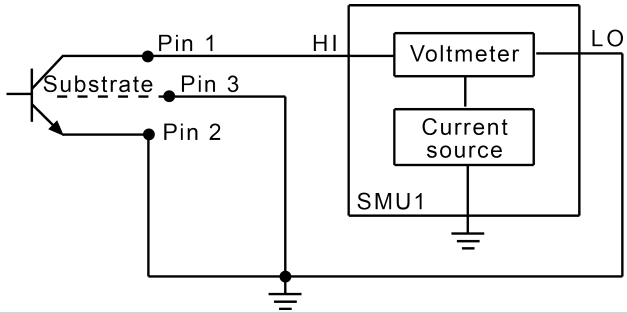
The limits set within a particular test sequence are cleared when the `devint` or `execut` command is called.

If you need a voltage limit greater than 20 V at a source-measure unit (SMU) that is set to force zero current, call the `measv` command to set the SMU to autorange to a higher range, or use the `rangev` command to set a higher voltage range. Similarly, if you need a current limit of greater than 10 mA at a SMU that is set to force zero volts, call the `measi` command to set the SMU to autorange to a higher range or use the `rangev` command to set a higher current range.

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

bsweepX (on page 4-7)
devint (on page 2-6)
execut (on page 2-8)
forceX (on page 4-11)
measX (on page 4-18)
pulseX (on page 4-21)
rangeX (on page 4-23)
searchX (on page 2-29)
sweepX (on page 4-29)

lorangeX

This command defines the bottom autorange limit.

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 4200A-SCS automatically provides any settling delay for the range change that may be necessary due to this potential range change.

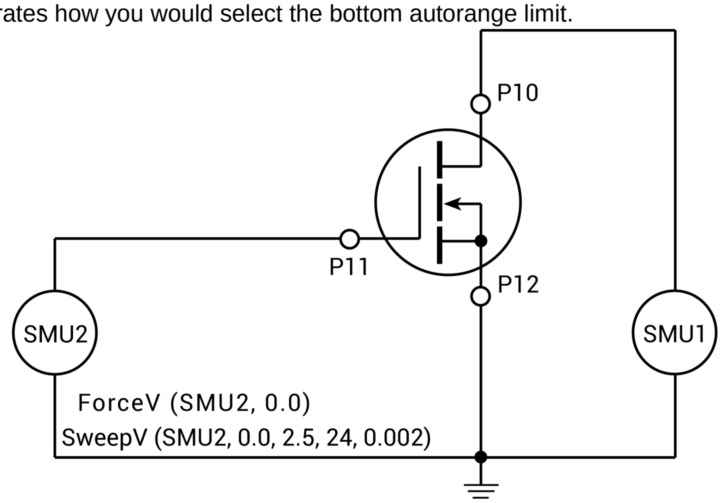
Once defined, the `lorangeX` command is in effect until a `devclr`, `devint`, `execut`, or another `lorangeX` command executes.

Example

```
double idatrg[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 4-9)
[devint](#) (on page 2-6)
[execut](#) (on page 2-8)

measX

This command allows the measurement of voltage, current, or time.

Usage

```
int meast(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
result	The variable assigned to the result of the measurement

Details

For a measurement conversion, the signal is sampled for a specific period of time. This sampling time for measurement is called the integration time. For the `measX` command, the integration time is fixed at 0.01 PLC. For 60 Hz line power, 0.01 PLC = 166.67 µs (0.01 PLC/60 Hz). For 50 Hz line power, 0.01 PLC = 200 µs (0.01 PLC/50 Hz).

NOTE

The only difference between `measX` and `intgX` is the integration time. For `measX`, the integration time is fixed at 0.01 PLC. For `intgX`, the default integration time is 1 PLC, but can set to any PLC value between 0.01 and 10.0.

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. The use of 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 4.0 V range. An overrange condition returns the value 1.0E+22 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.

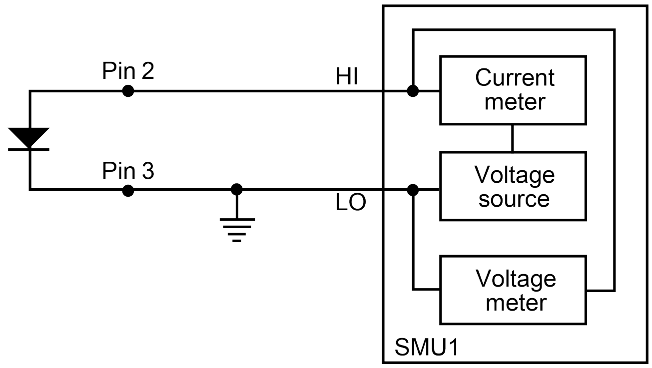
In general, measurement commands that return multiple results are more efficient than making multiple measurement commands.

A compliance limit setting goes into effect when the SMU is on a measure range that can accommodate the limit value. For manual ranging, the `rangeX` command is used to select the range. For autoranging, the `measi` or `measv` command will trigger a needed range change before the measurement is performed. See “Compliance limits” in the *Model 4200A-SCS Source-Measure Unit (SMU) User's Manual* for details.

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

intgX (on page 4-14)
rangeX (on page 4-23)

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.

Usage

```
int mpulse(long instr_id, double pulse_amplitude, double pulse_duration, double
    *v_meas, double *i_meas);
```

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
v_meas	The variable used to receive the voltage on the output of the instrument at the time the pulse terminates
i_meas	The variable used to receive the current drawn from the instrument; this measurement is made simultaneously with the voltage, so the combined values are an exact representation of the device load at pulse termination

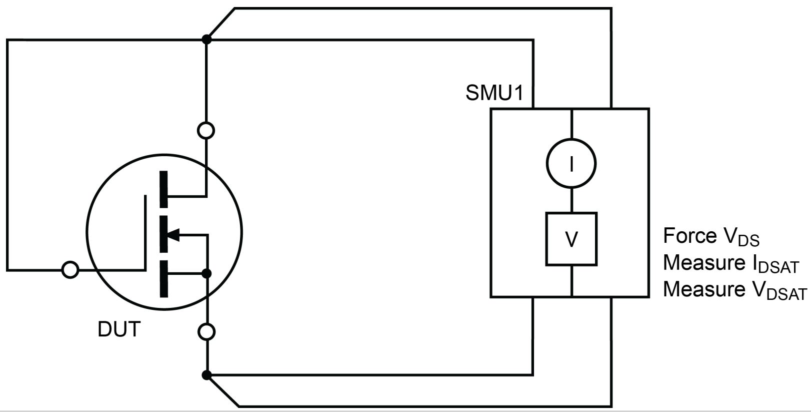
Details

Voltage and current are measured just before the pulse terminates. Pulsing is useful for devices that exhibit self-heating, which could damage the device or shift operating characteristics. Examples are high-power GaAs transistors or BJTs and some silicon devices.

Example

```
double vdsat, idsat, vds;
.
.
mpulse(SMU1, vds, l.0E-3, &vdsat, &idsat);
 /* Pulse output of SMU1. */
```

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

pulseX

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

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
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.

After `pulseX` is executed, the output is turned off. In order to make measurements, the output must be turned on again. `measX` 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 made after the pulse has completed.

NOTE

When the source is not operating, measurements are not allowed.

Whenever `pulseX` is executed, either a default or a programmed current or voltage limit is in effect. Refer to the `limitX` command for additional information.

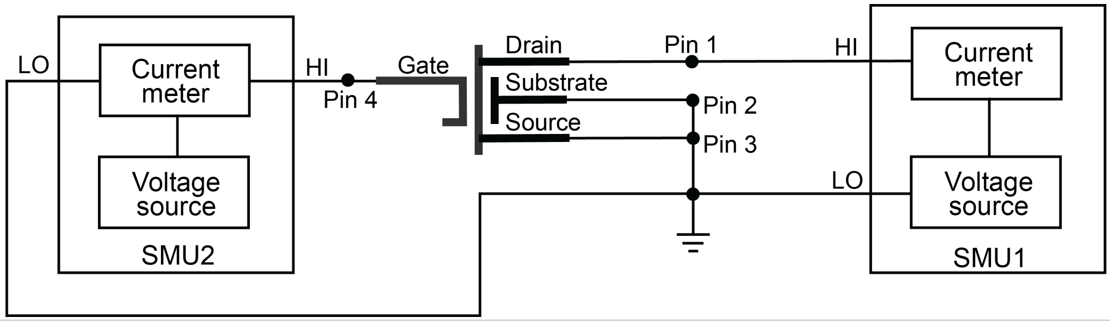
When using `limitX`, `rangeX`, and `pulseX` on the same source at the same time in a test sequence, call `limitX`, then `rangeX`, then `pulseX`.

Changing the source mode of the SMU can modify the measure range. If the sourcing mode is changed from voltage to current sourcing (or from current to voltage sourcing), the measure range may be changed to minimize variations in the SMU output level. See `rangeX` for recommended command order.

Example

```
double res1, res2;
.
.
conpin(GND, 2, 3, 0);
conpin(SMU1, 1, 0);
conpin(SMU2, 4, 0);
forcev(SMU1, .5);
trigig(SMU1, +1.E-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, 5.E-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 `trigig` command. This is possible because the clear trigger command (`clrtrg`) was not used.



Also see

limitX (on page 4-16)
rangeX (on page 4-23)

rangeX

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

Usage

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

instr_id	The instrument identification code
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 selects a range automatically

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. See “Compliance limits” in the *Model 4200A-SCS Source-Measure Unit (SMU) User's Manual* for details.

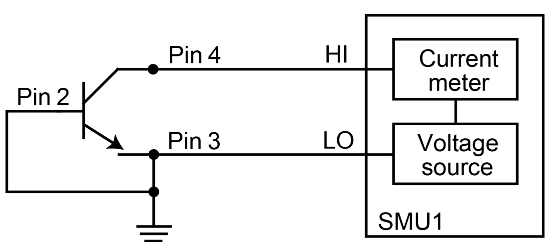
When changing the instrument range, be careful not to overrange the instrument. For example, a test initially performed on the 10 mA range with a 5 mA limit is changed to test in the 1 mA range with a 1 mA limit. Notice that the limit is lowered from 5 mA to 1 mA to avoid overranging the 1 mA setting.

When source mode of the SMU changes, the measure range may change. This change minimizes variations in the SMU output level. The source mode of the SMU refers to its voltage sourcing or current sourcing capability. Changing the source mode means using a command (such as `forceX`) to change the SMU mode from forcing voltage to forcing current (or from forcing current to forcing voltage). For example, if the SMU is programmed to force voltage (`forcev`), and then is programmed with to force current (`forcei`), to ensure a consistent output signal, the previously programmed current measure range may change. Make sure the correct measure range is set by sending the `rangeX` command after switching the source mode. The commands that can change the source mode are `asweepX`, `bsweepX`, `forceX`, `pulseX`, `searchX`, and `sweepX`.

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`.



Also see

asweepX (on page 4-2)
bsweepX (on page 4-7)
forceX (on page 4-11)
pulseX (on page 4-21)
searchX (on page 2-29)
sweepX (on page 4-29)

rtfary

This command returns the array of force values used during the subsequent voltage or frequency sweep.

Usage

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

forceArray	Array of force values for voltage or frequency

Details

This command returns an array of voltage or frequency force values for a sweep. Send this command before calling any sweep command.

> **NOTE**
>
> To prevent a memory exception error, make sure that the array that will receive the sourced values is large enough.

The following examples show the proper command sequence for using `rtfary`:

Example 1: Valid command sequence for voltage sweep	Example 2: Valid command sequence for frequency sweep
smeasz	smeasz
smeast	rtfary
rtfary	dsweepf
dsweepv	

Example

Programming example #2 (on page 5-35) returns the array of force values for the voltage sweep.

Also see

None

segment_sweepX_list

This command creates and returns up to a 4-segment linear sweep force table based on user-defined start, stop, and step values.

Usage

```
int segment_sweepv_list (double startVal, double *stopArray, double *stepArray, int
    numSegments, double *forceArray, int forceArraysize, int *numListpts);
int segment_sweepi_list (double startVal, double *stopArray, double *stepArray, int
    numSegments, double *forceArray, int forceArraysize, int *numListpts);
```

startVal	Starting voltage value
stopArray	A single dimension array containing stop values
stepArray	A single dimension array containing step values
numSegments	Number of segments
forceArray	A single dimension array returned with force values
forceArraysize	Size allocated for `forceArray`
numListpts	Number of total points in returned `forceArray`

Details

The `segment_sweepX` command is used with the `asweepX` command.

A forcing table is created with the `segment_sweepX_list` command and the force array table is sent using the `asweepX` command.

Example

```
startVoltage = 0.0V
stopArray[] = {5.0, -5.0, 0}
stepArray[] = {0.1, -0.5, 0.25}
segmentpts = 3
arraysize = 1000

    segment_sweepv_list(startVoltage, stopArray, stepArray, segmentpts,
   forceArray, arraysize, numListpts);
    forcev(SMU1, 0.0);
    rtfary(Programmed_V);
    smeasi(SMU1, Measured_I);
    asweepv(SMU1, *numListpts, delayValue, &forceArray[0]);
```

Also see

asweepX (on page 4-2)
forceX (on page 4-11)
rtfary (on page 4-25)
smeasX (on page 2-35)

setauto

This command re-enables autoranging and cancels any previous `rangeX` command for the specified instrument.

Usage

```
int setauto(int instr_id);
```

instr_id	The instrument identification code

Details

When an instrument is returned to the autorange mode, it remains in its present range for measurement purposes. The source range changes immediately.

Due to the dual-mode operation of the SMU (voltage versus current), `setauto` places both voltage and current ranges in autorange mode.

Example

```
double icer1;
double idatvg[25];
.
.
rangei(SMU1, 2.0E-9); /* Select manual range. */
delay(200); /* Delay after range change. */
measi(SMU1, &icer1); /* Measure leakage. */
.
.
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

ssmeasx

This command makes a series of readings until the change (delta) between readings is within a specified percentage.

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
result	The floating-point variable assigned to the result of the measurement
delta	The termination definition, which 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.

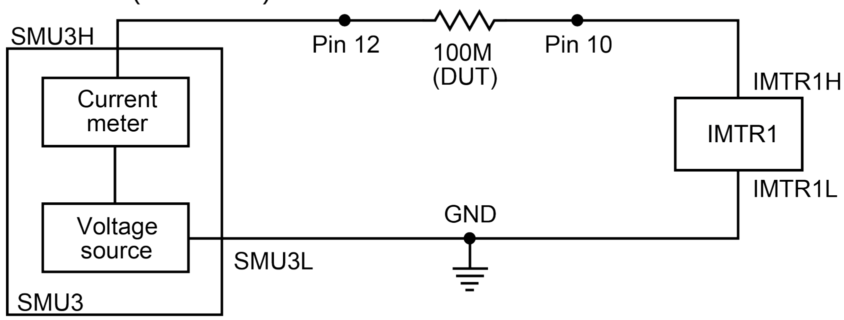
Under certain test conditions, the `ssmeas`*X* command is not ideal. For example, an oscillation where two contiguous measurements are within the given percentage will return a stable reading, even though the device cannot be measured.

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 (l.000E23)`



Also see

measX (on page 4-18)
rangeX (on page 4-23)
smeasX (on page 2-35)

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.

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 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 `sweep`*X* command is always used with the `smeas`*X*, `sintg`*X*, `savg`*X*, or `rtfary` command.

The `sweep`*X* 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 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 a one-dimensional array in the order they were made.

The `trigXg`, `trigXl`, and `trigcomp` 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 system maintains a measurement scan table consisting of devices to measure. This table is maintained by calls to the `smeasX`, `sintgX`, `savgX`, or `clrscn` command. As multiple calls to these commands are made, the commands are appended to this table.

When multiple calls to the `sweepX` command are executed in the same test sequence, the `smeasX`, `sintgX`, or `savgX` arrays are loaded sequentially. This appends the measurements from the second `sweepX` call to the previous results. If the arrays are not dimensioned correctly, access violations occur. The measurement table remains intact until the `clrscn`, `execut`, or `devint` command is executed.

Defining new test sequences using the `smeasX`, `sintgX`, or `savgX` commands adds commands to the active measure list. The previous measures are still defined and used. 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:

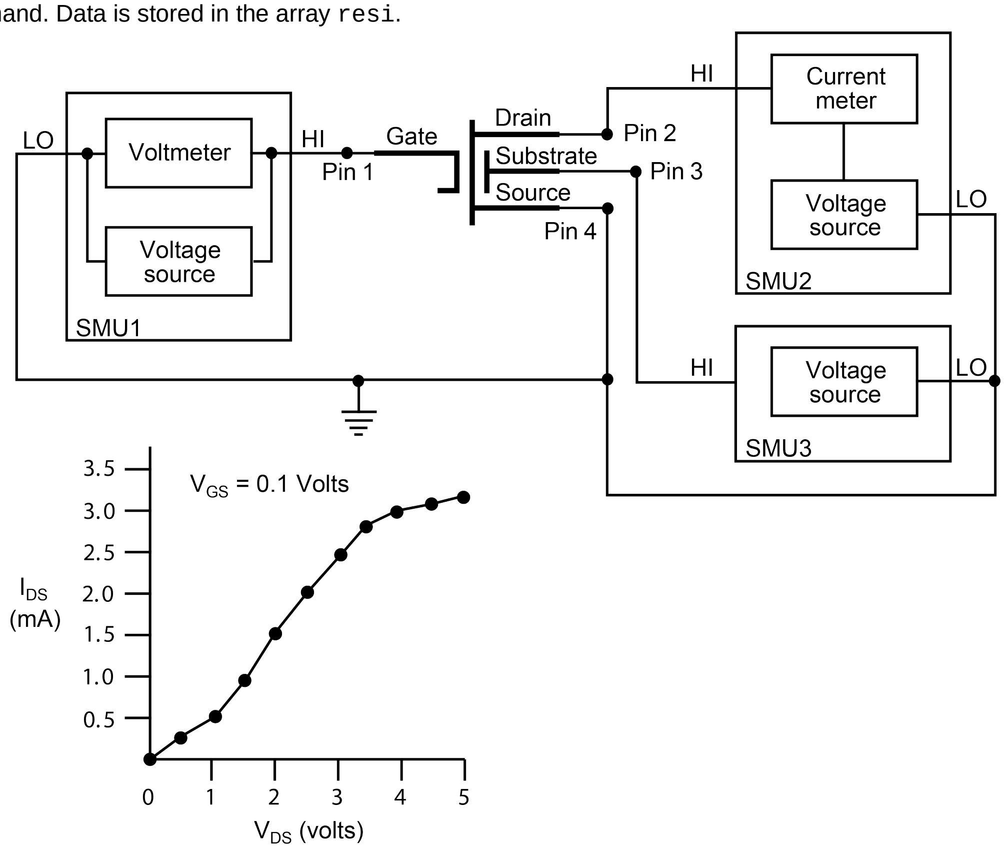
```
double ires[6];
conpin(SMU1, 10, 0);
conpin(2, GND 0);
forcev(SMU1, 5.0); /* Force 5 V to charge. */
delay(10); /* Wait for precharge. */
smeasi(SMU1, ires); /* Set up measurement. */
sweepv(SMU1, 5.0, 10.0, 5, 2.5E-3); /* Make the real measurement. */
```

If you change the source mode of the source-measure unit (SMU), it can modify the measure range. If the source mode is changed from voltage to current source (or from current to voltage source), the measure range may be changed to minimize variations in the SMU output level. See `rangeX` for the recommended command order.

Example

```
double resi[11], ssbiasv;
double vds[11];
.
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, -0.1); /* Apply a gate-to-source */
 /* voltage of -0.1V. */
rtfary(vds); /* Return force array*/
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 to 5 V. */
```

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

rangeX (on page 4-23)
rtfary (on page 2-27)
savgX (on page 2-27)
sintgX (on page 2-34)
smeasX (on page 2-35)

Section 5

LPT commands for CVUs

In this section:

LPT commands for the CVUs

The LPT commands for the 4210-CVU or 4215-CVU are listed in CVU commands (on page 1-7). LPT command details are presented here in alphabetic order.

adelay

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

Usage

```
int adelay(long numberOfPoints, double *delayArray);
```

numberOfPoints	Total number of sweep points
delayArray	An array of delay values (in seconds)

Details

NOTE

This command can be used with any of the `asweepX` commands. The following information pertains specifically to the 4210-CVU or 4215-CVU.

This command is used to define an array of delay values for the points in a voltage array sweep (`asweepv`). Each delay in the array is added to the delay time specified in `asweepv`. For example, if the array contained four delays (0.04 s, 0.05 s, 0.06 s, and 0.07 s) and the delay time specified in `asweepv` is 0.1 s, then the resulting delays are (0.14 s, 0.15 s, 0.16 s, and 0.17 s).

The number of delay values must match the number of points in the voltage array sweep. For example: Assume `asweepv` is configured to sweep four points, and the following delay times need to be set: 0.5 s, 0.25 s, 0.5 s, 0.25 s (in that order). With the delay time for `asweepv` set for 0 s, the array for the `adelay` command would be configured as follows:

```
delayArray(0) = 0.5
delayArray(1) = 0.25
delayArray(2) = 0.5
delayArray(3) = 0.25
```

Example

See Programming example #5 (on page 5-38), which shows how to set up an array of delay times for a voltage array sweep.

Also see

asweepX (on page 4-2)

asweepv

This command does a dc voltage sweep using an array of voltage values.

Usage

```
int asweepv(int instr_id, long numberOfPoints, double delayTime, double
    *forceArray);
```

instr_id	The instrument identification code of the 4210-CVU or 4215-CVU: `CVU1`
numberOfPoints	Total number of sweep points (`1` to `4096`)
delayTime	Delay time before each measurement in seconds (`0` to `999`)
forceArray	Array of dc voltage values

Details

NOTE

The following supplemental information on the voltage array sweep pertains specifically to the 4210-CVU or 4215-CVU. See `asweepX` in LPT commands for SMUs and general operations (on page 4-1) for additional information.

This command performs a dc voltage sweep using an array of voltage values. The number of voltage values in the array must match the *numberOfPoints* parameter value.

The *delayTime* parameter sets the user-programmed delay before each measurement. Note that there is an additional inherent system delay that occurs at the start of each step.

If different delay times are needed in the sweep, an array of delay time values can be set to adjust the delay times at each step (see `adelay` for details).

Use the `setfreq` and `setlevel` commands to set the ac drive frequency and voltage for the sweep.

Example

Refer to Programming example #4 (on page 5-37) for an example of a voltage array sweep.

Also see

adelay (on page 5-2)
asweepX (on page 4-2)
dsweepf (on page 5-8)
dsweepv (on page 5-10)
sweepf (on page 5-30)
setfreq (on page 5-19)
setlevel (on page 5-20)
sweepv (on page 5-33)

bsweepX

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

Usage

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

instr_id	The instrument identification code of the sourcing instrument
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 32,767)
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

`bsweepi` is only available for SMUs.

The `bsweep`*X* command is used with the `trig`*X*`g`, `trig`*X*`l`, or `trigcomp` 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 `bsweep`*X* command terminates the sweep and clears all sources by executing a `devclr` command internally. The standard `sweep`*X* 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 `bsweep`*X* command can also be used with the `smeas`*X*, `sintg`*X*, `savg`*X*, 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 `smeas`*X*, `sintg`*X*, `savg`*X*, or `clrscn` command. As multiple calls to `sweep`*X* 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 `bsweep`*X* command step.

When multiple calls to the `bsweep`*X* command are executed in the same test sequence, the arrays defined by calls to the `smeas`*X*, `sintg`*X*, or `savg`*X* command are all loaded sequentially. The results from the second call to the `bsweep`*X* command are appended to the results of the previous `bsweep`*X* command call. This can cause access violation errors if the arrays were not dimensioned for the absolute total. The measurement scan table remains intact until a `devint`, `execut`, or `clrscn` command completes.

Defining new test sequences using the `smeas`*X*, `sintg`*X*, or `savg`*X* command adds the command to the active measure list. The previous measurements are still defined and used; however, previous measurements for the second sweep can be eliminated by calling the `clrscn` command. New measurements are defined and used by calling the `smeasX`, `sintgX`, or `savgX` command after a `clrscn` command.

Note that changing the source mode of the SMU can modify the measure range. If the sourcing mode is changed from voltage to current sourcing (or from current to voltage sourcing), the measure range may be changed to minimize variations in the SMU output level. See `rangeX` for recommended command order.

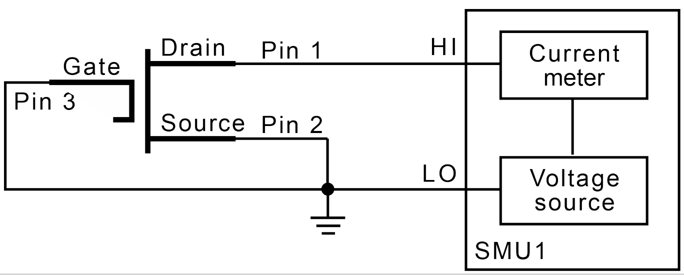
NOTE

It is recommended that you do not use GPIB instruments when doing sweeps with the `bsweepX` command. Refer to `kibdefint` for additional information.

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 2-2)
devclr (on page 4-9)
execut (on page 2-8)
kibdefint (on page 2-16)
rangeX (on page 4-23)
rtfary (on page 2-27)
savgX (on page 2-27)
sintgX (on page 2-34)
smeasX (on page 2-35)
sweepX (on page 4-29)
trigXg, trigXl (on page 2-37)
trigcomp (on page 2-36)

cvu_custom_cable_comp

This command determines the delays needed to accommodate custom cable lengths.

Usage

```
cvu_custom_cable_comp(int instr_id);
```

instr_id	The instrument identification code of the CVU (`CVU1`)

Details

The custom cable length measure gathers a specific set of timing coefficients to be applied during the C-V testing for a custom length cable. They are used to compensate the calibrated measurements made from the CVU.

Custom cable lengths are any lengths that are not 0 m, 1.5 m, or 3 m.

Once this command is run, these values are applied if you select a cable length of Custom in Clarius Tools > CVU Connection Compensation.

Possible return values are:

- `0`: OK
- `-907`: LPOT/LCUR fail
- `-908`: HPOT/HCUR fail

Also see

"Connection compensation" in the *Model 4200A-SCS Capacitance-Voltage Unit (CVU) User's Manual*

devclr

This command sets all sources to a zero state.

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. `devclr` is implicitly called by `clrcon`, `devint`, `execut`, and `tstdsl`.

For C-V testing, this command turns off the dc bias voltage.

Also see

clrcon (on page 7-2)
devint (on page 2-6)
execut (on page 2-8)
kibdefclr (on page 2-15)
tstdsl (on page 2-40)

devint

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

Usage

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

Details

Resets all active instruments, including the 4200A-CVIV, 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.

The `devint` command is implicitly called by the `execut` and `tstdsl` commands.

To abort a running `pulse_exec` pulse test, see `dev_abort`.

`devint` does the following:

1. Clears all sources by calling `devclr`.
2. Clears the matrix crosspoints by calling `clrcon`.
3. Clears the trigger tables by calling `clrtrg`.
4. Clears the sweep tables by calling `clrscn`.
5. Resets GPIB instruments by sending the string defined with `kibdefint`.
6. Resets the active instrument cards.

Instrument cards are reset in the following order:

1. SMU instrument cards
2. CVU instrument cards
3. Pulse instrument cards (4225-PMU or 4220-PGU)

The SMUs return to the following states:

- 100 µA and 10 V ranges
- Autorange on
- Voltage source
- 0 V dc bias

The 4210-CVU or 4215-CVU returns to the following states:

- 30 mV_{RMS} ac signal
- 0 V dc bias
- 100 kHz frequency
- Autorange on
- Cable length compensation set to 0 m
- Open/Short/Load compensation disabled

The 4225-PMU or 4220-PGU returns to the following states:

- The pulse mode is maintained. For example, if the pulse card is in Segment Arb mode, it is still in Segment Arb mode after the `devint` process is complete.
- 5 V and 10 mA ranges
- If in pulse mode:
  - Period of 1 µs
  - Transition times (rise and fall) of 100 ns
  - Width of 500 ns
  - Voltage high and low of 0 V
  - Load of 50 Ω
- If in segmented arb mode, Start Voltage is 0 V
- If in arbitrary waveform mode, Table Length is 100

Also see

clrcon (on page 7-2)
clrscn (on page 2-2)
clrtrg (on page 2-3)
dev_abort (on page 6-4)
devclr (on page 4-9)
kibdefint (on page 2-16)

dsweepf

This command performs a dual frequency sweep.

Usage

```
int dsweepf(int instr_id, double startf, double stopf, long *NumPts, double
    delaytime);
```

instr_id	The instrument identification code of the 4210-CVU or 4215-CVU: `CVU1`
startf	Initial frequency for the sweep
stopf	Final frequency for the first sweep
NumPts	Variable to receive the number of points sourced during the sweep
delaytime	Delay before each measurement (0 to 999 s)

Details

NOTE

Use the `sweepf` command to perform a single frequency sweep.

The CVU provides test frequencies from 1 kHz to 10 MHz. For the 4210-CVU, the frequencies are in the following steps:

- 1 kHz through 10 kHz in 1 kHz steps
- 10 kHz to 100 kHz in 10 kHz steps
- 100 kHz to 1 MHz in 100 kHz steps
- 1 MHz to 10 MHz in 1 MHz steps

If you are using a 4215-CVU, you can apply a resolution of 1 kHz to frequency values within the 1 kHz to 10 MHz limits. To set a frequency step size, set the `setmode KI_CVU_FREQ_STEPSIZE` modifier before calling `dsweepf()`. If `KI_CVU_FREQ_STEPSIZE`, is set to `0`, `dsweepf()` uses the discrete frequencies.

The frequency points to sweep are set using the *`startf`* and *`stopf`* parameters. If an entered value is not a supported frequency, the closest supported frequency is selected (for example, 15 kHz input selects 20 kHz). If a specified frequency is equidistant from two adjacent frequencies, it is rounded up to the higher frequency. The sweep can step forward (low frequency to high frequency) or it can step in reverse (high frequency to low frequency).

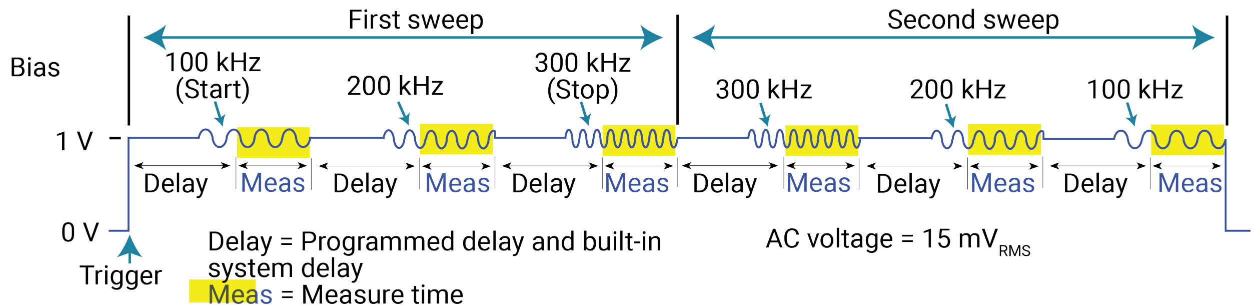
When the sweep is started, the CVU steps through all the supported frequency points from start to stop for the first sweep, and then repeats (in the reverse direction) from stop to start for the second sweep. For example, if the 4210-CVU start frequency is 800 kHz and the stop frequency is 3 MHz, the CVU steps through the frequency points 800 kHz, 900 kHz, 1 MHz, 2 MHz, 3 MHz, 3 MHz, 2 MHz, 1 MHz, 900 kHz, and 800 kHz.

The total number of sweep points is returned in the *`NumPts`* parameter. For the above example, *`NumPts`* is assigned a value of 10.

The *`delayTime`* parameter sets the delay that occurs before each measurement. Note that there is an inherent system overhead delay on each frequency step of the sweep.

Use the `forcev` command to set the dc bias level and `setlevel` command to set the ac drive voltage.

Dual frequency sweep example



Also see

forcev (on page 5-11)
setlevel (on page 5-20)
setmode (on page 5-21)
sweepf (on page 5-30)

dsweepv

This command performs a dual linear staircase voltage sweep.

Usage

```
int dsweepv(int instr_id, double startv, double stopv, long numSteps, double
    delaytime);
```

instr_id	The instrument identification code of the 4210-CVU or 4215-CVU: `CVU1`
startv	Initial force value for the sweep in volts (`-30` to `30`)
stopv	Final force value for the first sweep in volts (`-30` to `30`)
numSteps	Sets the number of points in the sweep (`1` to `4096`); see **Details**
delaytime	Delay before each measurement in seconds (`0` to `999`)

Details

This command is used to perform a dual staircase sweep (see the figure below). The linear step size to sweep is set using the *startv*, *stopv*, and *NumSteps* parameters. The linear step size for the sweep is then calculated as follows:

StepSize (in volts) = (*stopv* – *startv*) / (*numSteps*)

numSteps describes the first half of the sweep. For example, to do a dual sweep from 1 V to 10 V and back down in 1 V steps, set *numSteps* to 10. The result is a 20-point sweep (10 up and 10 down).

The first sweep can step forward (low voltage to high voltage) or it can step in reverse (high voltage to low voltage). After performing the first sweep, the second sweep will repeat in the reverse direction. For example, if configured to sweep from 1 V to 10 V, the second sweep will start at 10 V and step down to 1 V.

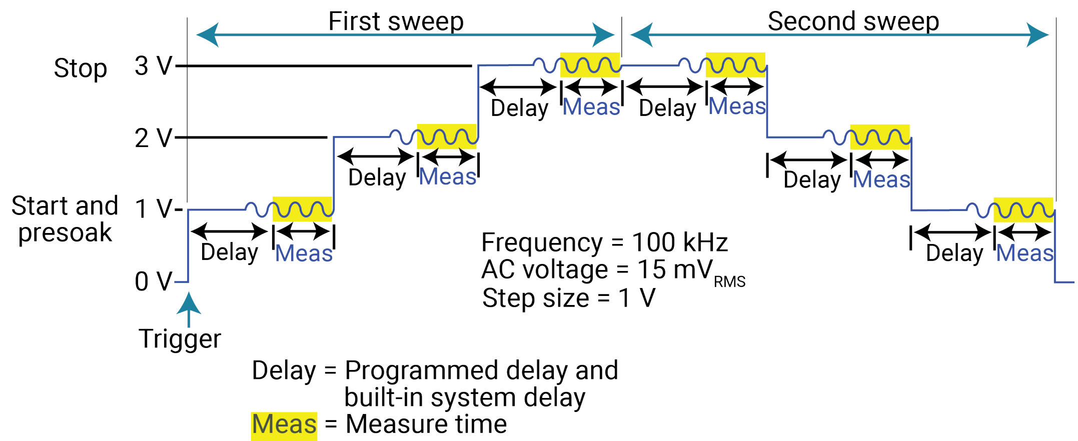
The *delayTime* parameter sets the delay that occurs before each measurement. Note that there is an inherent system overhead delay on each step of the sweep.

Use the `setfreq` and `setlevel` commands to set the ac drive frequency and voltage for the sweep.

NOTE

Use the `sweepv` command to perform a single linear staircase voltage sweep.

Example



Also see

asweepv (on page 5-3)
dsweepf (on page 5-8)
setfreq (on page 5-19)
setlevel (on page 5-20)
sweepf (on page 5-30)
sweepv (on page 5-33)

forcev

This command sets the dc bias voltage level.

Usage

`int forcev(int instr_id, double voltage);`

instr_id	The instrument identification code of the 4210-CVU or 4215-CVU: `CVU1`
voltage	The dc bias voltage level in volts (−30 to 30)

Details

This command sets a dc bias level for a single impedance measurement and a frequency sweep. Use the `setfreq` and `setlevel` commands to set the ac drive frequency and ac voltage for the sweep.

The dc source operates independently of the ac source. Changes to the level and state of the dc source take effect immediately; the ac frequency and source value are only used during `measz` operations.

Example

Programming example #1 (on page 5-34) makes a single impedance measurement. Note that the `rdelay` command provides a settling time before the measurement.

Also see

measz (on page 5-16)
setfreq (on page 5-19)
setlevel (on page 5-20)

getstatus

This command returns parameters that describe the state of the 4210-CVU or 4215-CVU.

Usage

```
int getstatus(int instr_id, long parameter, double *value);
```

instr_id	The instrument identification code of the 4210-CVU or 4215-CVU: CVU1
parameter	Parameter to be queried; the macros for the KI_CVU parameters are defined in lptparam.h; see **Details**
value	Returned value for the queried parameter

Details

Parameter:	**Returns:**
KI_CVU_LOAD_COMPENSATE	Load compensation: ON or OFF
KI_CVU_OPEN_COMPENSATE	Open compensation: ON or OFF
KI_CVU_SHORT_COMPENSATE	Short compensation: ON or OFF
KI_CVU_CABLE_CORRECT	Length setting for which the CVU card is correcting: 0, 1.5, or 3
KI_CVU_ACI_RANGE	AC current range in amps: 0 for autorange, or 1.5e-6, 50e-6 , or 1.5e-3 for fixed range (1.5 µA, 50 µA ,or 1.5 mA)
KI_CVU_ACI_PRESENT_RANGE	AC current range in amps: 1.5e-6, 50e-6 ,or 1.5e-3 (1.5 µA, 50 µA or 1.5 mA); returns range used for last measurement, even if on autorange
KI_CVU_ACV_LEVEL	AC voltage level in volts: ▪ 4210-CVU: 0.01 to 0.1 (10 mV_{RMS} to 100 mV_{RMS}) ▪ 4215-CVU: 0.01 to 1.0 (10 mV_{RMS} to 1 V_{RMS})
KI_CVU_APERTURE	A/D aperture time in PLCs: 0.006 to 10.002
KI_CVU_DCV_LEVEL	DC bias voltage level in volts: -30 to 30
KI_CVU_DELAY_FACTOR	Delay factor: 0 to 100
KI_CVU_FILTER_FACTOR	Filter factor: 0 to 707
KI_CVU_FREQUENCY	Drive frequency in Hertz: 1e3 to 10e6 (1 kHz to 10 MHz)
KI_CVU_MEASURE_MODEL	Impedance measure model: 0 through 5; see "Measurement model parameter values" table below)
KI_CVU_MEASURE_SPEED	Measurement speed (fast, normal, quiet, or custom)
KI_CVU_MEASURE_STATUS	Measurement status (for last reading); the measurement status codes are listed and explained in “Status codes” in the *Model 4200A-SCS Clarius User's Manual*
KI_CVU_MEASURE_TSTAMP	Measurement timestamp (for last reading)

Measurement model parameter values

Measurement model		Parameter value
ZTH	Impedance (Z) and phase (θ in degrees)	`KI_CVU_TYPE_ZTH` or `0`
RjX	Resistance and reactance	`KI_CVU_TYPE_RJX` or `1`
CpGp	Parallel capacitance and conductance	`KI_CVU_TYPE_CPGP` or `2`
CsRs	Series capacitance and resistance	`KI_CVU_TYPE_CSRS` or `3`
CpD	Parallel capacitance and dissipation factor	`KI_CVU_TYPE_CPD` or `4`
CsD	Series capacitance and dissipation factor	`KI_CVU_TYPE_CSD` or `5`
Y	YTH Admittance (1/Z) and phase (θ in degrees)	`KI_CVU_TYPE_YTH` or `7`

Also see

None

measf

This command returns the frequency sourced during a single measurement.

Usage

`int measf(int instr_id, double *freq);`

instr_id	The instrument identification code of the 4210-CVU or 4215-CVU: `CVU1`
freq	Returned frequency

Details

This command returns the present test frequency being used for a single impedance measurement. Use the `measz` command to make a single measurement.

NOTE

Use the `smeasf` or `smeasfRT` command to return the frequencies used for a sweep.

Also see

measz (on page 5-16)
smeasf (on page 5-23)
smeasfRT (on page 5-23)

meass

This command returns the status referenced to a single measurement.

Usage

```
int meass(INSTR_ID instr_id, double* result);
```

instr_id	The instrument identification code for the 4210-CVU or 4215-CVU: CVU1
result	Returned 32-bit measurement status

Details

This command returns the measurement status for a single measurement. See the following table for the results key.

This command returns the status in integer format. To compare this result to the Status Codes provided by Clarius, you must convert the values to hexadecimal.

Measurement status results key

Bit	Description	Value
31	Measurement timeout	Fault: 1 OK: 0
30 to 28	Not used	0
27	CVH1 ABB lock fault	Fault: 1 OK: 0
26	Not used	0
25 to 24	CVH1 overflow indicator (voltage and current)	Fault: 1 OK: 0
23 to 20	Not used	0
19	CVL1 ABB Lock Fault	Fault: 1 OK: 0
18	Not used	0
17 to 16	CVL1 overflow indicator (voltage and current)	Fault: 1 OK: 0
15 to 2	Not used	0
1 to 0	Current ac measurement range index	1.5 μA: 00 50 μA: 01 1.5 mA: 02

NOTE

Use the `measz` command to make a single measurement. Use the `smeass` command to return the measurement status values used for a sweep.

Also see

measz (on page 5-16)
smeass (on page 5-24)

meast

This command returns a timestamp referenced to a measurement or a system timer.

Usage

```
int meast(long timerID, double *timestamp);
```

timerID	The instrument identification code: CVU1, TIMER1, TIMER2, and so on
timestamp	Returned timestamp

Details

This command is used acquire the timestamp of the last single measurement, or return a timestamp referenced to a system timer.

When the *timerID* parameter is set for CVU1, calling the meast command after the call to perform a measurement (measz command) will return the timestamp for that measurement.

When the timerID parameter is set for a timer, the meast command can be called at any time and will return a timestamp that is referenced to a system timer. The enable command is used to start the timer (starts at zero when called).

NOTE

Use the smeast or smeastRT command to acquire timestamps for a sweep.

Examples

Programming example #1 (on page 5-34) acquires a timestamp for the measurement.

Programming example #2 (on page 5-35) measures the execution time of the code.

Also see

smeast (on page 5-25)
smeastRT (on page 5-26)

measv

This command returns the dc bias voltage sourced during a single measurement.

Usage

```
int measv(int instr_id, double *biasV);
```

instr_id	The instrument identification code of the 4210-CVU or 4215-CVU: CVU1
biasV	Returned dc bias voltage

Details

This command returns the dc bias voltage presently being used for a single measurement.

Use the measz command to make a single measurement.

NOTE

Use the `smeasv` or `smeasvRT` command to return the dc bias voltages used for a sweep.

Also see

measz (on page 5-16)
smeasv (on page 5-26)
smeasvRT (on page 5-27)

measz

This command makes an impedance measurement.

Usage

```
int measz(int instr_id, int model, int speed, double *result1 double *result2);
```

instr_id	The instrument identification code of the 4210-CVU or 4215-CVU: `CVU1`
model	Measurement model; see table in **Details**
speed	Measure speed: ■ `KI_CVU_SPEED_FAST`: Fast measurements (higher noise) ■ `KI_CVU_SPEED_NORMAL`: Selects a balance between speed and low noise ■ `KI_CVU_SPEED_QUIET`: Low-noise measurements ■ `KI_CVU_SPEED_CUSTOM`: Selects custom settings; the delay factor, filter factor, and aperture are set using the `setmode` command
result1	First result of the selected measure *model*
result2	Second result of the selected measure *model*

Details

This command makes a single impedance measurement.

Before calling `measz`, use the `forcev` command to set the dc bias level, the `setfreq` command to set the ac drive frequency, and the `setlevel` command to set the ac drive voltage.

The parameter values for the measurement *model* are listed in the following table.

Measurement model parameter values

Measurement model		Parameter value
ZTH	Impedance (Z) and phase (θ in degrees)	`KI_CVU_TYPE_ZTH` or `0`
RjX	Resistance and reactance	`KI_CVU_TYPE_RJX` or `1`
CpGp	Parallel capacitance and conductance	`KI_CVU_TYPE_CPGP` or `2`
CsRs	Series capacitance and resistance	`KI_CVU_TYPE_CSRS` or `3`
CpD	Parallel capacitance and dissipation factor	`KI_CVU_TYPE_CPD` or `4`
CsD	Series capacitance and dissipation factor	`KI_CVU_TYPE_CSD` or `5`
Y	YTH Admittance (1/Z) and phase (θ in degrees)	`KI_CVU_TYPE_YTH` or `7`

NOTE

Use the `smeasz` or `smeaszRT` command to measure and return the impedance readings for a sweep.

Also see

forcev (on page 5-11)
setfreq (on page 5-19)
setlevel (on page 5-20)
setmode (on page 5-21)
smeasz (on page 5-28)
smeaszRT (on page 5-29)

rangei

This command selects an impedance measurement range.

Usage

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

instr_id	The instrument identification code of the CVU: `CVU1`
range	Impedance measure range in amps: `0`, `1e-6`, `30e-6`, or `1e-3` (0, 1 µA, 30 µA, or 1 mA)

Details

Use this command to set the CVU to a current measure range for impedance measurements. To select autorange, set *range* to `0`. The CVU automatically goes to the most sensitive (optimum) range to make the measurement. This is the same as calling the `setauto` command.

The other `range` parameter values select a fixed measure range. The CVU remains on the fixed range until autorange is enabled or the CVU is reset (`devint` called).

Example

Programming example #1 (on page 5-34) uses the 1 mA measure range for the impedance measurement.

Also see

devint (on page 2-6)
setauto (on page 5-19)

rtfary

This command returns the array of force values used during the subsequent voltage or frequency sweep.

Usage

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

forceArray	Array of force values for voltage or frequency

Details

This command returns an array of voltage or frequency force values for a sweep. Send this command before calling any sweep command.

NOTE

To prevent a memory exception error, make sure that the array that will receive the sourced values is large enough.

The following examples show the proper command sequence for using `rtfary`:

Example 1: Valid command sequence for voltage sweep	Example 2: Valid command sequence for frequency sweep
smeasz	smeasz
smeast	rtfary
rtfary	dsweepf
dsweepv	

Example

Programming example #2 (on page 5-35) returns the array of force values for the voltage sweep.

Also see

None

setauto

This command selects the automatic measurement range.

Usage

```
int setauto(int instr_id);
```

instr_id	The instrument identification code of the CVU: CVU1

Details

This command sets the CVU for autorange measurements. When `setauto` is called, the CVU goes to the most sensitive range to make the measurement. Calling `devint` also selects autorange.

You can also use the `rangei` command to enable autorange or select a fixed measurement range. Autorange remains enabled until a fixed range is selected.

Example

Programming examples (on page 5-34) 2 through 5 use autorange for impedance measurements.

Also see

devint (on page 2-6)
rangei (on page 5-17)

setfreq

This command sets the frequency for the ac drive.

Usage

```
int setfreq(int instr_id, double frequency);
```

instr_id	The instrument identification code of the CVU: CVU1
frequency	Frequency of the ac drive

Details

The CVU provides test frequencies from 1 kHz to 10 MHz. For the 4210-CVU, the frequencies are in the following steps:

- 1 kHz through 10 kHz in 1 kHz steps
- 10 kHz to 100 kHz in 10 kHz steps
- 100 kHz to 1 MHz in 100 kHz steps
- 1 MHz to 10 MHz in 1 MHz steps

If you are using a 4215-CVU, you can apply a resolution of 1 kHz to frequency values within the 1 kHz to 10 MHz limits.

If an entered value is not a supported frequency, the closest supported frequency is selected (for example, with the 4210-CVU, 15 kHz input selects 20 kHz). You can use the `getstatus` command to retrieve the selected frequency value.

The ac drive (ac voltage level and frequency) does not turn on until a measurement is made. The ac drive turns off after the measurement is completed. Note that the dc voltage source stays on for the whole test.

Example

Programming examples (on page 5-34) 1, 2, 4 and 5 use the `setfreq` command to set the ac drive frequency.

Also see

getstatus (on page 5-12)
setlevel (on page 5-20)

setlevel

This command sets the voltage level of the ac drive.

Usage

`int setlevel(int instr_id, double signalLevel);`

instr_id	The instrument identification code of the CVU: `CVU1`
signalLevel	Voltage level of the ac drive in volts: ■ 4210-CVU: 10 mV to 100 mV_{RMS} ■ 4215-CVU: 10 mV to 1 V_{RMS}

Details

The ac drive (ac voltage level and frequency) does not turn on until a measurement is made. The ac drive turns off after the measurement is completed. The dc voltage source stays on for the whole test.

Example

All the Programming examples (on page 5-34) use the `setlevel` command to set the ac drive voltage.

Also see

setfreq (on page 5-19)

setmode (4210-CVU or 4215-CVU)

This command sets operating modes specific to the 4210-CVU or 4215-CVU.

Usage

```
int setmode(int instr_id, long modifier, double value);
```

instr_id	The instrument identification code of the CVU: CVU1
modifier	Specific operating characteristic to change; see table in **Details**
value	Parameter value for the *modifier*

Details

NOTE

The following information is specific to CVUs. For details on using setmode for other instruments, see the setmode (on page 2-32) command.

The setmode command allows control over the following CVU operating characteristics:

- Connection compensation control for open, short and load. When disabled, saved compensation constants are not applied to the measurements. Whenever the connection setup has changed, connection compensation needs to be performed to acquire and save new compensation constants. Connection compensation is performed from Clarius.
- Setting for cable length compensation (0 m, 1.5 m, or 3 m). This setting is made from the window that is used to enable compensation.
- 4215-CVU only: Setting the step size for a frequency sweep. Must be called before sweepf() or dsweepf().
- Settings (delay factor, filter factor and aperture) for KI_CUSTOM measurement speed, which is set by measz, smeasz, or smeaszRT.

For detail on connection compensation, refer to "Connection compensation" in the *Model 4200A-SCS Capacitance-Voltage Unit (CVU) User's Manual*.

Parameters for *modifier* and *value*

modifier	***value***	**Comment**
KI_CVU_OPEN_COMPENSATE	0 = OFF 1 = ON	Enable or disable compensation constants for open.
KI_CVU_SHORT_COMPENSATE	0 = OFF 1 = ON	Enable or disable compensation constants for short.
KI_CVU_LOAD_COMPENSATE	0 = OFF 1 = ON	Enable or disable compensation constants for load.
KI_CVU_CABLE_CORRECT	Cable length setting:	
	0.0	No cable compensation
	1.5	1.5 m CVU cable
	3.0	3.0 m CVU cable

Parameters for *modifier* and *value*

modifier	*value*	Comment
	5.0	1.5 m CVIV cable; 2-wire mode
	6.0	1.5 m CVU to CVIV cable with 0.75 m CVIV to DUT cable; 4-wire mode
	7.0	1.5 m CVU to CVIV cable with 0.61 m CVIV to DUT cable; 4-wire mode
KI_CVU_SET_CONSTANT_FILE	0 to 1000	The number in the file name that contains the open, short, and load compensation values for the CVU.
KI_CVU_MEASURE_SPEED	KI_CVU_SPEED_FAST	Fast measurements (higher noise).
	KI_CVU_SPEED_NORMAL	Balance between speed and low-noise.
	KI_CVU_SPEED_QUIET	Low-noise measurements.
	KI_CVU_SPEED_CUSTOM	Custom settings (see next modifiers).
KI_CVU_APERTURE KI_CVU_DELAY_FACTOR KI_CVU_FILTER_FACTOR	0.006 to 10.002 PLCs 0 to 100 0 to 707	Settings for the CUSTOM speed setting (see previous *modifier*).
KI_CVU_AC_SRC_HI KI_CVU_AC_MEAS_LO	1 or 2 (setting HI side to 1 sets LO side to 2, and vice versa)	Use to specify the ac source HI slice and ac source LO side.
KI_CVU_DC_SRC_HI KI_CVU_DC_SRC_LO	1 or 2 (setting HI side to 1 sets LO side to 2, and vice versa)	Use to specify the dc source HI slice and dc source LO side.
KI_CVU_DCV_OFFSET	−30 to +30 (default is 0)	Sets the dc bias offset (in volts).
KI_CVU_FREQ_STEPSIZE	0 to use discrete frequencies 1000 to 9.999e6	4215-CVU only. A devint() call resets the size to the default. If setmode() is not called before sweepf(), sweepf() uses the discrete frequencies.
KI_CVU_MEASURE_MODEL	KI_CVU_TYPE_ZTH	Impedance and phase (degrees).
	KI_CVU_TYPE_RJX	Resistance and reactance.
	KI_CVU_TYPE_CPGP	Parallel capacitance and conductance.
	KI_CVU_TYPE_CSRS	Series capacitance and resistance.
	KI_CVU_TYPE_CPD	Parallel capacitance and dissipation factor.
	KI_CVU_TYPE_CSD	Series capacitance and dissipation factor.
	KI_CVU_TYPE_YTH	1/X.

Also see

dsweepf (on page 5-8)
measz (on page 5-16)
setmode (on page 2-32)
smeasz (on page 5-28)
smeaszRT (on page 5-29)
sweepf (on page 5-30)

smeasf

This command returns the frequencies used for a sweep.

Usage

```
int smeasf(int instr_id, double *freq_arr);
```

instr_id	The instrument identification code of the CVU: CVU1
freq_arr	Returned array of test frequencies

Details

This command returns the present test frequencies used for a sweep. The frequency values are returned in an array. The frequency values are posted to Clarius in Analyze after the test has finished.

NOTE

You can use the `smeasfRT` command to return sourced sweep frequency values in an array. It posts the frequency values to Clarius in real time.

NOTE

Use the `measf` command to return the frequency used for a single measurement.

Also see

measf (on page 5-13)
smeasfRT (on page 5-23)

smeasfRT

This command returns the sourced frequencies (in real time) for a sweep.

Usage

```
int smeasfRT(int instr_id, double *freq_arr, char *colname);
```

instr_id	The instrument identification code of the CVU: CVU1
freq_arr	Returned array of test frequencies
colname	Column name (character string) to pass into Clarius for the data sheet column

Details

Like the `smeasf` command, the test frequencies for a sweep are returned in an array. However, the frequency values are posted to the Clarius Analyze sheet and graph in real time (after each step of the sweep is executed).

Note that the values are only available in real time if Clarius is running. Otherwise, they are stored in an array in the usual fashion.

Example

`smeasfRT(CVU1, freq_arr, "freq_arr");`
This command posts the frequency values into the Clarius Analyze sheet under a column named `freq_arr`.

Also see

smeasf (on page 5-23)

smeass

This command returns the measurement status values for every point in a sweep.

Usage

```
int meass(INSTR_ID instr_id, double* result);
```

instr_id	The instrument identification code for the CVU: `CVU1`
result	Returned array of 32-bit measurement status values

Details

This command returns the measurement status values for every point in a sweep. The values are returned in an array. See the following table for the results key.

This command returns the status in integer format. To compare this result to the Status Codes provided by Clarius, you must convert the values to hexadecimal.

Measurement status results key

Bit	Description	Value
31	Measurement timeout	Fault: `1` OK: `0`
30 to 28	Not used	`0`
27	CVH1 ABB lock fault	Fault: `1` OK: `0`
26	Not used	`0`
25 to 24	CVH1 overflow indicator (voltage and current)	Fault: `1` OK: `0`
23 to 20	Not used	`0`
19	CVL1 ABB Lock Fault	Fault: `1` OK: `0`
18	Not used	`0`
17 to 16	CVL1 overflow indicator (voltage and current)	Fault: `1` OK: `0`
15 to 2	Not used	`0`
1 to 0	Current ac measurement range index	1.5 μA: `00` 50 μA: `01` 1.5 mA: `02`

NOTE

Use the `meass` command to return the measurement status for a single measurement.

Also see

meass (on page 5-14)

smeast

This command returns timestamps referenced to sweep measurements or a system timer.

Usage

```
int meast(long timerID, double *tarray);
```

timerID	The ID of the CVU or timer: `CVU1`, `TIMER1`, `TIMER2`, and so on
tarray	Returned array of timestamps

Details

This command acquires the timestamp for each measurement step of a sweep. The timestamps are returned in an array. The timestamps are posted to Clarius Analyze after the test has finished.

NOTE

You can also use the `smeastRT` command to return timestamps in an array. It posts the frequency values to Clarius in real time.

The timestamp can be referenced to the CVU (*timerID* = `CVU1`) or to a system timer (for example, *timerID* = `TIMER1`). This command is similar to the `meast` command, but is synchronized with a sweep to return a timestamp referenced to each measurement. If you need a timestamp for a single measurement, use the `meast` command.

NOTE

LPT maintains a list of measurements to be done at each sweep point after the forcing instrument has stepped its source (V, I, or F). The *smeasX* and `smeasXRT` commands register the measurement with a master list. If the time measurement precedes the Z measurement, then the wrong timestamp is returned (the one from the previous measurement).

Example

Programming example #2 (on page 5-35) acquires a timestamp for each measurement in the sweep.

Also see

meast (on page 5-15)
smeastRT (on page 5-26)

smeastRT

This command returns timestamps (in real time) referenced to sweep measurements or a system timer.

Usage

```
int meastRT(long timerID, double *tarray, char *colname);
```

timerID	The ID of the CVU or timer: CVU1, TIMER1, TIMER2, and so on
tarray	Returned array of timestamps
colname	Column name to pass into Clarius (case-sensitive character string)

Details

Returns the timestamps are returned in an array and posts the timestamps to the Clarius Analyze sheet and graph in real time. Each timestamp appears in the sheet and graph after each measurement is made.

Note that the values are only available in real time if Clarius is running. Otherwise, they are stored in an array.

The *colname* parameter specifies the name for the data sheet column in Clarius.

NOTE

LPT maintains a list of measurements to be done at each sweep point after the forcing instrument has stepped its source (V, I, or F). The *smeasX* and smeas*X*RT commands register the measurement with a master list. If the time measurement precedes the Z measurement, then the wrong timestamp is returned (the one from the previous measurement).

Example

`smeastRT(CVU1, time_arr, "time_arr");`
This command posts the timestamp values into the Clarius Analyze sheet in the column named `time_arr`.

Also see

smeast (on page 5-25)

smeasv

This command returns the dc bias voltages used for a sweep.

Usage

```
int smeasv(int instr_id, double *varray);
```

instr_id	The instrument identification code of the CVU: CVU1
varray	Returned array of dc bias voltages

Details

This command returns the dc bias voltages used in a sweep. The values are returned in an array. The voltage values are posted to the Clarius Analyze sheet and graph after the test has finished.

NOTE

You can also use the `smeasvRT` command to return sourced sweep dc bias voltage values in an array. It posts the voltage values to Clarius in real time.

NOTE

Use the `measv` command to return the dc bias voltage used for a single measurement.

Also see

measv (on page 5-15)
smeasvRT (on page 5-27)

smeasvRT

This command returns the sourced dc bias voltages (in real time) for a sweep.

Usage

```
int smeasvRT(int instr_id, double *varray, char *colname);
```

instr_id	The instrument identification code of the CVU: `CVU1`
varray	Returned array of dc bias voltages
colname	Column name to pass into Clarius (character string)

Details

This command is similar to `smeasv` command. It returns the sourced dc bias voltages for a sweep in an array. However, the voltage values are posted to the Clarius Analyze sheet and graph in real time. Each voltage value appears in the sheet and graph after each step of the sweep is executed.

Note that the values are only available in real time if Clarius is running. Otherwise, they are stored in an array.

The *colname* parameter specifies a name for the data sheet column in Clarius.

Example

```
smeasvRT(CVU1, volt_arr, "volt_arr");
```

This command posts the voltage values into the Clarius data sheet under a column named `volt_arr`.

Also see

smeasv (on page 5-26)

smeasz

This command performs impedance measurements for a sweep.

Usage

```
int smeasz(int instr_id, long model, long speed, double *result1, double *result2);
```

instr_id	The instrument identification code of the CVU: `CVU1`
model	Measure model; refer to "Measurement model parameter values" table in the **Details**
speed	Speed settings: ▪ `KI_CVU_SPEED_FAST`: Fast measurements (higher noise) ▪ `KI_CVU_SPEED_NORMAL`: Selects a balance between speed and low noise ▪ `KI_CVU_SPEED_QUIET`: Low-noise measurements ▪ `KI_CVU_SPEED_CUSTOM`: Selects custom settings; the delay factor, filter factor, and aperture are set using the `setmode` command
result1	Array of the first result of the selected measure model
result2	Array of the second result of the selected measure model

Details

This command makes an impedance measurement on each step of a voltage or frequency sweep. The measured values for a sweep are returned in arrays. The measured readings are posted to the Clarius Analyze sheet and graph after the test has finished.

Before calling `smeasz`, use the `forcev` command to set the dc bias level, the `setfreq` command to set the ac drive frequency and the `setlevel` command to set the ac drive voltage.

Measurement model parameter values

Measurement model		**Parameter value**
ZTH	Impedance (Z) and phase (θ in degrees)	`KI_CVU_TYPE_ZTH` or `0`
RjX	Resistance and reactance	`KI_CVU_TYPE_RJX` or `1`
CpGp	Parallel capacitance and conductance	`KI_CVU_TYPE_CPGP` or `2`
CsRs	Series capacitance and resistance	`KI_CVU_TYPE_CSRS` or `3`
CpD	Parallel capacitance and dissipation factor	`KI_CVU_TYPE_CPD` or `4`
CsD	Series capacitance and dissipation factor	`KI_CVU_TYPE_CSD` or `5`
Y	YTH Admittance (1/Z) and phase (θ in degrees)	`KI_CVU_TYPE_YTH` or `7`

NOTE

You can also use the `smeaszRT` command to measure and return sweep impedance measurements. It posts the measured readings to Clarius in real time.

NOTE

To return a single impedance measurement, use `measz`.

Example

Programming examples (on page 5-34) 2 through 5 use the `smeasz` command for impedance measurements.

Also see

forcev (on page 5-11)
measz (on page 5-16)
setfreq (on page 5-19)
setlevel (on page 5-20)
setmode (on page 5-21)
smeaszRT (on page 5-29)

smeaszRT

This command makes and returns impedance measurements for a voltage or frequency sweep in real time.

Usage

```
int smeaszRT(int instr_id, long model, long speed, double *result1, char *colname1,
   double *result2, char *colname2);
```

instr_id	The instrument identification code of the CVU: `CVU1`
model	Measure model (see "Measurement model parameter values" table in **Details**)
speed	Speed settings: ▪ `KI_CVU_SPEED_FAST`: Fast measurements (higher noise) ▪ `KI_CVU_SPEED_NORMAL`: Selects a balance between speed and low noise ▪ `KI_CVU_SPEED_QUIET`: Low-noise measurements ▪ `KI_CVU_SPEED_CUSTOM`: Selects custom settings; the delay factor, filter factor, and aperture are set using the `setmode` command
result1	Array of the first result of the selected measure model
colname1	Column name to pass into Clarius for *result1* array (character string)
result2	Array of the second result of the selected measure model
colname2	Column name to pass into Clarius for *result2* array (character string)

Details

This command is similar to the `smeasz` command; both commands return the measured impedance readings for a sweep returned in arrays. However, the readings from `smeaszRT` are posted to the Clarius Analyze sheet and graph in real time. Two measurement results appear in the sheet and graph after each step of the sweep is executed.

Note that the values are only available in real-time if Clarius is running. Otherwise, they are stored in an array.

The *colname1* and *colname2* parameters specify names for data sheet columns in Clarius.

Measurement model parameter values

Measurement model		Parameter value
ZTH	Impedance (Z) and phase (θ in degrees)	`KI_CVU_TYPE_ZTH` or `0`
RjX	Resistance and reactance	`KI_CVU_TYPE_RJX` or `1`
CpGp	Parallel capacitance and conductance	`KI_CVU_TYPE_CPGP` or `2`
CsRs	Series capacitance and resistance	`KI_CVU_TYPE_CSRS` or `3`
CpD	Parallel capacitance and dissipation factor	`KI_CVU_TYPE_CPD` or `4`
CsD	Series capacitance and dissipation factor	`KI_CVU_TYPE_CSD` or `5`
Y	YTH Admittance (1/Z) and phase (θ in degrees)	`KI_CVU_TYPE_YTH` or `7`

Example

```
smeaszRT(CVU1, 2, KI_NORMAL, result1, "result1", result2, "result2");
```

This command posts the results into the Clarius data sheet under columns named *result1_arr* and *result2_arr*.

Also see

smeasz (on page 5-28)

sweepf

This command performs a frequency sweep.

Usage

```
int sweepf(int instr_id, double startf, double stopf, long *NumPts, double
    delaytime);
```

instr_id	The instrument identification code of the CVU: `CVU1`
startf	Initial frequency for the sweep in Hertz
stopf	Final frequency for the sweep in Hertz
NumPts	Query the number of sweep points
delayTime	Delay before each measurement in seconds: 0 to 999

Details

The CVU provides test frequencies from 1 kHz to 10 MHz. For the 4210-CVU, the frequencies are in the following steps:

- 1 kHz through 10 kHz in 1 kHz steps
- 10 kHz to 100 kHz in 10 kHz steps
- 100 kHz to 1 MHz in 100 kHz steps
- 1 MHz to 10 MHz in 1 MHz steps

If you are using a 4215-CVU, you can apply a resolution of 1 kHz to frequency values within the 1 kHz to 10 MHz limits. To set a frequency step size, set the `setmode KI_CVU_FREQ_STEPSIZE` modifier before calling `sweepf()`. If `KI_CVU_FREQ_STEPSIZE` is set to `0`, `sweepf()` uses the discrete frequencies.

The frequency points to sweep are set using the *`startf`* and *`stopf`* parameters. If an entered value is not a supported frequency, the closest supported frequency is selected (for example, 15 kHz input selects 20 kHz). The sweep can step forward from low frequency to high frequency or it can step in reverse from high frequency to low frequency.

When the sweep is started, the CVU steps through all the supported frequency points from start to stop. For example, if the 4210-CVU start frequency is 800 kHz and the stop frequency is 3 MHz, the CVU steps through the frequency points 800 kHz, 900 kHz, 1 MHz, 2 MHz, and 3 MHz.

The *`NumPts`* query returns the total number of sweep points. For the above example, *`NumPts`* returns 5.

The *`delayTime`* parameter sets the delay that occurs before each measurement. Note that there is an inherent system overhead delay on each frequency step of the sweep.

Use the `forcev` command to set the dc bias level and `setlevel` command to set the ac drive voltage.

NOTE

Use the `dsweepf` command to perform a dual frequency sweep.

Example

Programming example #3 (on page 5-36) performs a frequency sweep.

Also see

asweepv (on page 5-3)
dsweepf (on page 5-8)
dsweepv (on page 5-10)
forcev (on page 5-11)
setlevel (on page 5-20)
setmode (CVU) (on page 5-21)
sweepv (on page 5-33)

sweepf_log

This command performs a logarithmic frequency sweep using a 4215-CVU instrument. This is not available for the 4210-CVU.

Usage

```
int sweepf_log(int instr_id, double startf, double stopf, long *numPoints, double
   delaytime);
```

instr_id	The instrument identification code of the 4215-CVU: `CVU1`
startf	Initial frequency for the sweep
stopf	Final frequency for the sweep
numPoints	The number of sweep points
delayTime	Delay before each measurement in seconds: 0 to 999

Details

This command is used to perform a logarithmic base 10 frequency sweep.

The frequency points to sweep are set using the *startf* and *stopf* parameters. If an entered value is not a supported frequency, the closest supported frequency is selected. The sweep can step forward from low frequency to high frequency or it can step in reverse from high frequency to low frequency.

When the sweep is started, the CVU steps through all the supported frequency points from start to stop. You can apply a resolution of 1 kHz to frequency values within the 1 kHz to 10 MHz limits

The *delayTime* parameter sets the delay that occurs before each measurement. Note that there is an inherent system overhead delay on each frequency step of the sweep.

Use the `forcev` command to set the dc bias level and `setlevel` command to set the ac drive voltage.

A logarithmic sweep is also provided through the `cvuulib` user library.

Also see

asweepv (on page 5-3)
forcev (on page 5-11)
setlevel (on page 5-20)
setmode (4210-CVU or 4215-CVU) (on page 5-21)
sweepv (on page 5-33)

sweepv

This command performs a linear staircase dc voltage sweep.

Usage

```
int sweepv(int instr_id, double startv, double stopv, long NumSteps, double
    delaytime);
```

instr_id	The instrument identification code of the CVU: CVU1
startv	Initial force value for the sweep in volts: –30 to 30
stopv	Final force value for the sweep in volts: –30 to 30
NumSteps	Number of steps in the sweep: 1 to 4096
delaytime	Delay before each measurement in seconds: 0 to 999

Details

This command is used to perform a staircase sweep. The linear step size to sweep is set using the *startv*, *stopv*, and *NumSteps* parameters. The linear step size for the sweep is calculated as follows:

Step size (in volts) = (*stopv* – *startv*) / *NumSteps*

The sweep can step forward (low voltage to high voltage) or it can step in reverse (high voltage to low voltage).

The *delaytime* parameter sets the delay that occurs before each measurement. Note that there is an inherent system overhead delay on each step of the sweep.

Use the `setfreq` and `setlevel` commands to set the ac drive frequency and voltage for the sweep.

NOTE

Use the `dsweepv` command to do a dual linear staircase voltage sweep.

Example

Refer to Programming example #2 (on page 5-35) for an example of a single staircase voltage sweep.

Also see

asweepv (on page 5-3)
dsweepf (on page 5-8)
dsweepv (on page 5-10)
setfreq (on page 5-19)
setlevel (on page 5-20)
sweepf (on page 5-30)

Programming examples

These programming examples provide examples of how to use LPT commands to:

- Make a single CsRs impedance measurement: Programming example #1 (on page 5-34)
- Do a single staircase sweep and measure CpGp for each step: Programming example #2 (on page 5-35)
- Do a single frequency sweep and measure CpGp for each step: Programming example #3 (on page 5-36)
- Do a voltage array sweep: Programming example #4 (on page 5-37)
- Do a voltage array sweeps with an array of delay values used for the sweep: Programming example #5 (on page 5-38)

Programming example #1

Performs a single CsRs impedance measurement. Test parameters:

- DC bias voltage = 1 V
- AC drive frequency = 100 kHz
- AC drive voltage = 15 mV_{RMS}
- Measure model = CsRs

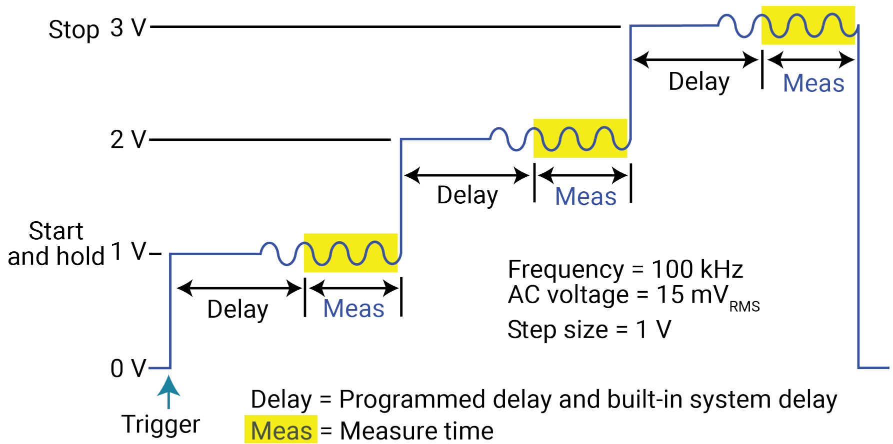
This example also acquires a timestamp for the measurement.

```
double result1, result2, timeStamp;
forcev(CVU1, 1);          /*  Set DC bias to 1 V. */
setfreq(CVU1, 100e3);     /*  Set AC drive frequency to 100 kHz. */
setlevel(CVU1, 15e-3);    /*  Set AC drive voltage to 15 mV RMS. */
rdelay(0.1);              /*  Set settling time to 100 ms. */
rangei(CVU1, 1.0e-3);     /* Select 1 mA measure range. */
measz(CVU1, KI_CVU_TYPE_CSRS, KI_CVU_SPEED_NORMAL, &result1, &result2);
                                   /*  Measure CsRs. */
meast(CVU1, &timeStamp);  /*  Return timestamp for measurement. */
devint();                 /*  Reset CVU. */
```

Programming example #2

Performs a single staircase voltage sweep, as shown in the figure below.

Figure 2: Voltage sweep example



CpGp is measured on each step of the sweep.

Test parameters:

- AC drive frequency = 100 kHz
- AC drive voltage = 15 mV_{RMS}
- Measure model = CpGp
- Measure range = Auto
- Sweep mode = single
- Start voltage = 1 V
- Stop voltage = 3 V
- Number of steps = 3
- Delay = 50 ms

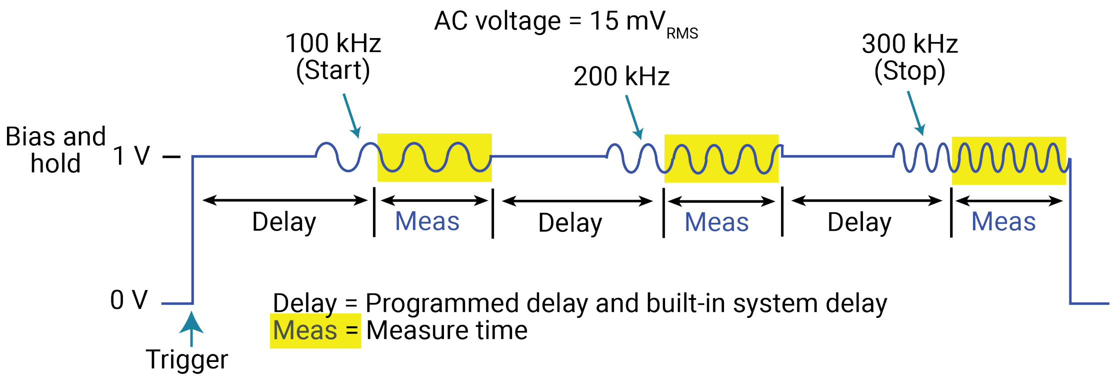
This example also returns a timestamp for each measurement and measures the execution time of the code.

```
double result1[4], result2[4], timeStamp1[4], timeStamp2;
enable(TIMER1);               /*  Start timer at 0 seconds. */
setfreq(CVU1, 100e3);         /*  Set AC drive frequency to 100 kHz. */
setlevel(CVU1, 15e-3);        /*  Set AC drive voltage to 15 mV RMS. */
setauto(CVU1);                /* Select auto measure range. */
smeasz(CVU1, KI_CVU_TYPE_CPGP, KI_CVU_SPEED_NORMAL, result1, result2);
                                       /*  Configure CpGp measurements. */
smeast(CVU1, timeStamp1);     /* Return timestamps for all measurements. */
rtfary(forceArray);           /*  Return array of force voltages. */
sweepv(CVU1, 1, 3, 3, 0.05);  /*  Configure and perform sweep. */
meast(TIMER1, &timeStamp2);   /*  Return execution time for above. */
                                       /*  block of code. */
devint();                              /*  Reset CVU. */
```

Programming example #3

Performs a single frequency sweep shown in the following figure.

Figure 3: Frequency sweep example



CpGp is measured on each frequency step of the sweep.

Test parameters:

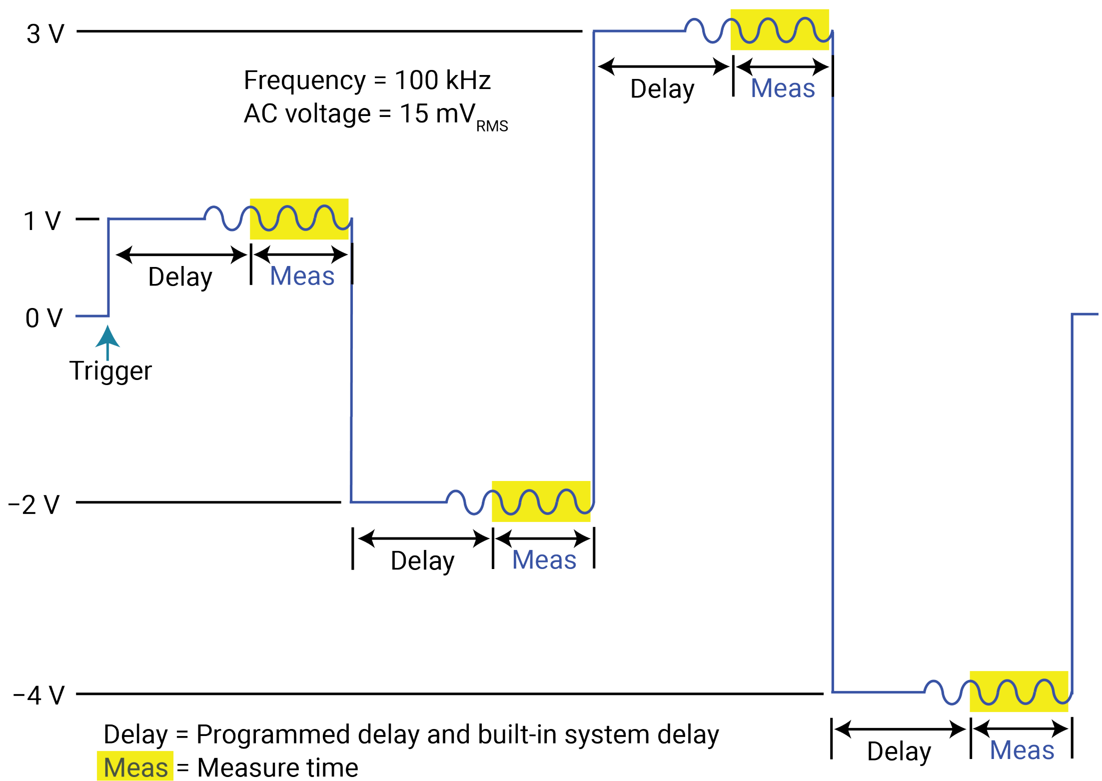
- AC drive voltage = 15 mV_{RMS}
- Measure mode = CpGp
- Measure range = Auto
- Start frequency = 100 kHz
- Stop frequency = 300 kHz
- Number of frequency steps = 3 (this value is returned from the command to the `NumPts` variable, and not passed by the user)
- Delay = 50 ms

```
double result1[6], result2[6], forceArray[6];
long NumPts;
setlevel(CVU1, 15e-3);        /*  Set AC drive voltage to 15 mV RMS. */
setauto(CVU1);                /* Select auto measure range. */
smeasz(CVU1, KI_CVU_TYPE_CPGP, KI_CVU_SPEED_NORMAL, result1, result2);
                                      /*  Configure CpGp measurements. */
rtfary(forceArray);           /*  Return array of force frequencies. */
dsweepf(CVU1, 100e3, 300e3, &NumPts, 0.05);  /*  Configure and perform sweep. */
devint();                     /*  Reset CVU. */
```

Programming example #4

This example performs a voltage array sweep as shown in the figure below.

Figure 4: Voltage array sweep example



The above figure shows an example of a voltage array sweep using the following voltage values: 1 V, -2 V, 3 V, -4 V. The voltage array is configured as follows:

```
forceArray[0] = 1
forceArray[1] = -2
forceArray[2] = 3
forceArray[3] = -4
```

CpGp is measured on each point of the sweep.

Test parameters:

- AC drive frequency = 100 kHz
- AC drive voltage = 15 mV_{RMS}
- Measure model = CpGp
- Measure range = Auto
- Force array = 1 V, -2 V, 3 V, -4 V
- Number of sweep points = 4
- Delay = 50 ms

```
double result1[4], result2[4], forceArray[4];
setfreq(CVU1, 100e3);          /*  Set AC drive frequency to 100 kHz. */
setlevel(CVU1, 15e-3);         /*  Set AC drive voltage to 15 mV RMS. */
setauto(CVU1);                 /* Select auto measure range. */
smeasz(CVU1, KI_CVU_TYPE_CPGP, KI_CVU_SPEED_NORMAL, result1, result2);
                                        /*  Configure CpGp measurements. */
asweepv(CVU1, 4, 0.05, forceArray);  /*  Configure and perform sweep. */
devint();                      /*  Reset CVU. */
```

Programming example #5

Similar to Programming example #4 (on page 5-37), but the Delay is set to 0 s, and an array of delay values is used for the sweep (0.5 s, 0.25 s, 0.5 s, and 0.25 s).

```
double result1[4], result2[4], forceArray[4], delayArray[4];
setfreq(CVU1, 100e3);         /*  Set AC drive frequency to 100 kHz. */
setlevel(CVU1, 15e-3);        /* Set AC drive voltage to 15 mV RMS. */
setauto(CVU1);                /* Select auto measure range. */
smeasz(CVU1, KI_CVU_TYPE_CPGP, KI_CVU_SPEED_NORMAL, result1, result2);
                                       /*  Configure CpGp measurements. */
adelay(4, delayArray);        /* Call a delay array for asweepv. */
asweepv(CVU1, 4, 0, forceArray);  /* Configure and perform sweep. */
devint();                     /* Reset CVU. */
```

Section 6

LPT commands for PGUs and PMUs

In this section:

LPT commands for PGUs and PMUs

The following information explains the commands in the LPT library for the 4220-PGU and 4225-PMU. The model names are abbreviated as PGU (pulse-generator unit) and PMU (pulse-measure unit). The PGU functions only as a pulse generator. The PMU has both pulse and measurement capabilities.

The pulse commands for the 4220-PGU and 4225-PMU require `pulse_exec` to execute. In addition, they support external triggering, but do not support trigger input from external input signals or instruments.

The PGU and PMU support the following pulse modes:

- Standard pulse mode: For this two-level pulse mode, the user defines an amplitude and base level for the pulse output.
- Segment Arb pulse mode: For this multi-level pulse mode, you define a pulse waveform that consists of three or more line segments. Segment Arb pulse mode for the PGU and PMU also includes sequencing and sequence looping. See seg_arb_sequence (on page 6-70) and seg_arb_waveform (on page 6-73) for the PGU and PMU.
- Full arb pulse mode: For this multilevel pulse mode, the waveform consists of a number of user-defined points. See arb_array (on page 6-3) and arb_file (on page 6-4).

Use the following instrument ID (identification) for LPT commands for the PGU and PMU:

- 4220-PGU: The instrument ID is VPU (VPU1, VPU2, and so on)
- 4225-PMU: The instrument ID is PMU (PMU1, PMU2, and so on)

See PGU (pulse only) and PMU (pulse and measure) group (on page 1-5) for a summary of the functions for the PGU and PMU.

The 4200A-SCS has built-in project tests that use the PGU and PMU LPT commands. In Clarius, see the `pmu-dut-examples` project for a simple example of coding a PMU user test module (UTM).

NOTE

The 4220-PGU and 4225-PMU support the pulsing and external triggering commands of the obsolete 4205-PG2.

arb_array

This command is used to define a full-arb waveform and name the file.

Usage

```
int arb_array(int instr_id, long ch, double TimePerPt, long length, double
    *levelArr, char *fname);
```

instr_id	The instrument identification code, such as VPU1 or VPU2
ch	The pulse card channel: `1` or `2`
TimePerPt	Sets the time interval between waveform points in seconds: `20e-9` to `1`
length	The number of waveform points (values): 262,144 maximum for each channel
levelArr	An array of voltage values for each point in the waveform (see **Details**)
fname	A name for the full-arb waveform

Pulse modes

Full Arb

Details

A Full Arb waveform can be defined for each pulse card channel. A Full Arb waveform is made up of user-defined points. A time interval is set to control the time between the waveform points.

This command defines the number of points in a waveform, the time interval between points, and the voltage value at each point.

The load time for a full-arb waveform is proportional to the number of points. The total time to load full-size full-arb waveforms for both channels is about one minute.

Once loaded, use `pulse_output` to turn on the appropriate channels, and then use `pulse_trig` to select the trigger mode and start (or arm) pulse output.

For additional information on this pulse mode and an example of a Full Arb waveform, refer to “Full arb waveform” in the *Model 4200A-SCS Pulse Card (PGU and PMU) User's Manual*.

.kaf waveform file for KPulse: You can copy the arbitrary waveform data defined by the `arb_file` command into a `.kaf` file. Use a text editor to format the file. You can then import the `.kaf` file into KPulse. By default, `.kaf` waveform files for KPulse are saved in the ArbFiles folder at the command path location `C:\s4200\kiuser\KPulse\ArbFiles`.

Also see

“KPulse (for Keithley Pulse Cards)” in the *Model 4200A-SCS Pulse Card (PGU and PMU) User's Manual*

arb_file (on page 6-4)
pulse_output (on page 6-38)
pulse_trig (on page 6-57)
seg_arb_define (on page 6-67)

arb_file

This command loads a waveform from an existing full-arb waveform file.

Usage

```
int arb_file(int instr_id, long chan, char *fname);
```

instr_id	The instrument identification code of the pulse card: VPU1, VPU2, and so on
chan	Channel number of the pulse card: 1 or 2
fname	The name of the waveform file; the name must be in quotes

Details

Use this command to load a waveform from an existing full-arb `.kaf` waveform file into the pulse card. You can load a full-arb waveform for each channel of the pulse card. Once loaded, use `pulse_output` to turn on the appropriate channel, and then use `pulse_trig` to select the trigger mode and start (or arm) pulse output.

When specifying the *fname*, include the full command path with the file name. Existing `.kaf` waveforms are typically saved in the `ArbFiles` folder at the following command path location:

```
C:\s4200\kiuser\KPulse\ArbFiles
```

You can create a full-arb waveform using KPulse and then save it as a `.kaf` waveform file.

You can modify a waveform in an existing `.kaf` file using a text editor or KPulse.

Example

`arb_file(VPU1, 1, "C:\\s4200\\kiuser\\KPulse\\ArbFiles\\SINE.kaf")`
This example loads a full-arb file named `SINE.kaf` (saved in the `ArbFiles` folder) into the pulse card for channel 1.

Also see

"KPulse (for Keithley Pulse Cards)" in the *Model 4200A-SCS Pulse Card (PGU and PMU) User's Manual*

arb_array (on page 6-3)
pulse_output (on page 6-38)
pulse_trig (on page 6-57)
seg_arb_file (on page 6-69)
seg_arb_define (on page 6-67)

dev_abort

This command programmatically ends (aborts) a test from within the user module that was started with the `pulse_exec` command.

Usage

```
int dev_abort(NULL);
```

Pulsers

4220-PGU
4225-PMU

Pulse mode

Standard and Segment Arb

Details

This command is useful during a longer `pulse_exec` test.

Because `pulse_exec` is nonblocking, you can fetch data during a longer test. An example of this is a long stress/measure test, using `seg_arb_sequence` and `seg_arb_waveform`, that evaluates degradation data during the test. Evaluating this data from within the user module may determine that a test should end. For example, if the degradation is greater than ten percent (>10%), then end the test to save test time.

Example

```
// Place code to configure the PMU test here
//
// Start the test (for seg-arb testing, or for standard pulsing
// with no ranging, LLEC, or I/V/P threshold detection)
status = pulse_exec(PULSE_MODE_SIMPLE);
   if ( status )
   {
      // Minimal error handling, release memory used to
        // fetch results and stop test
        Free_Used_Arrays();
        return status;
   }
// loop to fetch data, while waiting for test complete
abort_sent = 0;
while(pulse_exec_status(&elapsedt) == 1)
   {
     // Code to fetch and evaluate data here
     if (abort_sent == 0)
     {
          // Code to fetch PMU data
          // Code to evaluate data
          // Code to determine if an abort is required
     }

     // If the test must be aborted, send dev_abort
     if (abort_required && abort_sent == 0)
     {
       dev_abort(NULL);
        abort_sent = 1;
     }
      Sleep(100);
 }
```

This example illustrates placement of this command in a code a fragment. Note that after the `dev_abort` command is sent it is still necessary to use `pulse_exec_status` to poll and wait for the test to be ended.

Also see

None

devclr

This command sets all sources to a zero state.

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. `devclr` is implicitly called by `clrcon`, `devint`, `execut`, and `tstdsl`.

For C-V testing, this command turns off the dc bias voltage.

Also see

clrcon (on page 7-2)
devint (on page 2-6)
execut (on page 2-8)
kibdefclr (on page 2-15)
tstdsl (on page 2-40)

devint

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

Usage

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

Details

Resets all active instruments, including the 4200A-CVIV, 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.

The `devint` command is implicitly called by the `execut` and `tstdsl` commands.

To abort a running `pulse_exec` pulse test, see `dev_abort`.

`devint` does the following:

1. Clears all sources by calling `devclr`.
2. Clears the matrix crosspoints by calling `clrcon`.
3. Clears the trigger tables by calling `clrtrg`.
4. Clears the sweep tables by calling `clrscn`.
5. Resets GPIB instruments by sending the string defined with `kibdefint`.
6. Resets the active instrument cards.

Instrument cards are reset in the following order:

1. SMU instrument cards
2. CVU instrument cards
3. Pulse instrument cards (4225-PMU or 4220-PGU)

The SMUs return to the following states:

- 100 μA and 10 V ranges
- Autorange on
- Voltage source
- 0 V dc bias

The 4210-CVU or 4215-CVU returns to the following states:

- 30 mV_{RMS} ac signal
- 0 V dc bias
- 100 kHz frequency
- Autorange on
- Cable length compensation set to 0 m
- Open/Short/Load compensation disabled

The 4225-PMU or 4220-PGU returns to the following states:

- The pulse mode is maintained. For example, if the pulse card is in Segment Arb mode, it is still in Segment Arb mode after the `devint` process is complete.
- 5 V and 10 mA ranges
- If in pulse mode:
  - Period of 1 μs
  - Transition times (rise and fall) of 100 ns
  - Width of 500 ns
  - Voltage high and low of 0 V
  - Load of 50 Ω
- If in segmented arb mode, Start Voltage is 0 V
- If in arbitrary waveform mode, Table Length is 100

Also see

clrcon (on page 7-2)
clrscn (on page 2-2)
clrtrg (on page 2-3)
dev_abort (on page 6-4)
devclr (on page 4-9)
kibdefint (on page 2-16)

getstatus

This command returns the operating state of a specified instrument.

Usage

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

instr_id	The instrument identification code
parameter	The parameter of query; see **Details**
result	The data returned from the instrument; the `getstatus` command returns one item

Details

NOTE

If the `UT_INVLDPRM` invalid parameter error is 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) and a pulse card (VPU) are provided in the following tables.

No status values are provided for measurement-specific conditions.

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_IARANGE`	The presently active range	Current range (full-scale range value)
`KI_VARANGE`		Voltage range (full-scale range value)
`KI_COMPLNC`	Compliance status of last reading	Bitmapped values: 2 = LIMIT (at the compliance limit set by `limit`*`X`*) 4 = RANGE (at the top of the range set by `range`*`X`*)
`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_RANGE_COMPLIANCE`	Range compliance status of last reading	Returns 1 if in range compliance

Supported pulse card getstatus query parameters

Parameter	Returns	Comment
General parameters		
KI_VPU_PERIOD	Pulse period	Pulse period value in seconds
KI_VPU_TRIG_POLARITY	Trigger polarity	Rising or falling edge
KI_VPU_CARD_STATUS	Card status	Card level status
KI_VPU_TRIG_SOURCE	Trigger source	Trigger source value
Channel-based parameters		
KI_VPU_CH1_RANGE	Source range	Channel 1 range value in volts (5.0 or 20.0)
KI_VPU_CH2_RANGE	Source range	Channel 2 range value in volts (5.0 or 20.0)
KI_VPU_CH1_RISE	Rise time	Channel 1 rise time value in seconds
KI_VPU_CH2_RISE	Rise time	Channel 2 rise time value in seconds
KI_VPU_CH1_FALL	Fall time	Channel 1 fall time value in seconds
KI_VPU_CH2_FALL	Fall time	Channel 2 fall time value in seconds
KI_VPU_CH1_WIDTH	Pulse width	Channel 1 pulse width value in seconds
KI_VPU_CH2_WIDTH	Pulse width	Channel 2 pulse width value in seconds
KI_VPU_CH1_VHIGH	Pulse high	Channel 1 pulse high level value in volts
KI_VPU_CH2_VHIGH	Pulse high	Channel 2 pulse high level value in volts
KI_VPU_CH1_VLOW	Pulse low	Channel 1 pulse low level value in volts
KI_VPU_CH2_VLOW	Pulse low	Channel 2 pulse low level value in volts
KI_VPU_CH1_DELAY	Pulse delay	Channel 1 pulse delay from trigger value in seconds
KI_VPU_CH2_DELAY	Pulse delay	Channel 2 pulse delay from trigger value in seconds
KI_VPU_CH1_ILIMIT	Current limit	Channel 1 current Limit value in amps
KI_VPU_CH2_ILIMIT	Current limit	Channel 2 current Limit value in amps
KI_VPU_CH1_BURST_COUNT	Burst count	Channel 1 burst count value
KI_VPU_CH2_BURST_COUNT	Burst count	Channel 2 burst count value
KI_VPU_CH1_TEST_STATUS	Status	Channel 1 test status
KI_VPU_CH2_TEST_STATUS	Status	Channel 2 test status
KI_VPU_CH1_DC_OUTPUT	DC output	Channel 1 dc output value
KI_VPU_CH2_DC_OUTPUT	DC output	Channel 2 dc output value
KI_VPU_CH1_LOAD	Pulse load	Channel 1 pulse load value
KI_VPU_CH2_LOAD	Pulse load	Channel 2 pulse load value

Also see

getinstid (on page 2-10)

pg2_init

This command resets the pulse card to the specified pulse mode (standard, full arb, or Segment Arb) and its default conditions.

Usage

```
int pg2_init(int instr_id, long mode);
```

instr_id	The instrument identification code of the pulse card, such as VPU1 or VPU2
mode	The pulse mode: ■ Standard pulse: `0` ■ Segment Arb: `1` ■ Full Arb: `2`

Pulse modes

Standard, Full Arb, Segment Arb

Details

This command resets both channels of the pulse card to the default settings of the specified pulse mode. The default settings for each parameter are listed in the following table.

If you want to reset the pulse card for the presently selected pulse mode, use the `pulse_init` command.

Standard pulse defaults	Full Arb and Segment Arb pulse defaults
Pulse high and pulse low = 0 V Source range = 5 V fast speed Pulse period = 1e−6 s Pulse width = 500e−9 s Pulse count = 1 Rise and fall time = 10e−9 s Pulse delay = 0 s Pulse load = 50 Ω Pulse trigger source = Software Pulse trigger mode = Continuous Pulse trigger output = On* Trigger polarity = Positive Complement mode = Normal pulse Current limit = 105e−3 A Pulse output = Off	Source range = 5 V fast speed Pulse count = 1 Pulse delay = 0 s Pulse load = 50 Ω Pulse trigger source = Software Pulse trigger mode = Continuous Pulse trigger output = Off* Trigger polarity = Positive Current limit = 105e−3 A Pulse output = Off
* Turns on when a pulse is initiated with `pulse_trig`	

Example

```
pg2_init(VPU1, 1)
```

Resets the pulse card to the Segment Arb pulse mode and its default settings.

Also see

pulse_init (on page 6-30)

pmu_offset_current_comp

This command is used to collect offset current constants from the 4225-PMU. The offset (open) correction readings are stored in a local file.

Usage

```
int pmu_offset_current_comp(int instr_id);
```

instr_id	The instrument identification code of the pulse generator: PMU1, PMU2, and so on.

Pulsers

4225-PMU

Pulse mode

Segment Arb

Details

Use this command to collect current constants for offset current compensation. The correction readings are stored in a file. You can then use the `setmode` command to configure the state of the offset current compensation.

Example

`int pmu_offset_current_comp(PMU1)`
This example collects offset current constants from a 4225-PMU assigned PMU1.

Also see

setmode (4225-PMU) (on page 6-74)

PostDataDoubleBuffer

This buffer posts PMU data retrieved from the buffer into the Clarius Analyze sheet (large data sets).

Usage

```
int PostDataDoubleBuffer(char *ColName, double *array, int length);
```

ColName	Column name for the data array in the Clarius Analyze sheet
array	An array of data values for the Clarius Analyze sheet
length	Number of points (up to 65,535) to post into the Clarius Analyze sheet

Pulsers

4225-PMU

Pulse mode

Standard and Segment Arb

Details

You can use the `PostDataDouble` and `PostDataDoubleBuffer` commands to post double-precision floating-point data into the Clarius Analyze sheet. Up to 65,535 points (rows) can be posted into the Analyze sheet. These commands are used after one measurement is finished and a data value is assigned to the corresponding output variable.

You can use either of these commands to post data into the sheet. However, you should use the `PostDataDoubleBuffer` command to post the large data sets that are typically generated by PMU waveform measurements.

The following sequence summarizes the process to post data into the Analyze sheet:

- Run a test.
- Use `pulse_fetch` to retrieve the data from the buffer. You can analyze or manipulate the retrieved data.
- Use `PostDataDouble` or `PostDataDoubleBuffer` to post data into the Analyze sheet.

When you use `pulse_fetch`, you can either wait until the test is finished before retrieving data or you can retrieve blocks of data while the test is running, which is useful for a test that takes a long time. Instead of waiting for the entire test to finish, you can retrieve blocks of data at prescribed intervals. For details, see "Data retrieval options for `pulse_fetch`" in the `pulse_fetch` command Details section.

NOTE

If you do not need to analyze or manipulate the test data before posting it into the Analyze sheet in Clarius, you can use `pulse_measrt`.

NOTE

`PostDataDoubleBuffer` is not compatible with using KXCI to call user libraries remotely (see “Calling KULT user libraries remotely” in *Model 4200A-SCS KXCI Remote Control Programming*). Use `PostDataDouble` for user routines (UTMs) that will be called using KXCI.

Example

```
// Code to configure the PMU test here
// Start the test (no analysis)
status = pulse_exec(0);
// While loop (continues while test is still running), with
// delay (30 ms)
while (pulse_exec_status(&elapsedt) == 1)
{
   Sleep(30);
}
// Retrieve V, I, and timestamp data (no status)
status = pulse_fetch(PMU1, 1, 0, 20e3, Vmeas, Imeas, Tstamp, NULL);
// Separate V, I, and timestamp measurements
for (i = 0; i<20e3; i++)
{
   Vmeas_sheet[i] = Vmeas[2*i];
   Imeas_sheet[i] = Imeas[2*i];
   Tstamp_sheet[i] = Tstamp[2*i];
}
PostDataDoubleBuffer("DrainVmeas", Vmeas_sheet, 20e3);
PostDataDoubleBuffer("DrainImeas", Imeas_sheet, 20e3);
PostDataDoubleBuffer("Timestamp", Tstamp_sheet, 20e3);
```

Posts waveform measurement data into the Analyze sheet. This example assumes that a PMU waveform test is configured to perform 20,000 (or more) voltage and current measurements. Use `pulse_meas_wfm` to configure the waveform test.

The code:

- Starts the configured test.
- Uses a while loop to allow the waveform test to finish.
- Retrieves voltage, current, and timestamp readings (20,000 points) from the buffer.
- Separates the voltage, current, and timestamp readings.
- Posts the measurement data into the Clarius Analyze sheet.

Also see

PostDataDouble (on page 2-24)
pulse_fetch (on page 6-24)
pulse_meas_wfm (on page 6-36)
pulse_measrt (on page 6-37)

pulse_burst_count

For the burst mode, this command sets the number of pulses to output during a burst sequence.

Usage

```
int pulse_burst_count(int instr_id, long chan, unsigned long count);
```

instr_id	The instrument identification code of the pulse card, such as VPU1 or VPU2
chan	Channel number of the pulse card: 1 or 2
count	Number of pulses to output: 1 to (2^{32}-1); default 1

Pulse modes

Standard, Full Arb, Segment Arb

Details

Each channel of the pulse card can have a unique burst count. When a burst sequence is triggered, the card outputs the specified number of pulses and then stops. The `pulse_trig` command is used to start (or arm) the burst sequence (Burst or Trig Burst).

You can set burst count independently for each pulse card channel.

This command can also be used with `pulse_exec`.

NOTE

With an external trigger source selected, the burst count for channel 1 cannot be less than the burst count for channel 2. Setting the burst count for channel 2 higher than the burst count for channel 1 may cause your system to stop responding when pulse output is triggered to start. Also, when using one channel, set the unused channel to the same burst count value. See `pulse_trig_source` for details on selecting an external trigger source.

Example

`pulse_burst_count(VPU1, 1, 10)`
Sets the burst count for the pulse card channel 1 to a count of 10.

Also see

pulse_exec (on page 6-20)
pulse_trig (on page 6-57)
pulse_trig_source (on page 6-60)

pulse_chan_status

This command is used to determine how many readings are stored in the data buffer.

Usage

```
int pulse_chan_status(int instr_id, int chan, int *buffersize);
```

instr_id	The instrument identification code: PMU1, PMU2, and so on
chan	Channel number of the pulse card: 1 or 2
buffersize	User-defined name for the returned size value

Pulsers

4225-PMU

Pulse mode

Standard and Segment Arb

Details

Use this command to return the number of readings presently stored in the data buffer. This command can be called while a sweep is in progress or after it is completed.

For a short sweep (test time in seconds to a minute or more), this command is typically called after the sweep is complete to determine the total number of readings stored in the buffer. For a long test, you can use this command to track the progress of the test. A long test is typically Segment Arb with test time in minutes, hours, or days.

Example

`pulse_chan_status(PMU1, 1, buffersize);`
This command returns the number of readings stored in the buffer for channel 1.

Also see

None

pulse_conncomp

This command enables or disables connection compensation.

Usage

```
int pulse_conncomp(int instr_id, int chan, int type, int index);
```

instr_id	The instrument identification code: PMU1, PMU2, and so on
chan	Channel number of the PMU: 1 or 2
type	Type of compensation to enable: ▪ 1: Short ▪ 2: Delay
index	Connection compensation values: ▪ 0: Disable all connection compensation ▪ 1: Selects the default connection compensation values for a setup that uses the PMU only ▪ 2: Selects the default connection compensation values for a setup that uses the PMU and the RPM ▪ 3: Selects the custom connection compensation values (see **Details**)

Pulsers

4225-PMU

Pulse mode

Standard and Segment Arb

Details

Errors caused by the connections and cable length between the 4225-PMU and the device under test (DUT) can be corrected by using connection compensation. When connection compensation is enabled, each DUT measurement factors in either the default or measured (custom) compensation values.

Use this command to control connection compensation. You can select short or delay. Short compensation corrects for the measured resistance of the cabling and connections. Delay compensation measures and corrects for cable delay (the time it takes a signal to transit the cable).

You can use either default connection compensation values (PMU or RPM) or custom connection compensation values. The default values provide compensation for simple connection setups that use the supplied cables. The custom connection compensated values are generated when connection compensation is performed. The custom values provide optimum compensation.

Custom connection compensation is a two-part process:

1. Perform connection compensation from the Clarius interface. Connection compensation data is generated for short and delay conditions. The compensation values are stored in tables.
2. Use this command (`pulse_conncomp`) to enable the custom connection compensation values.

When a test is run, each measurement factors in the enabled compensation values. If connection compensation is disabled, the compensation values are not used by the test.

NOTE

For detail on performing connection compensation, refer to "Performing connection compensation" in the *Model 4200A-SCS Pulse Card (PGU and PMU) User's Manual*.

Example

```
pulse_conncomp(PMU1, 1, 1, 3);
```

This example assumes connection compensation was done using the Clarius interface. This command enables short connection compensation using the custom compensation values.

Also see

None

pulse_current_limit

This command sets the current limit of the pulse card.

Usage

```
int pulse_current_limit(int instr_id, long chan, double ilimit);
```

instr_id	The instrument identification code of the pulse card, such as VPU1 or VPU2
chan	Channel number of the pulse card: 1 or 2
ilimit	Current limit value (in amps; range and load dependent): ■ 5 V range, 50 Ω load : −0.2 to +0.2 ■ 20 V range, 50 Ω load: −0.4 to +0.4 ■ 20 V range: −0.8 to +0.8 Default is 5 V range, 105e−3 A (105 mA)

Pulse modes

Standard, Full Arb, Segment Arb

Details

You can set the current limit independently for each pulse card channel.

Current limit protects the DUT by using the specified DUT load to calculate the voltage required to reach the current limit. A pulse card channel will not exceed the voltage required to reach the set current limit value at the specified DUT load.

For information on the effect of loading on the limits, refer to the *Model 4200A-SCS Pulse Card (PGU and PMU) User's Manual*, "DUT resistance determines pulse voltage across DUT." For an example and values for load-line effect, refer to "Example 5: Maximum voltage and current, high voltage range."

Example

```
pulse_current_limit(VPU1, 1, 1e-3)
```

Sets the current limit of pulse card channel 1 to 1 mA.

Also see

pulse_load (on page 6-32)

pulse_dc_output

This command selects the dc output mode and sets the voltage level.

Usage

```
int pulse_dc_output(int instr_id, long chan, double dcvalue);
```

instr_id	The instrument identification code of the pulse card, such as VPU1 or VPU2
chan	Channel number of the pulse card: 1 or 2
dcvalue	The dc voltage output value (in volts; range and load dependent): ▪ 5 V range: –5 to +5 ▪ 20 V range: –20 to +20 (50 Ω load) ▪ Default: Not applicable

Pulse modes

Standard

Details

You can set each pulse card channel to output a fixed dc voltage level instead of pulses.

The maximum and minimum output voltage is range dependent. See `pulse_vhigh` and `pulse_vlow` for details.

CAUTION

The `pulse_vlow`, `pulse_vhigh`, and `pulse_dc_output` commands set the voltage value output by the pulse channel when it is turned on (using `pulse_output`). If the output is already enabled, these commands change the voltage level immediately, before the pulsing is started with a `pulse_trig` command.

Example

`pulse_dc_output(VPU1, 1, 10)`
Selects the dc voltage output for channel 1 and sets the voltage to +10 V.

Also see

pulse_load (on page 6-32)
pulse_vhigh (on page 6-62)
pulse_vlow (on page 6-63)

pulse_delay

This command sets the delay time from the trigger to when the pulse output starts.

Usage

```
int pulse_delay(int instr_id, long chan, double delay);
```

instr_id	The instrument identification code of the pulse card, such as VPU1 or VPU2
chan	Channel number of the pulse card: `1` or `2`
delay	Time delay in seconds: ■ Fast speed: `0` to (Period – 10e-9) ■ Slow speed: `0` to (Period – 10e-9) ■ Default: `0`

Pulse modes

Standard, Full Arb, Segment Arb

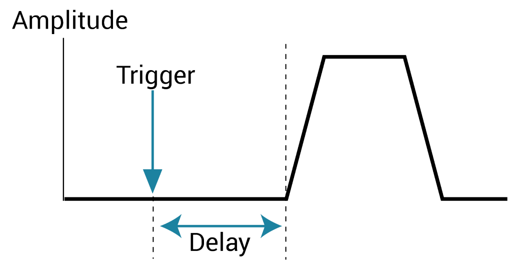
Details

NOTE

Use the `pulse_source_timing` command to set the pulse delay time for the 4220-PGU and 4225-PMU.

Pulse delay can be set independently for each pulse card channel. For both speeds, pulse delay can be set from 0 ns to (Period – 10 ns). The pulse delay is set in 10 ns increments. The `pulse_range` command is used to set pulse speed.

As shown below, pulse delay is the time from pulse trigger initiation to the start of the rise transition time.



The maximum pulse delay that can be set depends on the presently set period for the pulse. For example, if the period is set for 500 ns, the maximum pulse delay that can be set is 490 ns (500 ns – 10 ns = 490 ns).

Example

`pulse_delay(VPU1, 1, 300e-9)`
Sets the pulse delay for channel 1 to 300 ns.

Also see

pulse_period (on page 6-40)
pulse_range (on page 6-41)
pulse_source_timing (on page 6-47)
pulse_trig (on page 6-57)

pulse_exec

This command is used to validate the test configuration and start test execution.

Usage

```
int pulse_exec(long mode);
```

mode	The mode of execution: ■ `PULSE_MODE_SIMPLE` or `0`: No analysis performed during testing; no ranging, no load-line effect compensation, and no threshold checking ■ `PULSE_MODE_ADVANCED` or `1`: Enables the analytical sweep engine and incorporates the use of any combination of the options for the standard (2-level) pulse mode

Pulsers

4220-PGU
4225-PMU

Pulse mode

Standard and Segment Arb

Details

Use this command to validate the test configuration, select the simple or advanced mode, and execute the test. If there are any problems with the test configuration, the validation will stop and the test will not be executed.

The `pulse_exec` command is nonblocking, which means that if this command is called to execute the test, the program continues and does not wait for the test to finish. Therefore, after calling `pulse_exec`, the `pulse_exec_status` command must be called in a `while` loop to ensure the test is complete before fetching data or exiting the user test module (UTM).

There are two commands that affect a pulse test while it is running:

- The `pulse_remove` command removes a PMU channel from the test.
- The `dev_abort` command aborts the test.

NOTE

The Internal Trigger Bus trigger source (see `pulse_trig_source` command) is used only by the 4220-PGU and 4225-PMU for triggering. The `pulse_exec` command automatically uses the internal trigger bus. A trigger input to start a `pulse_exec` test is not available.

CAUTION

Do not exit the user module while the test is still running. Incorrect readings or device damage may result.

Example

```
// Code to configure the PMU test here
// Start the test (no analysis)
pulse_exec(0);
// while loop and short delay (10 ms)
while (pulse_exec_status(&elapsedt) == 1)
{
   Sleep(10);
}
// Retrieve all data
status = pulse_fetch(PMU1, 1, 0, 49, Drain_Vmeas, Drain_Imeas,
NULL, NULL);
// Code for data handling here
```

This example uses `pulse_exec` to set the execution type to simple two-level pulse operation (no analysis) and execute the test. The code pauses the program to monitor the status of the test. It uses a while loop to check the returned value of `pulse_exec_status`. When the test is completed, the program drops out of the loop and calls `pulse_fetch` to retrieve all the test data.

Also see

dev_abort (on page 6-4)
pulse_remove (on page 6-44)
pulse_trig_source (on page 6-60)

pulse_exec_status

This command is used to determine if a test is running or completed.

Usage

`int pulse_exec_status(double *elapsedt);`

elapsedt	Name of the user-defined pointer for elapsed time

Pulsers

4220-PGU
4225-PMU

Pulse mode

Standard and Segment Arb

Details

This command is required to determine when a test is complete or what is occurring during a test. The return value indicates whether the test is still running (`PMU_TEST_STATUS_RUNNING` or `1`) or completed (`PMU_TEST_STATUS_IDLE` or `0`). The primary use of this command is to ensure that the test is completed before fetching PMU data or ending the test.

The elapsed time is the Clarius test time, not the PMU or VPU card test time. For short test times, the returned elapsed time will be longer than the actual time required on-card.

This command is typically used in a `while` loop to allow the test to finish before retrieving the data using the `pulse_fetch` command.

It is the responsibility of the user test module (UTM) programmer to ensure that the pulse test is complete before exiting the UTM. If the UTM program ends before the test is complete, Clarius responds with two messages. These messages are displayed in the Clarius messages area:

1. Five seconds after the UTM ends prematurely (before the pulse test is finished), the message "*UTMname* ended before the test was complete. Waiting for test to finish (max wait = 5 minutes)" is displayed.
2. Clarius continues to wait for the UTM to finish, interrupting further test execution.
3. After the default of five minutes, the UTM is terminated and the following message is displayed, "*UTMname* did not finish before the maximum wait period. UTM aborted."
4. After this five minute wait, Clarius releases control to the user interface or to the next test in the project (if using repeat executing or looping).

Example

```
// Code to configure the PMU test here
// Start the test (no analysis)
pulse_exec(0);
// while loop and short delay (10 ms)
while (pulse_exec_status(&elapsedt) == 1)
{
   Sleep(10);
}
// Retrieve all data
status = pulse_fetch(PMU1, 1, 0, 49, Drain_Vmeas, Drain_Imeas,
NULL, NULL);
// Code for data handling here
```

This example uses `pulse_exec` to set the execution type to simple two-level pulse operation (no analysis) and execute the test. The code pauses the program to monitor the status of the test. It uses a while loop to check the returned value of `pulse_exec_status`. When the test is completed, the program drops out of the loop and calls `pulse_fetch` to retrieve all the test data.

Also see

pulse_exec (on page 6-20)
pulse_fetch (on page 6-24)

pulse_fall

This command sets the fall transition time for the pulse output.

Usage

`int pulse_fall(int instr_id, long chan, double fallt);`

instr_id	The instrument identification code of the pulse card, such as VPU1 or VPU2
chan	Channel number of the pulse card: `1` or `2`
fallt	Pulse fall time in seconds (floating-point number): ■ Fast speed: `10e-9` to `33e-3` (10 ns to 33 ms) ■ Slow speed: 4220-PGU and 4225-PMU: `50e-9` to `33e-3` (50 ns to 33 ms) ■ Default: `100e-9` (100 ns)

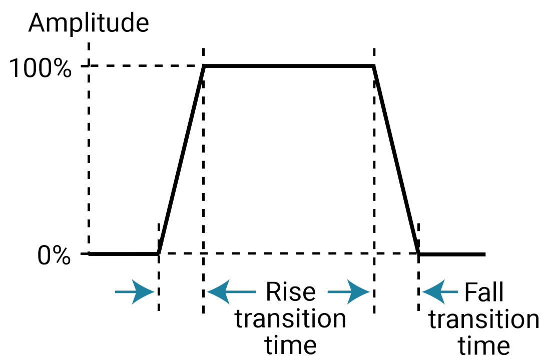
Pulse modes

Standard

Details

Rise and fall transition time can be set independently for each pulse card channel. There is a minimum slew rate for both the rise and fall transitions. For the fast speed range, the minimum is 362 µV/µs, or 1 V/2.7 ms. For the high voltage range, the minimum slew rate is 1.8 mV/µs, or 1 V/500 µs. The `pulse_range` command sets the pulse speed.

As shown below, the pulse fall time occurs between the 100 percent and 0 percent amplitude points on the falling edge of the pulse, where the amplitude is the difference between the V High and V Low pulse values.



The pulse fall time setting takes effect immediately during continuous pulse output. Otherwise, the fall time setting takes effect when the next trigger is initiated. The `pulse_trig` command is used to trigger continuous or burst output.

For slow speed, note that the minimum transition time for pulse source only (no measurement) on the 40 V range is 50 ns for the 4225-PMU and 4220-PGU.

NOTE

Use the `pulse_source_timing` command to set the pulse fall time for the 4220-PGU and 4225-PMU.

Example

```
pulse_fall(VPU1, 1, 50e-9)
```

For fast speed, the sets the pulse fall time for channel 1 of the pulse card to 50 ns.

Also see

pulse_range (on page 6-41)
pulse_rise (on page 6-45)
pulse_source_timing (on page 6-47)
pulse_trig (on page 6-57)

pulse_fetch

This command retrieves enabled test data and temporarily stores it in the data buffer.

Usage

```
int pulse_fetch(int instr_id, int chan, int StartIndex, int StopIndex, double
    *Vmeas, double *Imeas, double *Timestamp, unsigned long *Status);
```

instr_id	The instrument identification code: PMU1, PMU2, and so on
chan	Channel number of the pulse card: 1 or 2
StartIndex	Start index point for data (within the overall set of data)
StopIndex	Final index point to be retrieved
Vmeas	Name of the user-defined array for retrieved voltage measure readings; this is a single-dimension array
Imeas	Name of the user-defined array for retrieved current measure readings; this is a single-dimension array
Timestamp	Name of the user-defined array for retrieved timestamps; this is a single-dimension array
Status	Name of the user-defined array for retrieved status for the channel

Pulsers

4225-PMU

Pulse mode

Standard and Segment Arb

Details

When using `pulse_fetch` to retrieve data, you need to pause the program to allow time for the buffer to fill. You can use the sleep command to pause for a specified time, or you can use the `pulse_exec_status` command in a `while` loop to wait until the test is completed.

Use this command to retrieve a block of newly generated test data in pseudo real time and temporarily store it in the data buffer. The stored data can then be analyzed and manipulated as needed before posting it to the Clarius Analyze sheet.

Typically, this command is used with the `pulse_exec_status` command to allow the test to finish before retrieving the data.

The block of data to be retrieved is set by the *`StartIndex`* and *`StopIndex`* parameters. The start index parameter specifies the first index number in the buffer. The stop index parameter specifies the final index number. For example, assume there are 1000 data test points for a test, and you want to retrieve the first 50 points. The start index value is set to zero (`0`) and the stop index is set to `49`.

The *`Vmeas`*, *`Imeas`*, *`Timestamp`*, and *`Status`* parameters are array names defined by the user. If you do not want to retrieve the timestamp or status, `NULL` can be passed as valid parameters for these fields.

NOTE

The return of all readings must be enabled by the `pulse_meas_sm` command. If disabled, the arrays are not retrieved.

For spot mean measurements, data can be mixed; the amplitude and base level readings are returned in the same array buffer area and must be separated (or parsed) after the measurement cycle is complete. See the `pulse_meas_sm` command for details on spot mean measurements. Note that number of measurements returned is determined by the spot means enabled in `pulse_meas_sm`. With both amplitude and base measurements enabled, there will be two voltage and two current readings for each pulse (with spot mean discrete) or each pulse burst (with spot mean average). Voltage and current readings are returned in individual arrays: `Vmeas`, `Imeas`. When both amplitude and base readings are enabled, the readings are alternated. For example, the Vmeas array: Vampl_1, Vbase_1, Vampl_2, Vbase_2, Vampl_3, Vbase_3, and so on. To plot the amplitude values, separate the amplitude and base measurements into individual arrays before using `PostDataDouble` to post the measurements to the sheet.

The timestamps pertain to either per spot mean reading or per sample. Status is returned as a 32-bit word. The status code bit map is shown in the following table.

NOTE

If you do not need to analyze or manipulate the test data before posting it to the Clarius Analyze sheet, you can use the `pulse_measrt` command. The `pulse_measrt` command retrieves all the test data in pseudo real time and automatically posts it into the Clarius Analyze sheet.

Status-code bit map for `pulse_fetch`

Bit	Summary or description	Value (bit pattern)
31	Reserved	Reserved bit for future use
30	Sweep skipped	0 = Not skipped 1 = Skipped
29	Load-line effect compensation (LLEC) enabled (only valid when LLEC is enabled)	0 = Failed 1 = Successful
28	LLEC status	0 = Disabled 1 = Enabled
27 to 24	RPM mode settings	0 (0000) = No RPM 1 (0001) = RPM 2 (0010) = Bypass; PMU 3 (0011) = Bypass; SMU 4 (0100) = Bypass; CVU All other values (bit patterns) reserved
23 to 20	Reserved	Reserved bits for future use
19 to 16	Measurement type	1 (0001) = Spot mean 2 (0010) = Waveform All other values (bit patterns) reserved
15 to 12	Current threshold, voltage threshold, power threshold, and source compliance	0 (0000) = None 1 (0001) = Source compliance 2 (0010) = Current threshold reached or surpassed 4 (0100) = Voltage threshold reached or surpassed 8 (1000) = Power threshold reached or surpassed
11 to 10	Current measure overflow	0 (00) = No overflow 1 (01) = Negative overflow 2 (10) = Positive overflow
9 to 8	Voltage measure overflow	0 (00) = No overflow 1 (01) = Negative overflow 2 (10) = Positive overflow

Status-code bit map for `pulse_fetch`

Bit	Summary or description	Value (bit pattern)
7 to 4	Current measure range	0 (0000) = 100 nA (RPM only) 1 (0001) = 1 µA (RPM only) 2 (0010) = 10 µA (RPM only) 3 (0011) = 100 µA 4 (0100) = 1 mA (RPM only) 5 (0101) = 10 mA 6 (0110) = 200 mA 7 (0111) = 800 mA All other values (bit patterns) reserved
3 to 2	Voltage measure range	0 (00) = 10 V 1 (01) = 40 V
1 to 0	Channel number	1 (01) = Ch1 2 (10) = Ch2 Value 0 (00) not used

Data retrieval options for `pulse_fetch`

There are two options to retrieve data:

- Wait until the test is completed
- Retrieve blocks of data while the test is running

Because `pulse_exec` is a nonblocking command, the running user test module (UTM) will continue after it is called to start the test. This means that the program will not automatically pause to allow the pulse-measure test to finish.

CAUTION

The programmer must ensure that the test program does not finish or return to Clarius before the test is complete. Erroneous results and damage to test devices may occur.

If `pulse_fetch` is inadvertently called before the test is completed, the data buffer may not fill with all the requested readings. Array entries are designated as zero for test data that is not yet available.

Wait until the test is complete before retrieving data

An effective method to pause the program is to monitor the status of the test by using a `while` loop to check the returned value of `pulse_exec_status`. When the test is completed, the program drops out of the loop and calls `pulse_fetch` to retrieve all the test data. The following program fragment shows how to use a `while` loop.

Program fragment 1

```
// Code to configure the PMU test here
// Start the test (no analysis)
pulse_exec(0);
// while loop and short delay (10 ms)
while (pulse_exec_status(&elapsedt) == 1)
{
   Sleep(10);
}
// Retrieve all data
status = pulse_fetch(PMU1, 1, 0, 49, Drain_Vmeas, Drain_Imeas, NULL, NULL);
// Code for data handling here
```

After all the data is retrieved, it can be analyzed, manipulated, and then posted into the Clarius Analyze sheet. Use the `PostDataDouble` or `PostDataDoubleBuffer` command to post the data.

Retrieve blocks of data while the test is running

An advantage of the `pulse_exec` command being nonblocking is that it allows you to retrieve test data before the test is completed, which is useful for a test that takes a long time. Instead of waiting for the entire test to finish, you can retrieve blocks of data at prescribed intervals. The interval can be controlled by using the `sleep` command as shown in the following program fragment.

Program fragment 2

```
// Code to initialize the data arrays
for (i = 0; i < array_size; i++)
{
   Drain_Vmeas = 0.0;
   Drain_Imeas = 0.0;
}

// Code to configure the PMU test here
// Start the test and pause for 20 seconds
pulse_exec(0);
Sleep(20000);
// Retrieve a block of test data:
pulse_fetch(PMU1, 1, 0, 10e3, Drain_Vmeas, Drain_Imeas, 1, NULL);
// Code for data handling here
```

After retrieving a block of data, loop back to the sleep command to allow the next block of data to become available before fetching it. Repeat this loop until all the data is retrieved.

The `pulse_fetch` command will return all data available at the time of the call. The remaining array space will not be modified. To determine how much data was retrieved, it is recommended to initialize the arrays. **Program fragment 2** initializes the results arrays to 0.0, but other values may be used. After the retrieving the data, search the array for the first entry with this initialized value.

Retrieved blocks of data can be analyzed and manipulated while the test is still running. After data handling is completed, use the `PostDataDoubleBuffer` command to post the data to the Clarius Analyze sheet.

Example 1

```
// Code to configure the PMU test here
// Start the test (no analysis)
pulse_exec(0);
// while loop and short delay (10 ms)
while (pulse_exec_status(&elapsedt) == 1)
{
   Sleep(10);
}
// Retrieve all data
status = pulse_fetch(PMU1, 1, 0, 49, Drain_Vmeas, Drain_Imeas,
NULL, NULL);
// Code for data handling here
```

This example uses `pulse_exec` to set the execution type to simple two-level pulse operation (no analysis) and execute the test. The code pauses the program to monitor the status of the test. It uses a while loop to check the returned value of `pulse_exec_status`. When the test is completed, the program drops out of the loop and calls `pulse_fetch` to retrieve all the test data.

Example 2

```
pulse_fetch(PMU1, 1, 0, 49, Drain_Vmeas, Drain_Imeas, T_Stamp, NULL);
```

This command retrieves 50 points of data from the buffer, where:

- *`Instr_id`* = PMU1
- *`chan`* = 1 (channel 1)
- *`StartIndex`* = 0
- *`StopIndex`* = 49
- *`Vmeas`* = Drain_Vmeas (name of array)
- *`Imeas`* = Drain_Imeas (name of array)
- *`Timestamp`* = T_Stamp (name of array)
- *`Status`* = NULL (not retrieved)

Also see

PostDataDouble (on page 2-24)
PostDataDoubleBuffer (on page 6-11)
pulse_meas_sm (on page 6-33)
pulse_measrt (on page 6-37)

pulse_float

This command sets the state of the floating relay for the given pulse instrument.

Usage

`int pulse_float(int instr_id, int state);`

`instr_id`	The instrument identification code of the pulse card, such as PMU1 or PMU2
`state`	State of the relay: ■ `OFF` (default) ■ `ON` (float)

Pulsers

4220-PGU, 4225-PMU

Pulse mode

Standard

Details

This command is used to float the PGU/PMU card.

Example

`pulse_float(PMU1, OFF);`
This turns off the floating relay on PMU1 instrument.

Also see

None

pulse_halt

This command stops all pulse output from the pulse card.

Usage

```
int pulse_halt(int instr_id);
```

instr_id	The instrument identification code of the pulse card, such as VPU1 or VPU2

Pulse modes

Standard, Full Arb, Segment Arb

Details

This command stops all pulse output from the pulse card and turns the pulse card channels off. Pulse output can be restarted by turning the outputs on with `pulse_output` and then using the `pulse_trig` command to restart the test.

Example

`pulse_halt(VPU1)`
Stops pulse output.

Also see

pulse_output (on page 6-38)
pulse_trig (on page 6-57)

pulse_init

This command resets the pulse card to the default settings for the pulse mode that is presently selected.

Usage

```
int pulse_init(int instr_id);
```

instr_id	The instrument identification code of the pulse card, such as VPU1 or VPU2

Pulse modes

Standard, Full Arb, Segment Arb

Details

This command resets both channels of the pulse card to the default settings. The default settings are listed in the following table.

If you want to specify the pulse mode to reset, use the `pg2_init` command.

Standard pulse defaults	Full Arb and Segment Arb pulse defaults
Pulse high and pulse low = 0 V Source range = 5 V fast speed Pulse period = 1e−6 s Pulse width = 500e−9 s Pulse count = 1 Rise and fall time = 10e−9 s Pulse delay = 0 s Pulse load = 50 Ω Pulse trigger source = Software Pulse trigger mode = Continuous Pulse trigger output = On* Trigger polarity = Positive Complement mode = Normal pulse Current limit = 105e−3 A Pulse output = Off	Source range = 5 V fast speed Pulse count = 1 Pulse delay = 0 s Pulse load = 50 Ω Pulse trigger source = Software Pulse trigger mode = Continuous Pulse trigger output = Off* Trigger polarity = Positive Current limit = 105e−3 A Pulse output = Off
* Turns on when a pulse is initiated with `pulse_trig`	

Example

`pulse_init(VPU1)`
Resets the pulse card to the default settings for the presently selected pulse mode.

Also see

pg2_init (on page 6-10)

pulse_limits

This command sets measured voltage and current thresholds at the DUT and sets the power threshold for each channel.

Usage

```
int pulse_limits(int instr_id, int chan, double V_Limit, double I_Limit, double
   Power_Limit);
```

instr_id	The instrument identification code: PMU1, PMU2, and so on
chan	Channel number of the pulse card: 1 or 2
V_Limit	Measured voltage (V) threshold at the DUT
I_Limit	Measured current (A) threshold at the DUT
Power_Limit	Power (W) threshold for the channel (Power = V_{meas} x I_{meas})

Pulsers

4225-PMU

Pulse mode

Standard

Details

This feature differs from a SMU compliance setting in that threshold checking is done after each burst of pulses, using the spot mean values to compare to the specified thresholds. The thresholds are checked against all enabled measurements for the channel. If a threshold is reached or exceeded, the present sweep is stopped and testing continues with any subsequent sweeps.

This feature does not prevent the set thresholds from being reached or exceeded. After detecting a threshold breach, it aborts the sweep.

Maximum power for each PMU source range:

High-speed voltage source (10 V) range: Maximum power = 5 V x 0.1 A = 0.5 W

High-voltage source (40 V) range: Maximum power = 20 V x 0.4 A = 8 W

Example

```
pulse_limits(PMU1, 1, 42, 1, 10);
```

This example sets thresholds for channel 1 of the PMU, where:

- *instr_id* = PMU1
- *chan* = Channel 1
- *V_Limit* = 42 V
- *I_Limit* = 1 A
- *Power_Limit* = 10 W

Also see

None

pulse_load

This command sets the output impedance for the load (DUT).

Usage

```
int pulse_load(int instr_id, long chan, double load);
```

instr_id	The instrument identification code of the pulse card, such as VPU1 or VPU2
chan	Channel number of the pulse card: 1 or 2
load	Output impedance (in ohms): 1 to 10e6 (default 50)

Pulse modes

Standard, Full Arb, Segment Arb

Details

DUT impedance can be independently set for each pulse card channel. The DUT impedance can be set from 1 Ω to 10 MΩ, depending on the programmed pulse high and low values.

Maximum power transfer is achieved when the DUT impedance matches the output impedance of the pulse card. For example, if the DUT impedance is set to 1 MΩ, the voltage output settings will change to account for the higher DUT impedance, ensuring that the voltage at the DUT will not be double the voltage setting (caused by reflection due to load mismatching).

The purpose of setting the DUT load to a value other than 50 Ω is to simplify the calculation of the output levels. For example, if the DUT load is set to 50 Ω, but the actual DUT load has a high impedance of 1 MΩ, setting a voltage level of 2 V will result in a 4 V pulse at the DUT. Setting the DUT load to 1 MΩ will permit the set voltage to match the actual voltage, so setting a 2 V level will result in a 2 V pulse, with the pulse card taking the DUT impedance into account.

Example

```
pulse_load(VPU1, 1, 100)
```

Sets the output impedance of pulse card channel 1 to 100 Ω.

Also see

None

pulse_meas_sm

This command configures spot mean measurements.

Usage

```
int pulse_meas_sm(int instr_id, int chan, Int AcquireType, int AcquireMeasVAmpl,
    int AcquireMeasVBase, int AcquireMeasIAmpl, int AquireMeasIBase, int
    AquireTimeStamp, int LLEComp);
```

instr_id	The instrument identification code of the pulse card, such as VPU1 or VPU2
chan	Channel number of the pulse card: `1` or `2`
AcquireType	Acquisition type: ■ Discrete: `0` ■ Average: `1`
AcquireMeasVAmpl	Return amplitude voltage measurements: ■ Disable: `0` ■ Enable: `1`
AcquireMeasVBase	Return base level voltage measurements: ■ Disable: `0` ■ Enable: `1`
AcquireMeasIAmpl	Return amplitude current measurements: ■ Disable: `0` ■ Enable: `1`
AcquireMeasIBase	Return base current level measurements: ■ Disable: `0` ■ Enable: `1`
AcquireTimeStamp	Return timestamp readings: ■ Disable: `0` ■ Enable: `1`
LLEComp	Load-line effect compensation (LLEC): ■ All LLEC disabled: `0` ■ Voltage LLEC on for pulse amplitude only: `1`

Pulsers

4225-PMU

Pulse modes

Standard

Details

To use this command to configure spot mean measurements, you select the data acquisition type, set the readings to be returned, enable or disable timestamps, and set load-line effect compensation (LLEC).

LLEC is only performed for standard pulse IV testing using PMU measure ranges. It is not performed when using 4225-RPM measure ranges. The active RPM circuitry provides its own analog LLEC (assuming a short cable from the RPM to the DUT).

Also see

Model 4200A-SCS Pulse Card (PGU and PMU) User's Manual, "Load-line effect compensation," "Measurement types," "Spot mean measurements," and "Waveform measurements"

pulse_meas_timing

This command sets the measurement windows.

Usage

```
int pulse_meas_timing(int instr_id, int chan, double, StartPercent, double
   StopPercent, int NumPulses);
```

instr_id	The instrument identification code of the pulse card, such as VPU1 or VPU2
chan	Channel number of the pulse card: 1 or 2
StartPercent	Start location for measurements: ■ Spot mean measurements: Start location, specified as a percentage of the widths for the amplitude and base level (see **Details**) ■ Waveform: Pre-data for the amplitude, specified as a percentage of the amplitude pulse duration (see **Details**)
StopPercent	Stop location for measurements: ■ Spot mean measurements: Stop location, specified as a percentage of the widths for the amplitude and base level (see **Details**) ■ Waveform: Post-data for the amplitude, specified as a percentage of the amplitude pulse duration (see **Details**)
NumPulses	Number of pulses to output and measure (1 to 10,000)

Pulsers

4225-PMU

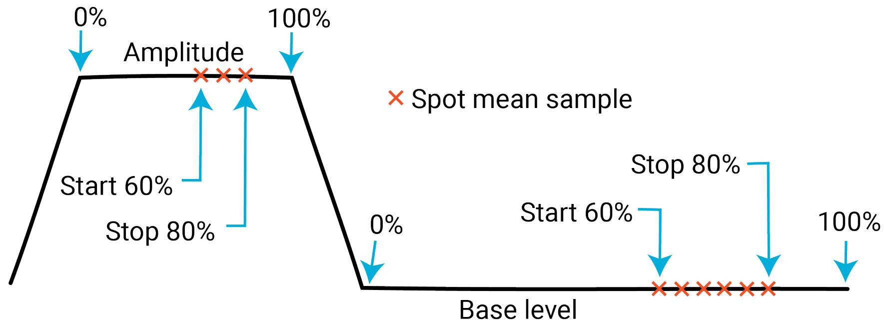
Pulse modes

Standard

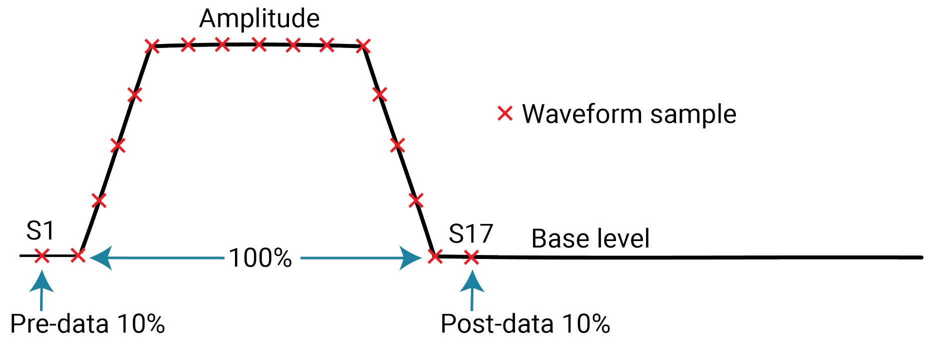
Details

Use the `pulse_meas_timing` command to set measurement timing. For spot mean measurements, portions of the amplitude and base levels are specified for sampling. For pre-data and post-data waveform measurements, a percentage of the entire pulse duration is specified.

The figure below shows example start and stop locations spot mean measurements are made. Three measured samples are taken on the amplitude and six samples are taken on the base level. The start and stop percentage values indicate the portions of the pulse that are sampled.



The figure below shows example a waveform measurement with pre-data and post-data. Pre-data is extra data taken before the rise time of the pulse; post-data is extra data taken after the fall time.



NOTE

Use the `pulse_sample_rate` command to set the sampling rate for pulse measurements.

NOTE

Before calling the `pulse_meas_timing` command, use the `pulse_meas_sm` or `pulse_meas_wfm` command to configure the measurement type.

Also see

Model 4200A-SCS Pulse Card (PGU and PMU) User's Manual, "Measurement timing" and "Measurement types"

pulse_meas_sm (on page 6-33)
pulse_sample_rate (on page 6-46)
pulse_meas_wfm (on page 6-36)

pulse_meas_wfm

This command configures waveform measurements.

Usage

```
int pulse_meas_wfm(int instr_id, int chan, int AcquireType, int AcquireMeasV, int
    AcquireMeasI, int AquireTimeStamp, int LLEComp);
```

instr_id	The instrument identification code of the PMU, such as PMU1 or PMU2
chan	Channel number of the pulse card: `1` or `2`
AcquireType	Acquisition type: ■ Discrete: `0` ■ Average: `1`
AcquireMeasV	Return voltage measurements: ■ Disable: `0` ■ Enable: `1`
AcquireMeasI	Return current measurements: ■ Disable: `0` ■ Enable: `1`
AcquireTimeStamp	Return timestamp readings (must be enabled to measure waveforms): ■ Disable: `0` ■ Enable: `1`
LLEComp	Load line effect compensation (LLEC): ■ LLEC disabled: `0` ■ LLEC enabled: `1`

Pulsers

4225-PMU

Pulse modes

Standard

Details

To use the `pulse_meas_wfm` command to configure waveform measurements, you select the data acquisition type, set the readings to be returned, enable or disable timestamps, and set load-line effect compensation (LLEC).

LLEC is only performed for standard pulse IV testing using PMU measure ranges. It is not performed when using 4225-RPM measure ranges. The active RPM circuitry provides its own analog LLEC (assuming a short cable from the RPM to the DUT).

Also see

Model 4200A-SCS Pulse Card (PGU and PMU) User's Manual, “Load-line effect compensation,” “Measurement types,” “Spot mean measurements,” and “Waveform measurements”

pulse_measrt

This command returns pulse source and measure data in pseudo real time.

Usage

```
int pulse_measrt(int instr_id, int chan, char *VMeasColName, char *IMeasColName,
    char *TimeStampColName, char *StatusColName);
```

instr_id	The instrument identification code: PMU1, PMU2, and so on
chan	Channel number of the pulse card: 1 or 2
VMeasColName	Column name for V-measure data in the Clarius Analyze sheet
IMeasColName	Column name for I-measure data in the Clarius Analyze sheet
TimeStampColName	Column name for the timestamp data in the Clarius Analyze sheet
StatusColName	Column name for the status data in the Clarius Analyze sheet

Pulsers

4225-PMU

Pulse mode

Standard and Segment Arb

Details

Use this command to return and display test data. The card returns data:

- If the time between measurements is very long.
- If the storage of the internal card (FIFO) is full.
- If a stepper is present, when the sweep is completed.
- At end of test.

The data is automatically placed in the Clarius Analyze sheet.

This command is also used to name the columns in the Clarius Analyze sheet.

This command must be called before calling `pulse_exec` to start the test.

NOTE

The `pulse_measrt` command is not compatible with using KXCI to call user libraries remotely (see “Calling KULT user libraries remotely” in *Model 4200A-SCS KXCI Remote Control Programming*. Use `PostDataDouble` for user test modules (UTMs) that will be called using KXCI.

Example

`pulse_measrt(PMU1, 1, "V-Measure", "I-Measure", "Timestamp", "Status");`
This example configures channel 1 of PMU1 to return data in pseudo real time.

Also see

pulse_exec (on page 6-20)
pulse_fetch (on page 6-24)

pulse_output

This command sets the pulse output of a pulse card channel on or off.

Usage

```
int pulse_output(int instr_id, long chan, long out_state);
```

instr_id	The instrument identification code of the pulse card, such as VPU1 or VPU2
chan	Channel number of the pulse card: `1` or `2`
out_state	Pulse output state: ■ Off: `0` (default) ■ On: `1`

Pulse modes

Standard, Full Arb, Segment Arb

Details

This command configures the channel to output and close the output relay.

If no 4225-RPM is used, this command connects the source to the device under test (DUT). The command `devclr` resets the pulse card source and disconnects the source from the DUT. The command `pulse_output(PMUx, chan, 0)` clears the physical connection to the DUT and resets the PMU source.

If a 4225-RPM is used with the PMU, this command prepares the pulse source when using a PMU with RPMs, but it does not close the output relay. The `rpm_config` command establishes the physical connection to the DUT. The `clrcon` command clears the physical connection to the DUT.

You can control each channel of the pulse card individually (on or off). When the channel is off, the output is in a high-impedance (open) state. After a channel is turned on, pulse output starts when a pulse trigger is initiated. Note that if a pulse delay has been set, pulse output starts after the delay period expires.

NOTE

It is good practice to routinely turn off the outputs of the pulse card after a test has been completed.

The `pulse_ssrc` command controls the high-endurance output relays (HEORs), and the `seg_arb_define` command defines a Segment Arb® waveform, which includes HEOR control.

Example

```
pulse_output(VPU1, 1, 0)
```

Turns off the output for pulse card channel 1.

Also see

clrcon (on page 7-2)
devclr (on page 4-9)
pulse_delay (on page 6-19)
pulse_ssrc (on page 6-48)
pulse_trig (on page 6-57)
pulse_current_limit (on page 6-17)
rpm_config (on page 6-66)
seg_arb_define (on page 6-67)

pulse_output_mode

This command sets the pulse output mode of a pulse card channel.

Usage

```
int pulse_output_mode(int instr_id, long chan, long mode);
```

instr_id	The instrument identification code of the pulse card, such as VPU1 or VPU2
chan	Channel number of the pulse card: 1 or 2
mode	Pulse output state: ■ NORMAL or 0 (default) ■ COMPLEMENT or 1

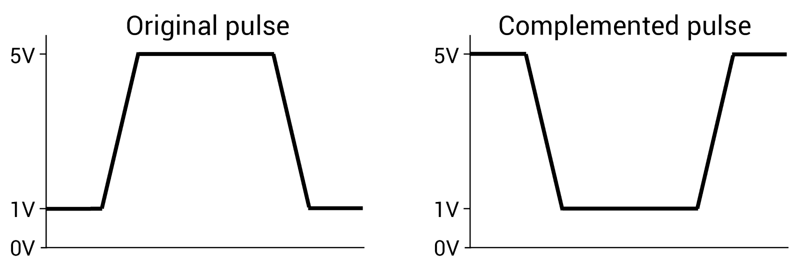
Pulse modes

Standard

Details

When a pulse card channel is set to COMPLEMENT, the V Low and V High voltage settings are swapped.

As shown in the following figure, when pulse is complemented, low pulse goes to the high level, and high pulse goes to the low level.



Example

`pulse_output_mode(VPU1, 1, COMPLEMENT)`
Sets the output mode for pulse card channel 1 to COMPLEMENT.

Also see

None

pulse_period

This command sets the period for pulse output.

Usage

```
int pulse_period(int instr_id, double period);
```

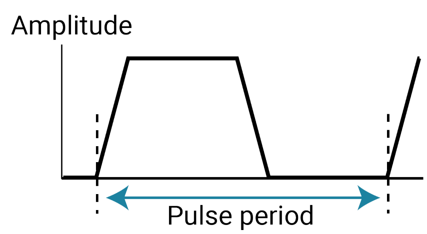
instr_id	The instrument identification code of the pulse card, such as VPU1 or VPU2
period	Pulse period (in seconds): ■ 5 V range: `20e-9` to `1` (20 ns to 1 s) ■ 20 V range: `500e-9` to `1` (500 ns to 1 s) ■ Default: `1e-6` (1 µs)

Pulse modes

Standard

Details

This command sets the pulse period for both channels of the pulse card. As shown below, the pulse period is measured at the median point (50 percent between the high and low pulse values) from the rising transition of the pulse to the rising transition of the next pulse.



The pulse period setting takes effect immediately during continuous pulse output. Otherwise, the period setting takes effect when the next trigger is initiated. The `pulse_trig` command is used to trigger continuous or burst output.

Example

`pulse_period(VPU1, 200e-9)`
Sets the pulse period of the pulse card to 200 ns.

Also see

pulse_trig (on page 6-57)

pulse_range

This command sets a pulse card channel for low voltage (fast speed) or high voltage (slow speed).

Usage

```
int pulse_range(int instr_id, long chan, double range);
```

instr_id	The instrument identification code of the pulse card, such as VPU1 or VPU2
chan	Channel number of the pulse card: `1` or `2`
range	Pulse range (in volts): 5 or `20` (default 5 V)

Details

Setting the pulse range of a pulse card channel to 5 V selects the low-voltage range. Selecting the low-voltage range also selects fast speed for pulse output. For fast speed, the minimum pulse width that can be set is 10 ns, and minimum rise and fall times can be set to 10 ns.

Setting the pulse range of a pulse card channel to 20 V selects the high-voltage range. Selecting the high-voltage range also selects slow speed for pulse output. For slow speed, the minimum pulse width that can be set is 250 ns, and the minimum rise and fall times can be set to 100 ns.

This setting takes effect when the next trigger is initiated. The following pulse parameters are then checked: period, width, rise time, fall time, and high and low voltage levels. If any of these parameters is out of bounds, it is reset to the default value.

NOTE

Use `pulse_range` before setting the voltage levels. When you use the `pulse_range` command, if you change the source range after setting the voltage levels in any pulse mode, it may result in voltage levels that are invalid for the new range setting.

NOTE

This command can also be used to set the voltage source range of the 4220-PGU and 4225-PMU. Use the `pulse_ranges` command to set the source and measure ranges of the 4225-PMU.

Example

`pulse_range(VPU1, 1, 20)`
Selects the high-voltage (slow speed) range for pulse card channel 1.

Also see

pulse_fall (on page 6-22)
pulse_vhigh (on page 6-62)
pulse_vlow (on page 6-63)
pulse_period (on page 6-40)
pulse_ranges (on page 6-42)
pulse_rise (on page 6-45)
pulse_width (on page 6-65)

pulse_ranges

This command sets the voltage pulse range and voltage/current measure ranges.

Usage

```
int pulse_ranges(int instr_id, int chan, double VSrcRange, int Vrange_type, double
   Vrange, int Irange_type, double IRange);
```

instr_id	The instrument identification code, such as VPU1, VPU2, PMU1, or PMU2
chan	Channel number of the pulse card: `1` or `2`
VSrcRange	Voltage source range: ■ `5` or `20` (into 50 Ω) ■ `10` or `40` (into ≥1 MΩ)
Vrange_type	Voltage measure range type (PMU): ■ Auto: `0` ■ Limited auto: `1` ■ Fixed: 2
Vrange	Voltage measure range (PMU) in volts: `10` or `40`; ignored if autorange is selected
Irange_type	Current measure range type (PMU): ■ Auto: `0` ■ Limited auto: `1` ■ Fixed: 2
IRange	Current measure range in amps; see **Details**; ignored if autorange is selected

Pulsers

4220-PGU
4225-PMU
4225-RPM

Pulse modes

Standard, Full Arb, Segment Arb

Details

The *`Vrange_type`*, *`Vrange`*, *`Irange_type`*, and *`Irange`* parameters are ignored by the PGU.

Autorange (`0`) and limited autorange (`1`) are not valid for the Segment Arb pulse mode.

You can set the source range independently for each PGU channel. There are two ranges for the output level: 5 V and 20 V (into a 50 Ω DUT load). Selecting the 5 V range also selects high-speed pulse output. For the 5 V high-speed range, the pulse period can be as short as 20 ns and pulse width can be set as short as 10 ns. This setting takes effect when the next pulse trigger is initiated.

For the PGU, use this command to set the voltage source range for pulse output.

For the PMU, use this command to:

- Set the voltage source range for pulse output.
- Set the voltage and current measure range types.
- Set the actual voltage and current measure ranges.

The measure range types for the PMU are:

- Fixed: Use this range type to specify a fixed measure range (*Vrange* or *Irange*).
- Limited Auto: Select this range type to use the fixed measure as the lowest range that will be used for automatic ranging.
- Auto: Use this range type to automatically select the optimum measure range. The specified fixed measure range (*Vrange* or *Irange*) is not used when autorange is enabled but must be a valid range.

The current ranges available depend on the source range and whether the system includes a 4225-RPM, as shown in the following table.

Current measure range (A)	PMU source range (V)	RPM source range (V)
0.8	n/a	40
0.2	10	n/a
0.01	10	10 or 40
0.001	n/a	10
0.0001	n/a	10 or 40
0.00001	n/a	10
0.000001	n/a	10
0.0000001	n/a	10

Auto or limited autoranging is available only when using the advanced mode in the `pulse_exec` command. Ranging is controlled per channel and may be combined with load-line effect compensation (LLEC) and thresholds. See `pulse_limits` command for thresholds.

The Segment Arb pulse mode does not allow range changes (no autorange) in a Segment Arb® waveform definition. Only fixed ranging is available for the Segment Arb pulse mode.

Example

```
pulse_ranges(PMU1, 1, 10, 0, 10, 0, 0.2);
```

This example sets the source-measure ranges for channel 1 of PMU1, where:

- *Instr_id* = PMU1
- *chan* = 1 (channel 1)
- *VSrcRange* = 10 V
- *Vrange_type* = Auto (0)
- *Vrange* = 10 V (value ignored because V-measure autorange is set)
- *Irange_type* = Auto (0)
- *Irange* = 200 mA (value ignored because I-measure autorange is set)

Also see

Model 4200A-SCS Pulse Card (PGU and PMU) User's Manual, "Setting up PMUs and PGUs in Clarius"

pulse_exec (on page 6-20)
pulse_limits (on page 6-31)
rpm_config (on page 6-66)

pulse_remove

This command removes a pulse channel from the test.

Usage

```
int pulse_remove(int instr_id, int chan, double voltage, unsigned long state);
```

instr_id	The instrument identification code: VPU1, VPU2, PMU1, PMU2, and so on
chan	Channel number of the pulse card: 1 or 2
voltage	Voltage to output when removing a channel
state	Output relay state: ■ PULSE OUTPUT OFF or 0: Open (disconnected) ■ PULSE OUTPUT ON or 1: Close (connected)

Pulsers

4220-PGU
4225-PMU

Pulse mode

Standard and Segment Arb

Details

This command is useful if you need one less channel for a pulse test that already exists. For example, you can use it to remove a channel from a long-term reliability test while allowing other channels to continue running.

Use the *voltage* and *state* parameters to remove a channel from a test that is running. Use the *voltage* parameter to set the output voltage. For example, you may want to set the output voltage to zero (0) when removing the channel. Use the *state* parameter to connect or disconnect the channel.

When you remove a channel from a test that is not running, the voltage and state parameters are ignored.

Example

```
pulse_remove(PMU2, 1, 0, 0);
```

This example removes channel 1 for PMU2, sets the voltage to 0 V, and opens the output relay.

Also see

None

pulse_rise

This command sets the rise transition time for the pulse card pulse output.

Usage

```
int pulse_fall(int instr_id, long chan, double riset);
```

instr_id	The instrument identification code of the pulse card, such as VPU1 or VPU2
chan	Channel number of the pulse card: `1` or `2`
riset	Pulse rise time in seconds (floating-point number): ■ Fast speed: `10e-9` to `33e-3` (10 ns to 33 ms) ■ Slow speed, 4220-PGU and 4225-PMU: `50e-9` to `33e-3` (50 ns to 33 ms) ■ Default: `100e-9` (100 ns)

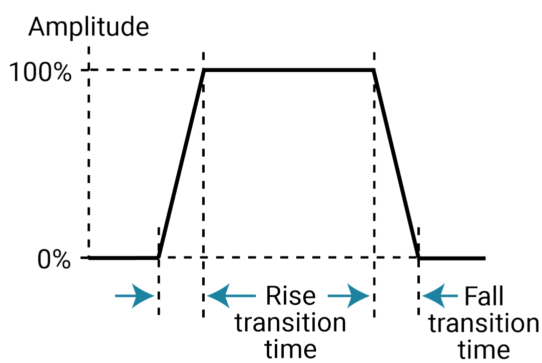
Pulse modes

Standard

Details

Rise and fall transition time can be set independently for each pulse card channel. There is a minimum slew rate for both the rise and fall transitions. For the fast speed range, the minimum is 362 µV/µs, or 1 V/2.7 ms. For the high-voltage range, the minimum slew rate is 1.8 mV/µs, or 1 V/500 µs. The `pulse_range` command is used to set pulse speed.

As shown below, the pulse rise time occurs between the 0 percent and 100 percent amplitude points on the rising edge of the pulse, where the amplitude is the difference between the V High and V Low pulse values.



The pulse rise time setting takes effect immediately during continuous pulse output. Otherwise, the rise time setting takes effect when the next trigger is initiated. The `pulse_trig` command is used to trigger continuous or burst output.

For slow speed, note that the minimum transition time for pulse source only (no measurement) on the 40 V range is 50 ns for the 4225-PMU and 4220-PGU.

NOTE

Use the `pulse_source_timing` command to set the pulse fall time for the 4220-PGU and 4225-PMU.

Example

`pulse_rise(VPU1, 1, 50e-9)`	
For fast speed, the sets the pulse rise time for channel 1 of the pulse card to 50 ns.	

Also see

pulse_fall (on page 6-22)
pulse_range (on page 6-41)
pulse_source_timing (on page 6-47)
pulse_trig (on page 6-57)

pulse_sample_rate

This command sets the measurement sample rate.

Usage

```
int pulse_sample_rate(INSTR_ID instr_id, double Sample_rate);
```

instr_id	The instrument identification code: PMU1, PMU2, and so on
Sample_rate	Sample rate: 200e6, 100e6, 50e6, 40e6, 33e6, 29e6, ... 1e3

Pulsers

4225-PMU

Pulse mode

Standard and Segment Arb

Details

Use this card-based command to set the measurement sample rate. The sample rate is the number of measurements (per second) that are performed by the PMU. The sample rate can be set from 200e6 to 200e6/n, where n = 1 to 200,000. The minimum sampling rate is 1E3 samples per second. The sample rate is a fixed rate (not adjustable within a test). For multi-card tests, set all cards to the same sample rate.

If a requested sample rate does not match an available rate, the next higher rate is used. For example, if 90e6 samples per second are sent, the sampling rate is set to 100e6 samples per second (200e6/2).

Example

`pulse_sample_rate(PMU1, 100E6);`	
This example command sets the sampling rate of the PMU to 100e6 samples per second.	

Also see

None

pulse_source_timing

This command sets the pulse period, pulse width, rise time, fall time, and delay time.

Usage

```
int pulse_source_timing(int instr_id, int chan, double period, double delay, double
   width, double rise, double fall);
```

instr_id	The instrument identification code: VPU1, VPU2, PMU1, PMU2, and so on
chan	Channel number of the pulse card: 1 or 2
period	Pulse period (in seconds) for both channels
delay	Delay time (in seconds) for the selected channel
width	Pulse width (in seconds for the selected channel
rise	Rise time (in seconds) for the selected channel
fall	Fall time (in seconds) for the selected channel

Pulsers

4220-PGU

4225-PMU

Pulse mode

Standard

Details

Use this command to set the timing parameters for the test. Pulse width, rise time, fall time, and delay are individually set for the selected channel. The pulse period setting applies to both channels.

This command returns errors if there is an invalid setting or combination of settings. The rise time of a pulse cannot be longer than the pulse width. The minimum time allowed for parameters width, rise, and fall is 20 ns. The minimum value for delay is 0 ns. When setting timing for a sample (waveform capture), setting the delay to a small value allows the PMU to better capture the rising edge of the pulse. This value is sample rate dependent, but for the 200 MSa/s rate, a pulse delay of 20 ns to 100 ns will allow the rising edge of the pulse to be captured.

Another internally enforced limit is the minimum off time. This is calculated as:

$$\text{minimum off time} = \text{period} - \text{delay} - \text{width} - 0.5 \times (\text{rise} + \text{fall})$$

The minimum off time may not be less than 40 ns. To see the whole pulse transition to high when capturing waveform data, use a small nonzero value like 10 ns for `pulse_delay`.

When a source timing parameter is already set to step or sweep, the step or sweep parameter overrides the timing parameter set by this command. For details, see `pulse_step_linear` and `pulse_sweep_linear`.

For example, if the `SWEEP_PERIOD_SP` parameter type is selected for the `pulse_sweep_linear` command, the period values for the sweep override the period setting for this command.

Example

```
pulse_source_timing(PMU1, 1, 0.02, 0.005, 0.01, 0.001, 0.001);
```

This example the following pulse source timing settings for the PMU, where:

- *instr_id* = PMU1
- *chan* = 1
- *period* = 0.02 (20 ms)
- *delay* = 0.005 (5 ms)
- *width* = 0.01 (10 ms)
- *rise* = 0.001 (1 ms)
- *fall* = 0.001 (1 ms)

Also see

Model 4200A-SCS Pulse Card (PGU and PMU) User's Manual, "Pulse parameter definitions"
pulse_step_linear (on page 6-50)
pulse_sweep_linear (on page 6-50)

pulse_ssrc

This command controls the high-endurance output relay (HEOR) for each output channel of the PGU.

Usage

```
int pulse_ssrc(int instr_id, long chan, long state, long ctrl);
```

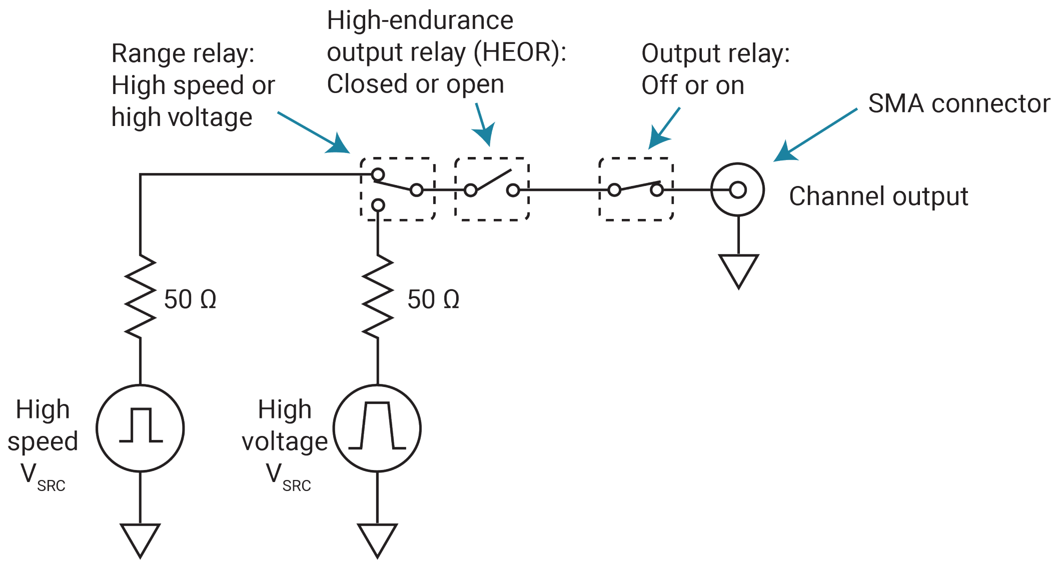
instr_id	The instrument identification code of the pulse card, such as VPU1 or VPU2
chan	Channel number of the pulse card: `1` or `2`
state	Open: `0` Close: `1` (default)
ctrl	How the HEOR will be controlled: ■ Auto (the Segment Arb pulse mode controls the HEOR): `0` (default) ■ Manual (*state* parameter opens or closes relay): `1` ■ Trigger out driven (relay state follows the trigger output): 2

Pulse modes

Standard, Full Arb, Segment Arb

Details

The high-endurance output relay (HEOR) is a solid-state relay (SSR) on each channel of the pulse card. Note that this setting is independent of the output relay (see `pulse_output`). A simplified schematic showing the relays is shown here.



Example

`pulse_ssrc(VPU1, 1, 0, 1)`
Selects manual control and opens the relay.

Also see

pulse_output (on page 6-38)
seg_arb_define (on page 6-67)
seg_arb_file (on page 6-69)

pulse_step_linear

This command configures the pulse stepping type.

Usage

```
int pulse_step_linear(int instr_id, int chan, int StepType, double start, double
    stop, double step);
```

instr_id	The instrument identification code: VPU1, VPU2, PMU1, PMU2, and so on
chan	Channel number of the pulse generator: `1` or `2`
StepType	Step type: ▪ `PULSE_AMPLITUDE_SP`: Sweeps pulse voltage amplitude ▪ `PULSE_BASE_SP`: Sweeps base voltage level ▪ `PULSE_DC_SP`: Sweeps dc voltage level ▪ `PULSE_PERIOD_SP`: Sweeps pulse period ▪ `PULSE_RISE_SP`: Sweeps pulse rise time ▪ `PULSE_FALL_SP`: Sweeps pulse fall time ▪ `PULSE_WIDTH_SP`: Sweeps full-width half-maximum pulse width ▪ `PULSE_DUAL_BASE_SP`: Dual sweeps base voltage level ▪ `PULSE_DUAL_AMPLITUDE_SP`: Dual sweeps pulse voltage amplitude ▪ `PULSE_DUAL_DC_SP`: Dual sweeps dc voltage level
start	Initial value for stepping
stop	Final value for stepping
step	Step size for stepping

Pulsers

4220-PGU

4225-PMU

Pulse mode

Standard

Details

The relationship between a step function and a sweep function for pulsing is similar to the same functions for SMUs. While a terminal of a device is at a pulse step, a pulse sweep is performed on another terminal.

A `pulse_step_linear` function cannot be used by itself. At least one PMU channel in a test must be a valid `pulse_sweep_linear` function call. The `PULSE_DUAL` options are for pulse dual sweeps. When you select Dual Sweep, the instrument sweeps from start to stop, then from stop to start. When you clear Dual Sweep, the instrument sweeps from start to stop only.

Use the *`start`*, *`stop`*, and *`step`* parameters to configure stepping. In addition, ensure that all pulse parameters are set before calling the `pulse_sweep_linear` or `pulse_step_linear` function. For example, when performing a pulse amplitude sweep (`PULSE_AMPLITUDE_SP`), use `pulse_vlow` to set the base voltage.

Amplitude and base level:

The pulse card can step or sweep amplitude (with base level fixed) or step or sweep base level (with amplitude fixed). Examples:

- `PULSE_AMPLITUDE_SP` (stepping or sweeping): Start = 1 V, stop = 5 V, step = 1 V
  Voltage amplitudes for pulse output sequence: 1 V, 2 V, 3 V, 4 V, and 5 V
  Note: Use the `pulse_vlow` function to set the base level voltage.
- `PULSE_BASE_SP` (stepping or sweeping): Start = 5 V, stop = 1 V, step = -1 V
  Voltage base levels for pulse output sequence: 5 V, 4 V, 3 V, 2 V, and 1 V
  Note: Use the `pulse_vhigh` function to set the amplitude voltage.

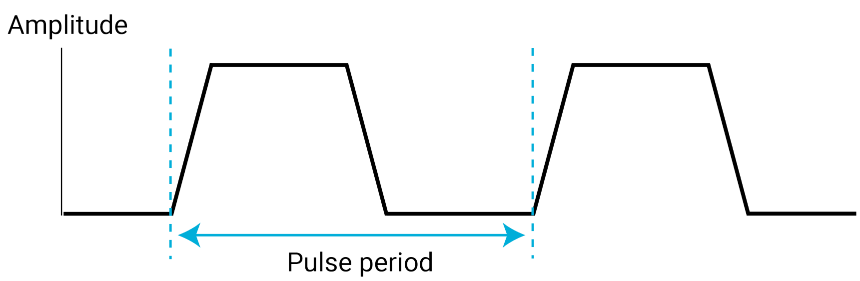
The dc voltage level: The pulse card can step or sweep a dc level. For example:

`PULSE_DC_SP` (stepping or sweeping): Start = 1 V, stop = 5 V, step = 1 V

The dc voltage output sequence: 1 V, 2 V, 3 V, 4 V, and 5 V

Pulse period:

The pulse period is the time interval between the start of the rising transition edge of consecutive output pulses, as shown in the following figure. To minimize self-heating effects, set a pulse period that is 10 to 100 times longer than the pulse width to produce a duty cycle that is 1 percent to 10 percent.



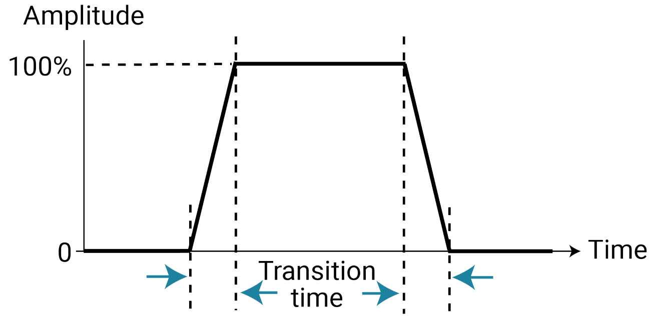
Pulse period example:

`PULSE_PERIOD_SP` (stepping or sweeping): Start = 0.01 s, stop = 0.05 s, step = 0.01 s

Pulse periods for output sequence: 0.01 s, 0.02 s, 0.03 s, 0.04 s, and 0.05 s

Pulse rise time and fall time:

Pulse rise time is the transition time (in seconds) from pulse low to pulse high. Pulse fall time is the transition time from pulse high to pulse low. The transition time is the interval between corresponding 0% and 100% amplitude points on the rising and falling edge of the pulse, as shown in the following figure.



Examples:

PULSE_RISE_SP (stepping or sweeping): Start = 0.001 s, stop = 0.005 s, step = 0.001 s

Rise times for pulse output sequence: 0.001 s, 0.002 s, 0.003 s, 0.004 s, and 0.005 s

PULSE_FALL_SP (stepping or sweeping): Start = 0.001 s, stop = 0.005 s, step = 0.001 s

Fall times for pulse output sequence: 0.001 s, 0.002 s, 0.003 s, 0.004 s, and 0.005 s

Pulse width:

The width of a pulse (in seconds) is measured at full-width half-maximum. For example:

PULSE_WIDTH_SP (stepping or sweeping): Start = 0.01 s, stop = 0.05 s, step = 0.01 s

Pulse widths for pulse output sequence: 0.01 s, 0.02 s, 0.03 s, 0.04 s, and 0.05 s

Dual Sweep:

The dual sweep allows for a voltage level sweep that goes up and down based on the voltage start stop and step. For example, a voltage amplitude sweep from 0 V to 4 V in 1 V steps. A single sweep (PULSE_AMPLITUDE_SP) would output 5 points: 0 V, 1 V, 2 V, 3 V, 4 V. A dual sweep version (PULSE_DUAL_AMPLITUDE_SP) outputs 10 points: 0 V, 1 V, 2 V, 3 V, 4 V, 4 V, 3 V, 2 V, 1 V, 0 V.

Also see

pulse_sweep_linear (on page 6-53)
pulse_vhigh (on page 6-62)
pulse_vlow (on page 6-63)
"Dual Sweep Option" in the *Model 4200A-SCS Clarius User's Manual*
"Operation mode timing diagrams" in the *Model 4200A-SCS Source-Measure Unit (SMU) User's Manual*
"PMU operation modes (PMU)" in the *Model 4200A-SCS Clarius User's Manual*
"Pulse width" in the *Model 4200A-SCS Clarius User's Manual*

pulse_sweep_linear

This command configures the pulse sweeping type.

Usage

```
int pulse_sweep_linear(int instr_id, int chan, int SweepType, double start, double
    stop, double step);
```

instr_id	The instrument identification code: VPU1, VPU2, PMU1, PMU2, and so on
chan	Channel number of the pulse generator: `1` or `2`
SweepType	Sweep type: ■ `PULSE_AMPLITUDE_SP`: Sweeps pulse voltage amplitude ■ `PULSE_BASE_SP`: Sweeps base voltage level ■ `PULSE_DC_SP`: Sweeps dc voltage level ■ `PULSE_PERIOD_SP`: Sweeps pulse period ■ `PULSE_RISE_SP`: Sweeps pulse rise time ■ `PULSE_FALL_SP`: Sweeps pulse fall time ■ `PULSE_WIDTH_SP`: Sweeps full-width half-maximum pulse width ■ `PULSE_DUAL_BASE_SP`: Dual sweeps base voltage level ■ `PULSE_DUAL_AMPLITUDE_SP`: Dual sweeps pulse voltage amplitude ■ `PULSE_DUAL_DC_SP`: Dual sweeps dc voltage level
start	Initial value for sweeping
stop	Final value for sweeping
step	Step size for sweeping

Pulsers

4220-PGU
4225-PMU

Pulse mode

Standard

Details

The relationship between a step function and a sweep function for pulsing is similar to the same functions for SMUs. While a terminal of a device is at a pulse step, a pulse sweep is performed on another terminal.

A `pulse_step_linear` function cannot be used by itself. At least one PMU channel in a test must be a valid `pulse_sweep_linear` function call. The `PULSE_DUAL` options are for pulse dual sweeps. When you select Dual Sweep, the instrument sweeps from start to stop, then from stop to start. When you clear Dual Sweep, the instrument sweeps from start to stop only.

Use the *`start`*, *`stop`*, and *`step`* parameters to configure stepping. In addition, ensure that all pulse parameters are set before calling the `pulse_sweep_linear` or `pulse_step_linear` function. For example, when performing a pulse amplitude sweep (`PULSE_AMPLITUDE_SP`), use `pulse_vlow` to set the base voltage.

Amplitude and base level:

The pulse card can step or sweep amplitude (with base level fixed) or step or sweep base level (with amplitude fixed). Examples:

- PULSE_AMPLITUDE_SP (stepping or sweeping): Start = 1 V, stop = 5 V, step = 1 V
  Voltage amplitudes for pulse output sequence: 1 V, 2 V, 3 V, 4 V, and 5 V
  Note: Use the pulse_vlow function to set the base level voltage.
- PULSE_BASE_SP (stepping or sweeping): Start = 5 V, stop = 1 V, step = -1 V
  Voltage base levels for pulse output sequence: 5 V, 4 V, 3 V, 2 V, and 1 V
  Note: Use the pulse_vhigh function to set the amplitude voltage.

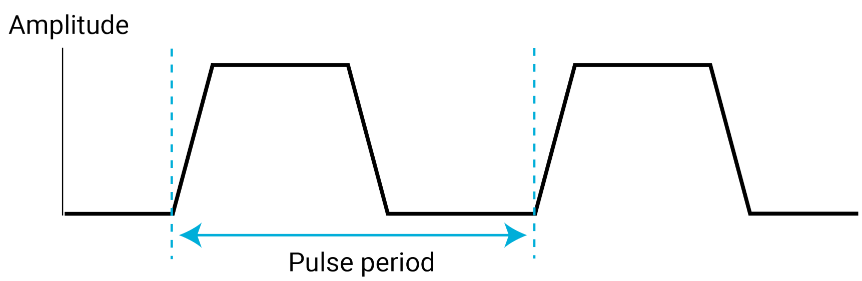
The dc voltage level: The pulse card can step or sweep a dc level. For example:

PULSE_DC_SP (stepping or sweeping): Start = 1 V, stop = 5 V, step = 1 V

The dc voltage output sequence: 1 V, 2 V, 3 V, 4 V, and 5 V

Pulse period:

The pulse period is the time interval between the start of the rising transition edge of consecutive output pulses, as shown in the following figure. To minimize self-heating effects, set a pulse period that is 10 to 100 times longer than the pulse width to produce a duty cycle that is 1 percent to 10 percent.



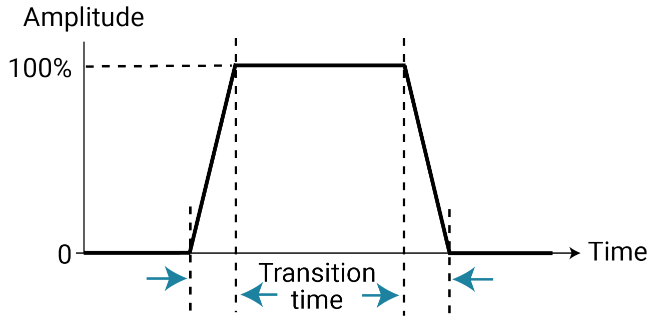
Pulse period example:

PULSE_PERIOD_SP (stepping or sweeping): Start = 0.01 s, stop = 0.05 s, step = 0.01 s

Pulse periods for output sequence: 0.01 s, 0.02 s, 0.03 s, 0.04 s, and 0.05 s

Pulse rise time and fall time:

Pulse rise time is the transition time (in seconds) from pulse low to pulse high. Pulse fall time is the transition time from pulse high to pulse low. The transition time is the interval between corresponding 0% and 100% amplitude points on the rising and falling edge of the pulse, as shown in the following figure.



Examples:

`PULSE_RISE_SP` (stepping or sweeping): Start = 0.001 s, stop = 0.005 s, step = 0.001 s

Rise times for pulse output sequence: 0.001 s, 0.002 s, 0.003 s, 0.004 s, and 0.005 s

`PULSE_FALL_SP` (stepping or sweeping): Start = 0.001 s, stop = 0.005 s, step = 0.001 s

Fall times for pulse output sequence: 0.001 s, 0.002 s, 0.003 s, 0.004 s, and 0.005 s

Pulse width:

The width of a pulse (in seconds) is measured at full-width half-maximum. For example:

`PULSE_WIDTH_SP` (stepping or sweeping): Start = 0.01 s, stop = 0.05 s, step = 0.01 s

Pulse widths for pulse output sequence: 0.01 s, 0.02 s, 0.03 s, 0.04 s, and 0.05 s

Dual Sweep:

The dual sweep allows for a voltage level sweep that goes up and down based on the voltage start stop and step. For example, a voltage amplitude sweep from 0 V to 4 V in 1 V steps. A single sweep (`PULSE_AMPLITUDE_SP`) would output 5 points: 0 V, 1 V, 2 V, 3 V, 4 V. A dual sweep version (`PULSE_DUAL_AMPLITUDE_SP`) outputs 10 points: 0 V, 1 V, 2 V, 3 V, 4 V, 4 V, 3 V, 2 V, 1 V, 0 V.

Example

```
pulse_sweep_linear(PMU1, 1, PULSE_AMPLITUDE_SP, 1, 5, 1);
```

This example configures channel 1 of the PMU to perform an amplitude sweep from 1 V to 5 V in 1 V steps.

Also see

pulse_step_linear (on page 6-50)
pulse_vhigh (on page 6-62)
pulse_vlow (on page 6-63)
"Dual Sweep Option" in the *Model 4200A-SCS Clarius User's Manual*
"Operation mode timing diagrams" in the *Model 4200A-SCS Source-Measure Unit (SMU) User's Manual*
"PMU operation modes (PMU)" in the *Model 4200A-SCS Clarius User's Manual*
"Pulse width" in the *Model 4200A-SCS Clarius User's Manual*

pulse_train

This command configures the pulse card to output a pulse train using fixed voltage values.

Usage

```
int pulse_train(int instr_id, int chan, double Vbase, double Vamplitude);
```

instr_id	The instrument identification code: VPU1, VPU2, PMU1, PMU2, and so on
chan	Channel number of the pulse card: 1 or 2
Vbase	Voltage level for pulse base level
Vamplitude	Voltage level for pulse amplitude

Pulsers

4220-PGU
4225-PMU

Pulse mode

Standard

Details

The configured pulse train will not change for the selected channel, but any sweep or step timing changes will affect the timing parameters of the train. For details on timing, see `pulse_step_linear` and `pulse_sweep_linear`. A `pulse_train` command cannot be used by itself in a test. When using a PMU, at least one PMU channel in a test must be a valid `pulse_sweep_linear` function call.

Example

`pulse_train(PMU1, 1, 0, 5);`
This example configures channel 1 of the PMU to output a 0 to 5 V pulse train.

Also see

pulse_step_linear (on page 6-50)
pulse_sweep_linear (on page 6-50)

pulse_trig

This command selects the trigger mode (continuous, burst, or trigger burst) and initiates the start of pulse output or arms the pulse card.

Usage

```
int pulse_trig(int instr_id, long mode);
```

instr_id	The instrument identification code of the pulse card, such as VPU1 or VPU2
mode	Trigger mode: ▪ Burst: 0 ▪ Continuous: 1 ▪ Trigger burst: 2

Pulse modes

Standard, Full Arb, Segment Arb

Details

With the software trigger source selected, this command sets the trigger mode (continuous, burst, or trig burst) for both pulse card channels, and initiates the start of pulse output.

A burst is a finite number of pulses (1 to $2^{32}-1$). The only difference between burst and trig burst is the behavior of trigger output. When using the burst or trig burst trigger mode, make sure to first set the pulse count before starting pulse output. The `pulse_burst_count` command is used to set the burst count.

If pulse delay is set to zero (0), pulse output will start immediately after it is triggered. If pulse delay is more than 0, pulse output will start after the delay period expires

This setting affects both output channels.

Example

`pulse_trig(VPU1, 0)`
Initiates (triggers) burst pulse output.

Also see

pulse_burst_count (on page 6-14)
pulse_delay (on page 6-19)
pulse_halt (on page 6-29)
pulse_output (on page 6-38)
pulse_trig_source (on page 6-60)
"Triggering" in the *Model 4200A-SCS Pulse Card (PGU and PMU) User's Manual*

pulse_trig_output

This command sets the output trigger on or off.

Usage

```
int pulse_trig_output(int instr_id, long state);
```

instr_id	The instrument identification code of the pulse card, such as VPU1 or VPU2
state	Output trigger state: ■ Off: `0` (default for Segment Arb and full arb) ■ On: `1` (default for standard pulse)

Pulse modes

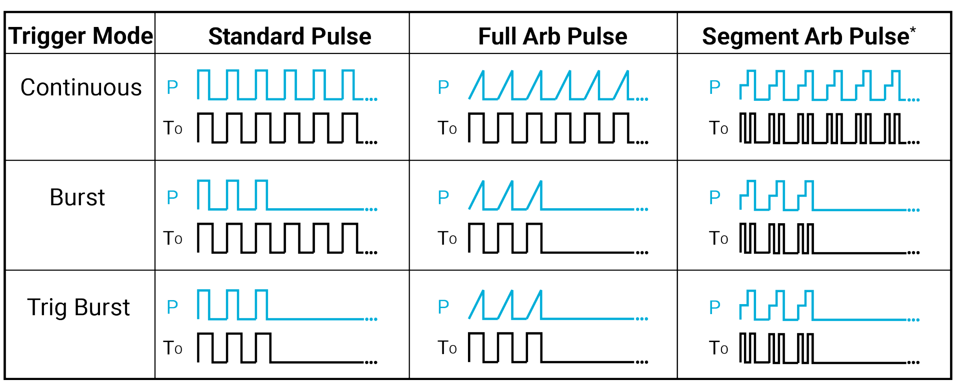
Standard, Full Arb, Segment Arb

Details

This command turns the TTL-level trigger output pulse on or off. The pulse is used to synchronize pulse output with the operations of an external instrument. When connected to a scope, each output pulse of triggers a scope waveform measurement.

When output trigger is enabled, an output pulse will initiate a TTL-level, 50% duty cycle output trigger pulse. The trigger pulses are available at the TRIGGER OUT connector of the pulse generator card.

The figure below shows the behavior of output triggers (T_O) for the three trigger modes. Notice that for the Burst mode, output triggers continue even though pulse output has stopped. For the trigger burst mode, output triggers stop when the pulse output stops.



P = Pulse output
T_O = Trigger output
*Segment Arb has user-defined trigger output (0 or 1) for each segment.

Example

`pulse_trig_output(VPU1, 1)`
Sets the pulse card trigger output on.

Also see

"Triggering" in the *Model 4200A-SCS Pulse Card (PGU and PMU) User's Manual*
pulse_trig_polarity (on page 6-59)

pulse_trig_polarity

This command sets the polarity (positive or negative) of the pulse card output trigger.

Usage

```
int pulse_trig_polarity(int instr_id, long polarity);
```

instr_id	The instrument identification code of the pulse card, such as VPU1 or VPU2
polarity	Output trigger polarity: ■ Negative, falling edge: `0` ■ Positive, rising edge: `1` ■ Default: `1`

Pulse modes

Standard, Full Arb, Segment Arb

Details

Trigger output provides a TTL-level output that is at the same frequency (period) as the pulse card output channels, but has a 50% duty cycle. It is used to synchronize pulse outputs with the operations of an external instrument.

The external instrument that is connected to the pulse card external trigger may require a positive-going (rising-edge) pulse or a negative-going (falling-edge) pulse for triggering.

If a polarity value other than 0 or 1 is sent, it will map to 0 or 1 in the following manner:

```
if(polarity <= 0)
   pol = NEGATIVE;
else
   pol = POSITIVE;
```

NOTE

4220-PGU and 4225-PMU: Do not use the two external falling trigger sources (`pulse_trig_source` function) with the positive trigger output polarity (`pulse_trig_polarity` function) on the master card that triggers itself and other subordinate cards. These two falling trigger sources should only be used when an external piece of equipment is used to supply the trigger pulses to the 4220-PGU and 4225-PMU. This applies to all three pulse modes (standard pulse, Segment Arb, and full arb).

Example

`pulse_trig_polarity(VPU1, 0)`
Sets the pulse card trigger output for negative polarity.

Also see

pulse_trig_output (on page 6-58)
pulse_trig_source (on page 6-60)
"Triggering" in the *Model 4200A-SCS Pulse Card (PGU and PMU) User's Manual*

pulse_trig_source

This command sets the trigger source.

Usage

```
int pulse_trig_source(int instr_id, long source);
```

instr_id	The instrument identification code of the pulse card, such as VPU1 or VPU2
source	Trigger source: ■ Software: `0` (default) ■ External – initial trigger only – rising: 1 ■ External – initial trigger only – falling: 2 ■ External – trigger per pulse – rising: 3 ■ External – trigger per pulse – falling: 4 ■ Internal trigger bus: 5

Pulse modes

Standard, Full Arb, Segment Arb

Details

This command sets the trigger source that is used to trigger the pulse card to start its output.

If the software trigger source selected, the `pulse_trig` command will select the trigger mode (continuous, burst, or trig burst), and initiate the start of pulse output.

If an external trigger source selected, the `pulse_trig` command will select the trigger mode and arm pulse output. Pulse output will start when the required external trigger pulse is applied to the Trigger In connector of the pulse card. There is a trigger-in delay of 560 ns. This is the delay from the trigger-in pulse to the time of the rising edge of the output pulse.

NOTE

4220-PGU and 4225-PMU: Do not use the two external falling trigger sources (`pulse_trig_source` function) with the positive trigger output polarity (`pulse_trig_polarity` function) on the master card that triggers itself and other subordinate cards. These two falling trigger sources should only be used when an external piece of equipment is used to supply the trigger pulses to the 4220-PGU and 4225-PMU. This applies to all three pulse modes (standard pulse, Segment Arb, and full arb).

NOTE

Because trigger source is a card-level setting and not a channel setting, using channel 1 or 2 will set the card to the specified source card 1. Similarly, channel 3 or 4 will set the source for card 2.

For an initial trigger only setting, only the first rising or falling trigger pulse will start and control pulse output.

For a trigger per pulse setting, rising or falling edge trigger pulses will start and control pulse output. After the initial pulse, the pulse output, either continuous or burst, will be output based on the internal pulse generator clock. If pulse-to-pulse synchronization is required over higher count pulse trains, use the trigger per pulse mode.

The Trigger In sources are:

- **External, initial trigger only (rising)**: The first rising-edge trigger pulse applied to TRIGGER In will start and control pulse output.
- **External, initial trigger only (falling)**: Same as above, except the initial falling-edge trigger will start and control pulse output.
- **External, trigger per pulse (rising)**: Rising-edge trigger pulses applied to TRIGGER IN will start and control pulse output.
- **External, trigger per pulse (falling)**: Same as above, except falling-edge triggers will start and control pulse output.
- **Internal Trigger Bus**: The internal bus trigger source is used for synchronizing multiple PMU/PGU cards for standard pulse using the legacy pulse commands (`pulse_vhigh`, `pulse_vlow`, `pulse_width`, and so on). This trigger source is used only by the 4220-PGU and 4225-PMU.

The internal bus trigger source is used for synchronizing multiple PMU/PGU cards for standard pulse using the legacy pulse commands (`pulse_vhigh`, `pulse_vlow`, `pulse_width`, and so on). This trigger source is used only by the 4220-PGU and 4225-PMU.

Example

`pulse_trig_source(VPU1, 1)`
Sets the trigger source to external – initial trigger only – rising.

Also see

pulse_trig (on page 6-57)
pulse_trig_polarity (on page 6-59)

pulse_vhigh

This command sets the pulse voltage high level.

Usage

```
int pulse_vhigh(INSTR_ID instr_id, long chan, double vhigh);
```

instr_id	The instrument identification code, such as VPU1 or VPU2
chan	Channel number of the pulse card: 1 or 2
vhigh	Pulse voltage high value in volts (floating-point number): ▪ Fast speed: –5 to +5 ▪ Slow speed: –20 to +20 ▪ Default: 0

Pulse modes

Standard

Details

Pulse voltage high can be set independently for each pulse card channel.

For a 50 Ω load:

- 5 V range (lower voltages and higher transitions): Pulse high and pulse low can be set from –5 V to +5 V.
- 20 V range (higher voltages and lower transitions): Pulse high and pulse low can be set from –20 V to +20 V.

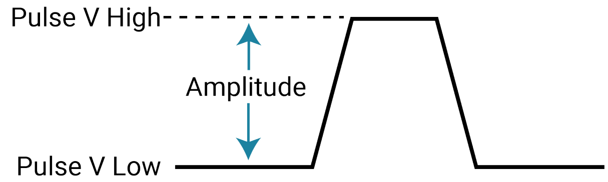
For a 1 MΩ load:

- 5 V range (high speed): Pulse high and pulse low can be set from –10 V to +10 V.
- 20 V range (high voltage): Pulse high and pulse low can be set from –40 V to +40 V.

The `pulse_range` command sets the pulse voltage range.

Set the `pulse_range` command before setting the voltage levels. When using the `pulse_range` command, changing the source range after setting voltage levels (in any pulse mode) will result in voltage levels that are invalid for the new range setting.

As shown in the following figure, the pulse voltage high is typically set as the greater pulse voltage value. However, voltage high can be any valid voltage value. That means pulse voltage high can be less than voltage low. When started, the pulse transitions from voltage low to voltage high and then back to voltage low. The voltage remains at voltage low for the remainder of the pulse period.



The pulse voltage high setting takes effect immediately during continuous pulse output. Otherwise, the voltage high setting takes effect when the next trigger is initiated. The `pulse_trig` command is used to trigger continuous or burst output.

CAUTION

The `pulse_vlow`, `pulse_vhigh`, and `pulse_dc_output` commands set the voltage value output by the pulse channel when it is turned on (using `pulse_output`). If the output is already enabled, these commands change the voltage level immediately, before the pulsing is started with a `pulse_trig` command.

Example

`pulse_vhigh(VPU1, 1, 2.5)`
Sets the pulse voltage high value for channel 1 of the pulse card to 2.5 V.

Also see

pulse_dc_output (on page 6-18)
pulse_output (on page 6-38)
pulse_range (on page 6-41)
pulse_trig (on page 6-57)
pulse_vlow (on page 6-63)

pulse_vlow

This command sets the pulse voltage low value.

Pulse modes

Standard

Usage

`int pulse_lhigh(int instr_id, long chan, double vlow);`

instr_id	The instrument identification code of the pulse card, such as VPU1 or VPU2
chan	Channel number of the pulse card: 1 or 2
vlow	Pulse voltage low value in volts (floating-point number): ■ Fast speed: –5 to +5 ■ Slow speed: –20 to +20 ■ Default: 0

Details

Pulse voltage low can be set independently for each pulse card channel.

For a 50 Ω load:

- 5 V range (lower voltages and higher transitions): Pulse high and pulse low can be set from –5 V to +5 V.
- 20 V range (higher voltages and lower transitions): Pulse high and pulse low can be set from –20 V to +20 V.

For a 1 MΩ load:

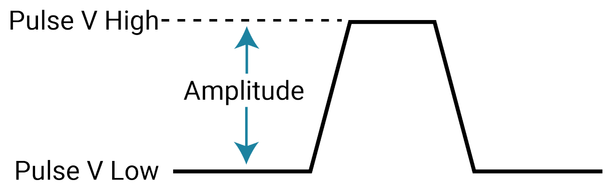
- 5 V range (high speed): Pulse high and pulse low can be set from –10 V to +10 V.
- 20 V range (high voltage): Pulse high and pulse low can be set from –40 V to +40 V.

The `pulse_range` command determines the pulse voltage range.

NOTE

Set the `pulse_range` command before setting the voltage levels. When using the `pulse_range` command, changing the source range after setting voltage levels (in any pulse mode) will result in voltage levels that are invalid for the new range setting.

As shown below, the pulse voltage low is typically set as the lower pulse voltage value. However, voltage low can be any valid voltage value. That means pulse voltage low can be less than voltage high. When started, the pulse transitions from voltage low to voltage high and then back to voltage low. The voltage remains at voltage low for the remainder of the pulse period.



The pulse voltage low setting takes effect immediately during continuous pulse output. Otherwise, the voltage low setting takes effect when the next trigger is initiated. The `pulse_trig` command is used to trigger continuous or burst output.

CAUTION

The `pulse_vlow`, `pulse_vhigh`, and `pulse_dc_output` commands set the voltage value output by the pulse channel when it is turned on (using `pulse_output`). If the output is already enabled, these commands change the voltage level immediately, before the pulsing is started with a `pulse_trig` command.

Example

`pulse_vlow(VPU1, 1, 0.5)`
Sets the pulse voltage low value for channel 1 of the pulse card to 0.5 V.

Also see

pulse_dc_output (on page 6-18)
pulse_output (on page 6-38)
pulse_range (on page 6-41)
pulse_trig (on page 6-57)
pulse_vhigh (on page 6-62)

pulse_width

This command sets the pulse width for pulse output.

Usage

```
int pulse_width(int instr_id, long chan, double width);
```

instr_id	The instrument identification code of the pulse card, such as VPU1 or VPU2
chan	Channel number of the pulse card: 1 or 2
width	Pulse width in seconds: ■ Fast speed (5 V): 10e−9 to (Period – 10e−9) ■ Slow speed (20 V): 250e−9 to (Period – 10e−9) ■ Default: `500e-9` (500 ns)

Pulse modes

Standard

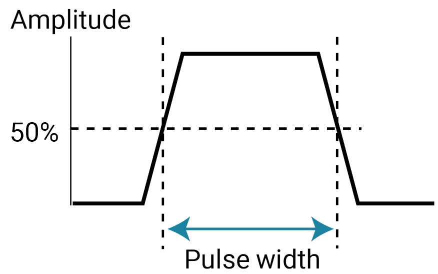
Details

NOTE

Use the `pulse_source_timing` command to set the pulse width for the 4220-PGU and 4225-PMU.

You can set the pulse width independently for each pulse card channel. The `pulse_range` command is used to set pulse speed.

Pulse card pulse width is based on the full width at half-maximum method (FWHM). As shown below, the pulse width is measured at the median (50 percent amplitude) point from the rising edge of the pulse to the falling edge of the pulse.



The maximum pulse width that can be set depends on the selected period for the pulse. For example, if the period is set for 500 ns, the maximum pulse width that can be set for the fast speed is 490 ns (500 ns – 10 ns = 490 ns).

The pulse width setting takes effect immediately during continuous pulse output. Otherwise, the width setting takes effect when the next trigger is initiated. The `pulse_trig` command is used to trigger continuous or burst output.

Example

`pulse_width(VPU1, 1, 250e-9)`	
Sets the pulse width for channel 1 to 250 ns.	

Also see

pulse_period (on page 6-40)
pulse_range (on page 6-41)
pulse_source_timing (on page 6-47)
pulse_trig (on page 6-57)

rpm_config

This command sends switching commands to the 4225-RPM.

Usage

```
int rpm_config(int instr_id, long chan, long modifier, long value);
```

instr_id	The instrument identification code: Identifier such as PMU1, SMU1, CVU1, PMU2, or SMU2
chan	Channel number of the pulse generator: `1` or `2`
modifier	Parameter to modify: `KI_RPM_PATHWAY`
value	Value to set modifier: ■ `KI_RPM_PULSE` or `0`: Selects pulsing (4225-RPM) ■ `KI_RPM_CV_2W` or `1`: Selects 2-wire CVU ■ `KI_RPM_CV_4W` or `2`: Selects 4-wire CVU ■ `KI_RPM_SMU` or `3`: Selects SMU (4200-SMU or 4201-SMU)

Pulsers

4225-PMU with the 4225-RPM

Pulse mode

Standard (two-level pulsing), Segment Arb, and full arb

Details

The 4225-RPM includes input connections for the CVU and SMU. Use this command to control switching inside the RPM to connect the PMU, CVU, or SMU to the output.

When using the PMU with the RPM, `rpm_config` must be called to connect the pulse source to the RPM output. Note that if there is no RPM connected to the PMU channel, the `rpm_config` command will not cause an error. The RPM connection is cleared by the `clrcon` command.

The ID of instrument to be used in the test sequence should be used as the setting for the *instr_id* parameter.

Example

`rpm_config(PMU1, 1, KI_RPM_PATHWAY, KI_RPM_PULSE);`	
This example sets channel 1 of the RPM for pulsing.	

Also see

clrcon (on page 7-2)

seg_arb_define

This command defines the parameters for a Segment Arb® waveform.

Usage

```
int seg_arb_define(int instr_id, long chan, long nsegments, double *startvals,
   double *stopvals, double *timevals, long *triggervals, long *outputRelayVals);
```

instr_id	The instrument identification code of the pulse card, such as VPU1 or VPU2
chan	Channel number of the pulse card: 1 or 2
nsegments	The number of values in each of the arrays (1024 maximum)
startvals	An array of start voltage values for each segment (in volts)
stopvals	An array of stop voltage values for each segment (in volts)
timevals	An array of time values for each segment: 20e-9 (20 ns) minimum
triggervals	An array of trigger values: ■ Trigger low: 0 ■ Trigger high: 1
outputRelayVals	An array of values to control the high endurance output relay: ■ Open: 0 ■ Closed: 1

Pulsers

4220-PGU
4225-PMU

Pulse modes

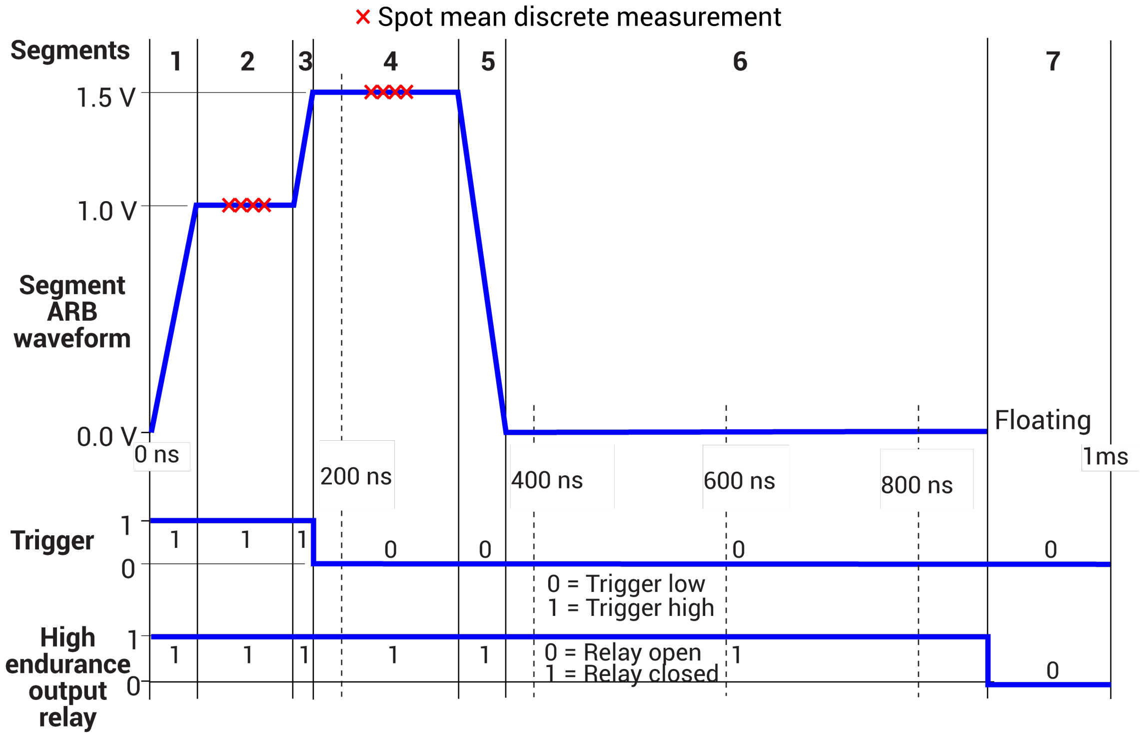
Source, Segment Arb

Details

You can configure each channel to output its own unique Segment Arb waveform. A Segment Arb waveform is made up of user-defined segments. Each segment can have a unique time interval, start value, stop value, output trigger level (TTL high or low), and output relay state (open or closed).

To configure each channel to output a unique Segment Arb® waveform, refer to seg_arb_sequence (on page 6-70).

The following arrays are required for the example Segment Arb waveform shown here.



Start	Stop	Time	Trigger	Output relay
startvals[0] = 0.0	stopvals[0] = 1.0	timevals[0] = 50e-9	triggervals[0] = 1	outputRelayVals[0] = 0
startvals[1] = 1.0	stopvals[1] = 1.0	timevals[1] = 100e-9	triggervals[1] = 1	outputRelayVals[1] = 0
startvals[2] = 1.0	stopvals[2] = 1.5	timevals[2] = 20e-9	triggervals[2] = 1	outputRelayVals[2] = 0
startvals[3] = 1.5	stopvals[3] = 1.5	timevals[3] = 150e-9	triggervals[3] = 0	outputRelayVals[3] = 0
startvals[4] = 1.5	stopvals[4] = 0.0	timevals[4] = 50e-9	triggervals[4] = 0	outputRelayVals[4] = 0
startvals[5] = 0.0	stopvals[5] = 0.0	timevals[5] = 500e-9	triggervals[5] = 0	outputRelayVals[5] = 0
startvals[6] = 0.0	stopvals[6] = 0.0	timevals[6] = 130e-9	triggervals[6] = 0	outputRelayVals[6] = 1

Also see

arb_file (on page 6-4)
arb_array (on page 6-3)
seg_arb_file (on page 6-69)
"Pulse-measure synchronization" in the *Model 4200A-SCS Pulse Card (PGU and PMU) User's Manual*
"Segment Arb waveform" in the *Model 4200A-SCS Pulse Card (PGU and PMU) User's Manual*

seg_arb_file

This command is used to load a waveform from an existing Segment Arb® waveform file.

Usage

```
int seg_arb_file(INSTR_ID instr_id, long chan, char *fname);
```

instr_id	The instrument identification code of the pulse card, such as VPU1 or VPU2
chan	Channel number of the pulse card: 1 or 2
fname	The name of the waveform file; name must be in quotes

Pulse modes

Source only, Segment Arb

Details

This command loads a waveform from an existing Segment Arb `.ksf` waveform file into the pulse card. A Segment Arb waveform can be loaded for each channel of the pulse card. Once loaded, use `pulse_output` to turn on the appropriate channel. Use `pulse_trig` to select the trigger mode and start (or arm) pulse output.

When specifying the file name, include the full command path with the file name. Existing `.ksf` waveforms are typically saved in the `SarbFiles` folder at the following command path location:

`C:\s4200\kiuser\KPulse\SarbFiles`

A Segment Arb waveform can be created using KPulse and saved as a `.ksf` waveform file.

You can modify a waveform in an existing `.ksf` file using a text editor.

Example

`seg_arb_file(VPU1, 1, "C:\\s4200\\kiuser\\KPulse\\SarbFiles\\sarb3.ksf")`
Loads a Segment Arb file named `sarb3.ksf` (saved in the `SarbFiles` folder) into the pulse card for channel 1.

Also see

arb_array (on page 6-3)
arb_file (on page 6-4)
"KPulse (for Keithley Pulse Cards)" in the *Model 4200A-SCS Pulse Card (PGU and PMU) User's Manual*
pulse_output (on page 6-38)
pulse_trig (on page 6-57)
seg_arb_define (on page 6-67)

seg_arb_sequence

This command defines the parameters for a Segment Arb waveform pulse-measure sequence.

Usage

```
int seg_arb_sequence(int instr_id, long chan, long SeqNum, long NumSegments, double
   *StartV, double *StopV, double *Time, long *Trig, long *SSR, long *MeasType,
   double *MeasStart, double *MeasStop);
```

instr_id	The instrument identification code of the pulse card, such as VPU1 or VPU2
chan	Channel number of the pulse card: 1 or 2
SeqNum	Sequence ID number (1 to 512, per channel) to uniquely identify this sequence
NumSegments	Total number of segments in this sequence
StartV	An array of start voltage levels
StopV	An array of stop voltage levels
Time	An array of segment time durations (in seconds with 10 ns resolution, 20 ns minimum)
Trig	An array of trigger values (for trigger output only): ■ Trigger low: 0 ■ Trigger high: 1
SSR	An array of values to control the high endurance output relay: ■ Open: 0 ■ Closed: 1
MeasType	PGU: 0 PMU: An array of measure types: ■ No measurements for this segment: 0 ■ Spot mean discrete: 1 ■ Waveform discrete: 2 ■ Spot mean average: 3 ■ Waveform average: 4
MeasStart	PGU: 0 PMU: An array of start measurement times (in seconds, with 10 ns resolution); a zero (0) second setting sets measure to start at the beginning of the segment
MeasStop	PGU: 0 PMU: An array of stop measurement times (in seconds, with 10 ns resolution); this is the elapsed time, within the segment, when the measurement stops

Pulsers

4220-PGU
4225-PMU

Pulse mode

Segment Arb

Details

Use this command to configure each channel to output a unique Segment Arb® waveform. For the PMU, this also configures each channel to make measurements.

A Segment Arb sequence is made up of user-defined segments (up to 2048 per channel). Each sequence can have a unique start voltage, stop voltage, time interval, output trigger level (TTL high or low), and output relay state (open or closed). For PMUs, each can have a unique pulse measurement type, measurement start time, and measurement stop time.

A defined sequence is uniquely identified by its specified channel number and sequence ID number. This command defines the sequences, or building blocks, that are typically used for a BTI (bias temperature instability semiconductor reliability) test.

A sequence is defined as three or more segments with seamless voltage transitions. Seamless means that there are no voltage differences — the voltage level for the last point in a segment must equal the voltage level for the first point of the next segment. Note that all segment transitions must be seamless. The minimum time per sequence is 20 ns.

One or more defined sequences are combined into a Segment Arb waveform using the `seg_arb_waveform` command. All sequence transitions must also be seamless. The example below shows an example of a waveform that consists of three sequences.

The 4220-PGU does not have pulse-measure capability. When this command for the PGU is called, the parameter values for *`MeasType`*, *`MeasStart`*, and *`MeasStop`* are ignored.

Example

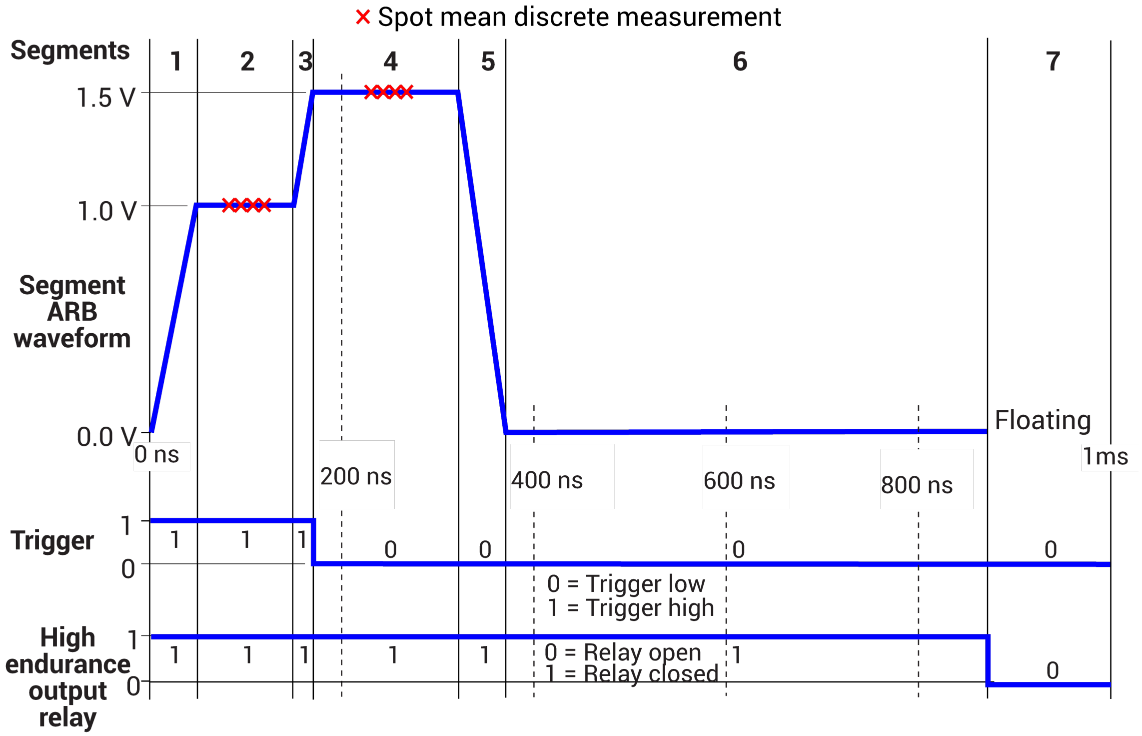
This command defines the Segment Arb sequence shown in the following figure.

```
seg_arb_sequence(PMU1, 1, 1, 7, Start_Volt, Stop_Volt, Time_Interval, Trig_Level,
   Output_Relay, Meas_Type, Meas_Start, Meas_Stop);
```

The arrays for the `seg_arb_function` are:

```
double Start_Volt[7] = {0, 1, 1, 1.5, 1.5, 0, 0};
double Stop_Volt[7] = {1, 1, 1.5, 1.5, 0, 0, 0};
double Time_Interval[7] = {50e-9, 100e-9, 20e-9, 150e-9, 50e-9, 500e-9, 130e-9};
int Trig_Level[7] = {1, 1, 1, 0, 0, 0, 0};
int Output_Relay[7] = {1, 1, 1, 1, 1, 1, 0};
int Meas_Type[7] = {0, 1, 0, 1, 0, 0, 0};
double Meas_Start[7] = {0, 25e-9, 0, 50e-9, 0, 0, 0};
double Meas_Stop[7] = {0, 75e-9, 0, 100e-9, 0, 0, 0};
```

The following figure shows an example of a Segment Arb sequence defined by the `seg_arb_sequence` command. Spot mean discrete measurements are performed on segments two and four.



The following table lists the `seg_arb_sequence` parameter arrays for the Segment Arb sequence shown in the example.

Parameter arrays for the seg_arb_sequence example

Parameter and Array Name	Value						
SegNum **Seg_Num**	1	2	3	4	5	6	7
StartV **Start_Volt**	0 V	1 V	1 V	1.5 V	1.5 V	0 V	0 V
StopV **Stop_Volt**	1 V	1 V	1.5 V	1.5 V	0 V	0 V	0 V
Time **Time_Interval**	50e-9 s	100e-9 s	20e-9 s	150e-9 s	50e-9 s	500e-9 s	130e-9 s
Trig **Trigger_Level**	1 (high)	1 (high)	1 (high)	0 (low)	0 (low)	0 (low)	0 (low)
SSR **Output_Relay**	1 (closed)	1 (closed)	1 (closed)	1 (closed)	1 (closed)	1 (closed)	0 (open)
MeasType **Meas_Type**	0 (none)	1 (spot mean)	0 (none)	1 (spot mean)	0 (none)	0 (none)	0 (none)
MeasStart **Meas_Start**	0 s	25e-9 s	0 s	50e-9 s	0 s	0 s	0 s
MeasStop **Meas_Stop**	0 s	75e-9 s	0 s	100e-9 s	0 s	0 s	0 s

Also see

seg_arb_waveform (on page 6-73)
"Measurement types" in the *Model 4200A-SCS Pulse Card (PGU and PMU) User's Manual*
"Segment Arb waveforms" in the *Model 4200A-SCS Pulse Card (PGU and PMU) User's Manual*

seg_arb_waveform

This command creates a voltage segment waveform.

Usage

```
int seg_arb_waveform(int instr_id, long chan, long NumSeq, long *Seq, double
   *SeqLoopCount);
```

instr_id	The instrument identification code, such as VPU1 or VPU2
chan	Channel number of the pulse card: 1 or 2
NumSeq	Total number of sequences in waveform definition (512 maximum)
Seq	An array of sequences using the sequence number ID
SeqLoopCount	An array of loop values (number of times to output a sequence); loop value range is 1 to 1E12

Pulsers

4220-PGU
4225-PMU

Pulse modes

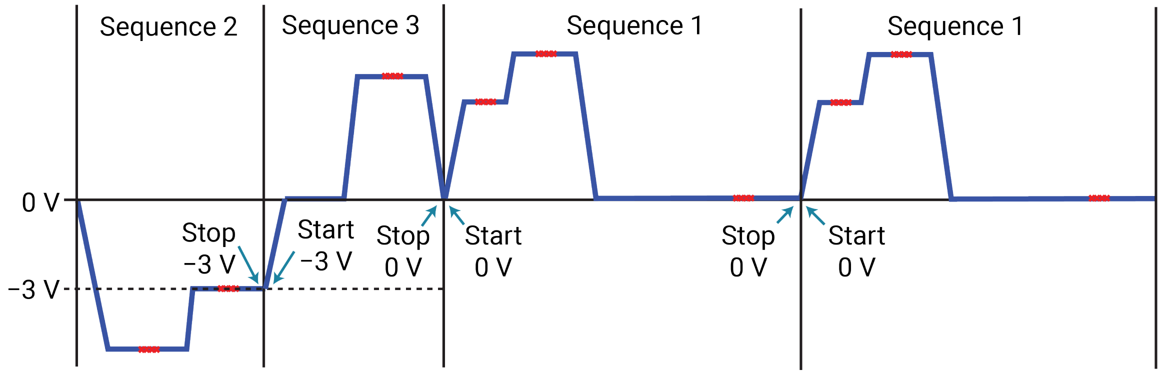
Segment Arb

Details

Use this command to create a voltage segment waveform from the sequences defined by the `seg_arb_sequence` command. The *NumSeq* parameter defines the number of sequences that make up the waveform. The *Seq* parameter is an array that indicates the identification (ID) number for each sequence in the waveform. The sequence ID numbers are set by the `seg_arb_sequence` command.

You can use this command to configure a waveform that repeats one or more of its sequences with the `SeqLoopCount` parameter.

All sequence transitions must be seamless. Seamless means that the voltage level for the last point in a sequence must equal the voltage level on the first point of the next sequence. The figure below shows an example of a three-sequence waveform that uses looping (Sequence 1 is repeated). Notice that the start and stop voltage values between sequences are the same, making it seamless.



Example

`seg_arb_waveform(PMU1, 1, 3, Seq_Num, Seq_Loop_Count);`

This function configures channel 1 of the PMU for a single three-sequence Segment Arb® waveform (as shown in the figure in the **Details**). This example assumes that the three sequences shown in the figure have already been defined by the `seg_arb_sequence` command.
The arrays for the waveform are:
`int Seq_Num[3] = {2, 3, 1};`
`double Seq_Loop_Count[3] = {1, 1, 2};`

Also see

seg_arb_sequence (on page 6-70)

setmode (4225-PMU)

This command sets operating modes specific to the PMU.

Usage

```
int setmode(int instr_id, long modifier, double value);
```

instr_id	The instrument identification code of the pulse generator, such as PMU1, PMU
modifier	Specific operating characteristic to change; see table in **Details**
value	Parameter value for the *modifier*; see table in **Details**

Pulsers

4225-PMU

Pulse mode

Standard

Details

The `setmode` command allows control over the 4225-PMU operating characteristics load-line effect compensation (LLEC) and offset current compensation.

LLEC is an algorithm that runs on each PMU in the test. It adjusts the output of the PMU to respond to the device-under-test (DUT) resistance and reach the programmed output value at the DUT. This algorithm is not guaranteed to reach the programmed target value. Therefore, there are controls to fine-tune the LLEC performance.

When enabled, the LLEC algorithm performs a number of iterations to determine the appropriate output voltage. The `pulse_meas_sm` and `pulse_meas_wfm` commands enable or disable LLEC.

LLEC is configured by setting the number of maximum iterations that will be performed and setting an acceptance window for one or both PMU channels. LLEC continues until either the output voltage to the DUT falls within the acceptance window or until the maximum number of iterations are performed. The LLEC tolerance window is:

LLEC window = LLC_TOLERANCE * Preferred Voltage + LLC_OFFSET

The LLEC is satisfied when:

Measured voltage < Preferred voltage ± LLEC Window

For example, assume the programmed pulse output is 1 V and the acceptance window is set to 0.1 (10%) and offset to 10 mV. LLEC performs iterations until the output voltage falls within the 0.9 V to 1.1 V window. Note that setting a smaller tolerance results in voltage steps that are much closer to the preferred voltage steps sizes, but at the expense of longer test times.

The offset current compensation method is configured by collecting offset current constants from the 4225-PMU and then enabling the constants. Use the `pmu_offset_current_comp` command to collect constants and then enable the constants with the `KI_PMU_Chx_OFFSET_CURR_COMP` parameter.

Parameters		
modifier	***value***	**Comment**
`KI_PXU_LLC_MAX_ITERATIONS`	1 to 1000; 20 to 30 typical	Set the maximum number of LLEC iterations
`KI_PXU_CHx_LLC_TOLERANCE`	0.0001 to 1 (0.01% to 100%); typical range is 0.001 to 0.01 (0.1% to 1%). The typical value is 0.003 (0.3%)	Set the gain of the channel 1 or channel 2 LLEC tolerance window as a percentage of the desired signal level.
`KI_PXU_CHx_LLC_OFFSET`	0 to 1.0	Sets the channel 1 or channel 2 LLEC DC bias offset.
`KI_PMU_CHx_OFFSET_CURR_COMP`	`0` = OFF `1` = ON	Enable or disable constants for channel 1 or channel 2 offset current compensation.

NOTE

When selecting and configuring an LLEC iteration method, remember that testing speed is affected by the maximum number of iterations as well as the tolerance window. Choosing a high maximum number of iterations and a tight tolerance will result in much longer test times.

Example

`setmode(PMU1, KI_PXU_CH1_LLC_TOLERANCE, 0.01);`
This command sets the LLEC for channel 1 of the PMU for a 1% acceptance window.

Also see

"Load-line effect compensation (LLEC) for the PMU" in the *Model 4200A-SCS Pulse Card (PGU and PMU) User's Manual*

pmu_offset_current_comp (on page 6-11)
pulse_meas_sm (on page 6-33)
pulse_meas_wfm (on page 6-36)
setmode (on page 6-74) (SMU)
setmode (on page 5-21) (4210-CVU)

Section 7

LPT commands for switching

In this section:

LPT commands for switching

These LPT commands are used with the Series 700 Switching System, the 4200A-CVIV Multi-Switch, and the 4225-RPM.

addcon

This command adds connections without clearing existing connections.

Usage

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

exist_connect	An 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

`addcon` can be used to make additional connections on a matrix. `addcon` will connect every item in the argument list together, and there is no real distinction between *exist_connect* and the rest of the connection list. `addcon` behaves like the `conpin` command, except previous connections are never cleared.

The value `-1` will be ignored by `addcon` and is considered a valid entry in the connection list. However, *exist_connect* may not be `-1`.

With the row-column connection scheme, only one instrument terminal may be connected to a pin.

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

Also see

clrcon (on page 7-2)
conpin (on page 7-2)
conpth (on page 7-3)
delcon (on page 7-6)

clrcon

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

Usage

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

Details

The `clrcon` command is called automatically by the `devint`, `pulse_output` (only for RPMs), and `execut` commands. 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.

Also see

devclr (on page 4-9)
devint (on page 2-6)
execut (on page 2-8)
pulse_output (on page 6-38)

conpin

This command connects pins and instruments.

Usage

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

InstrTermID	The instrument terminal ID, such as SMU1, GNDU, or PMU1CH1
connect1	A pin number or an instrument terminal ID
connectn	A pin number or an instrument terminal ID

Details

`conpin` connects every item in the argument list together. If no connection rules are violated, the pin or terminal is connected to the additional items, along with everything to which it is already connected.

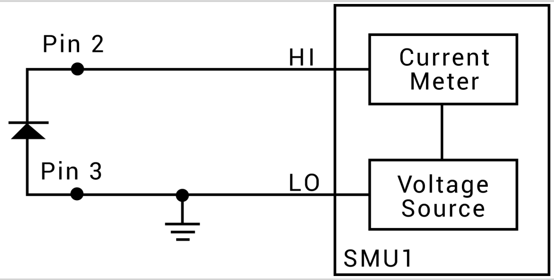
The first `conpin` or `conpth` after any other LPT library call clears all sources by calling `devclr` and then clears all matrix connections by calling `clrcon` before making the new connections.

The value `-1` is ignored by `conpin` and is considered a valid entry in the connection list.

With the row-column connection scheme, only one instrument terminal may be connected to a pin.

Example

```
conpin(3, GND, 0); /* Connect pin 3 to SMU1 */
 /* and ground. */
conpin(2, SMU1, 0); /* Connect pin 2 to SMU1. */
.
.
```



Also see

addcon (on page 7-1)
clrcon (on page 7-2)
conpth (on page 7-3)
delcon (on page 7-6)
devclr (on page 4-9)

conpth

This command connects pins and instruments using a specific pathway.

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

You can force the system to use a particular pathway by using `conpth` instead of `conpin`. This might be done to provide additional electrical isolation between two connections. The eight pathways are numbered 1 through 8.

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.

When the matrix is configured for remote sense, the only valid path values are 1, 3, 5, and 7.

Also see

addcon (on page 7-1)
clrcon (on page 7-2)
conpin (on page 7-2)
delcon (on page 7-6)
devclr (on page 4-9)

cviv_config

This command sends switching commands to the 4200A-CVIV Multi-Switch.

Usage

```
int cviv_config(int instr_id, int channel, int mode);
```

instr_id	The instrument identification code of the 4200A-CVIV: `CVIV1`
channel	4200A-CVIV channel: `1` to `4` 4200A-CVIV all channels: `5`
mode	For channels 1 to 4, the switch settings for the selected channel: ■ Open connection to output terminal: `KI_CVIV_OPEN` or `0` ■ Connect channel to SMU (4200-SMU, 4201-SMU, 4210-SMU, or 4211-SMU): `KI_CVIV_SMU` or `1` ■ Connect channel to CVU HI (4210-CVU or 4215-CVU: `KI_CVIV_CVH` or `2` ■ Connect channel to CVU LO (4210-CVU or 4215-CVU): `KI_CVIV_CVL` or `3` ■ Connect CV guard to the output connector shell with ac ground to center: `KI_CVIV_CV_GRD` or `4` ■ Connect channel to ground unit: `KI_CVIV_GNDU` or `5` ■ Connect channel to ac-coupled ac ground: `KI_CVIV_AC_COUPLED_AC_GND` or `6` ■ Connect channel to bias tee SMU CV HI: `KI_CVIV_BT_CVH` or `7` ■ Connect channel to bias tee SMU CV LO: `KI_CVIV_BT_CVL` or `8` ■ Connect channel to bias tee low current SMU CV HI: `KI_CVIV_BT_LOI_CVH` or `9` ■ Connect channel to bias tee low current SMU CV LO: `KI_CVIV_BT_LOI_CVL` or `10` ■ Connect channel to bias tee ac ground: `KI_CVIV_BT_AC_GND` or `11` If *channel* is set to 5 (all channels), the switch settings for the 4200A-CVIV instrument are: ■ All CV channels to C-V 2-wire: `KI_CVIV_CVU_2WIRE` or `1` ■ All CV channels to C-V 4-wire: `KI_CVIV_CVU_4WIRE` or `0`

Details

The 4200A-CVIV includes input connections for four SMU cards and one CVU card. Use this command to control switching inside the 4200A-CVIV to connect the SMU and CVU instruments to the output terminals.

The 4200A-CVIV connections are cleared by the `clrcon` command.

Example

```
cviv_config(CVIV1, 1, KI_CVIV_SMU);
```

This command connects channel 1 of the CVIV to a SMU.

Also see

clrcon (on page 7-2)
cviv_display_config (on page 7-5)
cviv_display_power (on page 7-6)

cviv_display_config

This command configures the LCD display on the 4200A-CVIV Multi-Switch.

Usage

```
int cviv_display_config(int instr_id, int channel, int identifier, char *value);
```

instr_id	The instrument identification code of the 4200A-CVIV: `CVIV1`
channel	4200A-CVIV channel (use to set a terminal name): `1` to `4` 4200A-CVIV virtual channel (use to set the test name): `5` See **Details**
identifier	Display the name of the terminal: `KI_CVIV_TERMINAL_NAME` or `1` Display the name of the test: `KI_CVIV_TEST_NAME` or `0` See **Details**
value	A string that defines the name (up to 16 characters for a test name or 6 characters for a terminal name)

Details

Sets the name for the channel terminal or test that is displayed on the 4200A-CVIV for the selected channel.

The *`channel`* and *`identifier`* settings must be set for either terminal or test name. For example, if *`channel`* is set to 2, *`identifier`* must be set to `KI_CVIV_TERMINAL_NAME`.

If the `clrcon` command is sent, the 4200A-CVIV display is updated to show the change in connections. If the 4200A-CVIV display is turned off, it remains off after a `clrcon`.

Example

```
cviv_display_config(CVIV1, 2, KI_CVIV_TERMINAL_NAME, "Source");
```

This command sets the name of the channel 2 terminal on the 4200A-CVIV display.

Also see

clrcon (on page 7-2)
cviv_config (on page 7-4)
cviv_display_power (on page 7-6)

cviv_display_power

This command sets the display state of the LCD display on the 4200A-CVIV.

Usage

```
int cviv_display_power(int instr_id, int state);
```

instr_id	The instrument identification code of the 4200A-CVIV: `CVIV1`
state	Display on: `KI_CVIV_DISPLAY_ON` or `1` Display off: `KI_CVIV_DISPLAY_OFF` or `0`

Details

This command turns the display of the 4200A-CVIV on or off.

When the display is turned off, the 4200A-CVIV clears the displays. A small green circle is displayed to indicate that the 4200A-CVIV instrument is powered.

When the display is turned on, the latest configuration is displayed.

If the `clrcon` command is sent, the 4200A-CVIV display is updated to show the change in connections. If the 4200A-CVIV display is turned off, it remains off after a `clrcon`.

Example

`cviv_display_power(CVIV1, KI_CVIV_DISPLAY_OFF);`
Turns off the 4200A-CVIV display.

Also see

cviv_config (on page 7-4)
cviv_display_config (on page 7-5)

delcon

This command removes specific matrix connections.

Usage

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

InstTermID	The instrument terminal ID, such as SMU1, GNDU, or PMU1CH1
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.

If a SMU remains connected, GND must be reconnected using `addcon` or an error is generated when the first LPT library command after the connection sequence executes.

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.

Example

```
double i1, i2;
conpin(3, GND, 0);
conpin(1, SMU1, 0);
conpin(2, SMU2, 0);
forcev(SMU1, 1.0);
forcei(SMU2, 0.001);
measi(SMU1, &i1);
delcon(SMU2, 0); /* Remove SMU2 from the circuit */
forcev(SMU1, 1.0); /* because delcon cleared sources. */
measi(SMU1, &i2);
```

Also see

addcon (on page 7-1)
clrcon (on page 7-2)
conpin (on page 7-2)
conpth (on page 7-3)
devclr (on page 4-9)

devint

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

Usage

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

Details

Resets all active instruments, including the 4200A-CVIV, 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.

The `devint` command is implicitly called by the `execut` and `tstdsl` commands.

To abort a running `pulse_exec` pulse test, see `dev_abort`.

`devint` does the following:

1. Clears all sources by calling `devclr`.
2. Clears the matrix crosspoints by calling `clrcon`.
3. Clears the trigger tables by calling `clrtrg`.
4. Clears the sweep tables by calling `clrscn`.
5. Resets GPIB instruments by sending the string defined with `kibdefint`.
6. Resets the active instrument cards.

Instrument cards are reset in the following order:

1. SMU instrument cards
2. CVU instrument cards
3. Pulse instrument cards (4225-PMU or 4220-PGU)

The SMUs return to the following states:

- 100 µA and 10 V ranges
- Autorange on
- Voltage source
- 0 V dc bias

The 4210-CVU or 4215-CVU returns to the following states:

- 30 mV_{RMS} ac signal
- 0 V dc bias
- 100 kHz frequency
- Autorange on
- Cable length compensation set to 0 m
- Open/Short/Load compensation disabled

The 4225-PMU or 4220-PGU returns to the following states:

- The pulse mode is maintained. For example, if the pulse card is in Segment Arb mode, it is still in Segment Arb mode after the `devint` process is complete.
- 5 V and 10 mA ranges
- If in pulse mode:
  - Period of 1 µs
  - Transition times (rise and fall) of 100 ns
  - Width of 500 ns
  - Voltage high and low of 0 V
  - Load of 50 Ω
- If in segmented arb mode, Start Voltage is 0 V
- If in arbitrary waveform mode, Table Length is 100

Also see

clrcon (on page 7-2)
clrscn (on page 2-2)
clrtrg (on page 2-3)
dev_abort (on page 6-4)
devclr (on page 4-9)
kibdefint (on page 2-16)

Keithley Instruments • 28775 Aurora Road • Cleveland, Ohio 44139 • 1-800-833-9200 • tek.com/keithley

KEITHLEY
A Tektronix Company

04/2022