# SDG Series
# Arbitrary Waveform Generator

Programming Guide

PG02_E05B

SIGLENT TECHNOLOGIES CO.,LTD

# Content

# 1 Programming Overview

By using USB and LAN interfaces, in combination with NI-VISA and programming languages, users can remotely control the waveform generator. Through the LAN interface, VXI-11, Sockets, and Telnet protocols can be used to communicate with the instruments. This chapter introduces how to build communication between the instrument and the PC. It also introduces how to configure a system for remote instrument control.

## 1.1 Build communication via VISA

### 1.1.1 Install NI-VISA

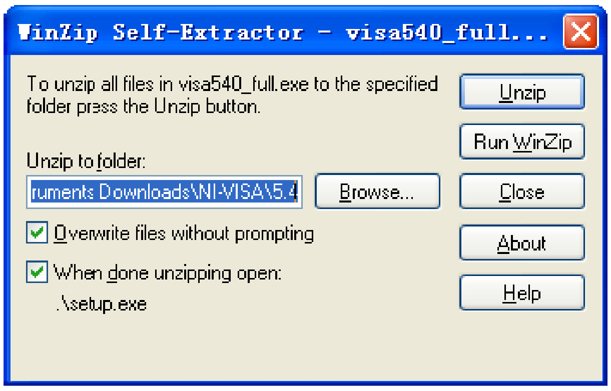
Before programming, please make sure that you have properly installed the latest version of National Instruments NI-VISA Software.

NI-VISA is a communication library that enables computer communications to instrumentation. There are two available VISA packages: A full version and the Run-Time Engine. The full version includes NI device drivers and a tool named NI MAX; a user interface to control the device. While the drivers and NI MAX can be useful, they are not required for remote control. The Run-Time Engine is a much smaller file and is recommanded for remote control.

For convenience, you can obtain the latest version of the NI-VISA run-time engine or the full version from the National Instruments website. The installation process is similar for both versions.

Follow these steps to install NI-VISA (The full version of NI-VISA 5.4 is used in this example):

a. Download the appropriate version of NI-VISA (the Run-time engine is recommanded)
b. Double click the visa540_full.exe and observe the dialog box as shown below:



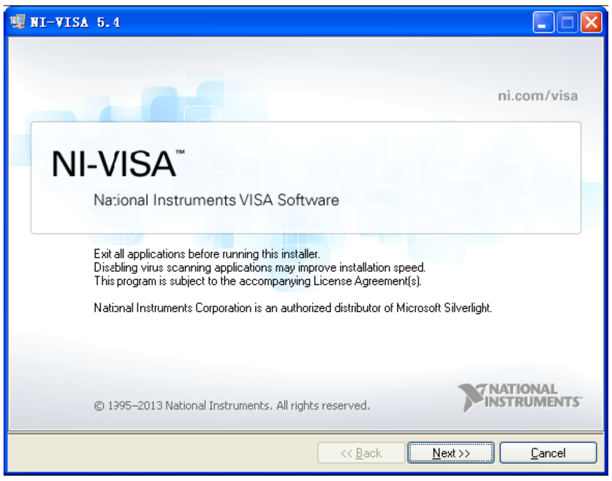
To unzip all files in visa540_full.exe to the specified folder press the Unzip button.

Unzip to folder: ruments Downloads\NI-VISA\5.4

☑ Overwrite files without prompting

☑ When done unzipping open: .\setup.exe

c. Click Unzip, the install process will launch after unzipping files. If your computer needs to install the .NET Framework 4, it may auto-start.

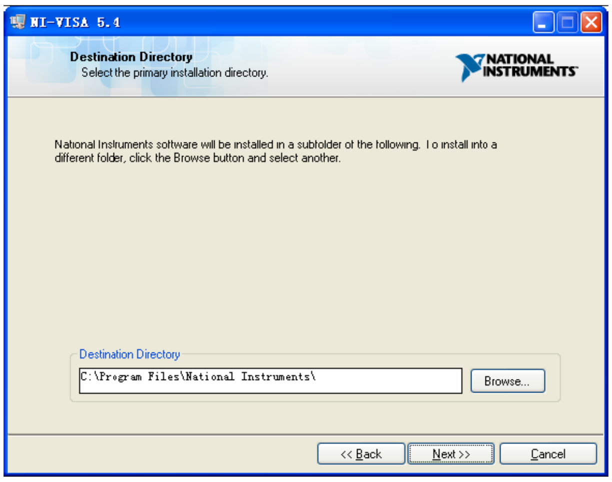


Exit all applications before running this installer.
Disabling virus scanning applications may improve installation speed.
This program is subject to the accompanying License Agreement(s).

National Instruments Corporation is an authorized distributor of Microsoft Silverlight.

© 1995–2013 National Instruments. All rights reserved.

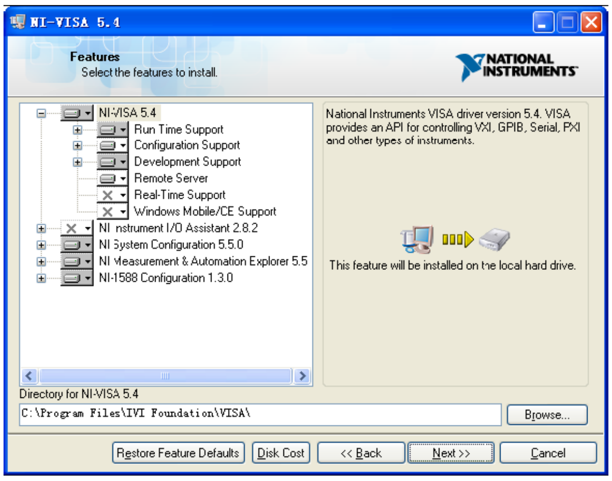
d. The NI-VISA install dialog is shown above. Click Next to start the installation process.



National Instruments software will be installed in a subfolder of the following. To install into a different folder, click the Browse button and select another.

Destination Directory: C:\Program Files\National Instruments\

e. Set the install path, the default path is “C:\Program Files\National Instruments\”, you can change it if you prefer. Click Next, dialog as shown above.



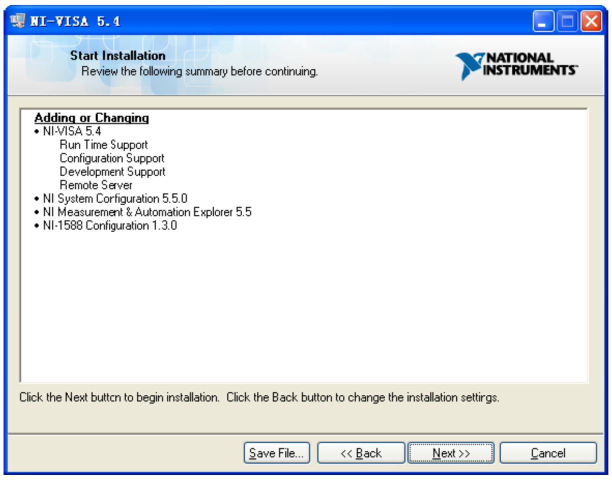
National Instruments VISA driver version 5.4. VISA provides an API for controlling VXI, GPIB, Serial, PXI and other types of instruments.

This feature will be installed on the local hard drive.

Directory for NI-VISA 5.4

C:\Program Files\IVI Foundation\VISA\

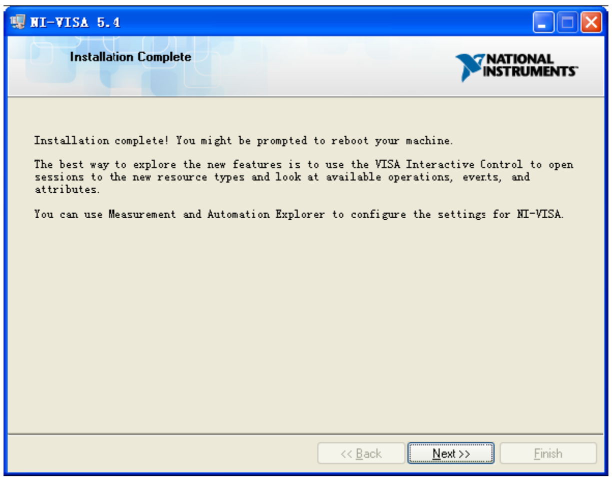
f. Click Next twice, in the License Agreement dialog, select the “I accept the above 2 License Agreement(s).”, and click Next, and a dialog box will appear as shown below:



**Adding or Changing**
- NI-VISA 5.4
  - Run Time Support
  - Configuration Support
  - Development Support
  - Remote Server
- NI System Configuration 5.5.0
- NI Measurement & Automation Explorer 5.5
- NI-1588 Configuration 1.3.0

Click the Next button to begin installation. Click the Back button to change the installation settings.

g. Click Next to begin installation.



Installation complete! You might be prompted to reboot your machine.

The best way to explore the new features is to use the VISA Interactive Control to open sessions to the new resource types and look at available operations, events, and attributes.

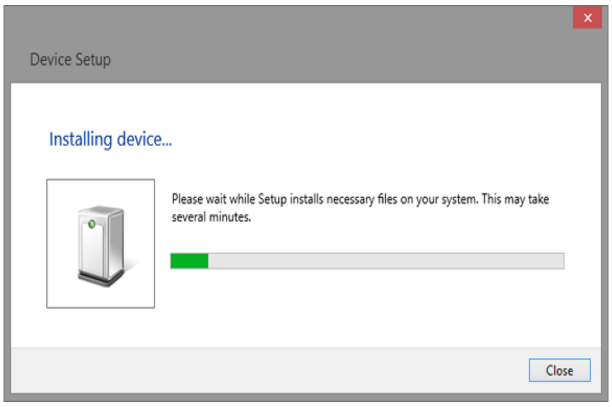
You can use Measurement and Automation Explorer to configure the settings for NI-VISA.

h. Now the installation is complete. Reboot your PC.

### 1.1.2 Connect the instrument

Depending on the specific model, the arbitrary waveform generator may be able to communicate with a PC through the USB or LAN interface.

Connect the arbitrary waveform generator and the USB Host interface of the PC using a USB cable. Assuming your PC is already turned on, turn on the SDG, and then the PC will display the “Device Setup” screen as it automatically installs the device driver as shown below.



Please wait while Setup installs necessary files on your system. This may take several minutes.

Wait for the installation to complete and then proceed to the next step.

## 1.2 Remote Control

### 1.2.1 User-defined Programming

Users can send SCPI commands via a computer to program and control the arbitrary waveform generator. For details, refer to the introductions in "Programming Examples".

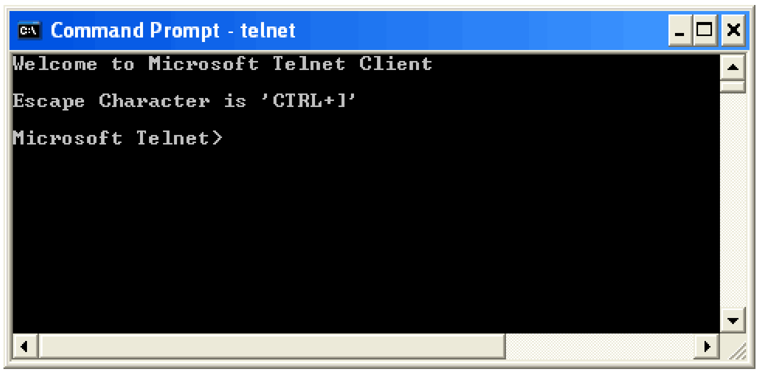
### 1.2.2 Using SCPI via NI-MAX

NI-MAX is a program created and maintained by National Instruments. It provides a basic remote control interface for VXI, LAN, USB, GPIB, and Serial communications. The SDG can be controlled remotely by sending SCPI commands via NI-MAX.

### 1.2.3 Using SCPI over Telnet

Telnet provides a means of communicating with the SDG over the LAN. The Telnet protocol sends SCPI commands to the SDG from a PC and is similar to communicating with the SDG over USB. It sends and receives information interactively: one command at a time. The Windows operating systems use a command prompt style interface for the Telnet client. The steps are as follows:

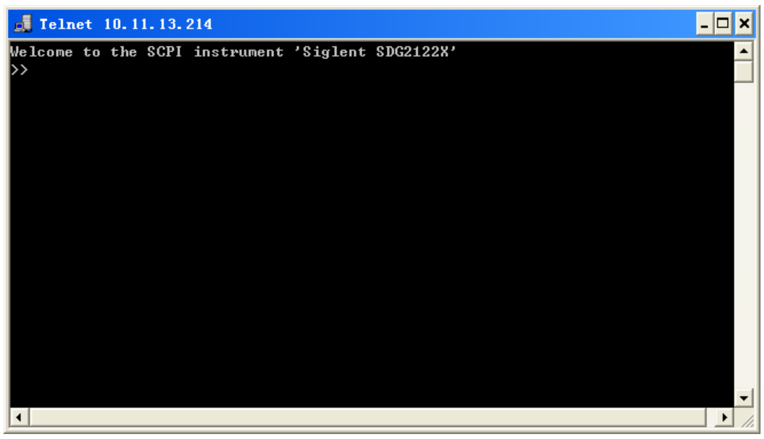
1. On your PC, click Start > All Programs > Accessories > Command Prompt.
2. At the command prompt, type in *telnet*.
3. Press the Enter key. The Telnet display screen will be displayed.



```
Welcome to Microsoft Telnet Client
Escape Character is 'CTRL+]'
Microsoft Telnet>
```

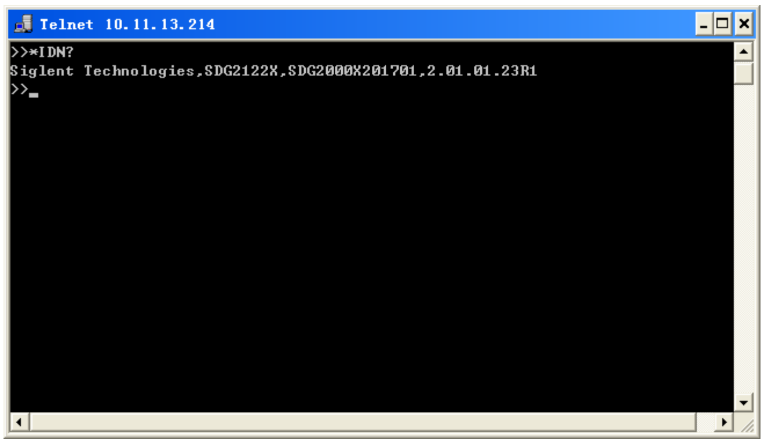
4. At the Telnet command line, type:open XXX.XXX.XXX.XXX 5024

   Where *XXX.XXX.XXX.XXX* is the instrument's IP address and 5024 is the port. You should see a response similar to the following:



```
Welcome to the SCPI instrument 'Siglent SDG2122X'
>>
```

5. At the SCPI> prompt, input the SCPI commands such as **IDN?* to return the company name, model number, serial number, and firmware version number.



```
>>*IDN?
Siglent Technologies,SDG2122X,SDG2000X201701,2.01.01.23R1
>>
```

6. To exit the SCPI> session, press the Ctrl+] keys simultaneously.
7. Type *quit* at the prompt or close the Telnet window to close the connection to the instrument and exit Telnet.

### 1.2.4 Using SCPI over Socket

Socket API can be used to control the SDG series by LAN without installing any other libraries. This can reduce the complexity of programming.

| | |
|---|---|
| **SOCKET ADDRESS** | IP address + port number |
| **IP ADDRESS** | SDG IP address |
| **PORT NUMBER** | 5025 |

Please see section 5.2 "Examples of Using Sockets" for the details.

# 2 Introduction to the SCPI Language

## 2.1 About Commands & Queries

This section lists and describes the remote control commands and queries recognized by the instrument. All commands and queries can be executed in either the local or remote state.

Each command or query, with syntax and other information, has some examples listed. The commands are given in both long and short format at "**COMMAND SYNTAX**" and "**QUERY SYNTAX**", and the subject is indicated as a command or query or both. Queries perform actions such as obtaining information from the instrument and are identified by a question mark (?) following the header.

## 2.2 Description

In the description, a brief explanation of the function performed is given. This is followed by a presentation of the formal syntax, with the header given in Upper-and-Lower-Case characters and the short form derived from it in ALL UPPER-CASE characters. Where applicable, the syntax of the query is given with the format of its response.

## 2.3 Usage

The commands and queries listed here can be used for SDGxxxx Series Arbitrary Waveform Generators.

## 2.4 Command Notation

The following notations are used in the commands:

< > Angular brackets enclose words that are used as placeholders, of which there are two types:

the header path and the data parameter of a command.

:= A colon followed by an equals sign separates a placeholder, from the description of the type and range of values that may be used in a command instead of the placeholder.

{ } Braces enclose a list of choices, one of which must be made.

[ ] Square brackets enclose optional items.

… ] Ellipsis (trailing dots) indicate that the preceding element may be repeated one or more times.

## 2.5 Table of Command & Queries

| Short | Long Form | Subsystem | What Command/Query does |
|---|---|---|---|
| *IDN | *IDN | SYSTEM | Gets identification from device. |
| *OPC | *OPC | SYSTEM | Gets or sets the OPC bit (0) in the Event Status Register (ESR). |
| *RST | *RST | SYSTEM | Restore default settings |
| CHDR | COMM_HEADER | SIGNAL | Sets or gets the command returned format |
| OUTP | OUTPUT | SIGNAL | Sets or gets the output state. |
| BSWV | BASIC_WAVE | SIGNAL | Sets or gets the basic wave parameters. |
| MDWV | MODULATEWAVE | SIGNAL | Sets or gets the modulation parameters. |
| SWWV | SWEEPWAVE | SIGNAL | Sets or gets the sweep parameters. |
| BTWV | BURSTWAVE | SIGNAL | Sets or gets the burst parameters. |
| PACP | PARACOPY | SIGNAL | Copies parameters from one channel to the other. |
| ARWV | ARBWAVE | DATA | Changes arbitrary wave type. |
| SYNC | SYNC | SIGNAL | Sets or gets the synchronization signal. |
| NBFM | NUMBER_FORMAT | SYSTEM | Sets or gets the data format. |
| LAGG | LANGUAGE | SYSTEM | Sets or gets the language. |

| Short | Long Form | Subsystem | What Command/Query does |
|---|---|---|---|
| SCFG | SYS_CFG | SYSTEM | Sets or gets the power-on system setting way. |
| BUZZ | BUZZER | SYSTEM | Sets or gets the buzzer state. |
| SCSV | SCREEN_SAVE | SYSTEM | Sets or gets the screen save state. |
| ROSC | ROSCILLATOR | SIGNAL | Sets or gets the state of the clock source. |
| FCNT | FREQCOUNTER | SIGNAL | Sets or gets the frequency counter parameters. |
| INVT | INVERT | SIGNAL | Sets or gets the polarity of the current channel. |
| COUP | COUPLING | SIGNAL | Sets or gets the coupling parameters. |
| VOLTPRT | VOLTPRT | SYSTEM | Sets or gets the state of over-voltage protection. |
| STL | STORELIST | SIGNAL | Lists all stored waveforms. |
| WVDT | WVDT | SIGNAL | Sets and gets the arbitrary wave data. |
| VKEY | VIRTUALKEY | SYSTEM | Sets the virtual keys. |
| SYST:COMM:LAN:IPAD | SYSTEM:COMMUNICATE:LAN:IPADDRESS | SYSTEM | The Command can set and get the system IP address. |
| SYST:COMM:LAN:SMAS | SYSTEM:COMMUNICATE:LAN:SMASK | SYSTEM | The Command can set and get the system subnet mask. |
| SYST:COMM:LAN:GAT | SYSTEM:COMMUNICATE:LAN:GATEWAY | SYSTEM | The Command can set and get the system Gateway. |
| SRATE | SAMPLERATE | SIGNAL | Sets or gets the arbitrary wave mode, sampling rate, and interpolation method. |
| HARM | HARMonic | SIGNAL | Sets or gets the harmonic information. |
| CMBN | CoMBiNe | SIGNAL | Sets or gets the wave combine information. |
| MODE | MODE | SIGNAL | Sets or gets the waveform phase mode |
| CASCADE | CASCADE | SYSTEM | Set up multi-device synchronization |

| Short | Long Form | Subsystem | What Command/Query does |
|---|---|---|---|
| IQ:CENT | IQ:CENTerfreq | SIGNAL | Sets the I/Q modulator center frequency. |
| IQ:SAMP | IQ:SAMPlerate | SIGNAL | Sets the I/Q sample rate. |
| IQ:SYMB | IQ:SYMBolrate | SIGNAL | Sets the I/Q symbol rate. |
| IQ:AMPL | IQ:AMPLitude | SIGNAL | Sets the I/Q amplitude. |
| IQ:IQAD:GAIN | IQ:IQADjustment:GAIN | SIGNAL | Adjusts the ratio of I to Q while preserving the composite. |
| IQ:IQAD:IOFF | IQ:IQADjustment:IOFFset | SIGNAL | Adjusts the I channel offset value. |
| IQ:IQAD:QOFF | IQ:IQADjustment:QOFFset | SIGNAL | Adjusts the I channel offset value. |
| IQ:IQAD:QSK | IQ:IQADjustment:QSKew | SIGNAL | Adjusts the phase angle between the I and Q vectors by increasing or decreasing the Q phase angle. |
| IQ:TRIG:SOUR | IQ:TRIGger:SOURce | SIGNAL | Sets the I/Q trigger source. |
| IQ:WAVE:BUIL | IQ:WAVEload:BUILtin | SIGNAL | Selects the I/Q wave from the built-in wave list. |
| IQ:WAVE:USER | IQ:WAVEload:USERstored | SIGNAL | Select the I/Q wave from user stored waveforms. |
| :IQ: FrequencySampling | :IQ: FrequencySampling | SIGNAL | Sets the I/Q Frequency sampling rate. |

# 3 Commands and Queries

## 3.1 IEEE 488.2 Common Command Introduction

The IEEE standard defines the common commands used for querying the basic information of the instrument or executing basic operations. These commands usually start with "*" and the length of the keywords of the command is usually 3 characters.

### 3.1.1 *IDN

**DESCRIPTION**

The *IDN? query causes the instrument to identify itself. The response is comprised of the manufacturer, model, serial number, and firmware version.

**QUERY SYNTAX**

*IDN?

**RESPONSE FORMAT**

Format 1: *IDN, <device id>,<model>,<serial number>,<firmware version>, <hardware version>

Format 2: <manufacturer>,<model>,<serial number>,<firmware version>

<device id>:= “SDG”.

<manufacturer>:= “Siglent Technologies”.

<model>:= A model identifier less than 14 characters, should not contain the word “MODEL”.

<serial number>:= The serial number.

<firmware version>:= The firmware version number.

<hardware version>:= The hardware level field, containing information about all separately revisable subsystems.

**EXAMPLE**

Reads version information:

**IDN?*

Return:

*Siglent Technologies,SDG6052X, SDG6XBAX1R0034, 6.01.01.28* (It may differ from each version)

Notes:

1. The table below shows the available response format of the command in each SDG series.

| **Parameter /command** | **SDG800** | **SDG1000** | **SDG2000X** | **SDG5000** | **SDG1000X** | **SDG6000X/X-E** | **SDG7000A** |
|---|---|---|---|---|---|---|---|
| Response Format | Format1 | Format1 | Format2 | Format1 | Format2 | Format2 | Format2 |

2. Format of <hardware version>: value1-value2-value3-value4-value5.

   value1: PCB version.

   value2: Hardware version.

   value3: Hardware subversion.

   value4: FPGA version.

   value5: CPLD version.

## 3.1.2 *OPC

**DESCRIPTION**

The *OPC (Operation Complete) command sets the OPC bit (bit 0) in the standard Event Status Register (ESR). This command has no other effect on the operation of the device because the instrument starts parsing a command or query only after it has completely processed the previous command or query. The *OPC? query always responds with the ASCII character 1 because the device only responds to the query when the previous command has been entirely executed.

**COMMAND SYNTAX** *OPC

**QUERY SYNTAX** *OPC?

**RESPONSE FORMAT**

Format 1: *OPC 1

Format 2: 1

Note: The table below shows the available response format of the command in each SDG series.

| **Parameter /command** | **SDG800** | **SDG1000** | **SDG2000X** | **SDG5000** | **SDG1000X** | **SDG6000X/X-E** | **SDG7000A** |
|---|---|---|---|---|---|---|---|
| Response Format | Format1 | Format1 | Format2 | Format1 | Format2 | Format2 | Format2 |

### 3.1.3 *RST

**DESCRIPTION** The *RST command initiates a device reset and recalls the default setup.

**COMMAND SYNTAX** *RST

**EXAMPLE** This example resets the signal generator to the default setup:

**RST*

## 3.2 Comm_Header Command

**DESCRIPTION** This command is used to change the query command returned format. “SHORT” parameter returns short format. “LONG” parameter returns long format. The “OFF” parameter returns nothing.

**COMMAND SYNTAX** Comm_HeaDeR <parameter>

<parameter>:= {SHORT,LONG,OFF}.

**QUERY SYNTAX** Comm_HeaDeR?

**RESPONSE FORMAT** CHDR <parameter>

**EXAMPLE** Set query command format to long:

*CHDR LONG*

Read query command format:

*CHDR?*

Return:

*COMM_HEADER LONG*

Note: The table below shows the availability of the command in each SDG series.

| Parameter /command | SDG800 | SDG1000 | SDG2000X | SDG5000 | SDG1000X | SDG6000X/X-E | SDG7000A |
|---|---|---|---|---|---|---|---|
| CHDR | yes | yes | no | yes | no | no | no |

## 3.3 Output Command

**DESCRIPTION**

This command enables or disables the output port(s) at the front panel. The query returns “ON” or “OFF” and “LOAD”, “PLRT”, and “RATIO” parameters.

**COMMAND SYNTAX**

<channel>:OUTPut ON|OFF,LOAD,<load>,PLRT, <polarity>

<channel>:= {C1, C2}.

<load>:= {see the note below}. The unit is ohms.

<polarity>:= {NOR, INVT}, in which NOR refers to normal, and INVT refers to invert.

NoiseSum: to add noise to the channel with specified signal-to-noise ratio.

<channel>:NOISE_ADD STATE,ON|OFF,RATIO,<S/N>.

or

<channel>:NOISE_ADD STATE,ON|OFF,RATIO_DB,<S/N(dB)>.

<S/N>:= {2.1-100000000}.

<S/N (dB)>:= {3.24886-80}.

Max Amplitude Output: to limit the maximum amplitude output.

<channel>:BSWV MAX_OUTPUT_AMP, <Amplitude>

<Amplitude>:={1-20}, Maximum output amplitude peak-to-peak voltages.

**QUERY SYNTAX**

<channel>:OUTPut?

<channel>:NOISE_ADD?

<channel>:BSWV?

**RESPONSE FORMAT**

<channel>:OUTP ON|OFF,LOAD,<load>,PLRT, <polarity>

<channel>:NOISE_ADDSTATE,ON|OFF,RATIO,<S/N >, RATIO_DB,<S/N (dB)>

<channel>:BSWV WVTP,<type>,FRQ,<frequency>,PERI,<period>,AMP,<amplitude>,AMPVRMS,<Amplitude>,MAX_OUTPUT_AMP, OFST, <offset>, HLEV, <high level>, LLEV, <low level>, PHSE, <phase>

**EXAMPLE**

Turn on CH1:

*C1:OUTP ON*

Read CH1 output state:

*C1:OUTP?*

Return:

*C1:OUTP ON,LOAD,HZ,PLRT,NOR*

Set the load of CH1 to 50 ohms:

*C1:OUTP LOAD,50*

Set the load of CH1 to HiZ:

*C1:OUTP LOAD,HZ*

Set the polarity of CH1 to normal:

*C1:OUTP PLRT,NOR*

turn on NoiseSum and set the signal-to-noise ratio

*C1:NOISE_ADD STATE,ON,RATIO,120*

Set the maximum output amplitude

*C1:BSWV MAX_OUTPUT_AMP,5*

Note: The table below shows the availability of the command in each SDG series.

| Parameter /command | SDG800 | SDG1000 | SDG2000X | SDG5000 | SDG1000X | SDG6000X/X-E | SDG7000A |
|---|---|---|---|---|---|---|---|
| <channel> | no | yes | yes | yes | yes | yes | yes |
| LOAD | 50,HiZ | 50~10000 , HiZ | 50~100000, HiZ | 50, HiZ | 50~100000, HiZ | 50~100000, HiZ | 50~100000, HiZ |

* "HiZ" refers to High Z.

**DESCRIPTION** Set two channels to open or close output at the same time

**COMMAND SYNTAX** OUT_BOTHCH <STATE>
<STATE>={ON,OFF}

**QUERY SYNTAX**

**RESPONSE FORMAT**

**EXAMPLE** Open two channels of output:
*OUT_BOTHCH ON*

Note: The table below shows the availability of the command in each SDG series.

| Parameter /command | SDG800 | SDG1000 | SDG2000X | SDG5000 | SDG1000X | SDG6000X/X-E | SDG7000A |
|---|---|---|---|---|---|---|---|
| | no | yes | yes | yes | yes | yes | yes |

## 3.4 Basic Wave Command

**DESCRIPTION** This command sets or gets the basic wave parameters.

**COMMAND SYNTAX** <channel>:BaSic_WaVe <parameter>,<value>
<channel>:={C1, C2}.

<parameter>:= {a parameter from the table below}.

<value>:={value of the corresponding parameter}.

| Parameters | Value | Description |
|---|---|---|
| WVTP | <type> | := {SINE, SQUARE, RAMP, PULSE, NOISE, ARB, DC, PRBS, IQ}. If the command doesn’t set basic waveform type, WVPT will be set to the current waveform. |
| FRQ | <frequency> | := frequency. The unit is Hertz “Hz”. Refer to the datasheet for the range of valid values. Not valid when WVTP is NOISE or DC. |
| PERI | <period> | := period. The unit is seconds "s". Refer to the datasheet for the range of valid values. Not valid when WVTP is NOISE or DC. |

| | | |
|---|---|---|
| AMP | <amplitude> | := amplitude. The unit is volts, peak-to-peak "Vpp". Refer to the datasheet for the range of valid values. Not valid when WVTP is NOISE or DC. |
| AMPVRMS | <amplitude> | := amplitude. The unit is Volts, root-mean-square “Vrms”. |
| AMPDBM | <amplitude> | := amplitude. The unit is dBm. |
| OFST | <offset> | := offset. The unit is volts "V". Refer to the datasheet for the range of valid values. Not valid when WVTP is NOISE. |
| COM_OFST | <common offset> | :={-1 to 1}.common offset. The unit is volts "V". It can be set only when the channel differential output is on. |
| SYM | <symmetry> | :={0 to 100}. Symmetry of RAMP. The unit is "%". Only settable when WVTP is RAMP. |
| DUTY | <duty> | := {0 to 100}. Duty cycle. The unit is "%". Value depends on frequency. Only settable when WVTP is SQUARE or PULSE. |
| PHSE | <phase> | := {0 to 360}. The unit is "degree". Not valid when WVTP is NOISE, PULSE or DC. |
| STDEV | <stdev> | := standard deviation of NOISE. The unit is volts "V". Refer to the datasheet for the range of valid values. Only settable when WVTP is NOISE. |
| MEAN | <mean> | := mean of NOISE. The unit is volts "V". Refer to the datasheet for the range of valid values. Only settable when WVTP is NOISE. |
| WIDTH | <width> | := positive pulse width. The unit is seconds "s". Refer to the datasheet for the range of valid values. Only settable when WVTP is PULSE. |
| RISE | <rise> | := rise time (10%~90%). The unit is seconds "s". Refer to the datasheet for the range of valid values. Only settable when WVTP is PULSE. |
| FALL | <fall> | := fall time (90%~10%). The unit is seconds "s". Refer to the datasheet for the range of valid values. Only settable when WVTP is PULSE. |
| DLY | <delay> | := waveform delay. The unit is seconds "s". Refer to the datasheet for the range of valid values. |
| HLEV | <high level> | := high level. The unit is volts "V". Not valid when WVTP is NOISE or DC. |
| LLEV | <low level> | := low level. The unit is volts "V". Not valid when WVTP is NOISE or DC. |
| BANDSTATE | <bandwidth switch > | := {ON,OFF}. Only settable when WVTP is NOISE. |

| | | |
|---|---|---|
| BANDWIDTH | <bandwidth value> | := noise bandwidth. The unit is Hertz "Hz". Refer to the datasheet for the range of valid values. Only settable when WVTP is NOISE. |
| LENGTH | <prbs length> | :={3~32}. Actual PRBS length = $2^{LENGTH}$-1. Only settable when WVTP is PRBS. |
| EDGE | <prbs rise/fall> | := rise/fall time of PRBS. The unit is seconds "s". Refer to the datasheet for the range of valid values. Only settable when WVTP is PRBS. |
| FORMAT | <output format> | :={DIFF, SINGLE }. Set channel differential output or single ended output |
| DIFFSTATE | <prbs differential switch> | :={ON, OFF}. State of PRBS differential mode. Only settable when WVTP is PRBS. |
| BITRATE | <prbs bit rate> | := PRBS bit rate. The unit is bits-per-second "bps". Refer to the datasheet for the range of valid values. Only settable when WVTP is PRBS. |
| LOGICLEVEL | <prbs logiclevel rate> | :={ TTL_CMOS, LVTTL_LVCMOS, ECL, LVPECL, LVDS CUSTOM (only on **SDG7000A**) }. Only settable when WVTP is PRBS. |

**QUERY SYNTAX**

<channel>: BaSic_WaVe?
<channel>:= {C1, C2}.

**RESPONSE FORMAT**

<channel>:BSWV <parameter>
<parameter>:= {All the parameters of the current basic waveform}.

**EXAMPLE**

Change the waveform type of C1 to Ramp:
*C1:BSWV WVTP,RAMP*

Change the frequency of C1 to 2000 Hz:
*C1:BSWV FRQ,2000*

Set the amplitude of C1 to 3 Vpp:
*C1:BSWV AMP,3*

Return parameters of C1 from the device:
*C1:BSWV?*
Return:
*C1:BSWV WVTP,SINE,FRQ,100HZ,PERI,0.01S,AMP,2V,*

*OFST,0V,HLEV,1V,LLEV,-1V,PHSE,0*

Set noise bandwidth of C1 to 100 MHz:
*C1:BSWV BANDWIDTH,100E6*
or
*C1:BSWV BANDWIDTH,100000000*

Set output amplitude of C1 to 3dBm:
*C1:BSWV AMPDBM,3*

Set the logic level of C1 to TTL_CMOS:
*C1:BSWV LOGICLEVEL,TTL_CMOS*

Notes:

1. The table below shows the availability of some command parameters in each SDG series.

| **Parameter /command** | **SDG 800** | **SDG 1000** | **SDG 2000X** | **SDG 5000** | **SDG 1000X** | **SDG 6000X** | **SDG 6000X-E** | **SDG 7000A** |
|---|---|---|---|---|---|---|---|---|
| <channel> | no | yes | yes | yes | yes | yes | yes | yes |
| RISE | yes | no | yes | yes | yes | yes | yes | yes |
| FALL | yes | no | yes | yes | yes | yes | yes | yes |
| DLY | no | yes | yes | yes | yes | yes | yes | yes |
| BANDSTATE | no | no | yes | no | no | yes | yes | yes |
| BANDWIDTH | no | no | yes | no | no | yes | yes | yes |
| LENGTH | no | no | no | no | no | yes | no | yes |
| EDGE | no | no | no | no | no | yes | no | yes |
| DIFFSTATE | no | no | no | no | no | yes | no | no |
| BITRATE | no | no | no | no | no | yes | no | yes |
| LOGICLEVEL | no | no | no | no | no | yes | no | yes |
| AMPDBM | no | no | yes | no | yes | yes | yes | yes |

2. With SDG1000X models, if Wave Combine is enabled, WVTP cannot be set to SQUARE.

## 3.5 Modulate Wave Command

**DESCRIPTION** This command sets or gets the modulation parameters.

**COMMAND SYNTAX**

```
<channel>:MoDulateWaVe <type>
<channel>:MoDulateWaVe <parameter>,<value>
<channel>:={C1, C2}

<type>:= {AM,DSBAM,FM,PM,PWM,ASK,FSK,PSK}.
<parameter>:= {a parameter from the table below}.
<value>:= {value of the from the table below}.
```

| Parameters | Value | Description |
|---|---|---|
| STATE | <state> | :={ON, OFF}. Enable or disable modulation. STATE must be set to ON before you set or read other parameters of the modulation. |
| AM,SRC | <src> | := {INT, EXT,CH1,CH2}. AM modulation source. |
| AM,MDSP | <mod wave shape> | := {SINE, SQUARE, TRIANGLE, UPRAMP, DNRAMP, NOISE, ARB}. AM modulation wave. Only settable when SRC is INT. |
| AM,FRQ | <AM frequency> | := AM frequency. The unit is Hertz "Hz". Refer to the datasheet for the range of valid values. Only settable when SRC is INT. |
| AM,DEPTH | <depth> | := {0 to 120}. AM depth. The unit is "%". Only settable when SRC is INT. |
| DSBAM,SRC | <src> | := {INT, EXT}. DSB-AM modulation source. |
| DSBSC,SRC | <src> | = {INT, EXT, CH1,CH2}. DSB-SC modulation source. (only SDG7000A) |
| DSBAM,MDSP | <mod wave shape> | := {SINE, SQUARE, TRIANGLE, UPRAMP, DNRAMP, NOISE, ARB}. DSB AM modulation wave. Only settable when SRC is INT. |
| DSBSC,MDSP | <mod wave shape> | := {SINE, SQUARE, TRIANGLE, UPRAMP, DNRAMP, NOISE, ARB}. DSB-SC modulation wave. Only settable when SRC is INT. (only SDG7000A) |
| DSBAM,FRQ | <DSB-AM frequency> | := DSB-AM frequency. The unit is Hertz "Hz". Refer to the datasheet for the range of valid values. Only |

| | | |
|---|---|---|
| | | settable when SRC is INT. |
| DSBSC,FRQ | <DSB-SC frequency> | := DSB-SC frequency. The unit is Hertz "Hz". Refer to the datasheet for the range of valid values. Only settable when SRC is INT.(only SDG7000A) |
| FM,SRC | <src> | := {INT, EXT, CH1,CH2}. FM modulation source. |
| FM,MDSP | <mod wave shape> | := {SINE, SQUARE, TRIANGLE, UPRAMP, DNRAMP, NOISE, ARB}.<br>FM modulation wave. Only settable when SRC is INT. |
| FM,FRQ | <FM frequency> | := FM frequency. The unit is Hertz "Hz". Refer to the datasheet for the range of valid values. Only settable when SRC is INT. |
| FM,DEVI | <FM frequency deviation > | := {0 to carrier frequency}.<br>FM frequency deviation. The value depends on the difference between the carrier frequency and the bandwidth frequency. Only settable when SRC is INT. |
| PM,SRC, | <src> | := {INT, EXT, CH1,CH2}. PM modulation source. |
| PM,MDSP | <mod wave shape> | := {SINE, SQUARE, TRIANGLE, UPRAMP, DNRAMP, NOISE, ARB}.<br>PM modulation wave. Only settable when SRC is INT. |
| PM,FRQ | <PM frequency> | := PM frequency. The unit is Hertz "Hz". Refer to the datasheet for the range of valid values. Only settable when SRC is INT. |
| PM,DEVI | <PM phase offset> | := {0 to 360}. PM phase deviation. The unit is "degree". Only settable when SRC is INT. |
| PWM,SRC | <src> | := {INT, EXT, CH1,CH2}. PWM modulation source. |
| PWM,FRQ | <PWM frequency> | := PWM frequency. The unit is Hertz "Hz". Refer to the datasheet for the range of valid values. Only settable when SRC is INT. |
| PWM,DEVI | <PWM dev> | := Duty cycle deviation. The unit is "%". Value depends on the carrier duty cycle. |
| PWM,MDSP | <mod wave shape> | := {SINE, SQUARE, TRIANGLE, UPRAMP, DNRAMP, NOISE, ARB}.<br>PWM modulation wave. Only settable when SRC is INT. |
| ASK,SRC | <src> | := {INT, EXT}. ASK modulation source. |
| ASK,KFRQ | < key frequency> | := ASK key frequency. The unit is Hertz "Hz". Refer to the datasheet for the range of valid values. Only |

| | | |
|---|---|---|
| | | settable when SRC is INT. |
| FSK,SRC | <src> | := {INT, EXT}. FSK modulation source. |
| FSK,KFRQ | < key frequency> | := FSK key frequency. The unit is Hertz "Hz". Refer to the datasheet for the range of valid values. Only settable when SRC is INT. |
| FSK,HFRQ | <FSK_hop_freq> | := FSK hop frequency. The same with basic wave frequency. The unit is Hertz "Hz". Refer to the datasheet for the range of valid values. |
| PSK,SRC | <src> | := {INT, EXT}. PSK modulation source. |
| PSK,KFRQ | < key frequency> | := PSK key frequency. The unit is Hertz "Hz". Refer to the datasheet for the range of valid values. Only settable when SRC is INT. |
| PSK,PLRT | <polarity> | :={POS,NEG}. |
| CARR,WVTP | <wave type> | := {SINE, SQUARE, RAMP, ARB, PULSE}. Carrier waveform type. |
| CARR,FRQ | <frequency> | := carrier frequency. The unit is Hertz "Hz". Refer to the datasheet for the range of valid values. |
| CARR,PHSE | <phase> | := {0 to 360}. Carrier phase. The unit is "degree". |
| CARR,AMP | <amplitude> | := carrier amplitude. The unit is volts, peak-to-peak "Vpp". Refer to the datasheet for the range of valid values. |
| CARR,OFST | <offset> | := carrier offset. The unit is volts "V". Refer to the datasheet for the range of valid values. |
| CARR,SYM | <symmetry> | := {0 to 100}. Carrier symmetry when the carrier is RAMP. The unit is "%". |
| CARR,DUTY | <duty> | := {0 to 100}. Carrier duty cycle when the carrier is SQUARE or PULSE. The unit is "%". |
| CARR,RISE | <rise> | := rise time when the carrier is PULSE. The unit is seconds "s". Refer to the datasheet for the range of valid values. |
| CARR,FALL | <fall> | := fall time when the carrier is PULSE. The unit is seconds "s". Refer to the datasheet for the range of valid values. |
| CARR,DLY | <delay> | := pulse delay when the carrier is PULSE. The unit is seconds "s". Refer to the datasheet for the range of valid values. |

Notes:

The range of some parameters depends on the model. Refer to the datasheet for details.

**QUERY SYNTAX**

<channel>:MoDulateWaVe?

<channel>:={C1, C2}.

**RESPONSE FORMAT**

<channel>:MDWV <parameter>

<parameter>:= {all parameters of the current modulation}.

**EXAMPLE**

Set CH1 modulation state to on:

*C1:MDWV STATE,ON*

Set CH1 modulation type to AM:

*C1:MDWV AM*

Set modulation to AM, and the modulating wave type to sine wave:

*C1:MDWV AM,MDSP,SINE*

Read CH1 modulation parameters when STATE is ON:

C1:MDWV?

Return:

*C1:MDWV AM,STATE,ON,MDSP,SINE,SRC,INT,FRQ,100HZ, DEPTH,100,CARR,WVTP,RAMP,FRQ,1000HZ,AMP,4V, AMPVRMS,1.15473Vrms,OFST,0V,PHSE,0,SYM,50*

Read CH1 modulate wave parameters when STATE is OFF:

*C1:MDWV?*

Return:

*C1:MDWV STATE,OFF*

Set CH1 FM frequency to 1000 Hz:

*C1:MDWV FM,FRQ,1000*

Set CH1 carrier to SINE:

*C1:MDWV CARR,WVTP,SINE*

Set CH1 carrier frequency to 1000 Hz:

*C1:MDWV CARR,FRQ,1000*

Note: The table below shows the availability of some command parameters in each SDG series.

| Parameter /command | SDG800 | SDG1000 | SDG2000X | SDG5000 | SDG1000X | SDG6000X/X-E | SDG7000A |
|---|---|---|---|---|---|---|---|
| <channel> | no | yes | yes | yes | yes | yes | yes |
| <type>, SRC | no | yes | yes | yes | yes | yes | yes |
| CARR, DLY | no | yes | yes | yes | yes | yes | yes |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| CARR, RISE | yes | no | yes | yes | yes | yes | yes |
| CARR, FALL | yes | no | yes | yes | yes | yes | yes |

<type>:= {AM, FM, PM, FSK, ASK, PSK, DSBAM, PWM}.

## 3.6 Sweep Wave Command

### 3.6.1 <channel>: SweepWaVe <para>,<value>

**DESCRIPTION** This command sets or gets the sweep parameters. This command is not valid in SDG7000A. SDG7000A sweep wave command is referenced in 3.6.2—3.6.38

**COMMAND SYNTAX** <channel>:SweepWaVe <parameter>,<value>
<channel>:={C1, C2}
<parameter>:= {a parameter from the table below}
<value>:={value of the corresponding parameter}

| Parameters | Value | Description |
|---|---|---|
| STATE | <state> | :={ON, OFF}. Enable or disable sweep. STATE must be set to ON before you set or read other parameters of the sweep. |
| TIME | <time> | := sweep time. The unit is seconds "s". Refer to the datasheet for the range of valid values. |
| STARTTIME | <time> | :={0 to 300}. The unit is seconds "s".Start hold time |
| ENDTIME | <time> | :={0 to 300}. The unit is seconds "s".End hold time |
| BACKTIME | <time> | :={0 to 300}. The unit is seconds "s".Back time |
| START | <start_freq> | := start frequency. The same with basic wave frequency. The unit is Hertz "Hz". |
| STOP | <stop_freq> | := stop frequency. The same with basic wave frequency. The unit is Hertz "Hz". |
| CENTER | <center_freq> | := center frequency. The unit is Hertz "Hz". Refer to the datasheet for the range of valid values. |
| SPAN | <span_freq> | := frequency span. The unit is Hertz "Hz". Refer to the datasheet for the range of valid values. |

| | | |
|---|---|---|
| SWMD | <sweep_mode> | := {LINE, LOG,STEP}, in which LINE refers to Linear ,LOG refers to Logarithmic and STEP refers to step. |
| DIR | <direction> | := {UP, DOWN, UP_DOWN}. Sweep direction. |
| SYM | <symmetry> | :={0% to 100%}. The symmetry when sweep direction is up. |
| TRSR | <trig_src> | := {EXT, INT, MAN}. Trigger source. EXT refers to External, INT refers to Internal and MAN refers to Manual. |
| MTRIG | | := send a manual trigger. Only valid when TRSR is MAN. |
| TRMD | <trig_mode> | := {ON, OFF}. State of trigger output. If TRSR is EXT, the parameter is invalid. |
| EDGE | <edge> | :={RISE, FALL}. Available trigger edge. Only valid when TRSR is EXT or MAN. |
| CARR, WVTP | <wave type> | := {SINE, SQUARE, RAMP, ARB}. Carrier waveform type. Modulation is not available if the carrier is PULSE, NOISE, or DC. |
| CARR, FRQ | <frequency> | := carrier frequency. The unit is Hertz "Hz". Refer to the datasheet for the range of valid values. |
| CARR, PHSE | <phase> | := {0 to 360}. Carrier phase. The unit is "degree". |
| CARR, AMP | <amplitude> | := carrier amplitude. The unit is volts, peak-to-peak "Vpp". Refer to the datasheet for the range of valid values. |
| CARR, OFST | <offset> | := carrier offset. The unit is volts "V". Refer to the datasheet for the range of valid values. |
| CARR, SYM | <symmetry> | := {0 to 100}. Carrier symmetry when the carrier is RAMP. The unit is percent "%". |
| CARR, DUTY | <duty> | := {0 to 100}. Carrier duty cycle when the carrier is SQUARE. The unit is "%". |
| MARK_STATE | <state> | := {ON, OFF}. |
| MARK_FREQ | <frequency> | := mark frequency. The unit is Hertz "Hz". The range is from the start frequency to the stop frequency. |

**QUERY SYNTAX** <channel>:SWeepWaVe?

<channel>:= {C1, C2}.

**RESPONSE FORMAT** <channel>:SWWV <parameter>

<parameter>:= {All parameters of the current sweep wave}

**EXAMPLE**

Set CH1 sweep state to ON:
*C1:SWWV STATE,ON*

Set CH1 sweep time to 1 s:
*C1:SWWV TIME,1*

Set CH1 stop frequency to 1000 Hz:
*C1:SWWV STOP,1000*

Set the trigger source of CH1 to Manual:
*C1:SWWV TRSR,MAN*
Send a manual trigger to CH1:
*C1:SWWV MTRIG*

Read CH2 sweep parameters when STATE is ON:
*C2:SWWV?*
Return:
*C2:SWWVSTATE,ON,TIME,1S,STOP,1500HZ,START,500HZ,CENTER,1000HZ,SPAN,1000HZ,TRSR,INT,TRMD,OFF,SWMD,LINE,DIR,UP,SYM,0,MARK_STATE,OFF,MARK_FREQ,1000HZ,CARR,WVTP,SINE,FRQ,1000HZ,AMP,4V,AMPVRMS,1.41421Vrms,OFST,0V,PHSE,0*

Read CH2 sweep parameters when STATE is OFF:
*C2:SWWV?*
Return:
*C2:SWWV STATE,OFF*
Set CH1 the FreqMarker of sweep to 1kHz
*C1:SWWV MARK_STATE,ON,MARK_FREQ,1000*

Note: The table below shows the availability of some command parameters in each SDG series.

| Parameter /command | SDG 800 | SDG 1000 | SDG 2000X | SDG 5000 | SDG 1000X | SDG6000 X/X-E |
|---|---|---|---|---|---|---|
| <channel> | no | yes | yes | yes | yes | yes |
| STATE | no | yes | yes | yes | yes | yes |
| TRMD | no | yes | yes | yes | yes | yes |
| EDGE | no | yes | yes | yes | yes | yes |
| START | | | | | | |

| **Parameter /command** | **SDG 800** | **SDG 1000** | **SDG 2000X** | **SDG 5000** | **SDG 1000X** | **SDG6000 X/X-E** |
|---|---|---|---|---|---|---|
| STOP | | | | | | |
| CENTER | | | | | | |
| SPAN | | | | | | |
| SWMD | | | | | | |
| DIR | | | | | | |
| TRSR | | | | | | |
| MTRIG | | | | | | |
| TRMD | | | | | | |
| CARR, WVTP | | | | | | |
| CARR, FRQ | | | | | | |
| CARR, PHSE | | | | | | |
| CARR, AMP | | | | | | |
| CARR, OFST | | | | | | |
| CARR, SYM | | | | | | |
| CARR, DUTY | | | | | | |

### 3.6.2 <channel>:SWEep <switch>

**DESCRIPTION** This command sets or gets the sweep state. This command is only used by the SDG7000A.

**COMMAND SYNTAX**
<channel>:SWEep <switch>
<channel>:= {C1, C2}.
<switch>:= {ON, OFF}.

**QUERY SYNTAX**
<channel>: SWEep?
<channel>:= {C1, C2}.

**EXAMPLE**
Set CH1 sweep state ON.
*:C1:SWEep ON*
Get CH1 sweep state.
*:C1:SWEep?*
Return:
*“ON”*

### 3.6.3 <channel>:SWEep:TYPE <type>

| | |
|---|---|
| **DESCRIPTION** | This command sets or gets the sweep type. This command is only used by the SDG7000A. |
| **COMMAND SYNTAX** | <channel>:SWEep:TYPE <type><br><channel>:= {C1, C2}.<br><type>:= {FREQ, AMP, BOTH}. |
| **QUERY SYNTAX** | <channel>: SWEep:TYPE?<br><channel>:= {C1, C2}. |
| **EXAMPLE** | Set CH1 sweep type to frequency sweep.<br>*:C1:SWEep:TYPE FREQ*<br>Get CH1 sweep type.<br>*:C1:SWEep:TYPE?*<br>Return:<br>*“FREQ”* |

### 3.6.4 <channel>:SWEep:SOURce <src>

| | |
|---|---|
| **DESCRIPTION** | This command sets or gets the sweep trig source. This command is only used by the SDG7000A. |
| **COMMAND SYNTAX** | <channel>:SWEep:SOURce <src><br><channel>:= {C1, C2}.<br><src>:= {INT,EXT, MAN}. |
| **QUERY SYNTAX** | <channel>: SWEep:SOURce?<br><channel>:= {C1,C2}. |
| **EXAMPLE** | Set CH1 sweep trig source to internal.<br>*:C1:SWEep:SOURce INT*<br>Get CH1 sweep trig source.<br>*:C1:SWEep:SOURce?*<br>Return:<br>*“INT”* |

### 3.6.5 <channel>:SWEep:FMODe <mode>

**DESCRIPTION** This command sets or gets the frequency sweep mode. This command is only used by the SDG7000A.

**COMMAND SYNTAX** <channel>:SWEep:FMODe <mode>

<channel>:= {C1, C2}.

<mode>:= {LINE,LOG,STEP}.

**QUERY SYNTAX** <channel>: SWEep:FMODe?

<channel>:= {C1, C2}.

**EXAMPLE** Set CH1 frequency sweep mode to linear.

*:C1:SWEep:FMODe linear.*

Get CH1 frequency sweep mode.

*:C1:SWEep:FMODe?*

Return?

*“LINE”*

### 3.6.6 <channel>:SWEep:AMODe <mode>

**DESCRIPTION** This command sets or gets the amplitude sweep mode. This command is only used by the SDG7000A.

**COMMAND SYNTAX** <channel>:SWEep:AMODe <mode>

<channel>:= {C1, C2}.

<mode>:= {LINE,STEP}.

**QUERY SYNTAX** <channel>: SWEep:AMODe?

<channel>:= {C1, C2}.

**EXAMPLE** Set CH1 amplitude sweep mode to linear.

*:C1:SWEep:AMODe linear.*

Get CH1 sweep mode.

*:C1:SWEep:AMODe?*

Return,

*“LINE”*

### 3.6.7 <channel>:SWEep:FSNumber <value>

| | |
|---|---|
| **DESCRIPTION** | This command sets or gets the frequency sweep step number. This command is only used by the SDG7000A. |
| **COMMAND SYNTAX** | <channel>:SWEep:FSNumber <value><br><channel>:= {C1, C2}.<br><value>:= integral number between 2 and 1024. |
| **QUERY SYNTAX** | <channel>: SWEep: FSNumber?<br><channel>:= {C1, C2}. |
| **EXAMPLE** | Sets CH1 frequency sweep step number to 10<br>*:C1:SWEep:FSNumber 10*<br>Gets CH1 frequency sweep step number<br>*:C1:SWEep:FSNumber?*<br>Return:<br>*“10”* |

### 3.6.8 <channel>:SWEep:ASNumber <value>

| | |
|---|---|
| **DESCRIPTION** | This command sets or gets the amplitude sweep step number. This command is only used by the SDG7000A. |
| **COMMAND SYNTAX** | <channel>:SWEep:ASNumber <value><br><channel>:= {C1, C2}.<br><value>:= integral number between 2 and 1024 |
| **QUERY SYNTAX** | <channel>: SWEep: ASNumber?<br><channel>:= {C, C2}. |
| **EXAMPLE** | Sets CH1 amplitude sweep step number to 10<br>*:C1:SWEep:ASNumber 10*<br>Gets CH1 amplitude sweep step number<br>*:C1:SWEep:ASNumber?*<br>Return:<br>*“10”* |

### 3.6.9 <channel>:SWEep:TIME <value>

| | |
|---|---|
| **DESCRIPTION** | This command sets or gets the sweep time. This command is only used by the SDG7000A.. |
| **COMMAND SYNTAX** | <channel>:SWEep:TIME <value><br><channel>:= {C1, C2}.<br><value>:= floating number with second unit |
| **QUERY SYNTAX** | <channel>: SWEep:TIME?<br><channel>:= {C1, C2}. |
| **EXAMPLE** | Sets CH1 sweep time to 10 seconds.<br>*:C1:SWEep:TIME 10*<br>Gets CH1 sweep time<br>*:C1:SWEep:TIME?*<br>Return:<br>*“10”* |

### 3.6.10 <channel>:SWEep:SHTime <value>

| | |
|---|---|
| **DESCRIPTION** | This command sets or gets the sweep start hold time. This command is only used by the SDG7000A. |
| **COMMAND SYNTAX** | <channel>:SWEep:SHTime <value><br><channel>:= {C1, C2}.<br><value>:= floating number with second unit |
| **QUERY SYNTAX** | <channel>: SWEep:SHTime?<br><channel>:= {C1, C2}. |
| **EXAMPLE** | Sets CH1 sweep start hold time to 100 milliseconds.<br>*:C1:SWEep:SHTime 0.1*<br>Gets CH1 sweep start hold time<br>*:C1:SWEep:SHTime?*<br>Return:<br>*“0.1”* |

### 3.6.11 <channel>:SWEep:EHTime <value>

**DESCRIPTION**

This command sets or gets the sweep end hold time. This command is only used by the SDG7000A.

**COMMAND SYNTAX**

<channel>:SWEep:EHTime <value>
<channel>:= {C1, C2}.
<value>:= floating number with second unit

**QUERY SYNTAX**

<channel>: SWEep:EHTime?
<channel>:= {C1, C2}.

**EXAMPLE**

Sets CH1 sweep end hold time to 100 milliseconds.
*:C1:SWEep:EHTime 0.1*
Gets CH1 sweep end hold time
*:C1:SWEep:EHTime?*
Return:
*"0.1"*

### 3.6.12 <channel>:SWEep:RTIMe <value>

**DESCRIPTION**

This command sets or gets the sweep return time. This command is only used by the SDG7000A.

**COMMAND SYNTAX**

<channel>:SWEep:RTIMe <value>
<channel>:= {C1, C2}.
<value>:= floating number with second unit

**QUERY SYNTAX**

<channel>: SWEep:RTIMe?
<channel>:= {C1, C2}.

**EXAMPLE**

Sets CH1 sweep return time to 100 milliseconds.
*:C1:SWEep:RTIMe 0.1*
Gets CH1 sweep return time
*:C1:SWEep:RTIMe?*
Return:
*"0.1"*

### 3.6.13 <channel>:SWEep:SFRequency <value>

| | |
|---|---|
| **DESCRIPTION** | This command sets or gets the sweep start frequency. This command is only used by the SDG7000A. |
| **COMMAND SYNTAX** | <channel>:SWEep:SFRequency <value><br><channel>:= {C1, C2}.<br><value>:= floating number with hertz unit |
| **QUERY SYNTAX** | <channel>: SWEep:SFRequency?<br><channel>:= {C1, C2}. |
| **EXAMPLE** | Sets CH1 sweep start frequency to 1kHz<br>*:C1:SWEep:SFRequency 1000*<br>Gets CH1 sweep start frequency.<br>*:C1:SWEep:SFRequency?*<br>Return:<br>*“1000”* |

### 3.6.14 <channel>:SWEep:EFRequency <value>

| | |
|---|---|
| **DESCRIPTION** | This command sets or gets the sweep stop frequency. This command is only used by the SDG7000A. |
| **COMMAND SYNTAX** | <channel>:SWEep:EFRequency <value><br><channel>:= {C1, C2}.<br><value>:= floating number with hertz unit |
| **QUERY SYNTAX** | <channel>: SWEep:EFRequency?<br><channel>:= {C1, C2}. |
| **EXAMPLE** | Sets CH1 sweep stop frequency to 9kHz<br>*:C1:SWEep:EFRequency 9000*<br>Gets CH1 sweep stop frequency.<br>*:C1:SWEep:EFRequency?*<br>Return:<br>*“9000”* |

### 3.6.15 <channel>:SWEep:CFRequency <value>

**DESCRIPTION** This command sets or gets the sweep center frequency. This command is only used by the SDG7000A.

**COMMAND SYNTAX** <channel>:SWEep:CFRequency <value>
<channel>:= {C1, C2}.
<value>:= floating number with hertz unit

**QUERY SYNTAX** <channel>: SWEep:CFRequency?
<channel>:= {C1, C2}..

**EXAMPLE** Sets CH1 sweep center frequency to 5kHz
*:C1:SWEep:CFRequency 5000*
Gets CH1 sweep center frequency.
*:C1:SWEep:CFRequency?*
Return:
*“5000”*

### 3.6.16 <channel>:SWEep:FSPan <value>

**DESCRIPTION** This command sets or gets the sweep frequency span. This command is only used by the SDG7000A.

**COMMAND SYNTAX** <channel>:SWEep:FSPan <value>
<channel>:= {C1, C2}.
<value>:= floating number with hertz unit

**QUERY SYNTAX** <channel>: SWEep:FSPan?
<channel>:= {C1, C2}.

**EXAMPLE** Sets CH1 sweep frequency span to 8kHz
*:C1:SWEep:FSPan 8000*
Gets CH1 sweep frequency span.
*:C1:SWEep:FSPan?*
Return:
*“8000”*

## 3.6.17 <channel>:SWEep:SAMPlitude <value>

| | |
|---|---|
| **DESCRIPTION** | This command sets or gets the sweep start amplitude. This command is only used by the SDG7000A. |
| **COMMAND SYNTAX** | <channel>:SWEep:SAMPlitude <value><br><channel>:= {C1, C2}.<br><value>:= floating number with volt unit |
| **QUERY SYNTAX** | <channel>: SWEep:SAMPlitude?<br><channel>:= {C1, C2}. |
| **EXAMPLE** | Sets CH1 sweep start amplitude to 100mv<br>*:C1:SWEep:SAMPlitude 0.1*<br>Gets CH1 sweep start amplitude.<br>*:C1:SWEep:SAMPlitude?*<br>Return:<br>*“0.1”* |

## 3.6.18 <channel>:SWEep:EAMPlitude <value>

| | |
|---|---|
| **DESCRIPTION** | This command sets or gets the sweep stop amplitude. This command is only used by the SDG7000A. |
| **COMMAND SYNTAX** | <channel>:SWEep:EAMPlitude <value><br><channel>:= {C1, C2}.<br><value>:= floating number with volt unit |
| **QUERY SYNTAX** | <channel>: SWEep:EAMPlitude?<br><channel>:= {C1, C2}. |
| **EXAMPLE** | Sets CH1 sweep stop amplitude to 900mv<br>*:C1:SWEep:EAMPlitude 0.9*<br>Gets CH1 sweep stop amplitude.<br>*:C1:SWEep:EAMPlitude?*<br>Return:<br>*“0.9”* |

## 3.6.19 <channel>:SWEep:CAMPlitude <value>

**DESCRIPTION** This command sets or gets the sweep center amplitude. This command is only used by the SDG7000A.

**COMMAND SYNTAX** <channel>:SWEep:CAMPlitude <value>
<channel>:= {C1, C2}.
<value>:= floating number with volt unit

**QUERY SYNTAX** <channel>: SWEep:CAMPlitude?
<channel>:= {C1, C2}.

**EXAMPLE** Sets CH1 sweep center amplitude to 500mv
*:C1:SWEep:CAMPlitude 0.5*
Gets CH1 sweep center amplitude.
*:C1:SWEep:CAMPlitude?*
Return:
*“0.5”*

## 3.6.20 <channel>:SWEep:ASPan <value>

**DESCRIPTION** This command sets or gets the sweep amplitude span. This command is only used by the SDG7000A.

**COMMAND SYNTAX** <channel>:SWEep:ASPan <value>
<channel>:= {C1, C2}.
<value>:= floating number with volt unit

**QUERY SYNTAX** <channel>: SWEep:ASPan?
<channel>:= {C1, C2}..

**EXAMPLE** Sets CH1 sweep amplitude span to 800mv
*:C1:SWEep:ASPan 0.8*
Gets CH1 sweep amplitude span.
*:C1:SWEep:ASPan?*
Return:
*“0.8”*

### 3.6.21 <channel>:SWEep:FDIRection <direction>

|  |  |
|---|---|
| **DESCRIPTION** | This command sets or gets the frequency sweep direction. This command is only used by the SDG7000A. |
| **COMMAND SYNTAX** | <channel>:SWEep:FDIRection <direction><br><channel>:= {C1, C2}.<br><direction>:= {UP, DOWN, UP_DOWN}. |
| **QUERY SYNTAX** | <channel>: SWEep:FDIRection?<br><channel>:= {C1, C2}. |
| **EXAMPLE** | Sets CH1 frequency sweep direction up.<br>*:C1:SWEep:FDIRection UP*<br>Gets CH1 frequency sweep direction.<br>*:C1:SWEep:FDIRection?*<br>Return:<br>*“UP”* |

### 3.6.22 <channel>:SWEep:ADIRection <direction>

|  |  |
|---|---|
| **DESCRIPTION** | This command sets or gets the amplitude sweep direction. This command is only used by the SDG7000A. |
| **COMMAND SYNTAX** | <channel>:SWEep:ADIRection <direction><br><channel>:= {C1, C2}.<br><direction>:= {UP, DOWN, UP_DOWN}. |
| **QUERY SYNTAX** | <channel>: SWEep:ADIRection?<br><channel>:= {C1, C2}. |
| **EXAMPLE** | Sets CH1 amplitude sweep direction up.<br>*:C1:SWEep:ADIRection UP*<br>Gets CH1 amplitude sweep direction.<br>*:C1:SWEep:ADIRection?*<br>Return:<br>*“UP”* |

### 3.6.23 <channel>:SWEep:FSYMmetry <value>

| | |
|---|---|
| **DESCRIPTION** | This command sets or gets the frequency sweep symmetry when direction is UP_DOWN. This command is only used by the SDG7000A. |
| **COMMAND SYNTAX** | <channel>:SWEep:FSYMmetry <value><br><channel>:= {C1, C2}.<br><value>:= floating number between 0 an 100. |
| **QUERY SYNTAX** | <channel>: SWEep:FSYMmetry?<br><channel>:= {C1, C2}. |
| **EXAMPLE** | Sets symmetry to 50% when CH1 frequency sweep direction is UP_DOWN<br>*:C1:SWEep:FSYMmetry 50*<br>Gets symmetry value when CH1 frequency sweep direction is UP_DOWN.<br>*:C1:SWEep:FSYMmetry?*<br>Return:<br>*“50”* |

### 3.6.24 <channel>:SWEep:ASYMmetry <value>

| | |
|---|---|
| **DESCRIPTION** | This command sets or gets the amplitude sweep symmetry when direction is UP_DOWN. This command is only used by the SDG7000A. |
| **COMMAND SYNTAX** | <channel>:SWEep:ASYMmetry <value><br><channel>:= {C1, C2}.<br><value>:= floating number between 0 an 100. |
| **QUERY SYNTAX** | <channel>: SWEep:ASYMmetry?<br><channel>:= {C1, C2}. |
| **EXAMPLE** | Sets symmetry to 50% when CH1 amplitude sweep direction is UP_DOWN<br>*:C1:SWEep:ASYMmetry 50*<br>Gets symmetry value when CH1 amplitude sweep direction is UP_DOWN. |

*:C1:SWEep:ASYMmetry?*

Return:

*"50"*

### 3.6.25 <channel>:SWEep:TOUT <switch>

| | |
|---|---|
| **DESCRIPTION** | This command sets or gets the sweep trig out state. This command is only used by the SDG7000A. |
| **COMMAND SYNTAX** | <channel>:SWEep:TOUT <switch><br><channel>:= {C1, C2}.<br><switch>:= {ON, OFF}. |
| **QUERY SYNTAX** | <channel>: SWEep:TOUT?<br><channel>:= {C1, C2}. |
| **EXAMPLE** | Sets CH1 sweep trig out state ON.<br>*:C1:SWEep:TOUT ON*<br>Gets CH1 sweep trig out state.<br>*:C1:SWEep:TOUT?*<br>Return:<br>*"ON"* |

### 3.6.26 <channel>:SWEep:EDGe <polarity>

| | |
|---|---|
| **DESCRIPTION** | This command sets or gets the sweep external trig edge. This command is only used by the SDG7000A. |
| **COMMAND SYNTAX** | <channel>:SWEep:EDGe <polarity><br><channel>:= {C1, C2}.<br><polarity>:= {RISE, FALL}. |
| **QUERY SYNTAX** | <channel>: SWEep:EDGe?<br><channel>:= {C1, C2}. |
| **EXAMPLE** | Sets CH1 sweep external trig edge RISE<br>*:C1:SWEep:EDGe RISE*<br>Gets CH1 sweep external trig edge.<br>*:C1:SWEep:EDGe?*<br>Return: |

*“RISE”*

### 3.6.27 <channel>:SWEep:MTRigger

| | |
|---|---|
| **DESCRIPTION** | This command trig sweep once when trig source is manual. This command is only used by the SDG7000A. |
| **COMMAND SYNTAX** | <channel>:SWEep:MTRigger<br><channel>:= {C1, C2}. |
| **EXAMPLE** | Sets CH1 sweep trig once when trig source is manual.<br>*:C1:SWEep:MTRigger* |

### 3.6.28 <channel>:SWEep:FMARker <switch>

| | |
|---|---|
| **DESCRIPTION** | This command sets or gets the sweep frequency marker state. This command is only used by the SDG7000A. |
| **COMMAND SYNTAX** | <channel>:SWEep:FMARker <switch><br><channel>:= {C1, C2}.<br><switch>:= {ON, OFF}. |
| **QUERY SYNTAX** | <channel>: SWEep:FMARker?<br><channel>:= {C1, C2}. |
| **EXAMPLE** | Sets CH1 sweep frequency marker state ON.<br>*:C1:SWEep:FMARker ON*<br>Gets CH1 sweep frequency marker state<br>*:C1:SWEep:FMARker?*<br>Return:<br>*“ON”* |

### 3.6.29 <channel>:SWEep:MFRequency <value>

| | |
|---|---|
| **DESCRIPTION** | This command sets or gets the sweep marker frequency. This command is only used by the SDG7000A. |
| **COMMAND SYNTAX** | <channel>:SWEep:MFRequency<value><br><channel>:= {C1, C2}. |

<value>:= floating number with hertz unit

**QUERY SYNTAX** <channel>: SWEep:MFRequency?
<channel>:= {C1, C2}.

**EXAMPLE** Sets CH1 sweep marker frequency 5kHz.
*:C1:SWEep:MFRequency 5000*
Gets CH1 sweep marker frequency.
*:C1:SWEep:MFRequency?*
Return:
*"5000"*

### 3.6.30 <channel>:SWEep:MSNumber <value>

**DESCRIPTION** This command sets or gets the sweep marker step number. This command is only used by the SDG7000A.

**COMMAND SYNTAX** <channel>:SWEep:MSNumber <value>
<channel>:= {C1, C2}.
<value>:= integral number between 2 and 1024

**QUERY SYNTAX** <channel>: SWEep:MSNumber?
<channel>:= {C1, C2}.

**EXAMPLE** Sets CH1 sweep marker the fifth step.
*:C1:SWEep:MSNumber 5*
Gets CH1 sweep marker step number.
*:C1:SWEep:MSNumber?*
Return:
*"5"*

### 3.6.31 <channel>:SWEep:CARRier:WTYPe <wave>

**DESCRIPTION** This command sets or gets the sweep carrier wave type. This command is only used by the SDG7000A.

**COMMAND SYNTAX** <channel>:SWEep:CARRier:WTYPe <wave>
<channel>:= {C1. C2}.

| | |
|---|---|
| | <wave>:={SINE,SQUARE,RAMP,AFG} |
| **QUERY SYNTAX** | <channel>: SWEep:CARRier:WTYPe?<br><channel>:= {C1, C2}. |
| **EXAMPLE** | Sets CH1 sweep carrier wave type sine.<br>*:C1:SWEep:CARRier:WTYPe SINE*<br>Gets CH1 sweep carrier wave type.<br>*:C1:SWEep:CARRier:WTYPe?*<br>Return:<br>*"SINE"* |

### 3.6.32 <channel>:SWEep:CARRier:FREQuency <value>

| | |
|---|---|
| **DESCRIPTION** | This command sets or gets the sweep carrier wave frequency. This command is only used by the SDG7000A. |
| **COMMAND SYNTAX** | <channel>:SWEep:CARRier:FREQuency <value><br><channel>:= {C1, C2}.<br><value>:= floating number with hertz unit |
| **QUERY SYNTAX** | <channel>: SWEep:CARRier:FREQuency?<br><channel>:={C1, C2}. |
| **EXAMPLE** | Sets CH1 sweep carrier wave frequency 1MHz.<br>*:C1:SWEep:CARRier:FREQuency 1000000*<br>Gets CH1 sweep carrier wave frequency.<br>*:C1:SWEep:CARRier:FREQuency?*<br>Return:<br>*"1000000"* |

### 3.6.33 <channel>:SWEep:CARRier:PHASe <value>

| | |
|---|---|
| **DESCRIPTION** | This command sets or gets the sweep carrier wave phase. This command is only used by the SDG7000A. |
| **COMMAND SYNTAX** | <channel>:SWEep:CARRier:PHASe <value><br><channel>:= {C1, C2}. |

<value>:= floating number with degree unit

**QUERY SYNTAX**

<channel>: SWEep:CARRier:PHASe?
<channel>:={C1, C2}.

**EXAMPLE**

Sets CH1 sweep carrier wave phase 90°.
*:C1:SWEep:CARRier:PHASe 90*
Gets CH1 sweep carrier wave phase.
*:C1:SWEep:CARRier:PHASe?*
Return:
*"90"*

## 3.6.34 <channel>:SWEep:CARRier:PAMPlitude <value>

**DESCRIPTION**

This command sets or gets the sweep carrier wave amplitude by Vpp. This command is only used by the SDG7000A.

**COMMAND SYNTAX**

<channel>:SWEep:CARRier:PAMPlitude <value>
<channel>:= {C1, C2}.
<value>:= floating number with volt unit

**QUERY SYNTAX**

<channel>: SWEep:CARRier:PAMPlitude?
<channel>:={C1, C2}.

**EXAMPLE**

Sets CH1 sweep carrier wave amplitude 4Vpp.
*:C1:SWEep:CARRier:PAMPlitude 4*
Gets CH1 sweep carrier wave amplitude by Vpp.
*:C1:SWEep:CARRier:PAMPlitude?*
Return:
*"4"*

## 3.6.35 <channel>:SWEep:CARRier:RAMPlitude <value>

**DESCRIPTION**

This command sets or gets the sweep carrier wave amplitude by Vrms. This command is only used by the SDG7000A.

**COMMAND SYNTAX**

<channel>:SWEep:CARRier:RAMPlitude <value>
<channel>:= {C1, C2}.

<value>:= floating number with unit Vrms.

**QUERY SYNTAX**

<channel>: SWEep:CARRier:RAMPlitude?
<channel>:={C1, C2}.

**EXAMPLE**

Sets CH1 sweep carrier wave amplitude 1.414Vrms.
*:C1:SWEep:CARRier:RAMPlitude 1.414*
Gets CH1 sweep carrier wave amplitude by Vrms.
*:C1:SWEep:CARRier:RAMPlitude?*
Return:
*"1.414"*

### 3.6.36 <channel>:SWEep:CARRier:OFFSet <value>

**DESCRIPTION**

This command sets or gets the sweep carrier wave offset. This command is only used by the SDG7000A.

**COMMAND SYNTAX**

<channel>:SWEep:CARRier:OFFSet <value>
<channel>:= {C1, C2}.
<value>:= floating number with volt unit.

**QUERY SYNTAX**

<channel>: SWEep:CARRier:OFFSet?
<channel>:={C1, C2}.

**EXAMPLE**

Sets CH1 sweep carrier wave offset 2V.
*:C1:SWEep:CARRier:OFFSet 2*
Gets CH1 sweep carrier wave offset.
*:C1:SWEep:CARRier:OFFSet?*
Return:
*"2"*

### 3.6.37 <channel>:SWEep:CARRier:SYMMetry <value>

**DESCRIPTION**

This command sets or gets the sweep carrier wave symmetry when carrier wave type is ramp. This command is only used by the SDG7000A.

**COMMAND SYNTAX**

<channel>:SWEep:CARRier:SYMMetry <value>

<channel>:= {C1, C2}.

<value>:= floating number between 0 an 100.

**QUERY SYNTAX**

<channel>: SWEep:CARRier:SYMMetry?

<channel>:= {C1, C2}.

**EXAMPLE**

Sets symmetry to 50% when CH1 sweep carrier wave type is ramp.

*:C1:SWEep:CARRier:SYMMetry 50*

Gets symmetry value when CH1 sweep carrier wave type is ramp.

*:C1:SWEep:CARRier:SYMMetry?*

Return:

*"50"*

### 3.6.38 <channel>:SWEep:CARRier:DUTY <value>

**DESCRIPTION**

This command sets or gets the sweep carrier wave duty when carrier wave type is square. This command is only used by the SDG7000A.

**COMMAND SYNTAX**

<channel>:SWEep:CARRier:DUTY <value>

<channel>:= {C1, C2}.

<value>:= floating number between 0 an 100.

**QUERY SYNTAX**

<channel>: SWEep:CARRier:DUTY?

<channel>:= {C1, C2}.

**EXAMPLE**

Sets duty to 50% when CH1 sweep carrier wave type is square.

*:C1:SWEep:CARRier:DUTY 50*

Gets duty value when CH1 sweep carrier wave type is square.

*:C1:SWEep:CARRier:DUTY?*

Return:

*"50"*

## 3.7 Burst Wave Command

**DESCRIPTION**

This command sets or gets the burst wave parameters.

**COMMAND SYNTAX** <channel>:BursTWaVe <parameter>,<value>

<channel>:= {C1, C2}.

<parameter>:= {a parameter from the table below}.

<value>:= {value of the corresponding parameter}.

| Parameters | Value | Description |
|---|---|---|
| STATE | <state> | := {ON, OFF}. Enable or disable burst.<br>STATE must be set to ON before you set or read other parameters of the burst. |
| PRD | <period> | := burst period. Refer to the datasheet for the range of valid values. The unit is seconds “s” Not valid when:<br>• Carrier is NOISE<br>• GATE_NCYC is GATE (except the "X" series)<br>• TRSR is EXT |
| STPS | <start_phase> | := {0 to 360}. Start phase of the carrier. The unit is "degree". Not valid when the carrier is NOISE or PULSE. |
| GATE_NCYC | <burst_mode> | := {GATE, NCYC}. Burst mode. Not valid when the carrier is NOISE. |
| TRSR | <trig_src> | := {EXT, INT, MAN}. Trigger source. EXT refers to External, INT refers to Internal and MAN refers to Manual. |
| MTRIG | | := send a manual trigger. Only when TRSR is MAN, the parameter is valid. |
| DLAY | <delay> | := trigger delay. The unit is seconds "s". Refer to the datasheet for the range of valid values. Available when GATE_NCYC is NCYC. Not valid when the carrier is NOISE. |
| PLRT | <polarity> | := {NEG, POS}. Gate polarity. Negative or Positive. |
| TRMD | <trig_mode> | := {RISE, FALL, OFF}. Trigger out mode. Available when GATE_NCYC is NCYC and TRSR is INT or MAN. Not valid when the carrier is NOISE. |
| EDGE | <edge> | :={RISE, FALL}. Available trigger edge. Only valid when TRSR is EXT or MAN. |
| EDGE | <edge> | := { RISE, FALL}. Available trigger edge. Available when GATE_NCYC is NCYC and TRSR is EXT. Not valid when the carrier is NOISE. |
| TIME | <circle_ time> | :={INF, 1, 2,..., M}, where M is the maximum supported Ncycle number which depends on the model; INF sets the burst to Infinite mode.<br>Available when GATE_NCYC is NCYC. Not valid when the carrier is NOISE. |
| COUNT | <counter> | :=Burst count, Only valid when TRSR is EXT or MAN. |
| CARR, WVTP | <wave type> | := {SINE, SQUARE, RAMP, ARB, PULSE, NOISE}. Carrier waveform type. |

| | | |
|---|---|---|
| CARR, FRQ | <frequency> | := carrier frequency. The unit is Hertz "Hz". Refer to the datasheet for the range of valid values. |
| CARR, PHSE | <phase> | := {0 to 360}. Carrier phase. The unit is "degree". |
| CARR, AMP | <amplitude> | := carrier amplitude. The unit is volts, peak-to-peak "Vpp". Refer to the datasheet for the range of valid values. |
| CARR, OFST | <offset> | := carrier offset. The unit is volts "V". Refer to the datasheet for the range of valid values. |
| CARR, SYM | <symmetry> | := {0 to 100}. Carrier symmetry when the carrier is RAMP. The unit is "%". |
| CARR, DUTY | <duty> | := {0 to 100}. Carrier duty cycle when the carrier is SQUARE or PULSE. The unit is "%". |
| CARR, RISE | <rise> | := rise time when the carrier is PULSE. The unit is seconds "s". Refer to the datasheet for the range of valid values. |
| CARR, FALL | <fall> | := fall time when the carrier is PULSE. The unit is seconds "s". Refer to the datasheet for the range of valid values. |
| CARR, DLY | <delay> | := pulse delay when the carrier is PULSE. The unit is seconds “s”. Refer to the datasheet for the range of valid values. |
| CARR, STDEV | <stdev> | := standard deviation of NOISE. The unit is volts “V”. Refer to the datasheet for the range of valid values. |
| CARR, MEAN | <mean> | := mean of NOISE. The unit is volts “V”. Refer to the datasheet for the range of valid values. |

**QUERY SYNTAX** <channel>:BTWV(BursTWaVe)?

<channel>:={C1, C2}

**RESPONSE FORMAT** <channel>:BTWV <parameter>

<parameter>:={All parameters of the current burst wave.}

**EXAMPLE** Set CH1 burst state to ON

*C1:BTWV STATE,ON*

Set CH1 burst period to 1 s.

*C1:BTWV PRD,1*

Set CH1 burst delay to 1 s

*C1:BTWV DLAY,1*

Set CH1 burst to infinite

*C1:BTWV TIME,INF*

Read CH2 burst parameters when the STATE is ON.

*C2:BTWV?*

Return:

*C2:BTWV STATE,ON,PRD,0.01S,STPS,0,TRSR,INT,
TRMD,OFF,TIME,1,DLAY,2.4e-07S,GATE_NCYC,NCYC,
CARR,WVTP,SINE,FRQ,1000HZ,AMP,4V,OFST,0V,PHSE,0*

Read CH2 burst parameters when the STATE is OFF.

*C2:BTWV?*

Return:

*C2:BTWV STATE,OFF*

Note: The table below shows the availability of some command parameters in each SDG series.

| Parameter /command | SDG800 | SDG1000 | SDG2000X | SDG5000 | SDG1000X | SDG6000X/X-E | SDG7000A |
|---|---|---|---|---|---|---|---|
| <channel> | no | yes | yes | yes | yes | yes | yes |
| TRMD | no | yes | yes | yes | yes | yes | yes |
| EDGE | no | yes | yes | yes | yes | yes | yes |
| CARR, DLY | yes | yes | yes | yes | yes | yes | yes |
| CARR, RISE | yes | no | yes | yes | yes | yes | yes |
| CARR, FALL | yes | no | yes | yes | yes | yes | yes |

## 3.8 Parameter Copy Command

**DESCRIPTION** This command copies parameters from one channel to another.

**COMMAND SYNTAX** ParaCoPy <destination_channel>,<src_channel>

< destination_channel>:= {C1, C2}.

<src_channel>:= {C1, C2}.

Note: the parameters C1 and C2 must be set to the device together.

**EXAMPLE** Copy parameters from CH1 to CH2.

*PACP C2,C1*

Note: The table below shows the availability of the command in each SDG series.

| Parameter /command | SDG800 | SDG1000 | SDG2000X | SDG5000 | SDG1000X | SDG6000X/X-E | SDG7000A |
|---|---|---|---|---|---|---|---|
| PACP | no | yes | yes | yes | yes | yes | yes |

## 3.9 Arbitrary Wave Command

### 3.9.1 Arbitrary Wave Switch Command

**DESCRIPTION**

This command sets and gets the arbitrary waveform type.

Note: The index number in the command syntax and the response format of the query omits the character and directly uses the value to represent the index number.

**COMMAND SYNTAX**

Format1: <channel>:ArbWaVe INDEX,<index>

Format2: <channel>:ArbWaVe NAME,<name>

Format3: <channel>:ArbWaVe NAME,<path>

<channel>:= {C1, C2}.

<index>: the index of the arbitrary waveform from the table below.

<name>: the name of the arbitrary waveform from the table below.

<path>: the path of waveform

**QUERY SYNTAX**

<channel>:ARbWaVe?

<channel>:= {C1, C2}.

**RESPONSE FORMAT**

<channel>:ARWV INDEX,<index>,NAME,<name>

**EXAMPLE**

Set CH1 current waveform by index 2:

*C1:ARWV INDEX,2*

Read CH1 current waveform:

*C1:ARWV?*

Return:

*C1:ARWV INDEX,2,NAME,StairUp*

Set CH1 current waveform to wave_1 by name.

*C1:ARWV NAME,”wave_1”*

Set the waveform of ch1 through the waveform path:

*C1:ARWV NAME,”Local/wave1.bin”*

*C1:ARWV NAME,"Local/wave2.mat"*
*C1:ARWV NAME,"Local/wave3.csv"*
*C1:ARWV NAME,"net_storage/wave4.bin"*
*C1:ARWV NAME,"U-disk0/wave1.bin"*

**NOTE** The specific path refers to the path in the file manager

**RELATED COMMANDS** STL

| Index | Name | Index | Name | Index | Name | Index | Name |
|---|---|---|---|---|---|---|---|
| 0 | Sine | 51 | AttALT | 102 | LFPulse | 153 | Duty18 |
| 1 | Noise | 52 | RoundHalf | 103 | Tens1 | 154 | Duty20 |
| 2 | StairUp | 53 | RoundsPM | 104 | Tens2 | 155 | Duty22 |
| 3 | StairDn | 54 | BlaseiWave | 105 | Tens3 | 156 | Duty24 |
| 4 | Stairud | 55 | DampedOsc | 106 | Airy | 157 | Duty26 |
| 5 | Ppulse | 56 | SwingOsc | 107 | Besselj | 158 | Duty28 |
| 6 | Npulse | 57 | Discharge | 108 | Bessely | 159 | Duty30 |
| 7 | Trapezia | 58 | Pahcur | 109 | Dirichlet | 160 | Duty32 |
| 8 | Upramp | 59 | Combin | 110 | Erf | 161 | Duty34 |
| 9 | Dnramp | 60 | SCR | 111 | Erfc | 162 | Duty36 |
| 10 | ExpFal | 61 | Butterworth | 112 | ErfcInv | 163 | Duty38 |
| 11 | ExpRise | 62 | Chebyshev1 | 113 | ErfInv | 164 | Duty40 |
| 12 | Logfall | 63 | Chebyshev2 | 114 | Laguerre | 165 | Duty42 |
| 13 | Logrise | 64 | TV | 115 | Legend | 166 | Duty44 |
| 14 | Sqrt | 65 | Voice | 116 | Versiera | 167 | Duty46 |
| 15 | Root3 | 66 | Surge | 117 | Weibull | 168 | Duty48 |
| 16 | X^2 | 67 | Radar | 118 | LogNormal | 169 | Duty50 |
| 17 | X^3 | 68 | Ripple | 119 | Laplace | 170 | Duty52 |
| 18 | Sinc | 69 | Gamma | 120 | Maxwell | 171 | Duty54 |
| 19 | Gaussian | 70 | StepResp | 121 | Rayleigh | 172 | Duty56 |
| 20 | Dlorentz | 71 | BandLimited | 122 | Cauchy | 173 | Duty58 |
| 21 | Haversine | 72 | CPulse | 123 | CosH | 174 | Duty60 |
| 22 | Lorentz | 73 | CWPulse | 124 | CosInt | 175 | Duty62 |
| 23 | Gauspuls | 74 | GateVibr | 125 | CotH | 176 | Duty64 |
| 24 | Gmonopuls | 75 | LFMPulse | 126 | CscH | 177 | Duty66 |
| 25 | Tripuls | 76 | MCNoise | 127 | SecH | 178 | Duty68 |
| 26 | Cardiac | 77 | AM | 128 | SinH | 179 | Duty70 |
| 27 | Quake | 78 | FM | 129 | SinInt | 180 | Duty72 |

| Index | Name | Index | Name | Index | Name | Index | Name |
|---|---|---|---|---|---|---|---|
| 28 | Chirp | 79 | PFM | 130 | TanH | 181 | Duty74 |
| 29 | Twotone | 80 | PM | 131 | ACosH | 182 | Duty76 |
| 30 | SNR | 81 | PWM | 132 | ASecH | 183 | Duty78 |
| 31 | Hamming | 82 | EOG | 133 | ASinH | 184 | Duty80 |
| 32 | Hanning | 83 | EEG | 134 | ATanH | 185 | Duty82 |
| 33 | Kaiser | 84 | EMG | 135 | ACsch | 186 | Duty84 |
| 34 | Blackman | 85 | Pulseilogram | 136 | ACoth | 187 | Duty86 |
| 35 | Gausswin | 86 | ResSpeed | 137 | Bartlett | 188 | Duty88 |
| 36 | Triang | 87 | ECG1 | 138 | BohmanWin | 189 | Duty90 |
| 37 | BlackmanH | 88 | ECG2 | 139 | ChebWin | 190 | Duty92 |
| 38 | Bartlett | 89 | ECG3 | 140 | FlattopWin | 191 | Duty94 |
| 39 | Tan | 90 | ECG4 | 141 | ParzenWin | 192 | Duty96 |
| 40 | Cot | 91 | ECG5 | 142 | TaylorWin | 193 | Duty98 |
| 41 | Sec | 92 | ECG6 | 143 | TukeyWin | 194 | Duty99 |
| 42 | Csc | 93 | ECG7 | 144 | Duty01 | 195 | demo1_375 |
| 43 | Asin | 94 | ECG8 | 145 | Duty02 | 196 | demo1_16k |
| 44 | Acos | 95 | ECG9 | 146 | Duty04 | 197 | demo2_3k |
| 45 | Atan | 96 | ECG10 | 147 | Duty06 | 198 | demo2_16k |
| 46 | Acot | 97 | ECG11 | 148 | Duty08 | | |
| 47 | Square | 98 | ECG12 | 149 | Duty10 | | |
| 48 | SineTra | 99 | ECG13 | 150 | Duty12 | | |
| 49 | SineVer | 100 | ECG14 | 151 | Duty14 | | |
| 50 | AmpALT | 101 | ECG15 | 152 | Duty16 | | |

Note: The below table shows the index of built-in waveforms of different models

| | SDG800 | SDG1000 | SDG2000X | SDG5000 | SDG1000X | SDG6000X/X-E | SDG7000A |
|---|---|---|---|---|---|---|---|
| INDEX | 0~46 | 2~198 | 2~198 | 2~198 | 2~198 | 2~198 | 0~198 |

Note: The table below shows the availability of some command parameters in each SDG series.

| Parameter /command | SDG800 | SDG1000 | SDG2000X | SDG5000 | SDG1000X | SDG6000X/X-E | SDG7000A |
|---|---|---|---|---|---|---|---|
| <channel> | no | yes | yes | yes | yes | yes | yes |
| INDEX | yes | yes | Yes (only built-in wave) | yes | Yes (only built-in wave) | Yes (only built-in wave) | Yes (only built-in wave) |
| NAME | yes | yes | Yes (only user- | yes | Yes (only user- | Yes (only user- | Yes (built-in or |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | defined wave) | | defined wave) | defined wave) Format 2 | user-defined wave) Format2 Format3 |

## 3.9.2 Arbitrary Wave Marker Setting Command(Only SDG7000A)

| | |
|---|---|
| **DESCRIPTION** | This command sets the switch of the markers. |
| **COMMAND SYNTAX** | <channel>:MSWitch <state><br><channel>:= {C1, C2}.<br><state>:= {ON, OFF}. |
| **QUERY SYNTAX** | <channel>: MSWitch?<br><channel>:= {C1, C2}. |
| **RESPONSE FORMAT** | <state> |
| **EXAMPLE** | Turn on the marker switch of CH1.<br>*C1:MSWItch ON*<br>Read the state of CH1 marker switch.<br>*C1:MSWItch?*<br>Return:<br>*ON* |
| **DESCRIPTION** | This command sets the position of the markers. |
| **COMMAND SYNTAX** | <channel>:MPOS<state><br><channel>:= {C1, C2}.<br><value>:= {integer number at the range of the length of the wave.}. |
| **QUERY SYNTAX** | <channel>: MPOS?<br><channel>:= {C1, C2}. |
| **RESPONSE FORMAT** | <value> |
| **EXAMPLE** | Set the position of marker of CH1 to 100. |

*C1:MPOS 100*

Read the position of the marker of CH1.

*C1:MPOS?*

Return:

*100*

## 3.10 Sync Command

**DESCRIPTION** This command sets the synchronization signal.

**COMMAND SYNTAX**

<channel>:SYNC <state>

<channel>:= {C1, C2}.

<state>:= {ON, OFF}.

SYNC TYPE, <TYPE>

<TYPE>:={CH1,CH2,MOD_CH1,MOD_CH2}.

**QUERY SYNTAX**

<channel>:SYNC?

<channel>:= {C1, C2}.

**RESPONSE FORMAT** <channel>:SYNC <state>

**EXAMPLE**

Turn on sync output and set the source as modulating signal of CH1:

*C1:SYNC ON,TYPE,MOD_CH1*

Read the state of CH1 sync.

*C1:SYNC?*

Return:

*C1:SYNC ON,TYPE,MOD_CH1*

Note: The table below shows the availability of the command in each SDG series.

| **Parameter /command** | **SDG800** | **SDG1000** | **SDG2000X** | **SDG5000** | **SDG1000X** | **SDG6000X/X-E** | **SDG7000A** |
|---|---|---|---|---|---|---|---|
| SYNC | no | yes | yes | yes | yes | yes | yes |

## 3.11 Equal Phase Command

| | |
|---|---|
| **DESCRIPTION** | This command is used to set the phase synchronization of two channels |
| **COMMAND SYNTAX** | EQPHASE |
| **RESPONSE FORMAT** | EQPHASE <state> |
| **EXAMPLE** | *EQPHASE* |

Note: The table below shows the availability of the command in each SDG series.

| Parameter /command | SDG800 | SDG1000 | SDG2000X | SDG5000 | SDG1000X | SDG6000X/X-E | SDG7000A |
|---|---|---|---|---|---|---|---|
| EQPHASE | no | yes | yes | yes | yes | yes | yes |

## 3.12 Number Format Command

**DESCRIPTION** This command sets or gets the number format.

**COMMAND SYNTAX** NumBer_ForMat PNT,<pnt>,
NumBer_ForMa SEPT,<sept>

<pnt>:= {Dot, Comma}. The point format.
<sept>:= {Space, Off, On}. The separator format.

**QUERY SYNTAX** NBFM?

**RESPONSE FORMAT** NBFM PNT,<pnt>, SEPT,<sept>

**EXAMPLE** Set point format to DOT:
*NBFM PNT,DOT*
Set separator format to ON:
*NBFM SEPT,ON*

Read the number format:
*NBFM?*
Return:

*NBFM PNT, DOT, SEPT, ON*

Note: The table below shows the availability of the command in each SDG series.

| **Parameter /command** | **SDG800** | **SDG1000** | **SDG2000X** | **SDG5000** | **SDG1000X** | **SDG6000X/X-E** | **SDG7000A** |
|---|---|---|---|---|---|---|---|
| NBFM | yes | yes | yes | yes | yes | yes | no |

## 3.13 Language Command

**DESCRIPTION** This command sets or gets the system language.

**COMMAND SYNTAX** LAnGuaGe <language>

<language>:= {EN,CH,RU}, where EN is English, CH is Chinese Simplified, and RU is Russian.

**QUERY SYNTAX** LAnGuaGe?

**RESPONSE FORMAT** LAGG <language>

**EXAMPLE** Set language to English:

*LAGG EN*

Read language

*LAGG?*

Return:

*LAGG EN*

Note: The table below shows the availability of some command parameters in each SDG series.

| **Parameter /command** | **SDG800** | **SDG1000** | **SDG2000X** | **SDG5000** | **SDG1000X** | **SDG6000X/X-E** | **SDG7000A** |
|---|---|---|---|---|---|---|---|
| RU | no | yes | no | no | no | no | no |

## 3.14 Configuration Command

**DESCRIPTION** This command sets or gets the power-on system setting.

| | |
|---|---|
| **COMMAND SYNTAX** | Format1: Sys_CFG <mode><br><mode>:= {DEFAULT, LAST,USER}<br><br>Format2: Sys_CFG<config><filepath><br><config>:={USER, PRESET}<br><filepath>:= { The path of the configuration file stored by the user (local, network storage, USB flash disk), including the file name and suffix } |
| **QUERY SYNTAX** | Sys_CFG? |
| **RESPONSE FORMAT** | SCFG <mode><br>Sys_CFG<config><filepath> |
| **EXAMPLE** | Set the power-on system setting to LAST:<br>*SCFG LAST*<br><br>Set boot recovery file:<br>*SCFG USER,"net_storage/config/state.xml"*<br>*or: SCFG USER,"U-disk0/config/state.xml"*<br>or: SCFG USER,*SCFG USER,"Local/state.xml"*<br><br>Set recovery file:<br>*SCFG PRESET,"net_storage/config/state.xml"*<br>*Or: SCFG PRESET,"U-disk0/config/state.xml"*<br>*Or: SCFG PRESET,"Local/state.xml"* |

Note 1: the path must be included in English in double quotation marks, and the suffix ".xml" must be added. Please refer to the file manager for specific available paths.

Note 2: Format 2 is only supported by SDG7000A

## 3.15 Date And Time Command

| | |
|---|---|
| **DESCRIPTION** | This command sets the date and time of the device |
| **COMMAND SYNTAX** | SYST:DATE < Date ><br>< Date >:= {Date to set, Format: yyyy/mm/dd }. |

SYST:TIME
< Time >:= { Time to set, Format: hh/mm/ss }.

**QUERY SYNTAX** SYST:DATE?
SYST:TIME?

**EXAMPLE 1** Set the date to 2021/01/10
*SYST:DATE 20210110*

**EXAMPLE 2** Set the time to 10:06:32
*SYST:TIME 100632*

Note: the following table shows the availability of some commands in different SDG series

| Parameter /command | SDG800 | SDG1000 | SDG2000X | SDG5000 | SDG1000X | SDG6000X/X-E | SDG7000A |
|---|---|---|---|---|---|---|---|
| MODE | no | no | no | no | no | no | yes |

## 3.16 Power On Command

**DESCRIPTION** This command is used to set direct power on or key on.

**COMMAND SYNTAX** POWER:ON:MODE<value>
< value >:= {1, 2}.

Mode 1:Press the power on button to power on
Mode 2:Turn on the power and start it directly

**QUERY SYNTAX** POWER:ON:MODE?

**RESPONSE FORMAT** POWER:ON:MODE< value >

**EXAMPLE** Turn on the power and start it directly:
*POWER:ON:MODE 2*

Note: the following table shows the availability of some commands in different SDG series

| Parameter /command | SDG800 | SDG1000 | SDG2000X | SDG5000 | SDG1000X | SDG6000X/X-E | SDG7000A |
|---|---|---|---|---|---|---|---|
| MODE | no | no | no | no | no | no | yes |

## 3.17 Key Command

| | |
|---|---|
| **DESCRIPTION** | This command is used to turn on or off the front panel keys. |
| **COMMAND SYNTAX** | KEY<state><br>< state >:= {ON, OFF}. |
| **QUERY SYNTAX** | KEY? |
| **RESPONSE FORMAT** | KEY< state > |
| **EXAMPLE** | Turn on the front panel key:<br>*KEY ON* |

Note: Only supported on the SDG7000A series.

## 3.18 Buzzer Command

| | |
|---|---|
| **DESCRIPTION** | This command turns on or off the buzzer. |
| **COMMAND SYNTAX** | BUZZer <state><br><state>:= {ON, OFF}. |
| **QUERY SYNTAX** | BUZZer? |
| **RESPONSE FORMAT** | BUZZ <state> |
| **EXAMPLE** | Turn on the buzzer:<br>*BUZZ ON* |

## 3.19 Channel Trigger Source Setting Command

**DESCRIPTION** This command is used to set whether to trigger both channels simultaneously when manually triggered.

**COMMAND SYNTAX** COUP TRDUCH, < parameter >
< parameter >:= {ON, OFF}.

**QUERY SYNTAX** COUP?

**RESPONSE FORMAT** COUP\sTRACE,OFF,FCOUP,OFF,PCOUP,OFF,ACOUP,OFF,TRDUCH,< parameter >

**EXAMPLE** When setting manual triggering, both channels are triggered simultaneously:
*COUP TRDUCH,ON*
Read the current channel trigger setting status:
*COUP?*
Return:
*COUP\sTRACE,OFF,FCOUP,OFF,PCOUP,OFF,ACOUP,OFF,TRDUCH,ON\n*

Note: The table below shows the availability of the command in each SDG series.

| Parameter /command | SDG800 | SDG1000 | SDG2000X | SDG5000 | SDG1000X | SDG6000X/X-E | SDG7000A |
|---|---|---|---|---|---|---|---|
| COUP TRDUCH | no | No | yes | no | no | no | no |

## 3.20 Screen Saver Command

**DESCRIPTION** This command sets or gets the screen saver time. The unit is minutes “m”.

**COMMAND SYNTAX** SCreen_SaVe <parameter>
<parameter>:= {OFF, 1, 5, 15, 30, 60, 120, 300}.

**QUERY SYNTAX** SCreen_SaVe?

**RESPONSE FORMAT** SCSV <parameter>

**EXAMPLE**

Set screen saver time to 5 minutes:

*SCSV 5*

Read the current screen saver time:

*SCreen_SaVe?*

Return:

*SCSV 5MIN*

Note: The table below shows the availability of the command in each SDG series.

| Parameter /command | SDG800 | SDG1000 | SDG2000X | SDG5000 | SDG1000X | SDG6000X/X-E | SDG7000A |
|---|---|---|---|---|---|---|---|
| SCSV | yes | yes | yes | yes | yes | yes | yes |

## 3.21 Clock Source Command

**DESCRIPTION**

This command sets or gets the clock source.

**COMMAND SYNTAX**

ROSCillator <src>

<src>:= {INT, EXT}

ROSCillator 10MOUT,<state>

<state>:={ ON,OFF }

**QUERY SYNTAX**

ROSC?

**RESPONSE FORMAT**

ROSC <src>,10MOUT,<state>

**EXAMPLE**

Set internal time base as the source:

*ROSC INT*

Enable 10MHz output:

*ROSC 10MOUT,ON*

Note: The table below shows the availability of the command in each SDG series.

| Parameter | SDG800 | SDG1000 | SDG2000X | SDG5000 | SDG1000X | SDG6000X/X-E | SDG7000A |
|---|---|---|---|---|---|---|---|

| /command | | | | | | | |
|---|---|---|---|---|---|---|---|
| ROSC | no | yes | yes | yes | yes | yes | yes |

## 3.22 Frequency Counter Command

| | |
|---|---|
| DESCRIPTION | This command sets or gets the frequency counter parameters. |
| COMMAND SYNTAX | FreqCouNTer <parameter>,<value><br><parameter>:= {a parameter from the table below}.<br><value>:= {value of the corresponding parameter}. |

| Parameters | Value | Description |
|---|---|---|
| STATE | <state> | :={ON, OFF}<br>State of frequency counter. |
| FRQ | <frequency> | Measured frequency. The unit is Hertz “Hz”.<br>Can’t be set. |
| PW | <pos_width> | Measured positive width. The unit is seconds “s”.<br>Can’t be set. |
| NW | <neg_width> | Measured negative width. The unit is seconds “s”.<br>Can’t be set. |
| DUTY | <duty> | Measured duty cycle. The unit is "%".<br>Can’t be set. |
| FRQDEV | <freq_dev> | Measured frequency deviation. The unit is "ppm".<br>Can’t be set. |
| REFQ | <ref_freq> | Expected frequency, for calculating the frequency deviation. The unit is Hertz “Hz”. |
| TRG | <triglev> | Trigger level. The range of valid values depends on the model. The unit is volts “V”. |
| MODE | <mode> | :={AC, DC}<br>Coupling mode. |
| HFR | <HFR> | :={ON, OFF}<br>State of High Frequency Rejection. |

| | |
|---|---|
| **QUERY SYNTAX** | FreqCouNTer? |
| **RESPONSE FORMAT** | FCNT <parameter> |

<parameter>:={All parameters of the frequency counter}

**EXAMPLE**

Turn frequency counter on:

*FCNT STATE,ON*

Set reference frequency to 1000 Hz:

*FCNT REFQ,1000*

Query frequency counter information:

*FCNT?*

Return:

*FCNT STATE,ON,FRQ,10000000HZ,DUTY,59.8568,REFQ, 1e+07HZ,TRG,0V,PW,5.98568e-08S,NW,4.01432e-08S,FRQDEV,0ppm,MODE,AC,HFR,OFF*

Note: The table below shows the availability of the command in each SDG series.

| Parameter /command | SDG800 | SDG1000 | SDG2000X | SDG5000 | SDG1000X | SDG6000X/X-E | SDG7000A |
|---|---|---|---|---|---|---|---|
| FCNT | no | yes | yes | yes | yes | yes | no |

## 3.23 Counter Command(Only SDG7000A)

**COMMAND SYNTAX** :SENSe:COUNTer

**DESCRIPTION** It is used to set and obtain various parameters of the counter. See 3.23.1 ~3.23.30 for detailed commands

This command sets the counter configuration

**COMMAND SYNTAX** SENSe:COUNTer:CONFig: < parameter > <value>

< parameter >:={ Parameters in the following table }

< value >:={ Values of related parameters }

| parameter | value | description |
|---|---|---|
| STATe | <state> | ={ OFF, ON}or{0, 1},turn on or off the counter |
| MODE | <state> | :={ FREQuency,TOTALizer }<br>Frequency meter mode or counter mode |

| | | |
|---|---|---|
| COUPLing | <mode> | :={AC, DC} Coupling mode |
| HFREJect | <HFR> | :={OFF,ON}or{0,1}High frequency suppression state |
| TLEVel | <triglev><unit> | Trigger level. The range of valid values depends on the model. The unit is volts “V”.<br>< unit > := {V, mV, uV} |
| SEXIT | <mode> | ={ OFF, ON}or{0, 1} |
| PAUSe | <state> | ={ OFF, ON},Pause switch |

**QUERY SYNTAX** :SENSe:COUNTer:CONFig: < parameter >?

**RESPONSE FORMAT** <mode>

**EXAMPLE** Set the counter coupling mode to AC mode:
*:SENSe:COUNTer:CONFig:COUPLing AC*

**DESCRIPTION** This command sets the measurement parameters in the frequency meter mode

**COMMAND SYNTAX** SENSe:COUNTer:FREQuency: < parameter > <value>
< parameter >:={ Parameters in the following table }
:< value >:={ Values of related parameters }.

| **parameter** | **value** | **description** |
|---|---|---|
| MEASure | < type > | <type>:={ FREQ, PERIOD, DUTY_CYCLe } |
| RFREQuency | <frequency><unit> | <unit> := {Hz, MHz, GHz}.The default unit is Hz |

**QUERY SYNTAX** :SENSe:COUNTer:FREQuency:MEASure[:type]?
RESPONSE FORMAT <mode>

**EXAMPLE** Set the measurement type in frequency meter mode as period:
*:SENSe:COUNTer:FREQuency:MEASure PERIOD*

**DESCRIPTION** This command is used to query the measurement results in the frequency meter mode

**COMMAND SYNTAX** :SENSe:COUNTer:FREQuency:

< Measurement type >:< parameter ><[type]>?>

< Measurement type >:={ default, PERiod, DUTY } Frequency, cycle and duty cycle

< parameter >:={ default, SNUMBer , MEAN, MAX, MIN, SDEViation } Real time value of measurement type, sampling times of measurement type, average value of measurement type, maximum value of measurement type, minimum value of measurement type and standard deviation of measurement type

< type >:={ FDEViation }. Available only when the measurement type is frequency mode

**QUERY SYNTAX** :SENSe:COUNTer:FREQuency:SNUMBer?

RESPONSE FORMAT <mode>

:SENSe:COUNTer:CONFig:COUPLing[:MODE] AC

**DESCRIPTION** This command is used to set the measurement parameters in counter mode

**COMMAND SYNTAX** :SENSe:COUNTer: TOTalizer: < parameter > < value >

< parameter >:={ Parameters in the following table }

< value >:={ Values of related parameters }.

| parameter | value | describe |
|---|---|---|
| GATE:STATe | < type > | <state>:={ OFF, ON} or {0, 1} |
| EDGE | <edge> | < edge >:={ RISE, FALL} |
| GATE:MODE | <mode> | <mode>:={LEVEL, AFTER_EDGE} |
| GOLarity:GATE:POLarity | < polarity > | < polarity >:={ NEGative, POSitive} |
| GOLarity:GATE:EDGE | < edge > | < edge >:={ RISE, FALL} |

## 3.23.1 :SENSe:COUNTer:CLEar

**DESCRIPTION** This command is used to clear the measurement results

**QUERY SYNTAX** :SENSe:COUNTer:CLEar

| | |
|---|---|
| **RESPONSE FORMAT** | |
| **EXAMPLE** | Clear the measurement results:<br>*:SENSe:COUNTer:CLEar* |

### 3.23.2 :SENSe:COUNTer:FREQuency:MEASure[:type]

| | |
|---|---|
| **DESCRIPTION** | This command sets the measurement type in the frequency meter mode |
| **COMMAND SYNTAX** | :SENSe:COUNTer:FREQuency:MEASure[:type] <type><br><type>:={ FREQ, PERIOD, DUTY_CYCLe } |
| **QUERY SYNTAX** | :SENSe:COUNTer:FREQuency:MEASure? |
| **RESPONSE FORMAT** | <type> |
| **EXAMPLE** | Set the measurement type in frequency meter mode as period:<br>*:SENSe:COUNTer:FREQuency:MEASure PERIOD* |
| **NOTE** | This command is only valid in frequency meter mode |

### 3.23.3 :SENSe:COUNTer:FREQuency:RFREQuency

| | |
|---|---|
| **DESCRIPTION** | This command sets the reference frequency in the frequency meter mode |
| **COMMAND SYNTAX** | :SENSe:COUNTer:FREQuency:<br>RFREQuency < frequency ><unit><br>< frequency >:= the reference frequency of frequency meter,Please refer to the data manual for the valid range of this parameter.<br><br>< unit > := {Hz, MHz, GHz}. The default unit is Hz |
| **QUERY SYNTAX** | :SENSe:COUNTer:FREQuency:RFREQuency? |
| **RESPONSE FORMAT** | < frequency > (Expressed in HZ) |

| | |
|---|---|
| **EXAMPLE** | Set the reference frequency in the frequency meter mode to 1MHz:<br>*:SENSe:COUNTer:FREQuency:RFREQuency 1MHZ* |
| **NOTE** | This command is only valid when the measurement type is frequency in counter mode |

### 3.23.4 :SENSe:COUNTer:FREQuency?

| | |
|---|---|
| **DESCRIPTION** | This command is used to query the frequency results measured in the frequency meter mode |
| **QUERY SYNTAX** | :SENSe:COUNTer:FREQuency? |
| **RESPONSE FORMAT** | < frequency > (Expressed in Hz) |
| **EXAMPLE** | Query the frequency results measured in the frequency meter mode:<br>*:SENSe:COUNTer:FREQuency?*<br>Return value<br>*9999996.75781744* |
| **NOTE** | This command is only valid in frequency meter mode |

### 3.23.5 :SENSe:COUNTer:FREQuency:SNUMBer?

| | |
|---|---|
| **DESCRIPTION** | This command is used to query the results of sampling times in frequency meter mode |
| **QUERY SYNTAX** | :SENSe:COUNTer:FREQuency:SNUMBer? |
| **RESPONSE FORMAT** | < sampling times > |
| **EXAMPLE** | Query the results of sampling times in frequency meter mode:<br>*:SENSe:COUNTer:FREQuency:SNUMBer?*<br>Return value:<br>*2294* |

| | |
|---|---|
| **NOTE** | This command is only valid in frequency meter mode |

### 3.23.6 :SENSe:COUNTer:FREQuency:FDEViation?

| | |
|---|---|
| **DESCRIPTION** | This command is used to query the frequency deviation results measured in the frequency meter mode |
| **QUERY SYNTAX** | :SENSe:COUNTer:FREQuency:FDEViation? |
| **RESPONSE FORMAT** | < frequency deviation > (Expressed in ppm) |
| **EXAMPLE** | Query the frequency deviation results measured in the frequency meter mode:<br>*:SENSe:COUNTer:FREQuency:FDEViation?*<br>Return value:<br>-0.324020794406533 |
| **NOTE** | This command is only valid in frequency meter mode |

### 3.23.7 :SENSe:COUNTer:FREQuency:MEAN?

| | |
|---|---|
| **DESCRIPTION** | This command is used to query the frequency average value measured in the frequency meter mode |
| **QUERY SYNTAX** | :SENSe:COUNTer:FREQuency:MEAN? |
| **RESPONSE FORMAT** | < frequency > (Expressed in Hz) |
| **EXAMPLE** | Query the frequency results measured in the frequency meter mode:<br>*:SENSe:COUNTer:FREQuency:MEAN?*<br>Return value:<br>*9999996.79101083* |
| **NOTE** | This command is only valid in frequency meter mode |

### 3.23.8 :SENSe:COUNTer:FREQuency:MEAN:FDEViation?

| | |
|---|---|
| **DESCRIPTION** | This command is used to query the deviation result of the frequency average value measured in the frequency meter mode |
| **QUERY SYNTAX** | :SENSe:COUNTer:FREQuency:MEAN:FDEViation? |
| **RESPONSE FORMAT** | < frequency deviation > (Expressed in ppm) |
| **EXAMPLE** | Query the deviation results of the average frequency measured in the frequency meter mode:<br>*:SENSe:COUNTer:FREQuency:MEAN:FDEViation?*<br>Return value:<br>*-0.322511510334804* |
| **NOTE** | This command is only valid in frequency meter mode |

### 3.23.9 :SENSe:COUNTer:FREQuency:MAX?

| | |
|---|---|
| **DESCRIPTION** | This command is used to query the maximum frequency measured in the frequency meter mode |
| **QUERY SYNTAX** | :SENSe:COUNTer:FREQuency:MAX? |
| **RESPONSE FORMAT** | < frequency > (Expressed in Hz) |
| **EXAMPLE** | Query the maximum frequency measured in the frequency meter mode:<br>*:SENSe:COUNTer:FREQuency:MAX?*<br>Return value:<br>*9999996.8775536* |
| **NOTE** | This command is only valid in frequency meter mode |

## 3.23.10 :SENSe:COUNTer:FREQuency:MAX:FDEViation?

| | |
|---|---|
| **DESCRIPTION** | This command is used to query the maximum deviation result of frequency measured in frequency meter mode |
| **QUERY SYNTAX** | :SENSe:COUNTer:FREQuency:MAX:FDEViation? |
| **RESPONSE FORMAT** | < frequency deviation > (Expressed in ppm) |
| **EXAMPLE** | Query the maximum deviation result of frequency measured in frequency meter mode: *:SENSe:COUNTer:FREQuency:MAX:FDEViation?* Return value: *-0.312244639918208* |
| **NOTE** | This command is only valid in frequency meter mode |

## 3.23.11 :SENSe:COUNTer:FREQuency:MIN?

| | |
|---|---|
| **DESCRIPTION** | This command is used to query the results of the minimum frequency measured in the frequency meter mode |
| **QUERY SYNTAX** | :SENSe:COUNTer:FREQuency:MIN? |
| **RESPONSE FORMAT** | < frequency > (Expressed in Hz) |
| **EXAMPLE** | Query the result of the minimum frequency measured in the frequency meter mode: *:SENSe:COUNTer:FREQuency:MIN?* Return value: *9999996.67704201* |
| **NOTE** | This command is only valid in frequency meter mode |

## 3.23.12 :SENSe:COUNTer:FREQuency:MIN:FDEViation?

| | |
|---|---|
| **DESCRIPTION** | This command is used to query the minimum deviation result of frequency measured in frequency meter mode |
| **QUERY SYNTAX** | :SENSe:COUNTer:FREQuency:MIN:FDEViation? |
| **RESPONSE FORMAT** | < frequency deviation > (Expressed in ppm) |
| **EXAMPLE** | Query the minimum deviation result of frequency measured in frequency meter mode: *:SENSe:COUNTer:FREQuency:MIN:FDEViation?* Return value: *-0.332295799069107* |
| **NOTE** | This command is only valid in frequency meter mode |

## 3.23.13 :SENSe:COUNTer:FREQuency:SDEViation?

| | |
|---|---|
| **DESCRIPTION** | This command is used to query the frequency standard deviation results measured in the frequency meter mode |
| **QUERY SYNTAX** | :SENSe:COUNTer:FREQuency:SDEViation? |
| **RESPONSE FORMAT** | < frequency > (Expressed in Hz) |
| **EXAMPLE** | Query the frequency standard deviation results measured in the frequency meter mode: *:SENSe:COUNTer:FREQuency:SDEViation?* Return value: *0.395284707521047* |
| **NOTE** | This command is only valid in frequency meter mode |

### 3.23.14 :SENSe:COUNTe:FREQuency:SDEViation:FDEViation?

| | |
|---|---|
| **DESCRIPTION** | This command is used to query the deviation results of the frequency standard deviation measured in the frequency meter mode |
| **QUERY SYNTAX** | :SENSe:COUNTe:FREQuency:SDEViation:FDEViation? |
| **RESPONSE FORMAT** | < frequency deviation > (Expressed in ppm) |
| **EXAMPLE** | Query the deviation result of frequency standard deviation measured in frequency meter mode:<br>*:SENSe:COUNTer:FREQuency:SDEViation:FDEViation?*<br>Return value:<br>*-0.332295799069107* |
| **NOTE** | This command is only valid in frequency meter mode |

### 3.23.15 :SENSe:COUNTer:FREQuency:PERiod?

| | |
|---|---|
| **DESCRIPTION** | This command is used to query the period results measured in the frequency meter mode |
| **QUERY SYNTAX** | :SENSe:COUNTer:FREQuency:PERiod? |
| **RESPONSE FORMAT** | < period > (Expressed in s) |
| **EXAMPLE** | Query the period results measured in the frequency meter mode:<br>*:SENSe:COUNTer:FREQuency:PERiod?*<br>Return value:<br>*1.00000032802047e-07* |
| **NOTE** | This command is only valid in frequency meter mode |

## 3.23.16 :SENSe:COUNTer:FREQuency:PERiod:MEAN?

| | |
|---|---|
| **DESCRIPTION** | This command is used to query the periodic average value measured in the frequency meter mode |
| **QUERY SYNTAX** | :SENSe:COUNTer:FREQuency:PERiod:MEAN? |
| **RESPONSE FORMAT** | < period > (Expressed in s) |
| **EXAMPLE** | Query the period average value measured in the frequency meter mode:<br>*:SENSe:COUNTer:FREQuency:PERiod:MEAN?*<br>Return value:<br>*1.00000032802047e-07* |
| **NOTE** | This command is only valid in frequency meter mode |

## 3.23.17 :SENSe:COUNTer:FREQuency:PERiod:MAX?

| | |
|---|---|
| **DESCRIPTION** | This command is used to query the average value of the period measured in the frequency meter mode |
| **QUERY SYNTAX** | :SENSe:COUNTer:FREQuency:PERiod:MAX? |
| **RESPONSE FORMAT** | < period > (Expressed in s) |
| **EXAMPLE** | Query the maximum value of the period measured in the frequency meter mode:<br>*:SENSe:COUNTer:FREQuency:PERiod:MAX?*<br>Return value:<br>*1.00000033229591e-07* |
| **NOTE** | This command is only valid in frequency meter mode |

### 3.23.18 :SENSe:COUNTer:FREQuency:PERiod:MIN?

| | |
|---|---|
| **DESCRIPTION** | This command is used to query the minimum value of the period measured in the frequency meter mode |
| **QUERY SYNTAX** | :SENSe:COUNTer:FREQuency:PERiod:MIN? |
| **RESPONSE FORMAT** | < period > (Expressed in s) |
| **EXAMPLE** | Query the minimum value of the period measured in the frequency meter mode:<br>*:SENSe:COUNTer:FREQuency:PERiod:MIN?*<br>Return value:<br>*1.000000312244 74e-07* |
| **NOTE** | This command is only valid in frequency meter mode |

### 3.23.19 :SENSe:COUNTer:FREQuency:PERiod:SDEViation?

| | |
|---|---|
| **DESCRIPTION** | This command is used to query the standard deviation results of the period measured in the frequency meter mode |
| **QUERY SYNTAX** | :SENSe:COUNTer:FREQuency:PERiod:SDEViation? |
| **RESPONSE FORMAT** | < period > (Expressed in s) |
| **EXAMPLE** | Query the standard deviation result of the period measured in the frequency meter mode:<br>*:SENSe:COUNTer:FREQuency:PERiod:SDEViation?*<br>Return value:<br>*6.52675178645049e-15* |
| **NOTE** | This command is only valid in frequency meter mode |

### 3.23.20 :SENSe:COUNTer:FREQuency:DUTY?

| | |
|---|---|
| **DESCRIPTION** | This command is used to query the duty cycle results measured in the frequency meter mode |
| **QUERY SYNTAX** | :SENSe:COUNTer:FREQuency:DUTY? |
| **RESPONSE FORMAT** | < duty cycle > (Expressed in %) |
| **EXAMPLE** | Query the duty cycle results measured in frequency meter mode: *:SENSe:COUNTer:FREQuency:DUTY?* Return value: *49.8178520697226* |
| **NOTE** | This command is only valid in frequency meter mode |

### 3.23.21 :SENSe:COUNTer:FREQuency:DUTY:MEAN?

| | |
|---|---|
| **DESCRIPTION** | This command is used to query the average value of the duty cycle measured in the frequency meter mode |
| **QUERY SYNTAX** | :SENSe:COUNTer:FREQuency:DUTY:MEAN? |
| **RESPONSE FORMAT** | < duty cycle > (Expressed in %) |
| **EXAMPLE** | Query the average value of duty cycle measured in frequency meter mode: *:SENSe:COUNTer:FREQuency:DUTY:MEAN?* Return value: *49.8204827053384* |
| **NOTE** | This command is only valid in frequency meter mode |

## 3.23.22 :SENSe:COUNTer:FREQuency:DUTY:MAX?

| | |
|---|---|
| **DESCRIPTION** | This command is used to query the result of the maximum duty cycle measured in the frequency meter mode |
| **QUERY SYNTAX** | :SENSe:COUNTer:FREQuency:DUTY:MAX? |
| **RESPONSE FORMAT** | < duty cycle > (Expressed in %) |
| **EXAMPLE** | Query the maximum value of duty cycle measured in frequency meter mode:<br>*:SENSe:COUNTer:FREQuency:DUTY:MAX?*<br>Return value:<br>*1.000000033229591e-07* |
| **NOTE** | This command is only valid in frequency meter mode |

## 3.23.23 :SENSe:COUNTer:FREQuency:DUTY:MIN?

| | |
|---|---|
| **DESCRIPTION** | This command is used to query the minimum value of duty cycle measured in frequency meter mode |
| **QUERY SYNTAX** | :SENSe:COUNTer:FREQuency:DUTY:MIN? |
| **RESPONSE FORMAT** | < duty cycle > (Expressed in %) |
| **EXAMPLE** | Query the minimum value of duty cycle measured in frequency meter mode:<br>*:SENSe:COUNTer:FREQuency:DUTY:MIN?*<br>Return value:<br>*49.8071405616451* |
| **NOTE** | This command is only valid in frequency meter mode |

### 3.23.24 :SENSe:COUNTer:FREQuency:DUTY:SDEViation?

| | |
|---|---|
| **DESCRIPTION** | This command is used to query the standard deviation results of the duty cycle measured in the frequency meter mode |
| **QUERY SYNTAX** | :SENSe:COUNTer:FREQuency:DUTY:SDEViation? |
| **RESPONSE FORMAT** | < duty cycle > (Expressed in %) |
| **EXAMPLE** | Query the results of periodic standard deviation measured in frequency meter mode: *:SENSe:COUNTer:FREQuency:DUTY:SDEViation?* Return value: *0.00505753455971934* |
| **NOTE** | This command is only valid in frequency meter mode |

### 3.23.25 :SENSe:COUNTer:TOTalizer:GATE:STATe

| | |
|---|---|
| **DESCRIPTION** | This command is used to set the gating state of the totalizer in the totalizer mode |
| **COMMAND SYNTAX** | :SENSe:COUNTer:TOTalizer:GATE:STATe <state> <state>:={ OFF, ON}or{0, 1} |
| **QUERY SYNTAX** | :SENSe:COUNTer:TOTalizer:GATE:STATe? |
| **RESPONSE FORMAT** | <state> |
| **EXAMPLE** | Set the counter gating status in totalizer mode to on: *:SENSe:COUNTer:TOTalizer:GATE:STATe ON* |
| **NOTE** | This command is only valid when the counter mode is totalizer |

### 3.23.26 :SENSe:COUNTer:TOTalizer:EDGE

| | |
|---|---|
| **DESCRIPTION** | This command is used to set the type of totalizer trigger edge in totalizer mode |
| **COMMAND SYNTAX** | :SENSe:COUNTer:TOTalizer:EDGE < edge ><br>< edge >:={ RISE, FALL} |
| **QUERY SYNTAX** | :SENSe:COUNTer:TOTalizer:EDGE? |
| **RESPONSE FORMAT** | < edge > |
| **EXAMPLE** | Set the trigger edge type of totalizer in totalizer mode as rising edge:<br>*:SENSe:COUNTer:TOTalizer:EDGE RISE* |
| **NOTE** | This command is only valid when the counter mode is totalizer |

### 3.23.27 :SENSe:COUNTer:TOTalizer:GATE:MODE

| | |
|---|---|
| **DESCRIPTION** | This command is used to set the totalizer gating mode in totalizer mode |
| **COMMAND SYNTAX** | :SENSe:COUNTer:TOTalizer:GATE:MODE < mode ><br>< mode >:={ LEVEL, AFTER_EDGE} |
| **QUERY SYNTAX** | :SENSe:COUNTer:TOTalizer:GATE:MODE? |
| **RESPONSE FORMAT** | <state> |
| **EXAMPLE** | Set the totalizer gating mode to level mode<br>:SENSe:COUNTer:TOTalizer:GATE:MODE LEVEL |
| **NOTE** | This command is only valid when the counter mode is totalizer |

## 3.23.28 :SENSe:COUNTer:TOTalizer:GOLarity:GATE:POLarity

| | |
|---|---|
| **DESCRIPTION** | This command is used to set the totalizer gating polarity in totalizer mode |
| **COMMAND SYNTAX** | :SENSe:COUNTer:TOTalizer:GATE:POLarity < polarity ><br>< polarity >:={ NEGative, POSitive} |
| **QUERY SYNTAX** | :SENSe:COUNTer:TOTalizer:GATE:POLarity? |
| **RESPONSE FORMAT** | < polarity > |
| **EXAMPLE** | Set the gating polarity of the counter to positive:<br>*:SENSe:COUNTer:TOTalizer:GATE:POLarity POSitive* |
| **NOTE** | This command is only valid when the counter mode is totalizer |

## 3.23.29 :SENSe:COUNTer:TOTalizer:GOLarity:GATE:EDGE

| | |
|---|---|
| **DESCRIPTION** | This command is used to set the totalizer gating edge in totalizer mode |
| **COMMAND SYNTAX** | :SENSe:COUNTer:TOTalizer:GATE:EDGE < edge ><br>< edge >:={ RISE, FALL} |
| **QUERY SYNTAX** | :SENSe:COUNTer:TOTalizer:GATE:EDGE? |
| **RESPONSE FORMAT** | < edge > |
| **EXAMPLE** | Set the gating edge of the totalizer as the rising edge:<br>*:SENSe:COUNTer:TOTalizer:GATE:EDGE RISE* |
| **NOTE** | This command is only valid when the counter mode is totalizer |

## 3.23.30 :SENSe:COUNTer:TOTalizer?

| | |
|---|---|
| **DESCRIPTION** | This command is used to query the cumulative value results measured in totalizer mode |
| **QUERY SYNTAX** | :SENSe:COUNTer:TOTalizer? |
| **RESPONSE FORMAT** | < totalize >(Expressed in hits) |
| **EXAMPLE** | Query cumulative value results measured in totalizer mode<br>*:SENSe:COUNTer:TOTalizer?*<br><br>Return value:<br>*13364879033* |
| **NOTE** | This command is only valid when the counter mode is totalizer |

## 3.24 Invert Command

**DESCRIPTION**

This command sets or gets the polarity of specified channel.

**COMMAND SYNTAX**

```
<channel>:INVerT <state>
<channel>:= {C1, C2}.
<state>:= {ON, OFF}.
```

**QUERY SYNTAX**

```
<channel>:INVerT?
<channel>:={C1, C2}.
```

**RESPONSE FORMAT**

```
<channel>:INVT <state>
```

**EXAMPLE**

Set CH1 polarity to invert

*C1:INVT ON*

Read the polarity of CH1.

*C1:INVT?*

Return:

*C1:INVT ON*

Note: The table below shows the availability of some command parameters in each SDG series.

| Parameter /command | SDG800 | SDG1000 | SDG2000X | SDG5000 | SDG1000X | SDG6000X/X-E | SDG7000A |
|---|---|---|---|---|---|---|---|
| <channel> | no | yes | yes | yes | yes | yes | yes |

## 3.25 Digital Filter Command

**DESCRIPTION** This command is used to set the switching and cut-off frequency of the digital filter.

**COMMAND SYNTAX** <channel>:FILTer< parameter >,<value>

< parameter >:={ Parameters in the following table}.

< value >:={Values of related parameters}.

| parameters | value | description |
|---|---|---|
| STATE | <state> | :={ON, OFF} Digital filter status |
| COFF_FRQ | <cutoff_freq> | :={ Device bandwidth },The unit is Hertz 'Hz", used to set the cut-off frequency of the digital filter |

**QUERY SYNTAX** < channel > FILTer?

< channel >:={C1,C2}.

**RESPONSE FORMAT** < channel >: FILT < parameter >,< value >

**EXAMPLE** Set the digital filter of CH1 on:

*C1:FILT STATE,ON*

Set the cut-off frequency of the digital filter of CH1 to 200 MHz:

*C1:FILT COFF_FRQ,200000000*

Query the digital filter information of CH1:

*C1:FILT?*

Return value:

*STATE,OFF,COFF_FRQ,350000000HZ*

Note: the following table shows the availability of some commands in different SDG series

| Parameters /Commands | SDG800 | SDG1000 | SDG2000X | SDG5000 | SDG1000X | SDG6000X/X-E | SDG7000A |
|---|---|---|---|---|---|---|---|
| STATE | no | no | no | no | no | no | yes |
| COFF_FRQ | no | no | no | no | no | no | yes |

## 3.26 Coupling Command

**DESCRIPTION**

This command sets or gets the channel coupling parameters. Only when TRACE is set to OFF, the other coupling parameters can be set.

**COMMAND SYNTAX**

COUPling <parameter>,<value>
<parameter>:= {a parameter from the table below}.
<value>:={value of the corresponding parameter}.

| Parameters | Value | Description |
|---|---|---|
| TRACE | <track_enble> | := {ON, OFF}<br>State of channel tracking. |
| STATE | <state> | := {ON, OFF}<br>State of channel coupling. |
| BSCH | <bsch> | := {CH1,CH2}<br>Base channel. |
| FCOUP | <fcoup> | := {ON, OFF}<br>State of frequency coupling |
| FDEV | <frq_dev> | := frequency deviation between the 2 channels. The unit is Hertz “Hz”. |
| FRAT | <frat> | := frequency ratio between the 2 channels. |
| PCOUP | <pcoup> | :={ON, OFF}<br>State of phase coupling |
| PDEV | <pha_dev> | := phase deviation between the 2 channels. The unit is degree “°”. |
| PRAT | <prat> | := phase ratio between the 2 channels. |
| ACOUP | <acoup> | := {ON, OFF}<br>State of amplitude coupling |
| ARAT | <arat> | := amplitude ratio between the 2 channels. |
| ADEV | <adev> | := amplitude deviation between the 2 channels. The unit is volts, peak-to-peak “Vpp”. |

**QUERY SYNTAX**

COUPling?

**RESPONSE FORMAT**

COUP <parameter>
<parameter>:= { All parameters of coupling}.

**EXAMPLE**

Set coupling state to ON:

*COUP STATE,ON*

Set frequency coupling state to ON:

*COUP FCOUP,ON*

Set frequency deviation to 5 Hz:

*COUP FDEV,5*

Query coupling information.

*COUP?*

Return:

*COUP TRACE,OFF,FCOUP,ON,PCOUP,ON,ACOUP,ON,FDEV,5HZ,PRAT,1,ARAT,2*

Note: The table below shows the availability of the command and some parameters in each SDG series.

| Parameter /command | SDG800 | SDG1000 | SDG2000X | SDG5000 | SDG1000X | SDG6000X/X-E | SDG7000A |
|---|---|---|---|---|---|---|---|
| COUP | no | yes | yes | yes | yes | yes | yes |
| TRACE | no | no | yes | no | yes | yes | yes |
| STATE | no | yes | no | yes | no | no | no |
| BSCH | no | yes | no | yes | no | no | no |
| FCOUP | no | no | yes | no | yes | yes | yes |
| FRAT | no | no | Yes | no | yes | yes | yes |
| PCOUP | no | no | Yes | no | yes | yes | yes |
| PRAT | no | no | Yes | no | yes | yes | yes |
| ACOUP | no | no | Yes | no | yes | yes | yes |
| ARAT | no | no | Yes | no | yes | yes | yes |
| ADEV | no | no | yes | no | yes | yes | yes |

## 3.27 Over-Voltage Protection Command

**DESCRIPTION** This command sets or gets the state of over-voltage protection.

**COMMAND SYNTAX** VOLTPRT <state>

<state>:= {ON, OFF}

**QUERY SYNTAX** VOLTPRT?

**RESPONSE FORMAT** VOLTPRT <state>

Note: The table below shows the availability of the command in each SDG series.

| **Parameter /command** | **SDG800** | **SDG1000** | **SDG2000X** | **SDG5000** | **SDG1000X** | **SDG6000X/X-E** | **SDG7000A** |
|---|---|---|---|---|---|---|---|
| VOLTPRT | no | yes | yes | no | yes | yes | yes |

## 3.28 Over-Current Protection Command

DESCRIPTION This command sets or gets the state of over-current protection.

COMMAND SYNTAX CURRPRT< state >
< state >:= {ON,OFF}

QUERY SYNTAX CURRPRT?

RESPONSE FORMAT CURRPRT < state >

Note: The table below shows the availability of the command in each SDG series.

| **Parameter /command** | **SDG800** | **SDG1000** | **SDG2000X** | **SDG5000** | **SDG1000X** | **SDG6000X/X-E** | **SDG7000A** |
|---|---|---|---|---|---|---|---|
| CURRPRT | no | yes | yes | no | yes | yes | yes |

## 3.29 Overload Status Query Command

**DESCRIPTION** This command is used to obtain the current overload status (overvoltage, overcurrent, none)

**COMMAND SYNTAX** VOLTSTAT?

**QUERY SYNTAX** VOLTSTAT?

Note: The table below shows the availability of the command in each SDG series.

| Parameter /command | SDG800 | SDG1000 | SDG2000X | SDG5000 | SDG1000X | SDG6000X/X-E | SDG7000A |
|---|---|---|---|---|---|---|---|
| VOLTSTAT? | no | yes | yes | no | yes | yes | yes |

## 3.30 Output Skew

| | |
|---|---|
| **DESCRIPTION** | This command is used toset or query output skew. |
| **COMMAND SYNTAX** | <channel>:OUTPut:SKEW <value><br>< channel >={C1,C2}<br>< value >={-0.2 to 0.2},unit “ns” |
| **QUERY SYNTAX** | < channel >:OUTPut:SKEW?<br>< channel >={C1,C2} |
| **RESPONSE FORMAT** | < value ><br>< value >:={ The value of the current channel setting } |
| **EXAMPLE** | Set the output skew of C1 to 0.2ns:<br>*C1:OUTPut:SKEW 0.2e-9*<br>Query the output skew of C1:<br>*C1:OUTPut:SKEW?*<br>Return value:<br>*2e-10* |

Note: The table below shows the availability of the command in each SDG series.

| Parameter /command | SDG800 | SDG1000 | SDG2000X | SDG5000 | SDG1000X | SDG6000X/X-E | SDG7000A |
|---|---|---|---|---|---|---|---|
| | no | no | no | no | no | no | yes |

## 3.31 Store List Command

| | |
|---|---|
| **DESCRIPTION** | This command is used to read the stored waveforms list with indexes and names. If the storage unit is empty, the command will return “EMPTY” |

**QUERY SYNTAX**

Format1: SToreList?

Format2: SToreList? <parameter>

< parameter >=:{ Parameters in the following table }

Format3: SToreList? <USER>,<path>

< path >=:{ Specify the path of network storage or USB flash disk, as shown in the table below }

**RESPONSE FORMAT**

<waveform name>

| parameter | | function |
|---|---|---|
| BUILDIN | < parameter > | Query built-in waveform name and index number |
| USER | < parameter > | Query locally saved waveform name |
| Path (for example only, you can specify any path under the network storage or USB flash disk directory) | | function |
| “net_storage/wave” | < path > | Query the waveform name saved under the network storage path |
| “U-disk0/wave” | < path > | Query the waveform name saved in the path of USB flash disk 0 |
| “U-disk1/wave” | < path > | Query the waveform name saved in the path of USB flash disk 1 |
| “U-disk2/wave” | < path > | Query the waveform name saved in the path of USB flash disk 2 |

**EXAMPLE**

(1) Read all arbitrary wave names saved in the device (excluding network storage and data in USB flash disk):

*STL?*

Return:

*STL M0, sine, M1, noise, M2, stairup, M3, stairdn, M4, stairud, M5, ppulse, M6, npulse, M7, trapezia, M8, upramp, M9, dnramp, M10, exp_fall, M11, exp_rise, M12, logfall, M13, logrise, M14, sqrt, M15, root3, M16, x^2, M17, x^3, M18, sinc, M19, gaussian, M20, dlorentz, M21, haversine, M22, lorentz, M23, gauspuls, M24, gmonopuls, M25, tripuls, M26, cardiac, M27, quake, M28, chirp, M29, twotone, M30, snr, M31, EMPTY, M32, EMPTY, M33, EMPTY, M34, hamming, M35, hanning, M36, kaiser, M37, blackman, M38,* *gaussiwin, M39, triangle, M40, blackmanharris, M41, bartlett, M42, tan, M43, cot, M44, sec, M45, csc, M46, asin, M47, acos, M48, atan, M49, acot, M50, EMPTY, M51, EMPTY, M52, EMPTY, M53, DDROPOUT, M54, FCLK1, M55, FSDA1, M56, EMPTY, M57, EMPTY, M58, EMPTY, M59, EMPTY*

(2) Read the built-in waveform name saved in the device:

*STL? BUILDIN*

Return:

*STL M10, ExpFal, M100, ECG14, M101, ECG15, M102, LFPulse, M103, Tens1, M104, Tens2, M105, Tens3, M106, Airy, M107, Besselj, M108, Bessely, M109, Dirichlet, M11, ExpRise, M110, Erf, M111, Erfc, M112, ErfcInv, M113, ErfInv, M114, Laguerre, M115, Legend, M116, Versiera, M117, Weibull, M118, LogNormal, M119, Laplace, M12, LogFall, M120, Maxwell, M121, Rayleigh, M122, Cauchy, M123, CosH, M124, CosInt, M125, CotH, M126, CscH, M127, SecH, M128, SinH, M129, SinInt, M13, LogRise, M130, TanH, M131, ACosH, M132, ASecH, M133, ASinH, M134, ATanH, M135, ACsch, M136, ACoth, M137, Bartlett, M138, BohmanWin, M139, ChebWin, M14, Sqrt, M140, FlattopWin, M141, ParzenWin, M142, TaylorWin, M143, TukeyWin, M144, SquareDuty01, M145, SquareDuty02, M146, SquareDuty04, M147, SquareDuty06, M148, SquareDuty08, M149, SquareDuty10, M15, Root3, M150, SquareDuty12, M151, SquareDuty14, M152, SquareDuty16, M153, SquareDuty18, M154, SquareDuty20, M155, SquareDuty22, M156, SquareDuty24, M157, SquareDuty26, M158, SquareDuty28, M159, SquareDuty30, M16, X^2, M160, SquareDuty32, M161, SquareDuty34, M162, SquareDuty36, M163, SquareDuty38, M164, SquareDuty40, M165, SquareDuty42, M166, SquareDuty44, M167, SquareDuty46, M168, SquareDuty48, M169, SquareDuty50, M17, X^3, M170, SquareDuty52, M171, SquareDuty54, M172, SquareDuty56, M173, SquareDuty58, M174, SquareDuty60, M175, SquareDuty62, M176, SquareDuty64, M177, SquareDuty66, M178, SquareDuty68, M179, SquareDuty70, M18, Sinc, M180, SquareDuty72, M181, SquareDuty74, M182, SquareDuty76, M183, SquareDuty78, M184, SquareDuty80, M185, SquareDuty82, M186, SquareDuty84, M187, SquareDuty86, M188, SquareDuty88, M189, SquareDuty90, M19, Gaussian, M190, SquareDuty92, M191, SquareDuty94, M192, SquareDuty96, M193, SquareDuty98, M194, SquareDuty99, M195, demo1_375pts, M196, demo1_16kpts, M197, demo2_3kpts, M198, demo2_16kpts, M2, StairUp, M20, Dlorentz, M21, Haversine, M22, Lorentz, M23, Gauspuls, M24, Gmonopuls, M25, Tripuls, M26, Cardiac, M27, Quake, M28, Chirp, M29, Twotone, M3, StairDn, M30,* *SNR, M31, Hamming, M32, Hanning, M33, kaiser, M34, Blackman, M35, Gausswin, M36, Triangle, M37, Bartlett-Hann, M38, Bartlett, M39, Tan, M4, StairUD, M40, Cot, M41, Sec, M42, Csc, M43, Asin, M44, Acos, M45, Atan, M46, Acot, M47, Square, M48, SineTra, M49, SineVer, M5, Ppulse, M50, AmpALT, M51, AttALT, M52, RoundHalf, M53, RoundsPM, M54, BlaseiWave, M55, DampedOsc, M56, SwingOsc, M57, Discharge, M58, Pahcur, M59, Combin, M6, Npulse, M60, SCR, M61, Butterworth, M62, Chebyshev1, M63, Chebyshev2, M64, TV, M65, Voice, M66, Surge, M67, Radar, M68, Ripple, M69, Gamma, M7, Trapezia, M70, StepResp, M71, BandLimited, M72, CPulse, M73, CWPulse, M74, GateVibr, M75, LFMPulse, M76, MCNoise, M77, AM, M78, FM, M79, PFM, M8, Upramp, M80, PM, M81, PWM, M82, EOG, M83, EEG, M84, EMG, M85, Pulseilogram, M86, ResSpeed, M87, ECG1, M88, ECG2, M89, ECG3, M9, Dnramp, M90, ECG4, M91, ECG5, M92, ECG6, M93, ECG7, M94, ECG8, M95, ECG9, M96, ECG10, M97, ECG11, M98, ECG12, M99, ECG13*

(3) Read user-defined waveform name from the device::

*STL? USER*

Return:

*STL WVNM,sinc_8M,sinc_3000000,sinc_1664000, ramp_8M,sinc_2000000,sinc_50000,square_8M,sinc_5000,wave1,square_1M*

(4) Read waveform data from network storage:

*STL? USER,”net_storage/wave”*

Return

*net_storage/wave,STLWVNM,AutoWave2,wave1,AutoWave, ExpFal,test-sq,Besselj,libEasyLib*

Note: The table below shows the availability of some command parameters in each SDG series.

| **Parameter /command** | **SDG800** | **SDG1000** | **SDG2000X** | **SDG5000** | **SDG1000X** | **SDG6000X/ X-E** | **SDG7000 A** |
|---|---|---|---|---|---|---|---|
| STL? return | BUILDIN USER | BUILDIN USER | BUILDIN | BUILDIN USER | BUILDIN | BUILDIN | BUILDIN\| USRE (Including network storage and USB flash disk storage) |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| BUILDIN | no | no | yes | no | yes | yes | Yes |
| USER | no | no | yes | no | yes | yes | Yes |
| PATH | no | no | no | no | no | no | Yes |

## 3.32 Arb Data Command

**DESCRIPTION** This command sets the arbitrary waveform data.

**COMMAND SYNTAX** <channel>:WVDT,<parameter>,<value>

<channel>:= {C1, C2, DIG}.

<parameter>:= {a parameter from the table below}.

<value>:= {value of the corresponding parameter}.

| Parameters | Value | Description |
|---|---|---|
| WVNM | <wave_name> | := waveform name. |
| LENGTH | <length> | := the number of waveform bytes, the valid range depends on the model. See note 1 below for the details.<br>This parameter is not necessary for the “X” series |
| FREQ | <frequency> | := frequency. The unit is Hertz “Hz”. |
| AMPL | <amplifier> | := amplitude. The unit is volts, peak-to-peak “Vpp”. |
| OFST | <offset> | := offset. The unit is volts “V”. |
| PHASE | <phase> | := phase. The unit is "degree". |
| WAVEDATA | <wave data> | := waveform data.<br>Data written to waveform file |

**QUERY SYNTAX** Format 1: WVDT? Mn

Format 2: WVDT? USER,<wave_name>

<wave name>:={The name of user-defined waveform}.

Format3: WVDT? USER,<PATH>,< wave_name>

< Path > specify storage path, such as USB flash disk path. See the following example for network storage path

< waveform name >: = {user defined waveform name}

**EXAMPLE**

Example1

*WVDT? USER,“net_storage/wave”,wave1*

Return

*WVDT POS,net_storage/wave, WVNM, wave1, LENGTH, 300B, TYPE, 10,WAVEDATA,*

Example2

*C1:WVDT WVNM,”wave1”,WAVEDATA, b'0x6000c0006000'.*

Notes1:

(1) The path query function must be under the network storage or USB flash disk directory. When using the command, English quotation marks must be added at both ends of the path.

(2) The top-level directory of network storage or USB flash disk can be obtained in the file manager.

(3) Do not specify a path. The default is the local path (format 2).

(4) Try to fix the directory where waveform files are stored to reduce the waiting time caused by file retrieval.

(5) The following table shows the paths of different storage methods and is only an example.

| path | | function |
|---|---|---|
| “net_storage/wave” | < path > | Read the waveform file information under the network storage path |
| “U-disk0/wave” | < path > | Read the waveform file information under the path of USB flash disk 0 |
| “U-disk1/wave” | < path > | Read the waveform file information under the path of USB flash disk 1 |
| “U-disk2/wave” | < path > | Read the waveform file information under the path of USB flash disk 2 |

Notes2:

(1) The table below shows the availability of some command parameters in each SDG series.

| Parameter /command | SDG800 | SDG1000 | SDG2000X | SDG5000 | SDG1000X | SDG6000 X-E | SDG6000X | SDG7000A |
|---|---|---|---|---|---|---|---|---|
| TYPE | yes | yes | no | yes | no | no | no | yes |
| <length> | 32KB | 32KB | 4B~16MB | 32KB, 1024KB | 4B~16MB | 4B~16MB | 4B~40MB | |
| USER | no | no | yes | no | yes | yes | yes | yes |
| Format of WVDT? | Format 1 | Format 1 | Built-in: Format1 User-defined: Format2 | Format 1 | Built-in: Format1 User-defined: Format2 | Built-in: Format1 User-defined: Format2 | Built-in: Format1 User-defined: Format2 Format3 | Built-in: Format1 User-defined: Format2 Format3 |

(2) The table below shows the details of Mn parameters in each SDG series.

| Model | Description of Mn |
|---|---|
| SDG800 | 0<=n<=59.<br>M0~M49: build-in (32KB).<br>M50~M59: user-defined (32KB). |
| SDG1000 | 0<=n<=59.<br>M0~M49: build-in (32KB).<br>M50~M59: user-defined (32KB). |
| SDG2000X | 0<=n<=196.<br>M0~M196: build-in (32KB).<br>Not necessary when sending waveform data. |
| SDG5000 | 0<=n<=68.<br>M0~M35: build-in (32KB).<br>M36~M59: user-defined (32KB).<br>M60~M67: user-defined (1024KB). |
| SDG1000X | 0<=n<=196.<br>M0~M196: build-in (32KB).<br>Not necessary when sending waveform data. |
| SDG6000X/X-E | 0<=n<=196.<br>M0~M196: build-in (32KB).<br>Not necessary when sending waveform data. |
| SDG7000A | 0<=n<=198<br>M0~M198: build-in (32KB). |

## 3.33 Sequence Command(Only SDG7000A)

### 3.33.1 <channel>:ARBMode <Mode>

| | |
|---|---|
| **DESCRIPTION** | This command is used to set or query the working mode of the ARB waveform |
| **COMMAND SYNTAX** | <channel>:ARBMode <mode><br><br><channel>:={C1,C2}.<br><mode>:={AFG,AWG}. |
| **QUERY SYNTAX** | <channel>:ARBMode?<br><br><channel>:={C1,C2}. |
| **EXAMPLE** | Set CH1 arb to AFG mode:<br>*:C1:ARBMode AFG*<br>Read the mode of CH1:<br>*:C1:ARBMode?*<br>Return:<br>*AWG* |

### 3.33.2 <channel>:SEQuence <switch>

| | |
|---|---|
| **DESCRIPTION** | This command is used to set (query) the output status of the sequence |
| **COMMAND SYNTAX** | <channel>:SEQuence <switch><br><br><channel>:={C1,C2}.<br><switch>:={0,1}or{ON,OFF}. |
| **QUERY SYNTAX** | <channel>:SEQuence?<br><channel>:={C1,C2}. |
| **EXAMPLE** | Turn on the sequence output of channel 1<br>*:C1:SEQuence ON* |

Read the sequence output status of channel 1

*:C1:SEQuence?*

Return:

*ON*

### 3.33.3 <channel>:SEQuence:BURSt <count>

| | |
|---|---|
| **DESCRIPTION** | This command is used to set (query) the number of cycles of each output waveform when the sequence is in single output mode |
| **COMMAND SYNTAX** | <channel>:SEQuence:BURSt <count><br><br><channel>:={C1,C2}.<br><count>:={1 to 65535}. |
| **QUERY SYNTAX** | <channel>:SEQuence:BURSt?<br><channel>:={C1,C2}. |
| **EXAMPLE** | Set channel 1 sequence to output 2 cycles each time in single output mode<br>*:C1:SEQuence:BURSt 2*<br>Query the number of cycles of each output waveform of channel 1 sequence in single output mode<br>*:C1:SEQuence:BURSt?*<br><br>Return:<br>*2* |

### 3.33.4 <channel>:SEQuence:RMODe <mode>

| | |
|---|---|
| **DESCRIPTION** | This command is used to set (query) the running mode of the sequence |
| **COMMAND SYNTAX** | <channel>:SEQuence:RMODe <mode><br><br><channel>:={C1,C2}.<br><mode>:={CONT,TCON,BURS,STEP,ADV}. |

| | |
|---|---|
| **QUERY SYNTAX** | <channel>:SEQuence:RMODe?<br><channel>:={C1,C2}. |
| **EXAMPLE** | Set the operation mode of channel 1 sequence to continuous operation<br>*:C1:SEQuence:RMODe CONT*<br>Query the operation mode of channel 1 sequence<br>*:C1:SEQuence:RMODe?*<br>Return:<br>*CONT* |

### 3.33.5 <channel>:TRIGger[:SEQuence]:SOURce

| | |
|---|---|
| **DESCRIPTION** | This command is used to set (query) the trigger mode when the sequence runs |
| **COMMAND SYNTAX** | <channel>:TRIGger:SOURce <src><br><br><channel>:={C1,C2}.<br><src>:={MAN,TIMe,EXT}. |
| **QUERY SYNTAX** | <channel>:TRIGger:SOURce?<br><channel>:={C1,C2}. |
| **EXAMPLE** | Set the trigger mode of channel 1 sequence to external trigger<br>*:C1:TRIGger:SOURce EXT*<br><br>Query the trigger mode of channel 1 sequence<br>*:C1:TRIGger:SOURce?*<br><br>Return:<br>*EXT* |

### 3.33.6 <channel>:TRIGger[:SEQuence][:IMMediate]

| | |
|---|---|
| **DESCRIPTION** | This command immediately triggers a sequence output |

| | |
|---|---|
| **COMMAND SYNTAX** | <channel>:TRIGger<br><channel>:={C1,C2}. |
| **EXAMPLE** | Immediately trigger the sequence output of channel 1<br>*:C1:TRIGger* |

### 3.33.7 <channel>:TRIGger:TIMer

| | |
|---|---|
| **DESCRIPTION** | This command is used to set (query) the time interval triggered by the timer of the sequence |
| **COMMAND SYNTAX** | <channel>:TRIGger:TIMer <time><br><channel>:={C1,C2}. |
| **QUERY SYNTAX** | <channel>:TRIGger:TIMer?<br><channel>:={C1,C2}. |
| **EXAMPLE** | Set the timer trigger interval of channel 1 sequence to 1ms<br>*:C1:TRIGger:TIMer 0.001*<br><br>Query the time interval triggered by the timer of channel 1 sequence<br>*:C1:TRIGger:TIMer?*<br>Return:<br>*“0.001”* |

### 3.33.8 <channel>:TRIGger[:SEQuence]:SLOPe

| | |
|---|---|
| **DESCRIPTION** | This command is used to set (query) the trigger edge of the external trigger of the sequence |
| **COMMAND SYNTAX** | <channel>:TRIGger:SLOPe <slope><br><br><channel>:={C1,C2}.<br><slope>:={RISe,FALL}. |
| **QUERY SYNTAX** | <channel>:TRIGger:SLOPe? |

<channel>:={C1,C2}.

**EXAMPLE**

Set the trigger polarity of the external trigger of channel 1 sequence as the rising edge

*:C1:TRIGger:SLOPe RISe*

Query trigger polarity of the external trigger of channel 1 sequence

*:C1:TRIGger:SLOPe?*

Return:

*RISe*

## 3.33.9 <channel>:SEQuence:COUNt

**DESCRIPTION**

This command is used to set (query) the total number of segments of the sequence

**COMMAND SYNTAX**

<channel>:SEQuence:COUNt <count>

<channel>:={C1,C2}.

**QUERY SYNTAX**

<channel>:SEQuence:COUNt?

<channel>:={C1,C2}.

**EXAMPLE**

Set channel 1 as a sequence of 10 segments

*:C1:SEQuence:COUNt 10*

Query the total number of segments of channel 1 sequence waveform

*:C1:SEQuence:COUNt?*

Return:

*10*

**NOTE**

Count= {1 to 1024}

## 3.33.10 <channel>:SEQuence:DEFAult

| | |
|---|---|
| **DESCRIPTION** | This command is used to set (query) sequence waveform parameters as default values |
| **COMMAND SYNTAX** | <channel>:SEQuence:DEFAult<br><channel>:={C1,C2}. |
| **EXAMPLE** | Set the sequence waveform parameters of channel 1 as the default setting<br>*:C1:SEQuence:DEFAult* |

## 3.33.11 <channel>:SEQuence:NEW

| | |
|---|---|
| **DESCRIPTION** | This command is used to create a new sequence waveform |
| **COMMAND SYNTAX** | <channel>:SEQuence:NEW <segment number><br><channel>:={C1,C2}.<br><br><segment number>:= Number of segments to be created |
| **EXAMPLE** | Create a new 10 segment waveform for the sequence of channel 1<br>*:C1:SEQuence:NEW 10* |
| **NOTE** | segment number = {1 to 1024} |

## 3.33.12 <channel>:SEQuence:SEGMent<x>:WAVeform

| | |
|---|---|
| **DESCRIPTION** | This command is used to set (query) the waveform of a certain segment of the sequence through the waveform name |
| **COMMAND SYNTAX** | <channel>:SEQuence:SEGMent<x>:WAVeform <name><br><br><channel>:={C1,C2}.<br><x>:=segment number<br><name>:=waveform name. See section 3.34.5 for available waveform names |

**QUERY SYNTAX** <channel>:SEQuence:SEGMent<x>:WAVeform?

<channel>:={C1,C2}.
<x>:= segment number

**EXAMPLE** Set the waveform of the third segment of channel 1 sequence as stairup
*:C1:SEQuence:SEGMent3:WAVeform stairup*

Query the waveform of the third segment of channel 1 sequence (return the waveform name)
*:C1:SEQuence:SEGMent3:WAVeform?*

Return:
*stairup*

**NOTE** segment number = {1 to 1024}

### 3.33.13 <channel>:SEQuence:SEGMent<x>:REPeat:COUNt

**DESCRIPTION** This command is used to set (query) the repetition times of a certain segment waveform of the sequence

**COMMAND SYNTAX** <channel>:SEQuence:SEGMent<x>:REPeat:COUNt <count>

<channel>:={C1,C2}.
<x>:=segment number

<count>:= The maximum value is related to the waveform length of this segment and the total length of other segments.

**QUERY SYNTAX** <channel>:SEQuence:SEGMent<x>:REPeat:COUNt?

<channel>:={C1,C2}.
<x>:= segment number

**EXAMPLE** Set the waveform of the third segment of the channel 1 sequence to repeat twice

*:C1:SEQuence:SEGMent3:REPeat:COUNt 2*

Query the waveform repetition times of the third segment of channel 1 sequence

*:C1:SEQuence:SEGMent3:REPeat:COUNt?*

Return:

*2*

**NOTE** segment number = {1 to 1024}

### 3.33.14 <channel>:SEQuence:SEGMent<x>:AMPlitude

**DESCRIPTION** This command is used to set (query) the waveform amplitude of a certain segment of the sequence

**COMMAND SYNTAX** <channel>:SEQuence:SEGMent<x>:AMPlitude <amp>

<channel>:={C1,C2}.

<x>:= segment number

<amp>:={0 to 24Vpp}

**QUERY SYNTAX** <channel>:SEQuence:SEGMent<x>:AMPlitude?

<channel>:={C1,C2}.

<x>:= segment number

**EXAMPLE** Set the amplitude of the third segment waveform of channel 1 sequence to 2Vpp

*:C1:SEQuence:SEGMent3:AMPlitude 2*

Query the amplitude of the waveform of the third segment of channel 1 sequence

*:C1:SEQuence:SEGMent3:AMPlitude?*

Return:

*2*

**NOTE** segment number = {1 to 1024}

### 3.33.15 <channel>:SEQuence:SEGMent<x>:OFFset

|  |  |
|---|---|
| **DESCRIPTION** | This command is used to set (query) the waveform offset of a certain segment of the sequence |
| **COMMAND SYNTAX** | <channel>:SEQuence:SEGMent<x>:OFFset <offset><br><br><channel>:={C1,C2}.<br><x>:= segment number<br><offset>:={-12V to 12V} |
| **QUERY SYNTAX** | <channel>:SEQuence:SEGMent<x>:OFFset?<br><channel>:={C1,C2}.<br><x>:= segment number |
| **EXAMPLE** | Set the offset of the third segment waveform of channel 1 sequence to 2Vdc<br>*:C1:SEQuence:SEGMent3:OFFset 2*<br><br>Query the offset of the third segment waveform of channel 1 sequence<br>*:C1:SEQuence:SEGMent3:OFFset?*<br><br>Return*:*<br>*2* |
| **NOTE** | segment number = {1 to 1024} |

### 3.33.16 <channel>:SEQuence:SEGMent<x>:VOLTage:HIGH

|  |  |
|---|---|
| **DESCRIPTION** | This command is used to set (query) the high-level value of the waveform of a certain segment of the sequence |
| **COMMAND SYNTAX** | <channel>:SEQuence:SEGMent<x>:VOLTage:HIGH <highLevel><br><br><channel>:={C1,C2}.<br><x>:= segment number<br><highLevel>:={lowLevel to 12V} |

**QUERY SYNTAX** <channel>:SEQuence:SEGMent<x>:VOLTage:HIGH?

<channel>:={C1,C2}.
<x>:= segment number

**EXAMPLE** Set the high level of the third segment waveform of channel 1 sequence to 10V
*:C1:SEQuence:SEGMent3:VOLTage:HIGH 10*

Query the high-level value of the waveform of the third segment of the channel 1 sequence
*:C1:SEQuence:SEGMent3:VOLTage:HIGH?*

Return:
*10*

**NOTE** segment number = {1 to 1024}

### 3.33.17 <channel>:SEQuence:SEGMent<x>:VOLTage: LOW

**DESCRIPTION** This command is used to set (query) the low-level value of the waveform of a certain segment of the sequence

**COMMAND SYNTAX** <channel>:SEQuence:SEGMent<x>:VOLTage:LOW <lowLevel>

<channel>:={C1,C2}.
<x>:= segment number
<lowLevel>:={-12V to highLevel}

**QUERY SYNTAX** <channel>:SEQuence:SEGMent<x>:VOLTage:LOW?

<channel>:={C1,C2}.
<x>:= segment number

**EXAMPLE** Set the low level of the third segment waveform of channel 1 sequence to 10V
*:C1:SEQuence:SEGMent3:VOLTage:LOW 10*

Query the low-level value of the third segment waveform of channel

1 sequence
*:C1:SEQuence:SEGMent3:VOLTage:LOW?*

Return:
*10*

**NOTE** segment number = {1 to 1024}

### 3.33.18 <channel>:SEQuence:SEGMent<x>:LENGth

**DESCRIPTION** This command is used to set (query) the length of the waveform of a certain segment of the sequence

**COMMAND SYNTAX** <channel>:SEQuence:SEGMent<x>:LENGth <len>

<channel>:={C1,C2}.
<x>:= segment number
<len>:= It will be automatically truncated to an integer multiple of 16

**QUERY SYNTAX** <channel>:SEQuence:SEGMent<x>:LENGth?

<channel>:={C1,C2}.
<x>:= segment number

**EXAMPLE** Set the length of the waveform of the third segment of the channel 1 sequence to 16384
*:C1:SEQuence:SEGMent3:LENGth 16384*

Query the length of the waveform of the third segment of the channel 1 sequence
*:C1:SEQuence:SEGMent3:LENGth?*

Return:
*16384*

**NOTE** segment number = {1 to 1024}

## 3.33.19 <channel>:SEQuence:SEGMent<x>:DELEte

| | |
|---|---|
| **DESCRIPTION** | This command is used to delete a segment in the sequence waveform |
| **COMMAND SYNTAX** | <channel>:SEQuence:SEGMent<x>:DELEte<br><br><channel>:={C1,C2}.<br><x>:= segment number |
| **EXAMPLE** | Delete the waveform of the third segment in the channel 1 sequence<br>*C1:SEQuence:SEGMent3:DELEte* |
| **NOTE** | segment number = {1 to 1024} |

## 3.33.20 <channel>:SEQuence:SEGMent<x>:INSErt

| | |
|---|---|
| **DESCRIPTION** | This command is used to insert a new segment after a certain segment in the sequence waveform |
| **COMMAND SYNTAX** | <channel>:SEQuence:SEGMent<x>:INSErt<br><br><channel>:={C1,C2}.<br><x>:= segment number |
| **EXAMPLE** | Insert a new segment after the third segment of the channel 1 sequence<br>*:C1:SEQuence:SEGMent3:INSErt* |
| **NOTE** | segment number = {1 to 1024} |

## 3.33.21 <channel>:SEQuence:STATe

| | |
|---|---|
| **DESCRIPTION** | This command is used to set (query) the running status of the sequence |
| **COMMAND SYNTAX** | <channel>:SEQuence:STATe <state> |

<channel>:={C1,C2}.
<state>:={RUN,STOP} or {ON,OFF}

**QUERY SYNTAX**

<channel>:SEQuence:STATe?
<channel>:={C1,C2}.

**EXAMPLE**

Set the status of channel 1 sequence to "run"
*:C1:SEQuence:STATe RUN*

Query the status of channel 1 sequence
*:C1:SEQuence:STATe?*
Return:
*RUN*

### 3.33.22 <channel>:SEQuence:SCALe

**DESCRIPTION**

This command is used to set (query) the output amplitude of the sequence

**COMMAND SYNTAX**

<channel>:SEQuence:SCALe <scale>
<channel>:={C1,C2}.
<scale>:={0.01 to 1}

**QUERY SYNTAX**

<channel>:SEQuence:SCALe?
<channel>:={C1,C2}.

**EXAMPLE**

Set the output amplitude of channel 1 sequence to 80%
*:C1:SEQuence:SCALe 0.8*

Query the output amplitude of channel 1 sequence
*:C1:SEQuence:SCALe?*
Return:
*0.8*

### 3.33.23 <channel>:SEQuence:INCReasing

| | |
|---|---|
| **DESCRIPTION** | This command is used to set (query) the interpolation method of the sequence |
| **COMMAND SYNTAX** | <channel>:SEQuence:INCReasing <mode><br><br><channel>:={C1,C2}.<br><mode>:={INT,ZERo,HLAS,DUPL} |
| **QUERY SYNTAX** | <channel>:SEQuence:INCReasing?<br><channel>:={C1,C2}. |
| **EXAMPLE** | Set the interpolation mode of channel 1 sequence to linear interpolation<br>*C1:SEQuence:INCReasing INT*<br>Query the interpolation mode of channel 1 sequence<br>*C1:SEQuence:INCReasing?*<br>Return:<br>*INT* |

### 3.33.24 <channel>:SEQuence:DECReasing

| | |
|---|---|
| **DESCRIPTION** | This command is used to set (query) the sampling method of the sequence |
| **COMMAND SYNTAX** | <channel>:SEQuence:DECReasing <mode><br><br><channel>:={C1,C2}.<br><mode>:={DECi,CTAi,CHEa} |
| **QUERY SYNTAX** | <channel>:SEQuence:DECReasing?<br><channel>:={C1,C2}. |
| **EXAMPLE** | Set the sampling method of channel 1 sequence to linear sampling<br>*:C1:SEQuence:DECReasing DECi*<br>Query the sampling method of channel 1 sequence<br>*:C1:SEQuence:DECReasing?* |

Return:
*DECi*

## 3.33.25 <channel>:SEQuence:RECall

| | |
|---|---|
| **DESCRIPTION** | This command is used to load sequence waveforms from a file |
| **COMMAND SYNTAX** | <channel>:SEQuence:RECall <path><br><br><channel>:={C1,C2}.<br><path>:= Complete path of waveform |
| **EXAMPLE** | From file sequence.csv loading sequence waveform<br>*:C1:SEQuence:RECall "Local/sequence.awg"*<br>*:C1:SEQuence:RECall "U-disk0/sequence.awg"*<br>*:C1:SEQuence:RECall "net_storage/sequence.awg"* |
| **NOTE** | The specific path refers to the path in the file manager |

## 3.33.26 <channel>:SEQuence:SAVe

| | |
|---|---|
| **DESCRIPTION** | This command is used to save the sequence waveform to a file |
| **COMMAND SYNTAX** | <channel>:SEQuence:SAVe<path><br><br><channel>:={C1,C2}.<br><path>:= Complete path of waveform |
| **EXAMPLE** | Save the sequence waveform to wave1.csv<br>*:C1:SEQuence:SAVe "Local/sequence.awg"*<br>*:C1:SEQuence:SAVe "U-disk0/sequence.awg"*<br>*:C1:SEQuence:SAVe "net_storage/sequence.awg"* |
| **NOTE** | The specific path refers to the path in the file manager |

## 3.34 Digital Channel Command(Only SDG7000A)

### 3.34.1 DIG:SRATe

| | |
|---|---|
| **DESCRIPTION** | This command is used to set (query) the bit rate of the digital channel |
| **COMMAND SYNTAX** | DIG:SRATe <value><br><value>:={0 to 1000000000} |
| **QUERY SYNTAX** | DIG:SRATe? |
| **EXAMPLE** | Set the bit rate of the digital channel to 100Mbps<br>*:DIG:SRATe 100000000*<br>Query bit rate of digital channel<br>*:DIG:SRATe?*<br><br>Return:<br>*100000000* |

### 3.34.2 DIG:PERiod

| | |
|---|---|
| **DESCRIPTION** | This command is used to set (query) the period of the digital channel |
| **COMMAND SYNTAX** | DIG:PERiod <value><br><value>:={1ns to 1Ks} |
| **QUERY SYNTAX** | DIG:PERiod? |
| **EXAMPLE** | Set the period of the digital channel to 10 ns<br>*:DIG:PERiod 0.00000001*<br>Query the period of the digital channel<br>*:DIG:PERiod?*<br><br>Return:<br>*0.00000001* |

### 3.34.3 DIG:CHANnel<x>:STATe

| | |
|---|---|
| **DESCRIPTION** | This command is used to set (query) the status of a digital channel |
| **COMMAND SYNTAX** | DIG:CHANnel<x>:STATe <value><br><X>:={1 to 16}<br><value>:={0 or 1} or {off or on} |
| **QUERY SYNTAX** | DIG:CHANnel<index>:STATe? |
| **EXAMPLE** | Set channel 2 to on<br>*:DIG:CHANnel2:STATe 1*<br>Query the status of digital channel 2<br>*:DIG:CHANnel2;STATe?*<br><br>Return:<br>*1* |

### 3.34.4 DIG:OUTPut

| | |
|---|---|
| **DESCRIPTION** | This command is used to set (query) the output status of the digital channel |
| **COMMAND SYNTAX** | DIG:OUTPut <value><br><value>:={0 or 1} or {off or on} |
| **QUERY SYNTAX** | DIG:OUTPut? |
| **EXAMPLE** | Turn on the digital channel output<br>*:DIG:OUTPut 1*<br>Query the output status of the digital channel<br>*:DIG:OUTPut?*<br><br>Return:<br>*1* |

### 3.34.5 DIG:WAVeform

**DESCRIPTION** This command is used to set (query) the data source of the digital channel through the waveform name

**COMMAND SYNTAX** DIG:WAVeform <name>

<name>:= The name of the built-in waveform, or the complete path of the waveform

**QUERY SYNTAX** DIG:WAVeform?

**EXAMPLE** Set the data source of the digital channel as sine

*:DIG:WAVeform sine*

Query the digital channel data source

*:DIG:WAVeform?*

Return:

*"sine"*

NOTE: Customized edit waveform, see 5.1.5

| | | | |
|---|---|---|---|
| sine | noise | erfc | erfcinv |
| stairup | stairdn | erfinv | laguerre |
| stairud | ppulse | legend | versiera |
| npulse | trapezia | weibull | lognormal |
| upramp | dnramp | laplace | maxwell |
| exp_fall | exp_rise | rayleigh | cauchy |
| logfall | logrise | cosh | cosint |
| sqrt | root3 | coth | csch |
| x^2 | x^3 | sech | sinh |
| sinc | gussian | sinint | tanh |
| dlorentz | haversine | acosh | asech |
| lorentz | gauspuls | asinh | atanh |
| gmonopuls | tripuls | acsch | acoth |
| cardiac | quake | bartlett | bohmanwin |
| chirp | twotone | chebwin | flattopwin |
| snr | digit_clock | parzenwin | taylorwin |
| digit_counter | digit_zero | tukeywin | square_duty01 |
| hamming | hanning | square_duty02 | square_duty04 |
| kaiser | blackman | square_duty06 | square_duty08 |

| | | | |
|---|---|---|---|
| gausswin | triang | square_duty10 | square_duty12 |
| blackmanharris | barthannwin | quare_duty14 | square_duty16 |
| tan | cot | square_duty18 | square_duty20 |
| sec | csc | square_duty22 | square_duty24 |
| asin | acos | square_duty26 | square_duty28 |
| atan | acot | square_duty30 | square_duty32 |
| square | sinetra | square_duty34 | square_duty36 |
| sinever | ampalt | square_duty38 | square_duty40 |
| attalt | roundhalf | square_duty42 | square_duty44 |
| roundspm | blaseiwave | square_duty46 | square_duty48 |
| dampedosc | swingosc | square_duty50 | square_duty52 |
| discharge | pahcur | square_duty54 | square_duty56 |
| combin | scr | square_duty58 | square_duty60 |
| butterworth | chebyshev1 | square_duty62 | square_duty64 |
| chebyshev2 | tv | square_duty66 | square_duty68 |
| voice | surge | square_duty70 | square_duty72 |
| radar | ripple | square_duty74 | square_duty76 |
| gamma | stepresp | square_duty78 | square_duty80 |
| bandlimited | cpulse | square_duty82 | square_duty84 |
| cwpulse | gatevibr | square_duty86 | square_duty88 |
| lfmpulse | mcnoise | square_duty90 | square_duty92 |
| am | fm | square_duty94 | square_duty96 |
| pfm | pm | square_duty98 | square_duty99 |
| pwm | eog | demo1_375pts | demo1_16kpts |
| eeg | emg | demo2_3kpts | demo2_16kpts |
| pulseilogram | resspeed | sine_harmonic2 | sine_harmonic3 |
| ecg1 | ecg2 | sine_harmonic4 | sine_harmonic5 |
| ecg3 | ecg4 | sine_harmonic6 | sine_harmonic7 |
| ecg5 | ecg6 | sine_harmonic8 | sine_harmonic9 |
| ecg7 | ecg8 | sine_harmonic10 | sine_harmonic11 |
| ecg9 | ecg10 | sine_harmonic12 | sine_harmonic13 |
| ecg11 | ecg12 | sine_harmonic14 | sine_harmonic15 |
| ecg13 | ecg14 | sine_harmonic16 | digit_one |
| ecg15 | lfpulse | | |
| tens1 | tens2 | | |
| tens3 | airy | | |
| besselj | bessely | | |
| dirichlet | erf | | |

## 3.35 Frequency Hop Command(Only SDG7000A)

### 3.35.1 <channel>:FHOP:SWITch

| | |
|---|---|
| **DESCRIPTION** | This command is used to set frequency hop switch state |
| **COMMAND SYNTAX** | <channel>:FHOP:SWITch <switch><br><channel>:={C1, C2}<br><switch>:= {ON, OFF} |
| **QUERY SYNTAX** | <channel>:FHOP:SWITch? |
| **EXAMPLE** | CH1 turn on frequency hop<br>*C1:FHOP:SWITch ON*<br>Query the switch state of CH1<br>*C1:FHOP:SWITch?*<br><br>Return:<br>*"ON"* |

### 3.35.2 <channel>:FHOP:STATe

| | |
|---|---|
| **DESCRIPTION** | This command is used to set the frequency hop run state |
| **COMMAND SYNTAX** | <channel>:FHOP:STATe <state><br><channel>:={C1, C2}<br><state>:= {ON, OFF} |
| **QUERY SYNTAX** | <channel>:FHOP:STATe? |
| **EXAMPLE** | CH1 set frequency hop run state on<br>*C1:FHOP:STATe ON*<br>Query the run state of CH1<br>*C1:FHOP:STATe?*<br><br>Return:<br>*"ON"* |

### 3.35.3 <channel>:FHOP:TYPE

**DESCRIPTION** This command is used to set the frequency hop type

**COMMAND SYNTAX** <channel>:FHOP:TYPE <type>
<channel>:={C1, C2}
<type>:= {"MANUAL", "RHOP", "RLIST"}

**QUERY SYNTAX** <channel>:FHOP:TYPE?

**EXAMPLE** Set CH1 frequency hop to manual type
*C1:FHOP:TYPE MANUAL*
Query CH1 frequency hop type
*C1:FHOP:TYPE?*

Return:
*“MANUAL”*

### 3.35.4 <channel>:FHOP:TIME

**DESCRIPTION** This command is used to set the frequency hop time

**COMMAND SYNTAX** <channel>:FHOP:TIME <time>
<channel>:={C1, C2}
<time>:= {float value from 0 to 1000.Unit second}

**QUERY SYNTAX** <channel>:FHOP:TIME?

**EXAMPLE** Set CH1 frequency hop time 1ms
*C1:FHOP:TIME 0.001*
Query CH1 frequency hop type time
*C1:FHOP:TIME?*

Return:
*“0.001”*

## 3.35.5 <channel>:FHOP:SFREquency

**DESCRIPTION** This command is used to set the min hop frequency in random mode.

**COMMAND SYNTAX** <channel>:FHOP:SFREquency <freq>

<channel>:={C1, C2}

<freq>:= {float value}

**QUERY SYNTAX** <channel>:FHOP:SFREquency?

**EXAMPLE** Set CH1 min hop frequency 1MHZ in random mode

*C1:FHOP:SFREquency 1000000*

Query CH1 min hop frequency in random mode

*C1:FHOP:SFREquency?*

Return:

*“1000000”*

## 3.35.6 <channel>:FHOP:EFREquency

**DESCRIPTION** This command is used to set the max hop frequency in random mode.

**COMMAND SYNTAX** <channel>:FHOP:EFREquency <freq>

<channel>:={C1， C2}

<freq>:= { float value }

**QUERY SYNTAX** <channel>:FHOP:EFREquency?

**EXAMPLE** Set CH1 max hop frequency 1MHZ in random mode

*C1:FHOP:EFREquency 100000000*

Query CH1 max hop frequency in random mode

*C1:FHOP:EFREquency?*

Return:

*“100000000”*

### 3.35.7 <channel>:FHOP:FSTep

| | |
|---|---|
| **DESCRIPTION** | This command is used to set the frequency step in random hop mode |
| **COMMAND SYNTAX** | <channel>:FHOP:FSTep <freq><br><channel>:={C1, C2}<br><freq>:= { float value } |
| **QUERY SYNTAX** | <channel>:FHOP:FSTep? |
| **EXAMPLE** | Set CH1 frequency step 10MHz in random hop mode<br>*C1:FHOP:FSTep 10000000*<br>Query CH1 frequency step in random hop mode<br>*C1:FHOP: FSTep?*<br><br>Return:<br>*“10000000”* |

### 3.35.8 <channel>:FHOP:RPATtern

| | |
|---|---|
| **DESCRIPTION** | This command is used to set the prbs pattern in random hop mode |
| **COMMAND SYNTAX** | <channel>:FHOP:RPATtern <pattern><br><channel>:={C1, C2}<br><pattern>:= {Int value between 3 and 32} |
| **QUERY SYNTAX** | <channel>:FHOP:RPATtern? |
| **EXAMPLE** | Set CH1 prbs pattern prbs-7 in random hop mode<br>*C1:FHOP:RPATtern 7*<br>Query CH1 prbs pattern in random hop mode<br>*C1:FHOP:RPATtern?*<br><br>Return:<br>*“7”* |

### 3.35.9 <channel>:FHOP:RLPAttern

**DESCRIPTION** This command is used to set the prbs pattern in random list mode

**COMMAND SYNTAX**

<channel>:FHOP:RLPAttern <pattern>
<channel>:={C1, C2}
<pattern>:= {Int value between 3 and 32}

**QUERY SYNTAX**

<channel>:FHOP:RLPAttern?

**EXAMPLE**

Set CH1 prbs pattern prbs-7 in random list mode
*C1:FHOP:RLPAttern 7*
Query CH1 prbs pattern in random list mode
*C1:FHOP:RLPAttern?*

Return:
*"7"*

### 3.35.10 <channel>:FHOP:ALSTate

**DESCRIPTION** This command is used to set the enable state of frequency avoid table

**COMMAND SYNTAX**

<channel>:FHOP:ALSTate <state>
<channel>:={C1, C2}
<state>:= {ON, OFF}

**QUERY SYNTAX**

<channel>:FHOP:ALSTate?

**EXAMPLE**

Set CH1 frequency avoid table enabled
*C1:FHOP:ALSTate ON*
Query CH1 frequency avoid table enable state
*C1:FHOP:ALSTate?*

Return:
*"ON"*

### 3.35.11 <channel>:FHOP:AFLIst

| | |
|---|---|
| **DESCRIPTION** | This command is used to insert a new element into the frequency table at the specified position |
| **COMMAND SYNTAX** | <channel>:FHOP:AFLIst <index>,<freq><br><channel>:={C1, C2}<br><index>:= {Int value between 1 and 4096}<br><freq>:= {float value} |
| **EXAMPLE** | Insert a new element 1KHz into CH1 frequency table at the position row 3<br>*C1:FHOP:AFLIst 3,1000* |

### 3.35.12 <channel>:FHOP:DFLIst

| | |
|---|---|
| **DESCRIPTION** | This command is used to delete a element from the frequency table at the specified position |
| **COMMAND SYNTAX** | <channel>:FHOP:DFLIst <index><br><channel>:={C1, C2}<br><index>:= {Int value between 1 and 4096} |
| **EXAMPLE** | Delete row 3 from CH1 frequency table<br>*C1:FHOP:DFLIst 3* |

### 3.35.13 <channel>:FHOP:CFLIst

| | |
|---|---|
| **DESCRIPTION** | This command is used to clear the frequency table |
| **COMMAND SYNTAX** | <channel>:FHOP:CFLIst<br><channel>:={C1, C2} |
| **EXAMPLE** | Clear CH1 frequency table<br>*C1:FHOP:CFLIst* |

### 3.35.14 <channel>:FHOP:MFLIst

| | |
|---|---|
| **DESCRIPTION** | This command is used to set the specified element value of frequency table |
| **COMMAND SYNTAX** | <channel>:FHOP:MFLIst <index>,<freq><br><channel>:={C1, C2}<br><index>:= { Int value between 1 and 4096}<br><freq>:= { float value } |
| **EXAMPLE** | Set CH1 frequency table row 3 element value 2KHz<br>*C1:FHOP:MFLIst 3,2000* |

### 3.35.15 <channel>:FHOP:AOLIst

| | |
|---|---|
| **DESCRIPTION** | This command is used to insert a new element into the hop order table at the specified position |
| **COMMAND SYNTAX** | <channel>:FHOP:AOLIst <index>,<freq_num><br><channel>:={C1, C2}<br><index>:= { Int value between 1 and 4096}<br><freq_num>:= { Int value between 1 and 4096} |
| **EXAMPLE** | Insert a new element 4 into CH1 hop order table at the position row 3<br>*C1:FHOP:AOLIst 3,4* |

### 3.35.16 <channel>:FHOP:DOLIst

| | |
|---|---|
| **DESCRIPTION** | This command is used to delete a element from the hop order table at the specified position |
| **COMMAND SYNTAX** | <channel>:FHOP:DOLIst <index><br><channel>:={C1, C2}<br><index>:= { Int value between 1 and 4096} |
| **EXAMPLE** | Delete row 3 from CH1 hop order table<br>*C1:FHOP:DOLIst 3* |

### 3.35.17 <channel>:FHOP:COLIst

| | |
|---|---|
| **DESCRIPTION** | This command is used to clear the hop order table |
| **COMMAND SYNTAX** | <channel>:FHOP:COLIst<br><channel>:={C1, C2} |
| **EXAMPLE** | Clear CH1 hop order table<br>*C1:FHOP:COLIst* |

### 3.35.18 <channel>:FHOP:MOLIst

| | |
|---|---|
| **DESCRIPTION** | This command is used to set the specified element value of hop order table |
| **COMMAND SYNTAX** | <channel>:FHOP:MOLIst <index>,<freq_num><br><channel>:={C1, C2}<br><index>:= { Int value between 1 and 4096}<br><freq_num>:= { Int value between 1 and 4096} |
| **EXAMPLE** | Set CH1 hop order table row 3 element value 8<br>*C1:FHOP:MOLIst 3,8* |

### 3.35.19 <channel>:FHOP:AALIst

| | |
|---|---|
| **DESCRIPTION** | This command is used to add a new element at the end of frequency avoid table |
| **COMMAND SYNTAX** | <channel>:FHOP:AALIst <start_freq>,<end_freq><br><channel>:={C1, C2}<br><start_freq>:= {float value}<br><end_freq>:= {float value} |
| **EXAMPLE** | Add a new element at the end of frequency avoid table, value 1KHz and 5KHz<br>*C1:FHOP:AALIst 1000,5000* |

### 3.35.20 <channel>:FHOP:DALIst

| | |
|---|---|
| **DESCRIPTION** | This command is used to delete the last element of frequency avoid table |
| **COMMAND SYNTAX** | <channel>:FHOP:DALIst<br><channel>:={C1, C2} |
| **EXAMPLE** | Delete the last element of CH1 frequency avoid table<br>*C1:FHOP:DALIst* |

### 3.35.21 <channel>:FHOP:CALIst

| | |
|---|---|
| **DESCRIPTION** | This command is used to clear the frequency avoid table |
| **COMMAND SYNTAX** | <channel>:FHOP:CALIst<br><channel>:={C1, C2} |
| **EXAMPLE** | Clear CH1 frequency avoid table<br>*C1:FHOP:CALIst* |

### 3.35.22 <channel>:FHOP:LFLIst

| | |
|---|---|
| **DESCRIPTION** | This command is used to load frequency table from file |
| **COMMAND SYNTAX** | <channel>:FHOP:LFLIst <file><br><channel>:={C1, C2}<br><file>:=file name |
| **EXAMPLE** | Load CH1 frequency table from file “freq.hop”<br>*C1:FHOP:LFLIst “freq.hop”* |

### 3.35.23 <channel>:FHOP:SFLIst

| | |
|---|---|
| **DESCRIPTION** | This command is used to save frequency table to file |
| **COMMAND SYNTAX** | <channel>:FHOP:SFLIst <file><br><channel>:={C1, C2}<br><file>:= file name |
| **EXAMPLE** | Save CH1 frequency table to file “freq.hop”<br>*C1:FHOP:SFLIst “freq.hop”* |

### 3.35.24 <channel>:FHOP:LOLIst

| | |
|---|---|
| **DESCRIPTION** | This command is used to load hop order table from file |
| **COMMAND SYNTAX** | <channel>:FHOP:LOLIst <file><br><channel>:={C1, C2}<br><file>:= file name |
| **EXAMPLE** | Load CH1 hop order table from file “order.hop”<br>*C1:FHOP:LOLIst “order.hop”* |

### 3.35.25 <channel>:FHOP:SOLIst

| | |
|---|---|
| **DESCRIPTION** | This command is used to save hop order table to file |
| **COMMAND SYNTAX** | <channel>:FHOP:SOLIst <file><br><channel>:={C1, C2}<br><file>:= file name |
| **EXAMPLE** | Save CH1 hop order table to file “order.hop”<br>*C1:FHOP:SOLIst “order.hop”* |

## 3.35.26 <channel>:FHOP:LALIst

| | |
|---|---|
| **DESCRIPTION** | This command is used to save frequency avoid table to file |
| **COMMAND SYNTAX** | <channel>:FHOP:LALIst <file><br><channel>:={C1, C2}<br><file>:= file name |
| **EXAMPLE** | Load CH1 frequency avoid table from file “avoid.hop”<br>*C1:FHOP:LALIst “avoid.hop”* |

## 3.35.27 <channel>:FHOP:SALIst

| | |
|---|---|
| **DESCRIPTION** | This command is used to save frequency avoid table to file |
| **COMMAND SYNTAX** | <channel>:FHOP:SALIst <file><br><channel>:={C1, C2}<br><file>:= file name |
| **EXAMPLE** | Save CH1 frequency avoid table to file “avoid.hop”<br>*C1:FHOP:SALIst “avoid.hop”* |

## 3.36 Virtual Key Command

**DESCRIPTION** This command is used to simulate pressing a key on the front panel.

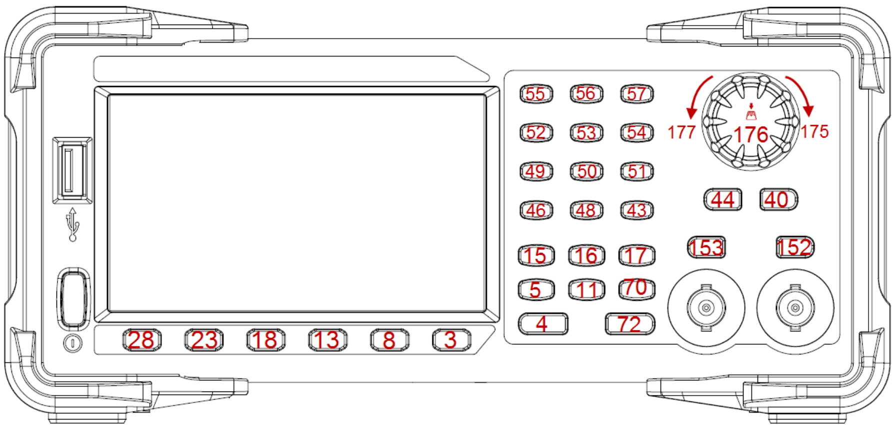
**COMMAND SYNTAX** VirtualKEY VALUE,<value>,STATE,<state>

<value>:= {a Name or Index of the virtual keys from the table below}.

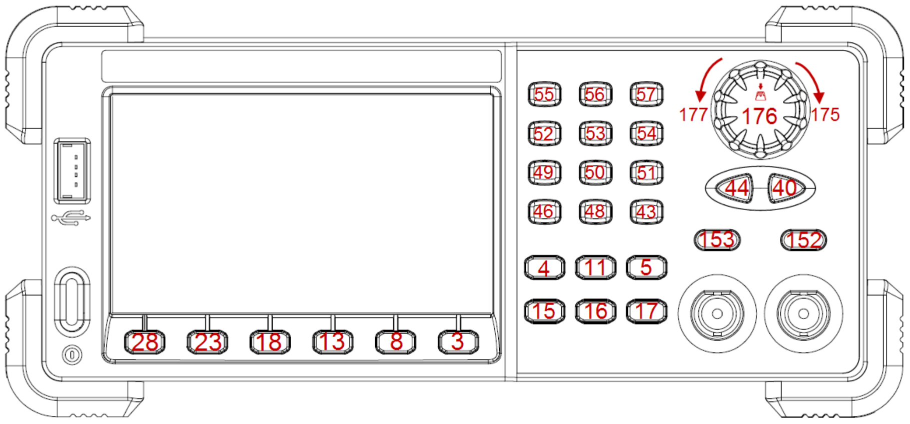
<state>:= {0,1}, where 1 is effective to virtual value, and 0 is useless.

| Name | Indexes | Name | Indexes |
|---|---|---|---|
| KB_FUNC1 | 28 | KB_NUMBER_4 | 52 |
| KB_FUNC2 | 23 | KB_NUMBER_5 | 53 |
| KB_FUNC3 | 18 | KB_NUMBER_6 | 54 |
| KB_FUNC4 | 13 | KB_NUMBER_7 | 55 |
| KB_FUNC5 | 8 | KB_NUMBER_8 | 56 |
| KB_FUNC6 | 3 | KB_NUMBER_9 | 57 |
| KB_SINE | 34 | KB_POINT | 46 |
| KB_SQUARE | 29 | KB_NEGATIVE | 43 |
| KB_RAMP | 24 | KB_LEFT | 44 |
| KB_PULSE | 19 | KB_RIGHT | 40 |
| KB_NOISE | 14 | KB_UP | 45 |
| KB_ARB | 9 | KB_DOWN | 39 |
| KB_MOD | 15 | KB_OUTPUT1 | 153 |
| KB_SWEEP | 16 | KB_OUTPUT2 | 152 |
| KB_BURST | 17 | KB_KNOB_RIGHT | 175 |
| KB_WAVES | 4 | KB_KNOB_LEFT | 177 |
| KB_UTILITY | 11 | KB_KNOB_DOWN | 176 |
| KB_PARAMETER | 5 | KB_HELP | 12 |
| KB_STORE_RECALL | 70 | KB_CHANNEL | 72 |
| KB_NUMBER_0 | 48 | KB_K | 59 |
| KB_NUMBER_1 | 49 | KB_M | 60 |
| KB_NUMBER_2 | 50 | KB_G | 61 |
| KB_NUMBER_3 | 51 | KB_DIGITAL | 20 |

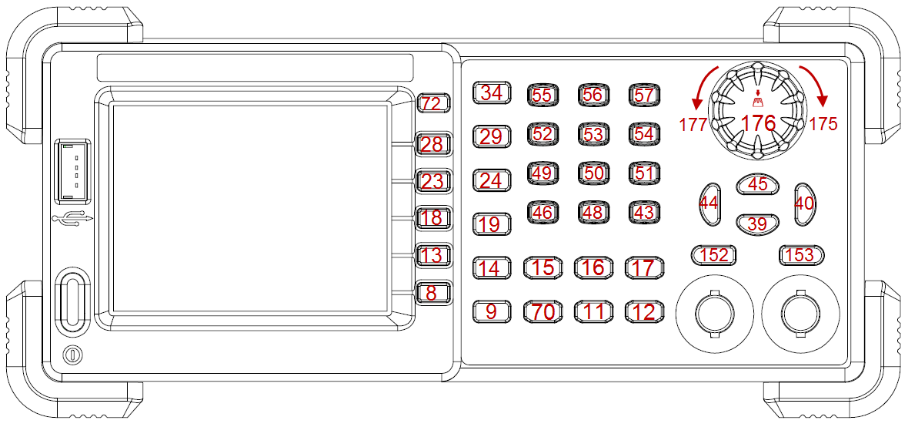
| | | | |
|---|---|---|---|
| KB_ENTER | 58 | KB_HOME | 21 |
| KB_AWG | 29 | KB_TOUCH | 22 |
| KB_IQ | 30 | | |



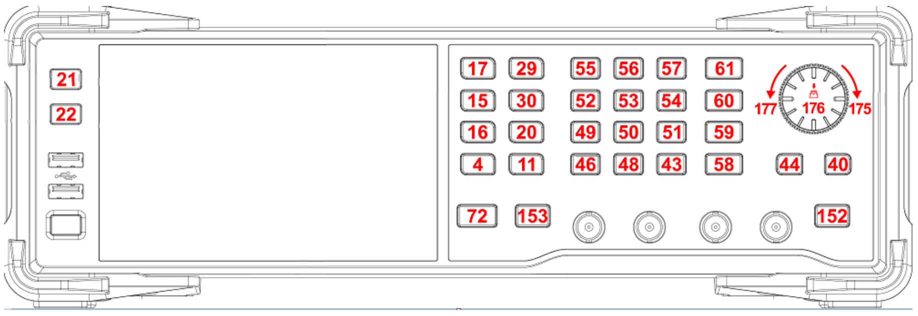
Keys and Indices on the SDG1000X/SDG2000X/SDG6000X/SDG6000X-E



Keys and Indices on the SDG5000



Keys and Indices on the SDG1000/SDG800



Keys and Indices on the SDG7000A

EXAMPLE *VKEY VALUE,15,STATE,1*

*VKEY VALUE,KB_SWEEP,STATE,1*

Note: The table below shows the availability of some command parameters in each SDG series.

| **Parameter /command** | **SDG 800** | **SDG 1000** | **SDG 2000X** | **SDG 5000** | **SDG 1000X** | **SDG 6000X/X-E** | **SDG 7000A** |
|---|---|---|---|---|---|---|---|
| KB_FUNC6 | no | no | yes | yes | yes | yes | no |
| KB_STORE_ RECALL | yes | yes | yes | no | yes | yes | no |
| KB_HELP | yes | yes | no | no | no | no | no |
| KB_CHANNEL | no | yes | yes | no | yes | yes | yes |
| KB_SINE | yes | yes | no | no | no | no | no |
| KB_SQUARE | yes | yes | no | no | no | no | no |
| KB_ RAMP | yes | yes | no | no | no | no | no |
| KB_PULSE | yes | yes | no | no | no | no | no |
| KB_NOISE | yes | yes | no | no | no | no | no |
| KB_ARB | yes | yes | no | no | no | no | no |
| KB_UP | yes | yes | no | no | no | no | no |
| KB_DOWN | yes | yes | no | no | no | no | no |

## 3.37 IP Command

**DESCRIPTION**

This command sets and gets the system IP address.

**COMMAND SYNTAX**

SYSTem:COMMunicate:LAN:IPADdress
"<parameter1>.<parameter2>.<parameter3>.<parameter4>"

<parameter1>:={an integer value between 1 and 223}.
<parameter2>:={an integer value between 0 and 255}.
<parameter3>:={an integer value between 0 and 255}.
<parameter4>:={an integer value between 0 and 255}.

**QUERY SYNTAX**

SYSTem:COMMunicate:LAN:IPADdress?

**EXAMPLES**

Set IP address to 10.11.13.203:
*SYST:COMM:LAN:IPAD "10.11.13.203"*

Get the IP address:
*SYST:COMM:LAN:IPAD?*
Return:
*"10.11.13.203"*

Note: The table below shows the availability of the command in each SDG series :

| Parameter /command | SDG 800 | SDG 1000 | SDG 2000X | SDG 5000 | SDG 1000X | SDG 6000X/X-E | SDG 7000A |
|---|---|---|---|---|---|---|---|
| SYST:COMM: LAN:IPAD | no | no | yes | no | yes | yes | yes |

## 3.38 Subnet Mask Command

**DESCRIPTION**

This command sets and gets the system subnet mask.

**COMMAND SYNTAX**

SYSTem:COMMunicate:LAN:SMASk
"<parameter1>.<parameter2>.<parameter3>.<parameter4>"

<parameter1>:={an integer value between 0 and 255}.
<parameter2>:={an integer value between 0 and 255}.

<parameter3>:={an integer value between 0 and 255}.
<parameter4>:={an integer value between 0 and 255}.

**QUERY SYNTAX** SYSTem:COMMunicate:LAN:SMASk?

**EXAMPLES** Set the subnet mask to 255.0.0.0:
*SYST:COMM:LAN:SMAS "255.0.0.0"*

Get the subnet mask:
*SYST:COMM:LAN:SMAS?*
Return:
*"255.0.0.0"*

Note: The table below shows the availability of the command in each SDG series.

| Parameter /command | SDG 800 | SDG 1000 | SDG 2000X | SDG 5000 | SDG 1000X | SDG 6000X/X-E | SDG 7000A |
|---|---|---|---|---|---|---|---|
| SYST:COMM: LAN:SMAS | no | no | yes | no | yes | yes | yes |

## 3.39 Gateway Command

**DESCRIPTION** This command sets and gets the system gateway.

**COMMAND SYNTAX** SYSTem:COMMunicate:LAN:GATeway
"<parameter1>.<parameter2>.<parameter3>.<parameter4>"

<parameter1>:={an integer value between 0 and 223}.
<parameter2>:={an integer value between 0 and 255}.
<parameter3>:={an integer value between 0 and 255}.
<parameter4>:={an integer value between 0 and 255}.

**QUERY SYNTAX** SYSTem:COMMunicate:LAN:GATeway?

**EXAMPLES** Set Gateway to 10.11.13.5:
*SYSTem:COMMunicate:LAN:GATeway "10.11.13.5"*

Get gateway:

*SYSTem:COMMunicate:LAN:GATeway?*
Return:
*"10.11.13.5"*

Note: The table below shows the availability of the command in each SDG series.

| **Parameter /command** | **SDG800** | **SDG1000** | **SDG2000X** | **SDG5000** | **SDG1000X** | **SDG6000X/X-E** | **SDG7000A** |
|---|---|---|---|---|---|---|---|
| SYST:COMM: LAN:GAT | no | no | yes | no | yes | yes | yes |

## 3.40 Sampling Rate Command

**DESCRIPTION**

This command sets or gets the Arb mode, sampling rate, and interpolation method. The sampling rate and interpolation method can only be set when MODE is TARB.

**COMMAND SYNTAX**

<channel>:SampleRATE MODE,<mode>,VALUE, <sample rate>, INTER,<interpolation>
<channel>:= <C1, C2>.

<mode>:= {DDS, TARB}, where TARB is TrueArb or
<mode>:={AFG, AWG} (only SDG7000A)

<sample rate>:= sample rate. The unit is samples per second "Sa/s".

<interpolation>:= {LINE, HOLD, SINC, SINC27, SINC13}, where LINE is linear, and HOLD is zero-order hold. SINC, SINC27 and SINC13 are only for SDG6000X/X-E and SDG7000A

**QUERY SYNTAX**

<channel>:SRATE?

**EXAMPLES**

Get the sampling rate of CH1:
*C1:SRATE?*
Return:
*C1:SRATE MODE,DDS*

Set CH1 to TureArb mode:

*C1:SRATE MODE,TARB*

Set sampling rate of CH1 to 1000000Sa/s:
*C1:SRATE VALUE,1000000*

Set CH1 to TureArb mode and set interpolation to SINC13
*C1:SRATE MODE,TARB,INTER,SINC13*

Note: The table below shows the availability of the command and some parameters in each SDG series.

| Parameter /command | SDG800 | SDG1000 | SDG2000X | SDG5000 | SDG1000X | SDG6000X/X-E | SDG7000A |
|---|---|---|---|---|---|---|---|
| SRATE | no | no | yes | no | no | yes | yes |
| INTER | no | no | no | no | no | yes | yes |

## 3.41 Harmonic Command

**DESCRIPTION**

This command sets or gets the harmonic parameters. Only available when the basic wave is SINE.

**COMMAND SYNTAX**

<channel>:HARMonic HARMSTATE,<state>,HARMTYPE, <type>,HARMORDER,<order>,<unit>,<value>, HARMPHASE,<phase>

<state>:= <ON, OFF>.
<type>:= <EVEN, ODD, ALL>.

<order>:= {1,2,...,M}, where M is the supported maximum order.
<unit>:= < HARMAMP, HARMDBC>.

<value>:= amplitude of specified harmonic. The range of valid values depends on the model. When <unit>= HARMAMP, the unit is volts, peak-to-peak “Vpp”, and when <unit>= HARMDBC, the unit is "dBc".

<phase>:= {0~360}, the unit is "degree"

**QUERY SYNTAX** <channel>:HARMonic?

<channel> : ={C1, C2}.

**EXAMPLES** Enable the harmonic function of CH1:

*C1:HARM HARMSTATE,ON*

Set the 2nd harmonic of CH1 to -6 dBc:

*C1:HARM HARMORDER,2,HARMDBC,-6*

Get the harmonic information of CH1:

*C1:HARM?*

Return:

*C1:HARM ,HARMSTATE,ON,HARMTYPE,EVEN,HARMORDER,2, HARMAMP,2.004748935V,HARMDBC,-6dBc,HARMPHASE,0*

Note: The table below shows the availability of the command in each SDG series.

| Parameter /command | SDG800 | SDG1000 | SDG2000X | SDG5000 | SDG1000X | SDG6000X/X-E | SDG7000A |
|---|---|---|---|---|---|---|---|
| HARM | no | no | yes | no | yes | yes | yes |

## 3.42 Waveform Combining Command

**DESCRIPTION** This command sets or gets the waveform combining parameters.

**COMMAND SYNTAX** <channel>:CoMBiNe <state>

<channel>:= {C1, C2}.

<state>:= {ON, OFF}.

**QUERY SYNTAX** <channel>:CoMBiNe?

<channel>:= {C1, C2}.

**RESPONSE FORMAT** <channel>:CMBN <state>

**EXAMPLES** Enable waveform combining forf CH1:

*C1:CMBN ON*

Query the waveform combining state of CH2:

*C2:CMBN?*

Return:

*C2:CMBN OFF*

Note: The table below shows the availability of the command in each SDG series.

| Parameter /command | SDG800 | SDG1000 | SDG2000X | SDG5000 | SDG1000X | SDG6000X/X-E | SDG7000A |
|---|---|---|---|---|---|---|---|
| CMBN | no | no | yes | no | yes | yes | yes |

## 3.43 Mode Select Command

**DESCRIPTION** This command sets or gets the phase mode.

**COMMAND SYNTAX** MODE <parameter>

<parameter>:= {PHASELOCKED, INDEPENDENT}.

**QUERY SYNTAX** MODE?

**RESPONSE FORMAT** MODE <parameter>

**EXAMPLE** Set the phase mode to INDEPENDENT:

*MODE INDEPENDENT*

Note: The table below shows the availability of the command in each SDG series.

| Parameter /command | SDG800 | SDG1000 | SDG2000X | SDG5000 | SDG1000X | SDG6000X/X-E | SDG7000A |
|---|---|---|---|---|---|---|---|
| MODE | no | no | yes | no | yes | yes | yes |

## 3.44 Multi-Device Sync

**DESCRIPTION** This command set up synchronization between two or more instruments and achieve in-phase output

**COMMAND SYNTAX**

CASCADE STATE,ON|OFF,MODE,<MODE>,DELAY,<DELAY>
<MODE>:={MASTER,SLAVE}

<DELAY>:={0-0.000025},UNIT=s，This parameter can only be set in slave mode

**QUERY SYNTAX**

CASCADE?

**RESPONSE FORMAT**

Return from SLAVE MODE :
CASCADE STATE,ON,MODE,SLAVE,DELAY, <DELAY>

Return from MASTER mode :
CASCADE STATE,ON,MODE,MASTER

**EXAMPLE**

Set the device as slave and the delay to 0.0000001s:
*CASCADE STATE,ON,MODE,SLAVE,DELAY,0.0000001*

Note: The table below shows the availability of the command in each SDG series.

| Parameter /command | SDG800 | SDG1000 | SDG2000X | SDG5000 | SDG1000X | SDG6000X/X-E | SDG7000A |
|---|---|---|---|---|---|---|---|
| CASCADE | no | no | yes | no | no | yes | yes |

## 3.45 IQ Commands

The table below shows the availability of IQ commands in each SDG series.

| Parameter /command | SDG 800 | SDG 1000 | SDG 2000X | SDG 5000 | SDG 1000X | SDG 6000X | SDG 6000X-E | SDG 7000A |
|---|---|---|---|---|---|---|---|---|
| IQ | no | no | no | no | no | yes | no | yes |

### 3.45.1 IQ:WAVeinfo?

| | |
|---|---|
| **DESCRIPTION** | This command queries the waveform information of I/Q. |
| **COMMAND SYNTAX** | IQ:WAVeinfo? |
| **EXAMPLE** | Query current I/Q waveform information:<br>*IQ:WAVeinfo?*<br><br>Return:<br>*WAVE_INFO,SYMBOL_LENTH,1024,OVER_SAMPLING, 4,MODULATION,2ASK,FILTER_TYPE,RootCosine, FILTER_ALPHA,0.35* |
| **NOTE** | Only on SDG7000A |

### 3.45.2 IQ:CENTerfreq

| | |
|---|---|
| **DESCRIPTION** | This command sets the center frequency of the I/Q modulator. |
| **COMMAND SYNTAX** | IQ:CENTerfreq <center_freq><unit><br><br><center_freq>:= the center frequency. Refer to the datasheet for the range of valid values.<br><unit>:= {Hz, kHz, MHz, GHz}. The default unit is Hertz “Hz”. |
| **QUERY SYNTAX** | IQ:CENTerfreq? |

| | |
|---|---|
| **RESPONSE FORMAT** | <center_freq> (expressed in Hz.) |
| **EXAMPLE** | Set the center frequency to 1 kHz:<br>*:IQ:CENTerfreq 1000Hz* |

### 3.45.3 IQ:SAMPlerate

| | |
|---|---|
| **DESCRIPTION** | This command sets the I/Q sampling rate. |
| **COMMAND SYNTAX** | IQ:SAMPlerate <sample_rate><unit><br><sample_rate>:= sample rate. Refer to the datasheet for the range of valid values.<br><unit>:= {Hz, kHz, MHz, GHz}. The default unit is Hertz “Hz”. |
| **QUERY SYNTAX** | IQ:SAMPlerate? |
| **RESPONSE FORMAT** | <sample_rate> (expressed in Hz.) |
| **EXAMPLE** | Set the sample rate to 100 kHz:<br>*:IQ:SAMPlerate 100000*<br>or:<br>*:IQ:SAMP 100kHz* |

### 3.45.4 IQ:SYMBolrate

| | |
|---|---|
| **DESCRIPTION** | This command sets the I/Q symbol rate. |
| **COMMAND SYNTAX** | IQ:SYMBolrate <symb_rate><unit><br><symb_rate>:= symbol rate. Refer to the datasheet for the range of valid values.<br><unit>:= {S/s, kS/s, MS/s}. The default unit is symbols-per-second “S/s”. |
| **QUERY SYNTAX** | IQ:SYMBolrate? |
| **RESPONSE FORMAT** | <symb_rate> (expressed in S/s.) |

| | |
|---|---|
| **EXAMPLE** | Set the symbol rate to 1 MS/s:<br>*:IQ:SYMB 1MS/s* |

### 3.45.5 IQ:AMPLitude

| | |
|---|---|
| **DESCRIPTION** | This command sets the I/Q amplitude. |
| **COMMAND SYNTAX** | IQ:AMPLitude <amplitude><unit><br><amplitude>:= amplitude. Refer to the datasheet for the range of valid values.<br><br><unit>:= {Vrms, mVrms, dBm}. The default unit is volts, root-mean-square “Vrms”. |
| **QUERY SYNTAX** | IQ:AMPLitude? |
| **RESPONSE FORMAT** | <amplitude> (expressed Vrms.) |
| **EXAMPLE** | Set the I/Q amplitude (sqrt($I^2$+$Q^2$)) to 0.2 Vrms:<br>*:IQ:AMPL 0.2* |

### 3.45.6 IQ:IQADjustment:GAIN

| | |
|---|---|
| **DESCRIPTION** | This command adjusts the ratio of I to Q while preserving the composite. |
| **COMMAND SYNTAX** | IQ:IQADjustment:GAIN <gain_ratio><br><gain_ratio>:= Gain ratio of I to Q. The default unit is dB. |
| **QUERY SYNTAX** | IQ:IQADjustment:GAIN? |
| **RESPONSE FORMAT** | <gain_ratio> (expressed in unit of dB.) |
| **EXAMPLE** | Set the gain ratio of I/Q to 0.1dB:<br>*:IQ:IQADjustment:GAIN 0.1* |

### 3.45.7 IQ:IQADjustment:IOFFset

| | |
|---|---|
| **DESCRIPTION** | This command adjusts the I channel offset value. |
| **COMMAND SYNTAX** | IQ:IQADjustment:IOFFset <offset><unit><br><offset>:= I offset.<br><unit>:= {V, mV, uV}. The default unit is volts "V". |
| **QUERY SYNTAX** | IQ:IQADjustment:IOFFset? |
| **RESPONSE FORMAT** | <offset> (expressed V.) |
| **EXAMPLE** | Set the I offset to 1 mV:<br>*IQ:IQADjustment:IOFFset 1mV* |

### 3.45.8 IQ:IQADjustment:QOFFset

| | |
|---|---|
| **DESCRIPTION** | This command adjusts the Q channel offset value. |
| **COMMAND SYNTAX** | IQ:IQADjustment:QOFFset <offset><unit><br><offset>:= Q offset.<br><unit>:= {V, mV, uV}. The default unit is volts "V". |
| **QUERY SYNTAX** | IQ:IQADjustment:QOFFset? |
| **RESPONSE FORMAT** | <offset> (expressed in V.) |
| **EXAMPLE** | Set the Q offset to -1 mV:<br>*IQ:IQAD:QOFF -0.001V* |

### 3.45.9 IQ:IQADjustment:QSKew

| | |
|---|---|
| **DESCRIPTION** | This command adjusts the phase angle (quadrature skew) between the I and Q vectors by increasing or decreasing the Q phase angle. |

| | |
|---|---|
| **COMMAND SYNTAX** | IQ:IQADjustment:QSKew <angle><br><angle>:= angle. The unit is degree. |
| **QUERY SYNTAX** | IQ:IQADjustment:QSKew? |
| **RESPONSE FORMAT** | <angle> (expressed in unit of degree.) |
| **EXAMPLE** | Set the Q angle to 1 degree:<br>*IQ:IQADjustment:QSKew 1.0* |

### 3.45.10 IQ:TRIGger:SOURce

| | |
|---|---|
| **DESCRIPTION** | This command sets the I/Q trigger source. |
| **COMMAND SYNTAX** | IQ:TRIGger:SOURce <src><br><br><src>:={INTernal,EXTernal,MANual} |
| **QUERY SYNTAX** | IQ:TRIGger:SOURce? |
| **RESPONSE FORMAT** | <src> |
| **EXAMPLE** | Set the trigger source to INT:<br>*IQ:TRIGger:SOURce INTernal* |

### 3.45.11 IQ:WAVEload:BUILtin

| | |
|---|---|
| **DESCRIPTION** | This command selects I/Q waveform from the built in waveform list. |
| **COMMAND SYNTAX** | IQ:WAVEload:BUILtin <wave_name><br><wave_name>:= {A waveform name from the table below}. |
| **QUERY SYNTAX** | IQ:WAVEload? |
| **RESPONSE FORMAT** | BUILtin\|USERstored <wave_name> |
| **EXAMPLE** | Set the I/Q waveform to built-in 2ASK:<br>*IQ:WAVE:BUIL “2ASK”* |

| | | | | |
|---|---|---|---|---|
| 2ASK | 4ASK | 8ASK | BPSK | 4PSK |
| 8PSK | DBPSK | 4DPSK | 8DPSK | 8QAM |
| 16QAM | 32QAM | 64QAM | 128QAM | 256QAM |

### 3.45.12 IQ:WAVEload:USERstored

| | |
|---|---|
| **DESCRIPTION** | This command selects I/Q waveform from the user stored waveforms. |
| **COMMAND SYNTAX** | Format1: IQ:WAVEload:USERstored "<wave_name>"<br><wave_name>:= { A waveform name from the user stored waveforms}.<br><br>Format2: IQ:WAVEload:USERstored<path><br><Path>: = {waveform path from user storage (local, network storage, USB flash disk), including file name and suffix}. |
| **QUERY SYNTAX** | IQ:WAVEload? |
| **RESPONSE FORMAT** | BUILtin\|USERstored <wave_name> |
| **EXAMPLE1** | Set the I/Q waveform to user stored UserIQ_1.arb:<br>*IQ:WAVEload:USERstored wave1.arb* |
| **EXAMPLE2** | Set the I/Q waveform as the user's locally stored waveform wave1.arb：<br>*IQ:WAVEload:USERstored "wave1.arb"*<br>*Or： IQ:WAVEload:USERstored "Local/wave1.arb"* |
| **EXAMPLE3** | Set the I / Q waveform to network storage waveform wave1.arb:<br>*IQ:WAVEload:USERstored "net_storage/wave/wave1.arb"* |
| **EXAMPLE4** | Set the I / Q waveform to the U disk storage waveform wave1.arb:<br>*IQ:WAVEload:USERstored "U-disk0/ wave/wave1.arb"* |

Note1:

(1) the path must be included in double quotation marks, for example: "net_storage/wave/wave1. Arb”. Please refer to file manager for specific available paths.

(2) If no path is specified, the default is the local path. If no suffix is added, the default is Wav suffix.

(3) SDG6000X, waveforms need to be stored in the specified path (Local/EasyIQ_arb), The wave file suffix is. arb

Note 2: the following table shows the availability of some commands in different SDG series.

| **Parameter /command** | **SDG 800** | **SDG 1000** | **SDG 2000X** | **SDG 5000** | **SDG 1000X** | **SDG 6000X** | **SDG 6000X-E** | **SDG 7000A** |
|---|---|---|---|---|---|---|---|---|
| USERstored | no | no | no | no | no | Format1 | no | Format2 |

### 3.45.13 IQ:FrequencySampling

| | |
|---|---|
| **DESCRIPTION** | This command sets the I/Q Frequency sampling rate. |
| **COMMAND SYNTAX** | IQ: FrequencySampling <sampling><br>< sampling >:= {1000-1250000000}. The unit is Hz |
| **QUERY SYNTAX** | IQ:FrequencySampling?<br><br>IQ:FrequencySamplingLimit? |
| **RESPONSE FORMAT** | <sampling><br>MAX,<max_sampling>,MIN,< min_sampling > |
| **EXAMPLE** | Set the I/Q frequency sampling to 2000000 Hz:<br>*IQ:FrequencySampling 2000000* |

## 3.46 File Operation Commands(Only SDG7000A)

### 3.46.1 MMEMory:DELete

| | |
|---|---|
| **DESCRIPTION** | This command deletes the file. |
| **COMMAND SYNTAX** | MMEMory:DELete <parameter> |
| | <parameter>:="The path of the file". |
| **QUERY SYNTAX** | |
| **RESPONSE FORMAT** | |
| **EXAMPLE** | Delete a file whose path is "Local/1000pts.bin": |
| | *MMEMory:DELete "Local/1000pts.bin"* |

### 3.46.2 MMEMory:RDIRectory

| | |
|---|---|
| **DESCRIPTION** | This command deletes the directory. |
| **COMMAND SYNTAX** | MMEMory:RDIRectory <parameter> |
| | <parameter>:="The path of the directory". |
| **QUERY SYNTAX** | |
| **EXAMPLE** | Delete a directory whose path is "Local/test": |
| | *MMEMory:RDIRectory "Local/test"* |

### 3.46.3 MMEMory:MDIRectory

| | |
|---|---|
| **DESCRIPTION** | This command creates a new directory. |
| **COMMAND SYNTAX** | MMEMory:MDIRectory <parameter> |
| | <parameter>:="The path of the directory". |
| **QUERY SYNTAX** | |

| | |
|---|---|
| **EXAMPLE** | Create a directory whose path is “Local/test”:<br>*MMEMory:MDIRectory “Local/test”* |

### 3.46.4 MMEMory:CATalog

| | |
|---|---|
| **DESCRIPTION** | This command checks the files and the directories from the path or checks the file with specific type. |
| **COMMAND SYNTAX** | MMEMory:CATalog? <parameter><br><parameter>:=”The path of the directory”.<br><br>MMEMory:CATalog:DATA:ARBitrary? <parameter><br><parameter>:=”The path of the directory”.<br><br>MMEMory:CATalog:STATe:XMLanguage? <parameter><br><parameter>:=”The path of the directory”. |
| **QUERY SYNTAX** | |
| **RESPONSE FORMAT** | remain space, used space<br>“File Name, File Type, File size” |
| **EXAMPLE** | Check the files and directories whose path is “Local/”:<br>*MMEMory:CATalog? “Local”*<br><br>Check the files with “.arb” or “.ARB” postfix whose path is “Local/”:<br>*MMEMory:CATalog:DATA:ARBitrary? “Local”*<br><br>Check the files with “.xml” or ”.XML” postfix whose path is “Local/”:<br>*MMEMory:CATalog:STATe:XMLanguage? “Local”* |

### 3.46.5 MMEMory:COPY

| | |
|---|---|
| **DESCRIPTION** | This command copies a file or a directory . |
| **COMMAND SYNTAX** | MMEMory:COPY <parameter><br><parameter>:= ”The path of the source”, “The path that the file is about to pasted to”. |

**QUERY SYNTAX**

**EXAMPLE**

Copy the file whose path is ”Local/test/1000pts.bin” and paste to “Local/1000pts.bin”

*MMEMory:COPY “Local/test/1000pts.bin”,”Local/1000pts.bin”*

Copy the directory whose path is ”Local/src” and paste to “Local/copy/”

*MMEMory:COPY “Local/src”, ”Local/copy”*

### 3.46.6 MMEMory:MOVE

**DESCRIPTION**

This command movesthe file or the directory to a new location.

**COMMAND SYNTAX**

MMEMory:MOVE<parameter>

<parameter>:= ”The path of the source”, “The path of the source that is about to move to”.

**QUERY SYNTAX**

**RESPONSE FORMAT**

**EXAMPLE**

Move the file whose path is ”Local/test/1000pts.bin” to “Local/1000pts.bin”

*MMEMory:MOVE “Local/test/1000pts.bin”,”Local/1000pts.bin”*

Move the directory whose path is ”Local/src” to “Local/copy/paste”

*MMEMory:MOVE “Local/src”, ”Local/copy/”*

### 3.46.7 MMEMory:SAVE:XML

**DESCRIPTION**

This command saves an xml configuration file to the default path or specified path

**COMMAND SYNTAX**

MMEMory:SAVE:XML<parameter>

<parameter>:= {save path, including file name and suffix}.

**QUERY SYNTAX**

**RESPONSE FORMAT**

**EXAMPLE**

Save the test.xml file to the local

*MMEMory:SAVE:XML "Local/test.xml" or*

*MMEMory:SAVE:XML "test.xml"*

Save the test.xml file to the network storage disk

*MMEMory:SAVE:XML "net_storage/test.xml"*

Save the test.xml file to the USB flash drive

*MMEMory:SAVE:XML "U-disk0/test.xml"*

### 3.46.8 MMEMory:LOAD:XML

**DESCRIPTION**

Load an xml configuration file from the default path or the specified path

**COMMAND SYNTAX**

MMEMory:LOAD:XML<parameter>

<parameter>:= {path, including file name and suffix}.

**QUERY SYNTAX**

**RESPONSE FORMAT**

**EXAMPLE**

Load the test.xml file locally

*MMEMory:SAVE:XML "Local/test.xml" or*

*MMEMory:SAVE:XML "test.xml"*

Load the test.xml file from the network storage disk

*MMEMory:SAVE:XML "net_storage/test.xml"*

Load the test.xml file from the USB flash drive

*MMEMory:SAVE:XML "U-disk0/test.xml"*

### 3.46.9 MMEMory:TRANsfer

**DESCRIPTION**

This command can send customized data to the specified bin file in the specified path

| | |
|---|---|
| **COMMAND SYNTAX** | MMEMory:TRANsfer < parameter >,#{data}<br><parameter>:= {path, including file name and suffix}.<br>{data}:= Length of data length+data length+binary data |
| **QUERY SYNTAX** | |
| **RESPONSE FORMAT** | |
| **EXAMPLE** | Send the data with the length of 1 and the length of 4 to the local wave1.bin<br>*MMEMory:TRANsfer "Local/wave1.bin",#14ABCD*<br><br>Send data with data length of 1 and data length of 4 to wave1.bin under USB flash disk<br>*MMEMory:TRANsfer "U-disk0/wave1.bin",#14ABCD*<br><br>Send data with data length of 1 and data length of 4 to wave1.bin under the network storage disk<br>*MMEMory:TRANsfer "net_storage /wave1.bin",#14ABCD* |

# 4 Waveform format

This chapter gives a description of the waveform file format used by the signal source. Through these instructions, you can learn how to customize and edit the waveform file.

Note: The following table shows the waveform file formats supported in each model

| Model / Format | SDG 800 | SDG 1000 | SDG 2000X | SDG 5000 | SDG 1000X | SDG 6000X/X-E | SDG 6000X-E | SDG 7000A |
|---|---|---|---|---|---|---|---|---|
| bin | y | y | y | y | y | y |  | y |
| csv |  |  | y |  | y | y |  | y |
| dat |  |  | y |  | y | y |  | y |
| mat |  |  |  |  |  |  |  | y |
| awg |  |  |  |  |  |  |  | y |
| hop |  |  |  |  |  |  |  | y |
| wav |  |  |  |  |  |  |  | y |
| arb |  |  |  |  |  | y |  | y |

## 4.1 bin

The bin file content is binary, and the file content is the codeword value of each point (codeword range - 32768~32767). Manual editing is not supported. When the machine imports the file, it maintains the current amplitude, frequency and offset information, and directly converts each codeword value into voltage output.

Note: The following table shows the bin file length of each model

| Model / param | SDG 800 | SDG 1000 | SDG 2000X | SDG 5000 | SDG 1000X | SDG 6000X-E | SDG 6000X | SDG 7000A |
|---|---|---|---|---|---|---|---|---|
| <length> | 32KB | 32KB | 4B~16MB | 32KB~ 1024KB | 4B~16MB | 4B~16MB | 4B~40MB | 4B~512M |
| codeword range | -32768~ 32767 | -32768~ 32767 | -32768~ 32767 | -32768~ 32767 | -32768~ 32767 | -32768~ 32767 | -32768~ 32767 | -32768~ 32767 |

## 4.2 csv/dat

The contents of the csv and dat files are text. The CSV file supported by SDG7000A is divided into header information and waveform data.

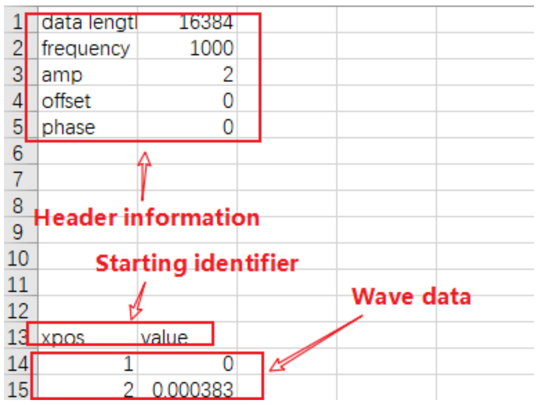
1. The content of the csv header information contains the following four items. There can be more information items, but the following four items must be included, and more items will be ignored. One line for each item, ending with a newline character.

| Header information | Describe |
|---|---|
| amp, value | Amplitude of waveform |
| offset, value | Offset of waveform |
| frequency, value | Frequency of waveform |
| data length, value | Length of waveform |

2. Each group of segment data occupies one row. One of the following four strings can be used as the starting identifier (identifier occupies one line), which is placed before the wave data.

| Starting identifier |
|---|
| Second, Volt |
| Xpos, value |
| Time, Ampl |
| Second, Value |

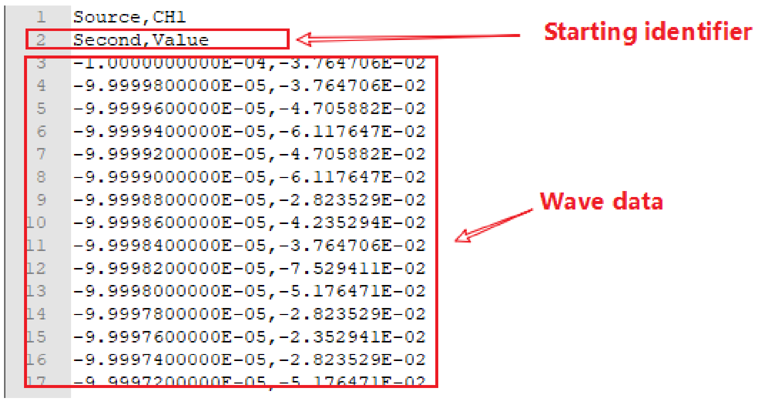
An example of csv format data is shown below:



| | | |
|---|---|---|
| 1 | data lengtl | 16384 |
| 2 | frequency | 1000 |
| 3 | amp | 2 |
| 4 | offset | 0 |
| 5 | phase | 0 |
| 6 | | |
| 7 | | |
| 8 | | |
| 9 | | |
| 10 | | |
| 11 | | |
| 12 | | |
| 13 | xpos | value |
| 14 | 1 | 0 |
| 15 | 2 | 0.000383 |

The header information is removed from the csv file, and only the data is retained, which is dat format.

An example of dat format data is shown below:



```
Source,CH1
Second,Value
-1.0000000000E-04,-3.764706E-02
-9.9999800000E-05,-3.764706E-02
-9.9999600000E-05,-4.705882E-02
-9.9999400000E-05,-6.117647E-02
-9.9999200000E-05,-4.705882E-02
-9.9999000000E-05,-6.117647E-02
-9.9998800000E-05,-2.823529E-02
-9.9998600000E-05,-4.235294E-02
-9.9998400000E-05,-3.764706E-02
-9.9998200000E-05,-7.529411E-02
-9.9998000000E-05,-5.176471E-02
-9.9997800000E-05,-2.823529E-02
-9.9997600000E-05,-2.352941E-02
-9.9997400000E-05,-2.823529E-02
-9.9997200000E-05,-5.176471E-02
```

## 4.3 mat

The mat file is composed of a fixed 128-byte file header information (mat_head) and two data units

The file header is represented by the following structure, and only the endian_Indicator needs to be concerned, SDG7000A only supports IM mode:

```
typedef struct
{
    char descriotive[116];
    char data_offset[8];
    uint16 version;             //Version of the mat file
    char endian_indicator[2];   // The value is IM or MI, indicates the size end mode
}cfg_mat_header_t;
```

There is a data header information (data_head, 88 bytes) at the beginning of each data unit, which is used to describe the number of bytes occupied by the data unit, data type, data block name and other information.

The data block header is represented by the following structure:

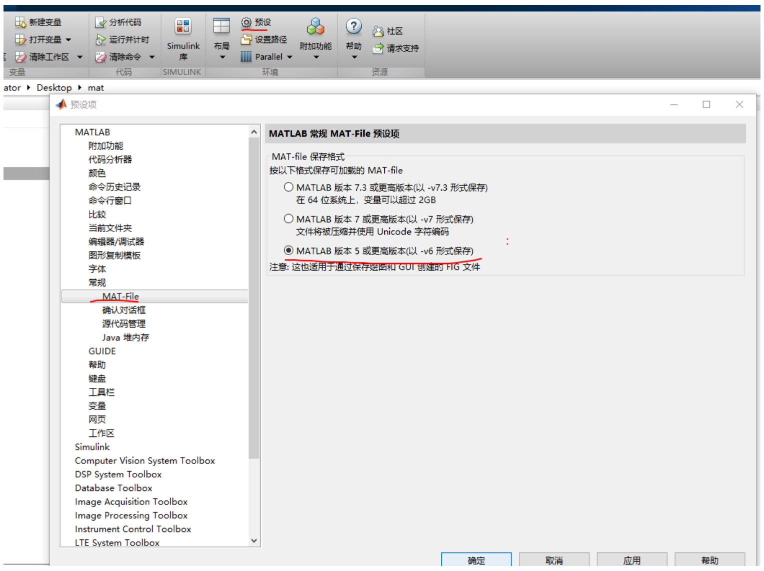
```
typedef struct
{
     u32 data_type;                // Value must be CFG_ MI_ MATRIX
     u32 array_len;
     u32 array_flag_data_type;
     u32 array_flag_data_len;
     u32 array_flag_data_1;
```

```
    u32 array_flag_data_2;
    u32 dimensions_array_data_type;
    u32 dimensions_array_data_len;
    u32 dimensions_array_data_1;
    u32 dimensions_array_data_2;
    u32 array_name_data_type;
    u32 array_name_data_len;
    char array_name_data[32];      // The oscilloscope export file is XX_Time or XX_data
    u32 data_tag_data_type;        // Type of waveform data
    u32 data_tag_data_len;         // Size of waveform data
}matlab_data_head_t;
```

XX_ data_ Type represents the data type, which corresponds to the following enumeration values:

```
typedef enum :
{
    CFG_MI_INT8=1,          //8 bit, signed
    CFG_MI_UINT8,           //8 bit, unsigned
    CFG_MI_INT16,           //16-bit, signed
    CFG_MI_UINT16,          //16-bit, unsigned
    CFG_MI_INT32,           //32-bit, signed
    CFG_MI_UINT32,          //32-bit, unsigned
    CFG_MI_SINGLE,          //IEEE 754 single format
    CFG_MI_RESERVED1,
    CFG_MI_DOUBLE,          //=9, IEEE 754 double format
    CFG_MI_RESERVED2,
    CFG_MI_RESERVED3,
    CFG_MI_INT64,           //=12, 64-bit, signed
    CFG_MI_UINT64,          //64-bit, unsigned
    CFG_MI_MATRIX,          //MATLAB array
    CFG_MI_COMPRESSED,      //Compressed Data
    CFG_MI_UTF8,            //Unicode UTF-8 Encoded Character Data
    CFG_MI_UTF16,           //Unicode UTF-16 Encoded Character Data
    CFG_MI_UTF32,           //Unicode UTF-32 Encoded Character Data
}cfg_mat_data_type_t;
```

Mat files exported by matlab are in compressed format by default, and need to be saved in non-compressed format as follows. And only two data segment files are supported. Time data needs double, and waveform data needs single



MATLAB
附加功能
代码分析器
颜色
命令历史记录
命令行窗口
比较
当前文件夹
编辑器/调试器
图形复制模板
字体
常规
MAT-File
确认对话框
源代码管理
Java 堆内存
GUIDE
帮助
键盘
工具栏
变量
网页
工作区
Simulink
Computer Vision System Toolbox
DSP System Toolbox
Database Toolbox
Image Acquisition Toolbox
Image Processing Toolbox
Instrument Control Toolbox
LTE System Toolbox

MATLAB 常规 MAT-File 预设项

MAT-file 保存格式
按以下格式保存可加载的 MAT-file
- ○ MATLAB 版本 7.3 或更高版本(以 -v7.3 形式保存)
  在 64 位系统上，变量可以超过 2GB
- ○ MATLAB 版本 7 或更高版本(以 -v7 形式保存)
  文件将被压缩并使用 Unicode 字符编码
- [✓ MATLAB 版本 5 或更高版本(以 -v6 形式保存)]

注意: 这也适用于通过保存绘图和 GUI 创建的 FIG 文件

## 4.4 awg

The content of the awg file is text. The awg file is divided into the following two parts:

1. Sequence setting information. The following fields can be included (not required) to describe the sequence setting. The following fields are not required. Missing some fields will not affect the file recall. The missing parameter will remain the current parameter. There may also be redundant fields, which will be ignored.

| Fields | Describe |
|---|---|
| amplitude_scale, value | Amplitude of sequence. Value={Number between 0-1, data type is float} |
| run_mode, mode | Run mode. mode =<br>{ countinuous ,tcontinuous ,burst ,step ,advance } |
| burst_count, value | The number of burst count at a single trigger. Value=integer. |
| timer_interval, value | The timing time when the timer is triggered. Value=float. |
| trigger_mode, mode | Trigger mode.<br>mode={ button ,timer ,external } |
| burst_hold_type, type | Type of hold value in burst of run mode.<br>type ={ Zero ,Start Value ,End Value } |
| trigger_edge, edge | Trigger edge polarity in case of external trigger.<br>Edge={ raise ,fall } |

| | |
|---|---|
| interp_mode, mode | Interpolation algorithm type.<br>Mode = { zero_hold ,linear ,sinc ,sinc_27 ,sinc_13 } |
| increasing_strategy, value | Interpolation strategy.<br>value={ interpolation ,return_zer ,hold_last ,multiplication } |
| decreasing_strategy, value | Decreasing_strategy .value={ decimation ,cut_tail ,cut_head } |
| mark_skew, value | Value of maker_skew.Value=float |
| file_type, SDS | The siglent oscilloscope exports the awg file flag. (Only the awg file exported by the oscilloscope has this field). If there is this field, the waveform file described by the segment information should be in the same directory as the awg file. |
| start_play_segment,value | Output start segment. value=integer. Without this field, the sequence will output from the first segment |

2. Segment information. Each segment information occupies one line. The parameters are separated by commas. The information of each segment is divided into segments_ N starts. Where N is the segment number, starting from 0 and adding 1 successively.

   There are two formats for segment information, depending on whether the field file is included file_name.

   Format 1: Include file_name, Store the name and value of each parameter in a certain order. The format is as follows:

   segment_0,file_name,2_stairup_ram.bin,offset,0,amplitude,4,cfg_len,32768,repetition,1,goto,0, goto_mode,next,marker_switch,1,marker_pos,0,segment_numb,0,segment_store_addr,0,wait_ event, none.

   Value of goto_mode can be one of them: next, item.

   Value of wait_event can be one of them:none, button, timer, external.

   Format 2: No file_name Store the value of each parameter in a certain order. (no parameter name). The format is as follows:

   segment_0,C1_seg00001.csv,-0.047059,4.141177,50000,1,-1,next,0,0.

Definition of parameters: Segment number (segment_0), ile name (C1_seg00001.csv), offset (-0.047059), amplitude (4.141177), wave points (50000), Repetitions (1), Go to segment number (-1, Because the next parameter jump mode is next), jump (next), marker_switch (0), marker_pos (0).

## 4.5 hop

The hop file is used to store the frequency hopping sequence. The file is divided into three data blocks to store three tables: frequency list, order list, and frequency avoid list. The file content is text, which can be edited manually

The Hop file has three data blocks, which are stored in different lines. Each line stores one parameter or data.

The first line is file version number: **Ver:1.0**

The second line is frequency list version number: **FreqListVer:1.0**

The third line is the data starting mark of the frequency list: **freq_list_ start**

The following is the frequency list data

The last line of frequency list ends with an end tag: **freq_list_end**

Next is the order list and frequency avoid list. The structure is the same as the frequency list, but the keywords are different, as follows:

**OrderListVer:1.0** -- Version number of order list

**order_list_start** -- Start tag of order list data

**order_list_end** -- End tag of order list data

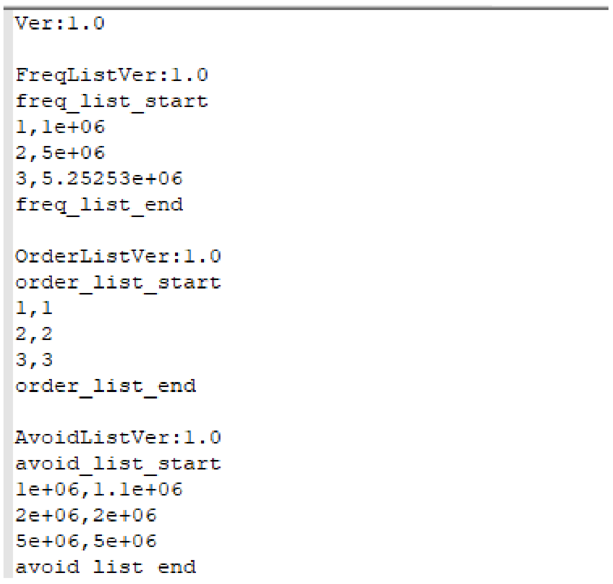
**AvoidListVer:1.0** -- Version number of frequency avoid list

**avoid_list_start** -- Start tag of frequency avoid list data

**avoid_list_end** -- End tag of frequency avoid list data

Two data in each row of the data block are separated by commas.

Example:



```
Ver:1.0

FreqListVer:1.0
freq_list_start
1,1e+06
2,5e+06
3,5.25253e+06
freq_list_end

OrderListVer:1.0
order_list_start
1,1
2,2
3,3
order_list_end

AvoidListVer:1.0
avoid_list_start
1e+06,1.1e+06
2e+06,2e+06
5e+06,5e+06
avoid_list_end
```

## 4.6 wav/arb

The wav/arb file is an IQ waveform file. The wav/arb file consists of two parts: header and data. The header is text data. Waveform data is binary data.

The header must end with "IQData,". From the beginning of the file to "IQData," is the file information in text format. "IQData," is followed by binary waveform data.

When loading the wav/arb file, the following keywords (key_str) will be read from the header for parsing. If some of the following keywords do not exist in the header, the corresponding information will be set to the default value or left blank. If there is extra information, it will be ignored. " key_ Str, value "form a group of description information, and each group of description information is separated by commas. The following is a header instance of a wav/arb file and the meaning of each keyword FileType,IQ,Version,2.0,FileName,test.ARB,DataSourceType,PN9,SymbolLength,512,SymbolRate,1000000,ModulationType,16QAM,FilterType,RootCosine,FilterBandwidth,0,FilterAlpha,0.5,FilterLength,32,OverSampling,2,ActualSampleLength,512,SampleRate,2000000.000000,RMS,0.684953707203608,DataLength,1024,IQData.

| keyword | Describe |
|---|---|
| FileType | Type of file (IQ) |
| Version | Version number |
| DataSourceType | |
| SymbolLength | Symbol length |

| | |
|---|---|
| SymbolRate | Symbol rate |
| ModulationType | Modulation type |
| FilterType | Filter type |
| FilterBandwidth | |
| FilterAlpha | Filter Alpha |
| FilterLength | Filter length |
| OverSampling | Oversampling |
| ActualSampleLength | |
| SampleRate | sampling rate |
| DataLength | |
| RMS | |
| IQData | |

# 5 Programming Examples

This chapter gives some examples for the programmer. In these examples, you can see how to use VISA or sockets, in combination with the commands described above to control the generator. By following these examples, you can develop many more applications.

## 5.1 Examples of Using VISA

### 5.1.1 VC++ Example

**Environment:** Windows 7 32-bit, Visual Studio.

**Description:** Query the instrument information using "*IDN?" command over NI-VISA, with the access through USBTMC and TCP/IP separately.

**Steps:**

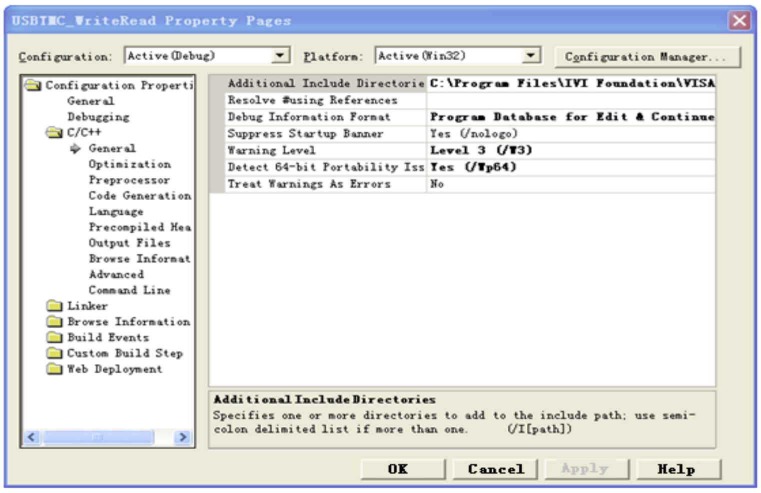
1. Open Visual Studio and create a new VC++ win32 console project.
2. Set the project environment to use the NI-VISA lib, there are two ways to specify NI-VISA, static or automatic:
    a) Static:

       Find the files visa.h, visatype.h, and visa32.lib in the NI-VISA installation path, copy them to the root path of the VC++ project, and add them to the project. In the project name.cpp file, add the following two lines:

       ```
       #include "visa.h"
       #pragma comment(lib,"visa32.lib")
       ```

    b) Dynamic:

       In "project---properties---c/c++---General---Additional Include Directories" set the value to the NI-VISA installation path (e.g. C:\Program Files\IVI Foundation\VISA\WinNT\include), as shown in the figure below:


USBTMC_WriteRead Property Pages

Configuration: Active(Debug) Platform: Active(Win32) Configuration Manager...

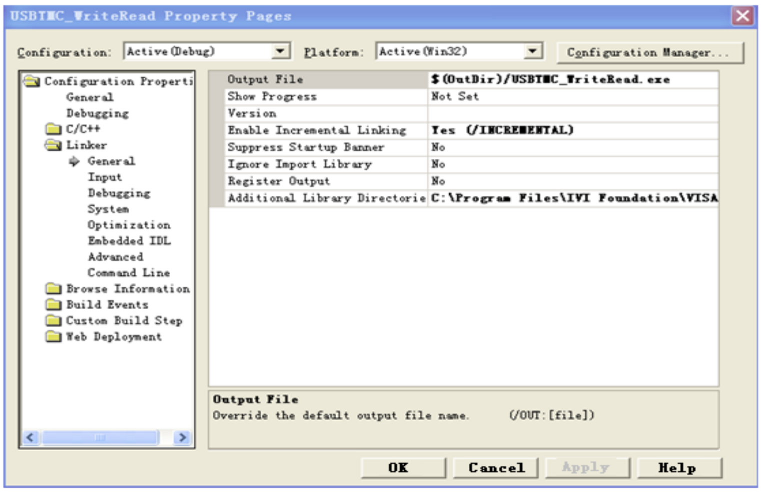
Configuration Properti
- General
- Debugging
- C/C++
  - General
  - Optimization
  - Preprocessor
  - Code Generation
  - Language
  - Precompiled Hea
  - Output Files
  - Browse Informat
  - Advanced
  - Command Line
- Linker
- Browse Information
- Build Events
- Custom Build Step
- Web Deployment

| | |
|---|---|
| Additional Include Directorie | C:\Program Files\IVI Foundation\VISA |
| Resolve #using References | |
| Debug Information Format | Program Database for Edit & Continue |
| Suppress Startup Banner | Yes (/nologo) |
| Warning Level | Level 3 (/W3) |
| Detect 64-bit Portability Iss | Yes (/Wp64) |
| Treat Warnings As Errors | No |

**Additional Include Directories**
Specifies one or more directories to add to the include path; use semi-colon delimited list if more than one. (/I[path])

OK Cancel Apply Help

In " project---properties---Linker---General---Additional Library Directories " set the value to the NI-VISA installation path (e.g. C:\Program Files\IVI Foundation\VISA\WinNT\include), as shown in the figure below:


USBTMC_WriteRead Property Pages

Configuration: Active(Debug) Platform: Active(Win32) Configuration Manager...

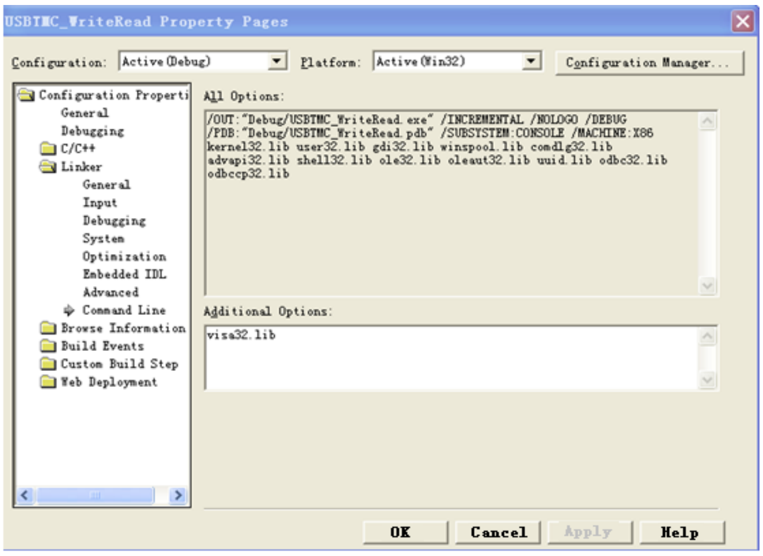
Configuration Properti
- General
- Debugging
- C/C++
- Linker
  - General
  - Input
  - Debugging
  - System
  - Optimization
  - Embedded IDL
  - Advanced
  - Command Line
- Browse Information
- Build Events
- Custom Build Step
- Web Deployment

| | |
|---|---|
| Output File | $(OutDir)/USBTMC_WriteRead.exe |
| Show Progress | Not Set |
| Version | |
| Enable Incremental Linking | Yes (/INCREMENTAL) |
| Suppress Startup Banner | No |
| Ignore Import Library | No |
| Register Output | No |
| Additional Library Directorie | C:\Program Files\IVI Foundation\VISA |

**Output File**
Override the default output file name. (/OUT:[file])

OK Cancel Apply Help

In "project---properties---Linker---Command Line---Additional" set the value to visa32.lib, as shown in the figure below:


USBTMC_WriteRead Property Pages

Configuration: Active(Debug)  Platform: Active(Win32)

All Options:

```
/OUT:"Debug/USBTMC_WriteRead.exe" /INCREMENTAL /NOLOGO /DEBUG
/PDB:"Debug/USBTMC_WriteRead.pdb" /SUBSYSTEM:CONSOLE /MACHINE:X86
kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib
odbccp32.lib
```

Additional Options:

```
visa32.lib
```

Include visa.h file in the projectname.cpp file:

```
#include <visa.h>
```

3. Coding:

a) USBTMC:

```
int Usbtmc_test()
{
        /* This code demonstrates sending synchronous read & write commands */
        /* to an USB Test & Measurement Class (USBTMC) instrument using      */
        /* NI-VISA                                  */
        /* The example writes the "*IDN?\n" string to all the USBTMC          */
        /* devices connected to the system and attempts to read back          */
        /* results using the write and read functions.                        */
        /* The general flow of the code is   */
        /*      Open Resource Manager              */
        /*      Open VISA Session to an Instrument                            */
        /*      Write the Identification Query Using viPrintf                   */
        /*      Try to Read a Response With viScanf                                     */
        /*      Close the VISA Session            */
```

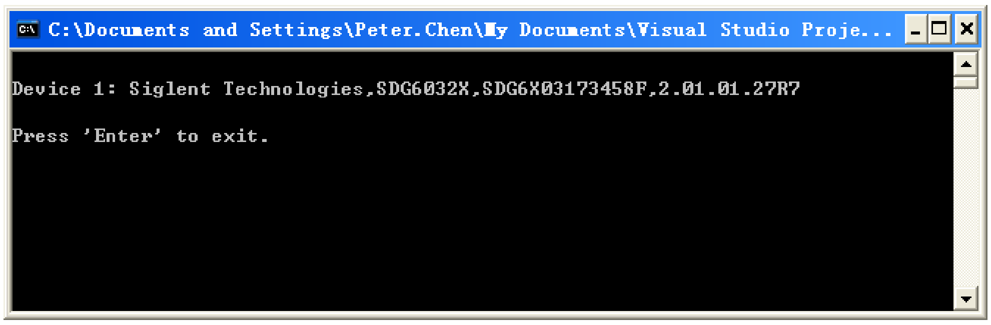
```
/***********************************************************/
ViSession   defaultRM;
ViSession   instr;
ViUInt32   numInstrs;
ViFindList   findList;
ViStatus   status;
char      instrResourceString[VI_FIND_BUFLEN];
unsignedchar   buffer[100];
int        i;
/** First we must call viOpenDefaultRM to get the manager
* handle.   We will store this handle in defaultRM.*/
status=viOpenDefaultRM (&defaultRM);
if (status<VI_SUCCESS)
{
          printf ("Could not open a session to the VISA Resource Manager!\n");
          return   status;
}
/* Find all the USB TMC VISA resources in our system and store the   number of
resources in the system in numInstrs.                              */
status   =   viFindRsrc   (defaultRM,   "USB?*INSTR",   &findList,   &numInstrs,
instrResourceString);
if (status<VI_SUCCESS)
{
         printf ("An error occurred while finding resources.\nPress 'Enter' to continue.");
          fflush(stdin);
          getchar();
          viClose (defaultRM);
          return   status;
}
```

```
/** Now we will open VISA sessions to all USB TMC instruments.
* We must use the handle from viOpenDefaultRM and we must
* also use a string that indicates which instrument to open.   This
* is called the instrument descriptor.   The format for this string
* can be found in the function panel by right-clicking on the
* descriptor parameter. After opening a session to the
* device, we will get a handle to the instrument which we
* will use in later VISA functions.   The AccessMode and Timeout
* parameters in this function are reserved for future
* functionality.   These two parameters are given the value VI_NULL.*/
for (i=0; i<int(numInstrs); i++)
{
        if (i> 0)
        {
                viFindNext (findList, instrResourceString);
        }
       status = viOpen (defaultRM, instrResourceString, VI_NULL, VI_NULL, &instr);
        if (status<VI_SUCCESS)
        {
                printf ("Cannot open a session to the device %d.\n", i+1);
                continue;
        }
        /* * At this point we now have a session open to the USB TMC instrument.
        * We will now use the viPrintf function to send the device the string "*IDN?\n",
        * asking for the device's identification.   */
        char * cmmand ="*IDN?\n";
        status = viPrintf   (instr, cmmand);
        if (status<VI_SUCCESS)
        {
```

```
                        printf ("Error writing to the device %d.\n", i+1);
                        status = viClose (instr);
                        continue;
                }
                /** Now we will attempt to read back a response from the device to
                * the identification query that was sent.   We will use the viScanf
                * function to acquire the data.
                * After the data has been read the response is displayed.*/
                status =   viScanf(instr, "%t", buffer);
                if (status<VI_SUCCESS)
                {
                        printf ("Error reading a response from the device %d.\n", i+1);
                }
                else
                {
                        printf ("\nDevice %d: %s\n", i+1 , buffer);
                }
                status = viClose (instr);

        }
        /** Now we will close the session to the instrument using
        * viClose. This operation frees all system resources.                    */
        status = viClose (defaultRM);
        printf("Press 'Enter' to exit.");
        fflush(stdin);
        getchar();
        return 0;
}
```

```
int _tmain(int argc, _TCHAR* argv[])

{

        Usbtmc_test();

        return 0;

}
```

**Run result:**

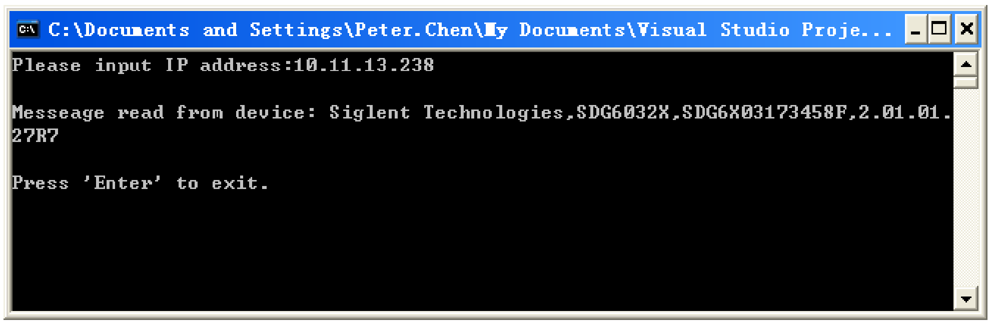


```
C:\Documents and Settings\Peter.Chen\My Documents\Visual Studio Proje...
Device 1: Siglent Technologies,SDG6032X,SDG6X03173458F,2.01.01.27R7
Press 'Enter' to exit.
```

b) TCP/IP:

```
int TCP_IP_Test(char *pIP)

{

        char outputBuffer[VI_FIND_BUFLEN];

        ViSession defaultRM, instr;

        ViStatus status;

        /* First we will need to open the default resource manager. */

        status = viOpenDefaultRM (&defaultRM);

        if (status<VI_SUCCESS)

        {

                printf("Could not open a session to the VISA Resource Manager!\n");

        }

        /* Now we will open a session via TCP/IP device */

        char head[256] ="TCPIP0::";

        char tail[] ="::INSTR";
        strcat(head,pIP);

        strcat(head,tail);

        status = viOpen (defaultRM, head, VI_LOAD_CONFIG, VI_NULL, &instr);

        if (status<VI_SUCCESS)

        {

                printf ("An error occurred opening the session\n");

                viClose(defaultRM);

        }

        status = viPrintf(instr, "*idn?\n");

        status = viScanf(instr, "%t", outputBuffer);

        if (status<VI_SUCCESS)

        {

                printf ("viRead failed with error code: %x \n",status);

                viClose(defaultRM);

        }

        else

        {

                printf ("\nMesseage read from device: %*s\n", 0,outputBuffer);

        }

        status = viClose (instr);

        status = viClose (defaultRM);

        printf("Press 'Enter' to exit.");

        fflush(stdin);

        getchar();

        return 0;

}


int _tmain(int argc, _TCHAR* argv[])

{
        printf("Please input IP address:");
        char ip[256];
        fflush(stdin);
        gets(ip);
        TCP_IP_Test(ip);
        return 0;
}
```

**Run result:**



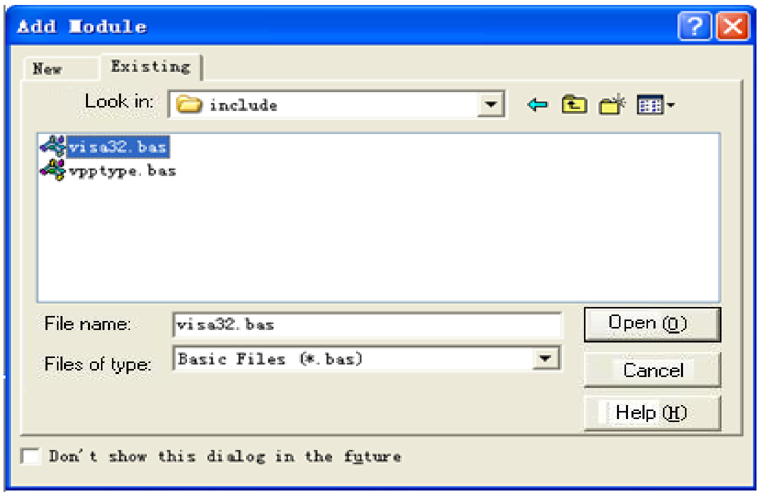
```
C:\Documents and Settings\Peter.Chen\My Documents\Visual Studio Proje...
Please input IP address:10.11.13.238

Messeage read from device: Siglent Technologies,SDG6032X,SDG6X03173458F,2.01.01.
27R7

Press 'Enter' to exit.
```

VB Example**Environment:** Windows 7 32-bit, Microsoft Visual Basic 6.0

**Description:** Query the instrument information using the “*IDN?" command over NI-VISA, via USBTMC and TCP/IP separately.

**Steps:**

1. Open Visual Basic, and build a standard application program project.
2. Set the project environment to use the NI-VISA lib: Click the Existing tab of Project>>Add Existing Item, search the visa32.bas file in the “include” folder under the NI-VISA installation path and add the file, as shown in the figure below:



3. Coding:

a) USBTMC:

```
Private Function Usbtmc_test() As Long
    ' This code demonstrates sending synchronous read & write commands
    ' to an USB Test & Measurement Class (USBTMC) instrument using
    ' NI-VISA
    ' The example writes the "*IDN?\n" string to all the USBTMC
    ' devices connected to the system and attempts to read back
    ' results using the write and read functions.
    ' The general flow of the code is
    '       Open Resource Manager
    '       Open VISA Session to an Instrument
    '       Write the Identification Query Using viWrite
    '       Try to Read a Response With viRead
    '       Close the VISA Session
    Const MAX_CNT = 200

    Dim defaultRM As Long
    Dim instrsesn As Long
    Dim numInstrs As Long
```

```
 Dim findList As Long
 Dim retCount As Long
 Dim status As Long
 Dim instrResourceString As String * VI_FIND_BUFLEN
 Dim Buffer As String * MAX_CNT
 Dim I As Integer

' First we must call viOpenDefaultRM to get the manager
' handle. We will store this handle in defaultRM.
 status = viOpenDefaultRM(defaultRM)
 If (status < VI_SUCCESS) Then
     resultTxt.Text =“"Could not open a session to the VISA Resource Manager”"
     Usbtmc_test = status
     Exit Function
 End If

' Find all the USB TMC VISA resources in our system and store the
' number of resources in the system in numInstrs.
 status = viFindRsrc(defaultRM,“"USB?*INST”", findList, numInstrs, instrResourceString)
 If (status < VI_SUCCESS) Then
     resultTxt.Text =“"An error occurred while finding resources”"
     viClose(defaultRM)
     Usbtmc_test = status
     Exit Function
 End If

' Now we will open VISA sessions to all USB TMC instruments.
' We must use the handle from viOpenDefaultRM and we must
' also use a string that indicates which instrument to open.   This
' is called the instrument descriptor.   The format for this string
' can be found in the function panel by right-clicking on the
' descriptor parameter. After opening a session to the
' device, we will get a handle to the instrument which we
' will use in later VISA functions.   The AccessMode and Timeout
' parameters in this function are reserved for future
' functionality.   These two parameters are given the value VI_NULL.
 For i = 0 To numInstrs
     If (i > 0) Then
         status = viFindNext(findList, instrResourceString)
     End If
     status = viOpen(defaultRM, instrResourceString, VI_NULL, VI_NULL, instrsesn)
     If (status < VI_SUCCESS) Then
         resultTxt.Text ="“Cannot open a session to the device“" + CStr(i + 1)
         GoTo NextFind
     End If

    ' At this point we now have a session open to the USB TMC instrument.
    ' We will now use the viWrite function to send the device the string“"*IDN”",
    ' asking for the devic''s identification.
     status = viWrite(instrsesn,“"*IDN”", 5, retCount)
     If (status < VI_SUCCESS) Then
         resultTxt.Text =“"Error writing to the device”"
         status = viClose(instrsesn)
         GoTo NextFind
     End If
    ' Now we will attempt to read back a response from the device to
    ' the identification query that was sent.   We will use the viRead
    ' function to acquire the data.
```

```
            '' After the data has been read the response is displayed.
             status = viRead(instrsesn, Buffer, MAX_CNT, retCount)
             If (status < VI_SUCCESS) Then
                 resultTxt.Text =“"Error reading a response from the device”" + CStr(i + 1)
             Else
                 resultTxt.Text =“"Read from device:“" + CStr(i + 1) +“"“" + Buffer
             End If
             status = viClose(instrsesn)
         Next i

        '' Now we will close the session to the instrument using
        '' viClose. This operation frees all system resources.
         status = viClose(defaultRM)
         Usbtmc_test = 0
     End Function
```

b) TCP/IP:

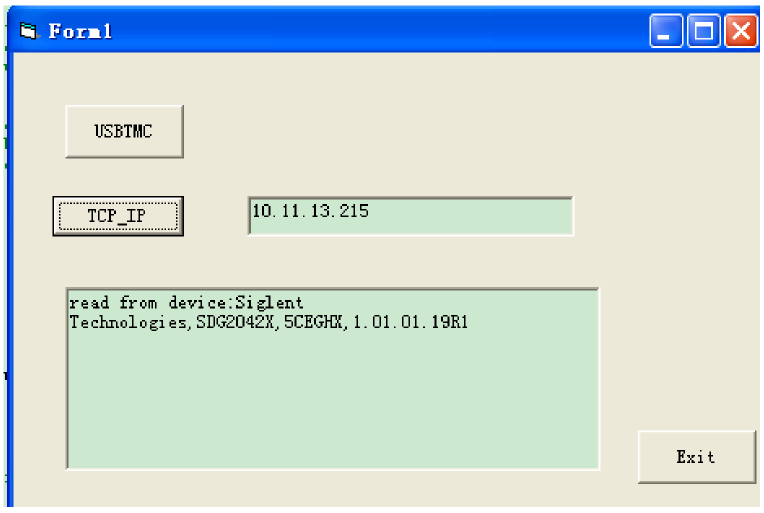
```
     Private Function TCP_IP_Test(ByVal ip As String) As Long
         Dim outputBuffer As String * VI_FIND_BUFLEN
         Dim defaultRM As Long
         Dim instrsesn As Long
         Dim status As Long
         Dim count As Long

        '' First we will need to open the default resource manager.
         status = viOpenDefaultRM(defaultRM)
         If (status < VI_SUCCESS) Then
             resultTxt.Text =“"Could not open a session to the VISA Resource Manager”"
```

```
            TCP_IP_Test = status
            Exit Function
        End If


      '' Now we will open a session via TCP/IP device
        status = viOpen(defaultRM,“"TCPIP0:”" + ip +“"::INST”", VI_LOAD_CONFIG, VI_NULL,
    instrsesn)
        If (status < VI_SUCCESS) Then
            resultTxt.Text =“"An error occurred opening the sessio”"
            viClose(defaultRM)
            TCP_IP_Test = status
            Exit Function
        End If


        status = viWrite(instrsesn,“"*IDN”", 5, count)
        If (status < VI_SUCCESS) Then
            resultTxt.Text =“"Error writing to the device”"
        End If
        status = viRead(instrsesn, outputBuffer, VI_FIND_BUFLEN, count)
        If (status < VI_SUCCESS) Then
            resultTxt.Text =“"Error reading a response from the device”" + CStr(i + 1)
        Else
            resultTxt.Text =“"read from device”" + outputBuffer
        End If
        status = viClose(instrsesn)
        status = viClose(defaultRM)
        TCP_IP_Test = 0
    End Function
```

c) Button control code:

```
Private Sub exitBtn_Click()
    End
End Sub
Private Sub tcpipBtn_Click()
    Dim stat As Long
    stat = TCP_IP_Test(ipTxt.Text)
    If (stat < VI_SUCCESS) Then
        resultTxt.Text = Hex(stat)
    End If
End Sub
Private Sub usbBtn_Click()
    Dim stat As Long
    stat = Usbtmc_test
    If (stat < VI_SUCCESS) Then
        resultTxt.Text = Hex(stat)
    End If
End Sub
```

**Run result:**



```
10.11.13.215
```

```
read from device:Siglent
Technologies,SDG2042X,5CEGHX,1.01.01.19R1
```

MATLAB Example**Environment:** Windows 7 32-bit, MATLAB R2013a

**Description:** Query the instrument information using the "*IDN?" command over NI-VISA, with the access through USBTMC and TCP/IP separately.

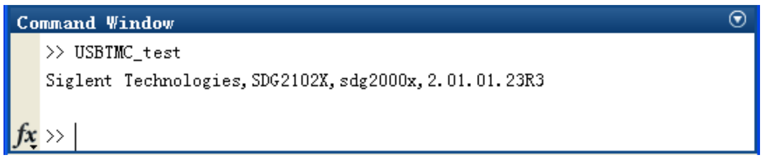
**Steps:**

1. Open MATLAB, and modify the current directory. In this demo, the current directory is modified to "D:\USBTMC_TCPIP_Demo".
2. Click File>>New>>Script in the Matlab interface to create an empty M file.
3. Coding:
   a) USBTMC:

```
function USBTMC_test()
% This code demonstrates sending synchronous read & write commands
% to an USB Test & Measurement Class (USBTMC) instrument using
% NI-VISA

%Create a VISA-USB object connected to a USB instrument
vu = visa('ni','USB0::0xF4ED::0xEE3A::sdg2000x::INSTR');

%Open the VISA object created
fopen(vu);

%Send the string "*IDN?",asking for the device's identification.
fprintf(vu,'*IDN?');

%Request the data
outputbuffer = fscanf(vu);
disp(outputbuffer);
```

```
%Close the VISA object
fclose(vu);
delete(vu);
clear vu;

end
```

**Run result:**



```
Command Window
>> USBTMC_test
Siglent Technologies,SDG2102X,sdg2000x,2.01.01.23R3

>>
```

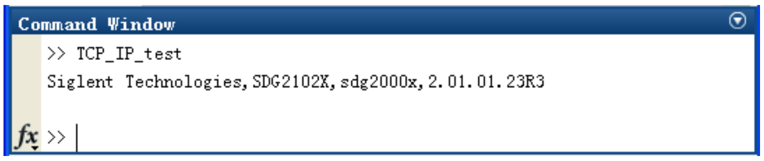
b) TCP/IP:

Write a function TCP_IP_Test:

```
function TCP_IP_test()
% This code demonstrates sending synchronous read & write commands
% to a TCP/IP instrument using NI-VISA

%Create a VISA-TCPIP object connected to an instrument
%configured with IP address.
vt = visa('ni',['TCPIP0::','10.11.13.32','::INSTR']);

%Open the VISA object created
fopen(vt);

%Send the string "*IDN?",asking for the device's identification.
fprintf(vt,'*IDN?');

%Request the data
```

```
outputbuffer = fscanf(vt);
disp(outputbuffer);

%Close the VISA object
fclose(vt);
delete(vt);
clear vt;

end
```

**Run result:**



```
Command Window
>> TCP_IP_test
Siglent Technologies,SDG2102X,sdg2000x,2.01.01.23R3

fx >>
```

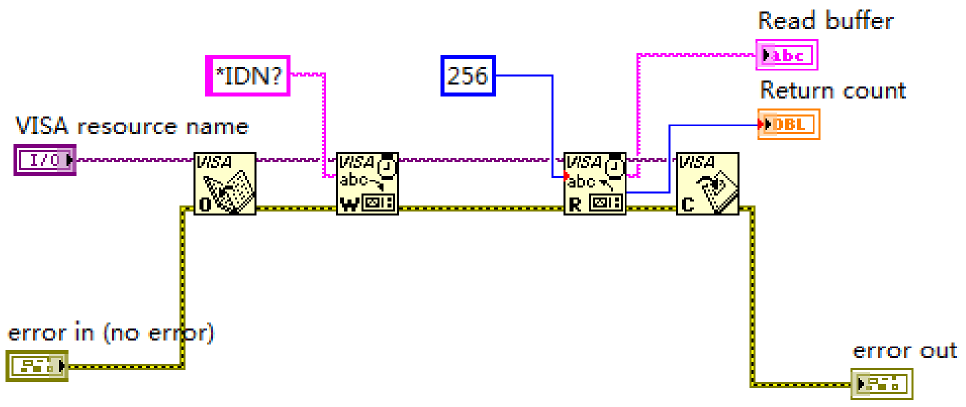
### 5.1.2 LabVIEW Example

**Environment:** Windows 7 32-bit, LabVIEW 2011

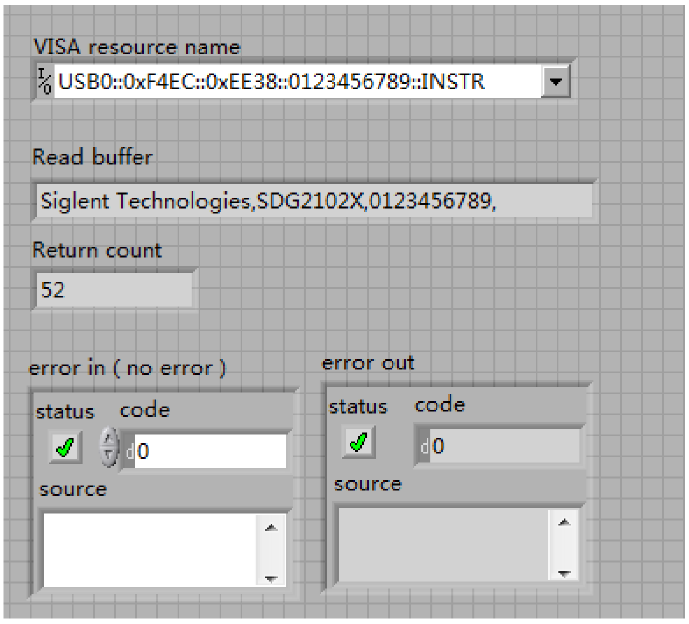
**Description:** Query the instrument information using the "*IDN?" command over NI-VISA, with the access through USBTMC and TCP/IP separately.

**Steps:**

1. Open LabVIEW, and create a VI file.
2. Add controls. Right-click in the **Front Panel** interface, select and add **VISA resource name**, error in, error out and some indicators from the Controls column.
3. Open the **Block Diagram** interface. Right-click on the **VISA resource name**, select and add the following functions from VISA Palette from the pop-up menu: **VISA Write**, **VISA Read**, **VISA Open,** and **VISA Close**.
4. The connection is as shown in the figure below:



5. Select the device resource from the VISA Resource Name list box and run the program.

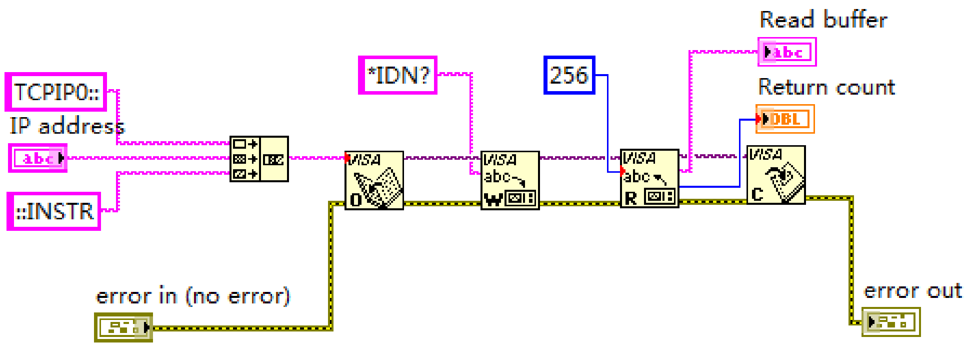


VISA resource name
USB0::0xF4EC::0xEE38::0123456789::INSTR
Read buffer
Siglent Technologies,SDG2102X,0123456789,
Return count
52
error in ( no error )
status ✓ code 0
source
error out
status ✓ code 0
source

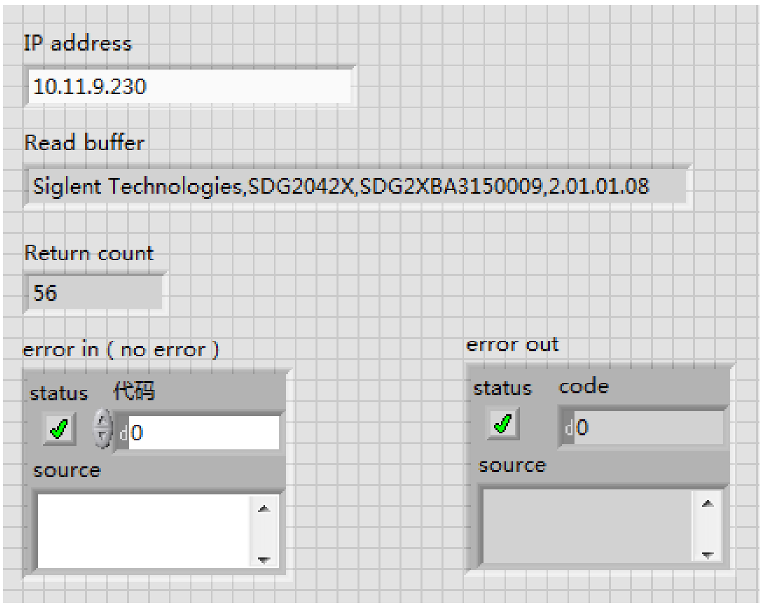
In this example, the VI opens a VISA session to a USBTMC device, writes a "*IDN?" command to the device, and reads back the response. After all communication is complete, the VI closes the VISA session.

6. Communicating with the device via TCP/IP is similar to USBTMC. But you need to change VISA Write and VISA Read Function to Synchronous I/O. The LabVIEW default is asynchronous I/O. Right-click the node and select Synchronous I/O Mod>>Synchronous from the shortcut menu to write or read data synchronously.

7. The connection is as shown in the figure below:



8. Input the IP address and run the program.


IP address
10.11.9.230
Read buffer
Siglent Technologies,SDG2042X,SDG2XBA3150009,2.01.01.08
Return count
56
error in ( no error )
status 代码
0
source
error out
status code
0
source

## 5.1.3 Python2 Example

**Environment:** Python2.7, PyVISA 1.4

(Please install PyVISA after installing Python2.7. Please refer to https://pyvisa.readthedocs.io/en/stable/getting.html for the PyVISA installation guide.

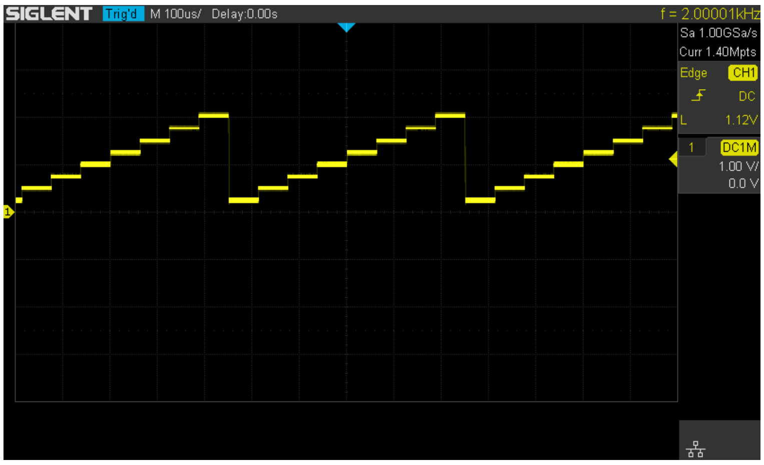
**Description**: Use Python script to build an 8-point 16-bit arbitrary waveform (0x1000, 0x2000, 0x3000, 0x4000, 0x5000, 0x6000, 0x7000, 0x7fff) and save the waveform data in "wave1.bin", then download it to the instrument, finally read it back from the instrument and save it as "wave2.bin".

Below is the code of the script:

```
#!/usr/bin/env python2.7
# -*- coding: utf-8 -*-

import visa
import time
import binascii

#USB resource of Device
device_resource = "USB0::0xF4EC::0x1101::#15::INSTR"

#Little endian, 16-bit 2's complement
wave_points = [0x0010, 0x0020, 0x0030, 0x0040, 0x0050, 0x0060, 0x0070, 0xff7f]

def create_wave_file():
    """create a file"""
    f = open("wave1.bin", "wb")
    for a in wave_points:
        b = hex(a)
        b = b[2:]
        len_b = len(b)
        if (0 == len_b):
            b = '0000'
        elif (1 == len_b):
            b = '000' + b
        elif (2 == len_b):
            b = '00' + b
        elif (3 == len_b):
            b = '0' + b
        c = binascii.a2b_hex(b)     #Hexadecimal integer to ASCii encoded string
        f.write(c)
    f.close()

def send_wawe_data(dev):
    """send wave1.bin to the device"""
    f = open("wave1.bin", "rb")     #wave1.bin is the waveform to be sent
    data = f.read()
    print 'write bytes:',len(data)
    dev.write("C1:WVDT
WVNM,wave1,FREQ,2000.0,AMPL,4.0,OFST,0.0,PHASE,0.0,WAVEDATA,%s" % (data))
#"X" series (SDG1000X/SDG2000X/SDG6000X/X-E)
    dev.write("C1:ARWV NAME,wave1")
    f.close()

def get_wave_data(dev):
    """get wave from the devide"""
    f = open("wave2.bin", "wb")     #save the waveform as wave2.bin
    dev.write("WVDT? user,wave1")     #"X" series (SDG1000X/SDG2000X/SDG6000X/X-E)
    time.sleep(1)
    data = dev.read()
    data_pos = data.find("WAVEDATA,") + len("WAVEDATA,")
    print data[0:data_pos]
    wave_data = data[data_pos:]
    print 'read bytes:',len(wave_data)
    f.write(wave_data)
    f.close()
```

```
if __name__ == '__main__':
    """"""
    device = visa.instrument(device_resource, timeout=5000, chunk_size = 40*1024)
    create_wave_file()
    send_wawe_data(device)
   get_wave_data(device)
```

**Output waveform:**



## 5.1.4 Python3 Example

**Environment:** Python3.6.5， PyVISA 1.9

Same example, but for Python 3.6.5

```
#!/usr/bin/env python3.6.5
# -*- coding: utf-8 -*-

import visa
import time
import binascii

#USB resource of Device
```

```
device_resource = 'USB0::0xF4EC::0x1102::SDG7ABAQ5R0010::INSTR'


#Little endian, 16-bit2's complement
wave_points = [0x0080, 0x0070, 0x0060, 0x0040, 0x0050, 0x0060, 0x0070, 0xff7f,0x0050]


def create_wave_file():
    """create a file"""
    f = open("wave1.bin", "wb")
    for a in wave_points:
        b = hex(a)
        b = b[2:]
        len_b = len(b)
        if (0 == len_b):
            b = '0000'
        elif (1 == len_b):
            b = '000' + b
        elif (2 == len_b):
            b = '00' + b
        elif (3 == len_b):
            b = '0' + b
        c = binascii.a2b_hex(b)     #Hexadecimal integer to ASCii encoded string
        f.write(c)
    f.close()


def send_wawe_data(dev):
    """send wave1.bin to the device"""
    f = open("wave1.bin", "rb")     #wave1.bin is the waveform to be sent
    data = f.read()
    print ('write bytes:%s'%len(data))
    dev.write("C1:WVDT
WVNM,wave1,FREQ,2000.0,AMPL,4.0,OFST,0.0,PHASE,0.0,WAVEDATA,%s"    %    (data))
#"X" series (SDG1000X/SDG2000X/SDG6000X/X-E)
    dev.write("C1:ARWV NAME,wave1")
    f.close()

def get_wave_data(dev):
    """get wave from the devide"""
    f = open("wave2.bin", "wb")     #save the waveform as wave2.bin
    dev.write("WVDT? user,wave1")     #"X" series (SDG1000X/SDG2000X/SDG6000X/X-E)
    time.sleep(1)
    data = dev.read()
    data_pos = data.find("WAVEDATA,") + len("WAVEDATA,")
    print (data[0:data_pos])
    wave_data = data[data_pos:]
    print ('read bytes:%s'%len(wave_data))
    f.write(wave_data)
    f.close()

if __name__ == '__main__':
    """"""
    rm=visa.ResourceManager()
    device    =rm.open_resource(device_resource,    timeout=50000,    chunk_size    =
24*1024*1024)
    create_wave_file()
    send_wawe_data(device)
    #get_wave_data(device)
```

### 5.1.5 Python3 Example(Digital)

**Environment:** Python3.6.5， PyVISA 1.9

Same example, but for the digital output channels.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import pyvisa as visa
import time
import binascii

# resource of Device
device_resource = 'USB0::0xF4EC::0x1102::SDG7ACBC5M0005::INSTR'

d7 = '000011110000'   # Data stream of ch7 in digital
d6 = '101010101010'   # Data stream of ch6 in digital
d5 = '010101010101'   # Data stream of ch5 in digital
d4 = '110011001100'   # Data stream of ch4 in digital
d3 = '000000111111'   # Data stream of ch3 in digital
d2 = '111000111000'   # Data stream of ch2 in digital
d1 = '001100110011'   # Data stream of ch1 in digital
d0 = '110011001100'   # Data stream of ch0 in digital
other = '00000000'   # The last 8ch data is 0
wave_points = []
for i7, i6, i5, i4, i3, i2, i1, i0 in zip(d7, d6, d5, d4, d3, d2, d1, d0):
    a = i7 + i6 + i5 + i4 + i3 + i2 + i1 + i0 + other
    wave_points.append(int(a, 2))

def create_wave_file():
    """create a file"""
    f = open("wave1.bin", "wb")
    for a in wave_points:
        b = hex(a)
        b = b[2:]
        len_b = len(b)
        if (0 == len_b):
            b = '0000'
        elif (1 == len_b):
            b = '000' + b
        elif (2 == len_b):
            b = '00' + b
        elif (3 == len_b):
            b = '0' + b
        c = binascii.unhexlify(b)   # Hexadecimal integer to ASCii encoded string
        f.write(c)
    f.close()


def send_wave_data(dev):
    """send wave1.bin to the device"""
    f = open("wave1.bin", "rb")   # wave1.bin is the waveform to be sent
    data = f.read().decode("latin1")
    print('write class:', type(data))
    print('write bytes:', len(data))
    dev.write_termination = ''
    dev.write("DIG:WVDT WVNM,digital, WAVEDATA,%s" % (data), encoding='latin1')
    f.close()
    return data

if __name__ == '__main__':
    """"""
    rm = visa.ResourceManager()
    device = rm.open_resource(device_resource, timeout=50000, chunk_size=24 * 1024 *
1024)
    create_wave_file()
    send = send_wave_data(device)
   print('Done')
```

## 5.2 Examples of Using Sockets

### 5.2.1 Python Example

Python has a low-level networking module that provides access to the socket interface. Python scripts can be written for sockets to do a variety of tests and measurement tasks.

**Environment:** Windows 7 32-bit, Python v2.7.5

**Description:** Open a socket, send a query, and repeat this loop 10 times, finally close the socket. Note that SCPI command strings must be terminated with a “\n” (new line) character in programming.

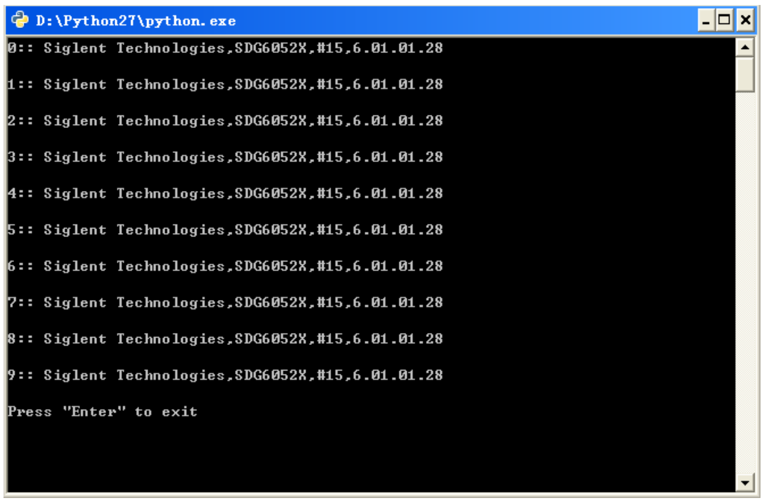
Below is the code of the script:

```
#!/usr/bin/env python
#-*- coding:utf-8 –*-
#-----------------------------------------------------------------------------
# The short script is an example that opens a socket, sends a query,
# print the return message and closes the socket.
#-----------------------------------------------------------------------------

import socket    # for sockets
import sys   # for exit
import time # for sleep

#-----------------------------------------------------------------------------

remote_ip = "10.11.13.40"   # should match the instrument’s IP address
port = 5025 # the port number of the instrument service
count = 0
```

```
def SocketConnect():
    try:
        #create an AF_INET, STREAM socket (TCP)
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    except socket.error:
        print ('Failed to create socket.')
        sys.exit();
    try:
        #Connect to remote server
        s.connect((remote_ip , port))
    except socket.error:
        print ('failed to connect to ip ' + remote_ip)
    return s

def SocketQuery(Sock, cmd):
    try :
        #Send cmd string
        Sock.sendall(cmd)
        time.sleep(1)
    except socket.error:
        #Send failed
        print ('Send failed')
        sys.exit()
    reply = Sock.recv(4096)
    return reply

def SocketClose(Sock):
    #close the socket
    Sock.close()
```

```
        time.sleep(.300)

def main():
        global remote_ip
        global port
        global count

        # Body: send the SCPI commands “*IDN?” 10 times and print the return message
        s = SocketConnect()
        for i in range(10):
                qStr = SocketQuery(s, b'*IDN?\n')
                print (str(count) + ":: " + str(qStr))
                count = count + 1
        SocketClose(s)
        input('Press "Enter" to exit')

if __name__ == '__main__':
        proc = main()
```

**Run result:**



D:\Python27\python.exe

```
0:: Siglent Technologies,SDG6052X,#15,6.01.01.28
1:: Siglent Technologies,SDG6052X,#15,6.01.01.28
2:: Siglent Technologies,SDG6052X,#15,6.01.01.28
3:: Siglent Technologies,SDG6052X,#15,6.01.01.28
4:: Siglent Technologies,SDG6052X,#15,6.01.01.28
5:: Siglent Technologies,SDG6052X,#15,6.01.01.28
6:: Siglent Technologies,SDG6052X,#15,6.01.01.28
7:: Siglent Technologies,SDG6052X,#15,6.01.01.28
8:: Siglent Technologies,SDG6052X,#15,6.01.01.28
9:: Siglent Technologies,SDG6052X,#15,6.01.01.28
Press "Enter" to exit
```

# 6 Index

## About SIGLENT

SIGLENT is an international high-tech company, concentrating on R&D, sales, production and services of electronic test & measurement instruments.

SIGLENT first began developing digital oscilloscopes independently in 2002. After more than a decade of continuous development, SIGLENT has extended its product line to include digital oscilloscopes, isolated handheld oscilloscopes, function/arbitrary waveform generators, RF/MW signal generators, spectrum analyzers, vector network analyzers, digital multimeters, DC power supplies, electronic loads and other general purpose test instrumentation. Since its first oscilloscope was launched in 2005, SIGLENT has become the fastest growing manufacturer of digital oscilloscopes. We firmly believe that today SIGLENT is the best value in electronic test & measurement.

**Headquarters:**
SIGLENT Technologies Co., Ltd
Add: Bldg No.4 & No.5, Antongda Industrial Zone, 3rd Liuxian Road, Bao'an District, Shenzhen, 518101, China
Tel: + 86 755 3688 7876
Fax: + 86 755 3359 1582
Email: sales@siglent.com
Website: int.siglent.com

**North America:**
SIGLENT Technologies America, Inc
6557 Cochran Rd Solon, Ohio 44139
Tel: 440-398-5800
Toll Free: 877-515-5551
Fax: 440-399-1211
Email: info@siglentna.com
Website: www.siglentna.com

**Europe:**
SIGLENT Technologies Germany GmbH
Add: Staetzlinger Str. 70
86165 Augsburg, Germany
Tel: +49(0)-821-666 0 111 0
Fax: +49(0)-821-666 0 111 22
Email: info-eu@siglent.com
Website: www.siglenteu.com

**Follow us on**
**Facebook: SiglentTech**