

Programming Guide

SDG Series Function/Arbitrary Waveform Generator

Catalogue

1	PROGRAMMING OVERVIEW	4
1.1	BUILD COMMUNICATION	4
1.1.1	<i>Install NI-VISA.....</i>	4
1.1.2	<i>Connect the instrument.....</i>	6
1.2	HOW TO REMOTE CONTROL	7
1.2.1	<i>User-defined Programming.....</i>	7
1.2.2	<i>Send SCPI Commands via NI-VISA.....</i>	7
2	INTRODUCTION TO THE SCPI LANGUAGE	8
2.1	ABOUT COMMANDS & QUERIES	8
2.2	HOW THEY ARE LISTED	8
2.3	HOW THEY ARE DESCRIBED	8
2.4	WHEN CAN THEY BE USED	8
2.5	COMMAND NOTATION	8
2.6	TABLE OF COMMAND & QUERIES	9
3	COMMANDS AND QUERIES.....	11
3.1	IEEE 488.2 COMMON COMMAND INTRODUCTION	11
3.1.1	<i>IDN.....</i>	11
3.1.2	<i>OPC.....</i>	12
3.1.3	<i>CLS.....</i>	12
3.1.4	<i>ESE.....</i>	13
3.1.5	<i>ESR.....</i>	13
3.1.6	<i>RST.....</i>	14
3.1.7	<i>SRE.....</i>	14
3.1.8	<i>STB.....</i>	15
3.1.9	<i>TST.....</i>	15
3.1.10	<i>WAI.....</i>	16
3.1.11	<i>DDR.....</i>	16
3.1.12	<i>CMR.....</i>	17
3.2	COMM_HEADER COMMAND.....	18
3.3	OUTPUT COMMAND	18
3.4	BASIC WAVE COMMAND	20
3.5	MODULATE WAVE COMMAND.....	22
3.6	SWEEP WAVE COMMAND.....	26
3.7	BURST WAVE COMMAND	28
3.8	PARAMETER COPY COMMAND	31
3.9	ARBITRARY WAVE COMMAND	31
3.10	SYNC COMMAND	33
3.11	NUMBER FORMAT COMMEND	33
3.12	LANGUAGE COMMAND	34

3.13	CONFIGURATION COMMAND	35
3.14	BUZZER COMMAND	35
3.15	SCREEN SAVE COMMAND	35
3.16	CLOCK SOURCE COMMAND.....	36
3.17	FREQUENCY COUNTER COMMAND.....	36
3.18	INVERT COMMAND	37
3.19	COUPLING COMMAND	38
3.20	VOLTAGE OVERLOAD COMMAND	39
3.21	STORE LIST COMMAND	40
3.22	ARBITRARY WAVE DATA COMMAND.....	41
3.23	VIRTUAL KEY COMMAND.....	43
3.24	IP COMMAND	44
3.25	SUBNET MASK COMMAND	45
3.26	GATEWAY COMMAND.....	46
3.27	SAMPLING RATE COMMAND	46
3.28	HARMONIC COMMAND.....	47
3.29	WAVEFORM COMBINING COMMAND	48
4	PROGRAMMING EXAMPLES	49
4.1	EXAMPLE OF VC++	49
4.2	EXAMPLE OF VB.....	55
4.3	EXAMPLE OF MATLAB	62
4.4	EXAMPLE OF LABVIEW.....	64
5	INDEX.....	67

1 Programming Overview

This chapter introduces how to build communication between SDG series function/arbitrary waveform generator and the PC. It also introduces how to remote control.

1.1 Build communication

1.1.1 Install NI-VISA

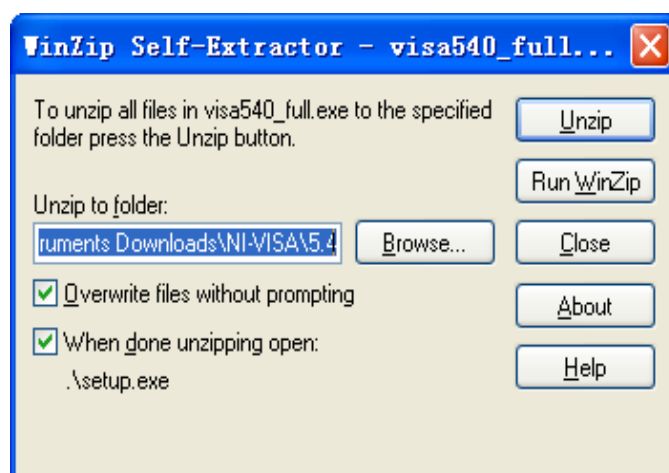
Before programming, you need to install NI-VISA, which you can download from the NI-VISA web site. About NI-VISA, there are full version and Run-Time Engine version. The full version include NI device driver and a tool named NI MAX that is a user interface to control the device. The Run-Time Engine version which is much smaller than the full version only include NI device driver.

For example, you can get NI-VISA 5.4 full version from:
<http://www.ni.com/download/ni-visa-5.4/4230/en/>.

You can also download NI-VISA Run-Time Engine 5.4 to your PC and install it as default selection. This installation process is similar with the full version.

After you downloaded the file you can follow the steps below to install it:

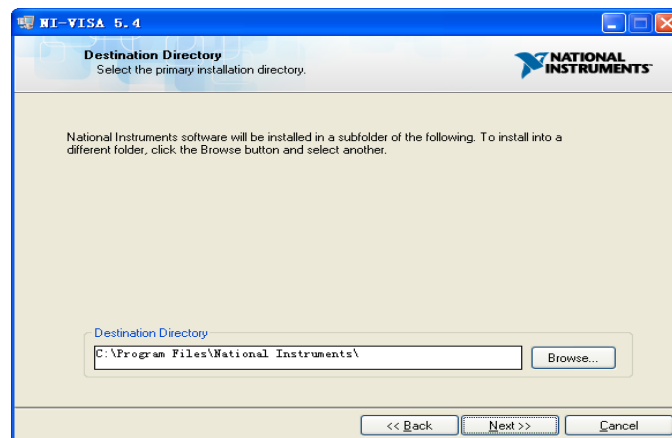
- i. Double click the visa540_full.exe, dialog shown as below:



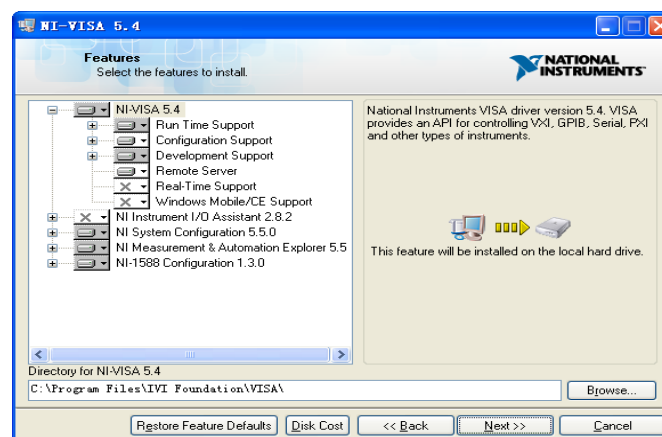
- ii. Click Unzip, the installation process will automatically launch after unzipping files. If your computer needs to install .NET Framework 4, its Setup process will auto start.



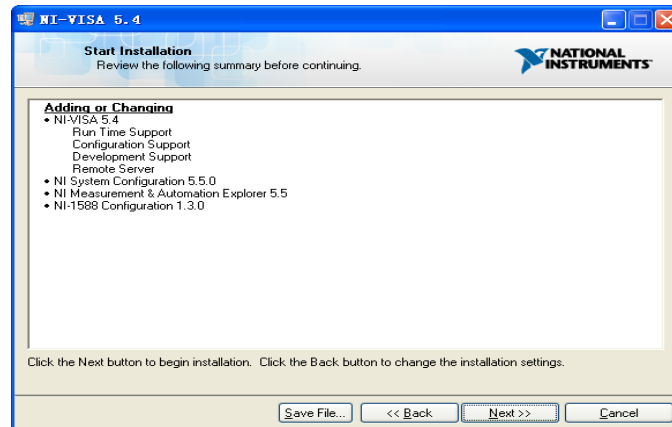
- iii. The NI-VISA installing dialog is shown above. Click Next to start the installation process.



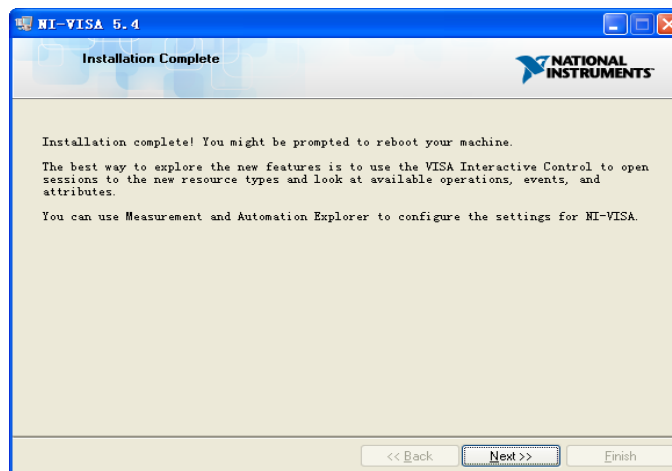
Set the install path, default path is “C:\Program Files\National Instruments\”, you can change it. Click Next, dialog shown as above.



- iv. Click Next twice, in the License Agreement dialog, select the “I accept the above 2 License Agreement(s).”, and click Next, dialog shown as below:



- v. Click Next to run installation.

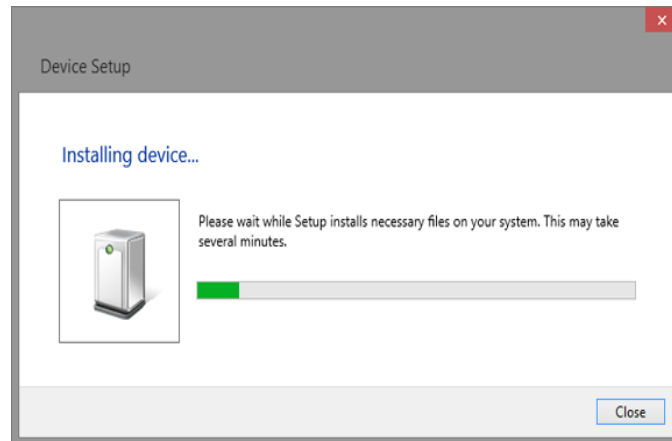


Now the installation is complete, reboot your PC.

1.1.2 Connect the instrument

Depending on your specific model your function/arbitrary waveform generator may be able to communicate with a PC through the USB or LAN interface. This manual takes the USB as an example. But some examples may involve LAN.

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



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

1.2 How To Remote Control

1.2.1 User-defined Programming

Users can use SCPI commands to program and control the function/arbitrary waveform generator. For details, refer to the introductions in "**Programming Examples**".

1.2.2 Send SCPI Commands via NI-VISA

You can control the SDG remotely by sending SCPI commands via NI-VISA software.

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 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, and are recognized by the question mark (?) following the header.

2.2 How They are Listed

The descriptions are listed in alphabetical order according to their short format.

2.3 How They are Described

In the descriptions themselves, 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.

2.4 When can They be Used

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

2.5 Command Notation

The following notations are used in the commands:

- < > Angular brackets enclose words that are used 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.

... An ellipsis indicates that the items both to its left and right may be repeated for a number of times.

2.6 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).
*CLS	*CLS	SYSTEM	Clears all the status data registers.
*ESE	*ESE	SYSTEM	Sets or gets the Standard Event Status Enable register (ESE).
*ESR	*ESR	SYSTEM	Reads and clears the contents of the Event Status Register (ESR).
*RST	*RST	SYSTEM	Initiates a device reset.
*SRE	*SRE	SYSTEM	Sets the Service Request Enable register (SRE).
*STB	*STB	SYSTEM	Gets the contents of the IEEE 488.2 defined status register.
*TST	*TST	SYSTEM	Performs an internal self-test.
*WAI	*WAI	SYSTEM	Wait to continue command.
DDR	DDR	SYSTEM	Reads and clears the Device Dependent Register (DDR).
CMR	CMR	SYSTEM	Reads and clears the command error register.
CHDR	COMM_HEADER	SIGNAL	Sets or gets the command returned format
OUTP	OUTPUT	SIGNAL	Sets or gets output state.
BSWV	BASIC_WAVE	SIGNAL	Sets or gets basic wave parameters.
MDWV	MODULATEWAVE	SIGNAL	Sets or gets modulation parameters.
SWWV	SWEEPWAVE	SIGNAL	Sets or gets sweep parameters.
BTWV	BURSTWAVE	SIGNAL	Sets or gets 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 synchronization signal.
NBFM	NUMBER_FORMAT	SYSTEM	Sets or gets data format.
LAGG	LANGUAGE	SYSTEM	Sets or gets language.
SCFG	SYS_CFG	SYSTEM	Sets or gets the power-on system setting way.
BUZZ	BUZZER	SYSTEM	Sets or gets buzzer state.

SCSV	SCREEN_SAVE	SYSTEM	Sets or gets screen save state.
ROSC	ROSCILLATOR	SIGNAL	Sets or gets state of clock source.
FCNT	FREQCOUNTER	SIGNAL	Sets or gets frequency counter parameters.
INVT	INVERT	SIGNAL	Sets or gets polarity of current channel.
COUP	COUPLING	SIGNAL	Sets or gets coupling parameters.
VOLTPRT	VOLTPRT	SYSTEM	Sets or gets state of over-voltage protection.
STL	STORELIST	SIGNAL	Lists all stored waveforms.
WVDT	WVDT	SIGNAL	Sets and gets 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 system IP address.
SYST:COMM:LAN:SMAS	SYSTEM:COMMUNICATE:LAN:SMASK	SYSTEM	The Command can set and get system subnet mask.
SYST:COMM:LAN:GAT	SYSTEM:COMMUNICATE:LAN:GATEWAY	SYSTEM	The Command can set and get system Gateway.
SRATE	SAMPLERATE	SIGNAL	Sets or gets sampling rate. You can only use it in TrueArb mode
HARM	HARMonic	SIGNAL	Sets or gets harmonic information.
CMBN	CoMBiNe	SIGNAL	Sets or gets wave combine information.

3 Commands and Queries

3.1 IEEE 488.2 Common Command Introduction

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 comprises manufacturer, model, serial number, software version and firmware version.

QUERY SYNTAX *IDN?

RESPONSE FORMAT *IDN, <device id>,<model>,<serial number>,<software version>, <hardware version>.

<device id>:= "SDG" is used to identify instrument.

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

<serial number>: Each product has its own number, the serial number can labeled product uniqueness.

<software version>:= A serial numbers about software version.

<hardware version>:= The hardware level field, should contain information about all separately revisable subsystems. This information can be contained in single or multiple revision codes.

EXAMPLE

Reads version information.

*IDN?

Return:

*IDN SDG, SDG5162, 120465, 5.01.02.05, 02-00-00-21-24 (It may differ from each version)

Notes:

1)

Parameter/command	SDG800	SDG1000	SDG2000X	SDG5000	SDG1000X
<hardware version>	yes	yes	no	yes	no

2) Explain for <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

*OPC 1

3.1.3 CLS

DESCRIPTION

The *CLS command clears all the status data registers.

COMMAND SYNTAX

*CLS

EXAMPLE

The following command causes all the status data registers to be cleared:

*CLS

3.1.4 ESE

DESCRIPTION	The *ESE command sets the Standard Event Status Enable register (ESE). This command allows one or more events in the ESR register to be reflected in the ESB summary message bit (bit 5) of the STB register. The *ESE? query reads the contents of the ESE register.
COMMAND SYNTAX	*ESE <value> <value> : = 0 to 255.
QUERY SYNTAX	*ESE?
RESPONSE FORMAT	*ESE <value>
EXAMPLE	The following instruction allows the ESB bit to be set if a user request (URQ bit 6, i.e. decimal 64) and/or a device dependent error (DDE bit 3, i.e. decimal 8) occurs. Summing these values yields the ESE register mask $64+8=72$. *ESE? Return: *ESE 72
RELATED COMMANDS	*ESR

3.1.5 ESR

DESCRIPTION	The *ESR? query reads and clears the contents of the Event Status Register (ESR). The response represents the sum of the binary values of the register bits 0 to 7.
QUERY SYNTAX	*ESR?
RESPONSE FORMAT	*ESR <value> <value> : = 0 to 255
EXAMPLE	The following instruction reads and clears the content of the ESR register: *ESR? Return: *ESR 0

RELATED COMMANDS *CLS, *ESE

3.1.6 RST

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

COMMAND SYNTAX * RST

EXAMPLE This example resets the signal generator:
*RST

3.1.7 SRE

DESCRIPTION The *SRE command sets the Service Request Enable register (SRE). This command allows the user to specify which summary message bit(s) in the STB register will generate a service request.

A summary message bit is enabled by writing a '1' into the corresponding bit location. Conversely, writing a '0' into a given bit location prevents the associated event from generating a service request (SRQ). Clearing the SRE register disables SRQ interrupts.

The *SRE? query returns a value that, when converted to a binary number represents the bit settings of the SRE register. Note that bit 6 (MSS) cannot be set and it's returned value is always zero.

COMMAND SYNTAX *SRE <value>
<value> : = 0 to 255

QUERY SYNTAX *SRE?

RESPONSE FORMAT *SRE <value>

EXAMPLE The following instruction allows a SRQ to be generated as soon as the MAV summary bit (bit 4, i.e. decimal 16) or the INB summary bit (bit 0, i.e. decimal 1) in the STB register, or both are set. Summing these two values yields the SRE mask $16+1 = 17$.
*SRE?
Return:

*SRE 17

3.1.8 STB

DESCRIPTION	The *STB? query reads the contents of the 488.2 defined status register (STB), and the Master Summary Status (MSS). The response represents the values of bits 0 to 5 and 7 of the Status Byte register and the MSS summary message. The response to a *STB? query is identical to the response of a serial poll except that the MSS summary message appears in bit 6 in place of the RQS message.
QUERY SYNTAX	*STB?
RESPONSE FORMAT	*STB <value> <value> : = 0 to 255
EXAMPLE	The following reads the status byte register: *STB? Return: *STB 0
RELATED COMMANDS	*CLS, *SRE

3.1.9 TST

DESCRIPTION	The *TST? query performs an internal self-test and the response indicates whether the self-test has detected any errors. The self-test includes testing the hardware of all channels. Hardware failures are identified by a unique binary code in the returned <status> number. A "0" response indicates that no failures occurred.
QUERY SYNTAX	*TST?
RESPONSE FORMAT	*TST <status> <status> : = 0 self-test successful
EXAMPLE	The following causes a self-test to be performed: TST? Return(if no failure): *TST 0

RELATED COMMANDS *CAL

Note:

Parameter/command	SDG800	SDG1000	SDG2000X	SDG5000	SDG1000X
TST	no	yes	yes	yes	yes

3.1.10 WAI

DESCRIPTION The *WAI (WAIT to continue) command, requires by the IEEE 488.2 standard, has no effect on the instrument, as the signal generator only starts processing a command when the previous command has been entirely executed.

COMMAND SYNTAX *WAI

RELATED COMMANDS *OPC

3.1.11 DDR

DESCRIPTION The DDR? query reads and clears the contents of the device dependent or device specific error register (DDR). In case of a hardware failure, the DDR register specifies the origin of the failure.

QUERY SYNTAX DDR?

RESPONSE FORMAT DDR <value>
<value> : = 0 to 65535

EXAMPLE DDR?
Return:
DDR 0

The following table gives details:

Bit	Bit Value	Description
15...14		Reserved
13	8192	Time-base hardware failure detected
12	4096	Trigger hardware failure detected
11		Reserved
10		Reserved
9	512	Channel 2 hardware failure detected
8	256	Channel 1 hardware failure detected

7	128	External input overload condition detected
6...4		Reserved
3		Reserved
2		Reserved
1	2	Channel 2 overload condition detected
0	1	Channel 1 overload condition detected

Note:

Parameter/command	SDG800	SDG1000	SDG2000X	SDG5000	SDG1000x
DDR	yes	yes	no	yes	no

3.1.12 CMR

DESCRIPTION The CMR? query reads and clears the contents of the command error register (CMR) .See the table below which specifies the last syntax error type detected by the instrument.

QUERY SYNTAX CMR?

RESPONSE FORMAT CMR <value>
<value> : = 0 to 14

EXAMPLE CMR?
Return:
CMR 0

Value	Description
0	
1	Unrecognized command/query header
2	Invalid character
3	Invalid separator
4	Missing parameter
5	Unrecognized keyword
6	String error
7	Parameter can't allowed
8	Command String Too Long
9	Query cannot allowed
10	Missing Query mask
11	Invalid parameter
12	Parameter syntax error
13	Filename too long
14	Directory not exist

Note:

Parameter/command	SDG800	SDG1000	SDG2000X	SDG5000	SDG1000X
CMR	yes	yes	no	yes	no

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. "OFF" returns nothing.

COMMAND SYNTAX CHDR (Comm_HeaDeR) <parameter>
<parameter>:= {SHORT, LONG, OFF}

QUERY SYNTAX CHDR (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:

Parameter/command	SDG800	SDG1000	SDG2000X	SDG5000	SDG1000X
CHDR	yes	yes	no	yes	no

3.3 Output Command

DESCRIPTION Enable or disable the output of the [Output] connector at the front panel corresponding to the channel.
The query returns "ON" or "OFF" and "LOAD", "PLRT" parameters.

COMMAND SYNTAX <channel>:OUTP (OUTPut) <parameter>
<channel>:={C1, C2}
<parameter>:= {a parameter from the table below}

Parameters	Value	Description
------------	-------	-------------

ON	---	Turn on
OFF	---	Turn off
LOAD	<load>	Value of load (default unit is ohm)
PLRT	<NOR, INVT>	Value of polarity parameter

< load>:= {please see the note below.}

QUERY SYNTAX

<channel>: OUTP(OUTPut)?

RESPONSE FORMAT

<channel>: OUTP <load>

EXAMPLE

Turn on channel one.

C1: OUTP ON

Read channel one output state.

C1: OUTP?

Return:

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

Set the load to 50.

C1: OUTP LOAD, 50

Set the load to HZ.

C1: OUTP LOAD, HZ

Set the polarity normal.

C1: OUTP PLRT, NOR

Set the polarity inverted.

C1: OUTP PLRT, INVT

Note:

Parameter/command	SDG800	SDG1000	SDG2000X	SDG5000	SDG1000X
<channel>	no	yes	yes	yes	yes
LOAD	50, HZ	50~10000, HZ	50~100000, HZ	50, HZ	50~100000, HZ

3.4 Basic Wave Command

DESCRIPTION

Sets or gets basic wave parameters. In SDG1000X if turn on wave combine, you can't set wave to square. Combining a square waveform is not possible.

COMMAND SYNTAX

<channel>:BSWV(BaSiC_WaVe) <parameter>

<channel>:={C1, C2}

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

Parameters	Value	Description
WVTP	<type>	Type of wave
FRQ	<frequency>	Value of frequency. If wave type is Noise or DC, you can't set this parameter.
PERI	<period>	Value of period. If wave type is Noise or DC, you can't set this parameter.
AMP	<amplitude>	Value of amplitude. If wave type is Noise or DC, you can't set this parameter.
OFST	<offset>	Value of offset. If wave type is Noise or DC, you can't set this parameter.
SYM	<symmetry>	Value of symmetry. Only when wave type is Ramp, you can set this parameter.
DUTY	<duty>	Value of duty cycle. Only when wave type is Square and Pulse, you can set this parameter.
PHSE	<phase>	Value of phase. If wave type is Noise or Pulse or DC, you can't set this parameter.
STDEV	<standard deviation>	Value of Noise wave standard deviation. Only when wave type is Noise, you can set this parameter.
MEAN	<mean>	Value of Noise wave mean. Only when wave type is Noise, you can set this parameter.
WIDTH	<width>	Value of width. Only when wave type is Pulse, you can set this parameter.
RISE	<rise>	Value of rise time. Only when wave type is Pulse, you can set this parameter.
FALL	<fall>	Value of fall time. Only when wave type is Pulse, you can set this parameter.
DLY	<delay>	Value of delay. Only when wave type is Pulse, you can set this parameter.
HLEV	<high level>	Value of high level. If wave type is Noise or DC, you can't set this parameter.
LLEV	<low level>	Value of low level. If wave type is Noise or DC, you can't set this parameter.
BANDSTATE	<bandwidth>	State of noise bandwidth switch. Only when wave type

	switch >	is Noise, you can set this parameter.
BANDWIDTH	<bandwidth value>	Value of noise bandwidth. Only when wave type is noise, you can set this parameter.

Note: if the command doesn't set basic wave type, WVPT parameter will be set to current wave type.

where:

<type>:= {SINE, SQUARE, RAMP, PULSE, NOISE, ARB ,DC}
 <frequency>:= {Default unit is "Hz". Value depends on the model.}
 <amplitude>:= {Default unit is "V". Value depends on the model.}
 <offset>:= {Default unit is "V". Value depends on the model.}
 <duty>:= {0% to 100%. Value depends on frequency.}
 <symmetry> := { 0% to 100%}
 <phase>:= {0 to 360, In SDG2000X/SDG1000X, if you set 400, it will set 40(400-360)}
 < standard deviation >:= {Default unit is "V". Value depends on the model.}
 <mean>:= {Default unit is "V". Value depends on the model.}
 <width>:= {Max_width < (Max_duty * 0.01) * period and Min_width > (Min_duty * 0.01) * period.}
 <rise>:= {Value depends on the model.}
 <fall>:= {Value depends on the model.}
 <delay>:= {Unit is S. Maximal is Pulse period, minimum value is 0.}
 <bandwidth switch >:= {ON,OFF}
 <bandwidth value>:= {value between 20MHz and 120MHz}

QUERY SYNTAX

<channel>: BSWV (BaSic_WaVe)?
 <channel>:= {C1, C2}

RESPONSE FORMAT

<channel>:BSWV<type>,<frequency>,<amplitude>,<offset>,
 <duty>,<symmetry>,<phase>,<variance>,<mean>,<width>,
 <rise>,<fall>,<delay>.

EXAMPLE

Change channel one wave type to ramp.

C1: BSWV WVTP, RAMP

Change frequency of channel one to 2000 Hz.

C1: BSWV FRQ, 2000

Set amplitude of channel one to 3Vpp.

C1: BSWV AMP, 3

Read channel basic wave parameters from 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 value of channel one to 100MHz

C1: BSWV BANDWIDTH, 100000000

Note:

Parameter/command	SDG800	SDG1000	SDG2000X	SDG5000	SDG1000X
<channel>	no(single channel)	yes	yes	yes	yes
RISE	yes	no	yes	yes	yes
FAL	yes	no	yes	yes	yes
DLY	no	yes	yes	yes	yes
BANDSTATE	no	no	yes	no	no
BANDWIDTH	no	no	yes	no	no

3.5 Modulate Wave Command

DESCRIPTION Sets or gets modulation parameters.

COMMAND <channel>:MDWV(MoDulateWaVe)<parameter>

SYNTAX <channel>:={C1, C2}

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

Parameters	Value	Description
STATE	<state>	Turn on or off modulation. Note: if you want to set or read other parameters of modulation, you must set STATE to ON at first.
AM, SRC	<src>	AM signal source.
AM, MDSP	<mod wave shape>	AM modulation wave. Only when AM signal source is set to INT, you can set the parameter.
AM, FRQ	<AM frequency>	AM frequency. Only when AM signal source is set to INT, you can set the parameter.
AM, DEPTH	<depth>	AM depth. Only when AM signal source is set to INT, you can set the parameter.
DSBAM, SRC	<src>	DSBAM signal source.
DSBAM, MDSP	<mod wave shape>	DSBAM modulation wave. Only when AM signal source is set to INT, you can set the parameter.
DSBAM, FRQ	<DSB-AM	DSBAM frequency. Only when AM signal

	frequency>	source is set to INT, you can set the parameter.
FM, SRC	<src>	FM signal source.
FM, MDSP	<mod wave shape>	FM modulation wave. Only when FM signal source is set to INT, you can set the parameter.
FM, FRQ	<FM frequency>	FM frequency. Only when FM signal source is set to INT, you can set the parameter.
FM, DEVI	<FM frequency deviation >	FM frequency deviation. Only when FM signal source is set to INT. you can set the parameter.
PM, SRC,	<src>	PM signal source.
PM, MDSP	<mod wave shape>	PM modulation wave. Only when PM signal source is set to INT, you can set the parameter.
PM, FRQ	<PM frequency>	PM frequency. Only when PM signal source is set to INT, you can set the parameter.
PWM, FRQ	<PWM frequency>	PWM frequency. Only when carrier wave is PULSE wave, you can set the parameter.
PWM, DEVI	<PWM dev>	Duty cycle deviation. Only when carrier wave is PULSE wave, you can set the parameter.
PWM, MDSP	<mod wave shape>	PWM modulation wave. Only when carrier wave is PULSE wave, you can set the parameter.
PWM, SRC	<src>	PWM signal source.
PM, DEVI	<PM phase offset>	PM phase deviation. Only when PM signal source is set to INT, you can set the parameter.
ASK, SRC	<src>	ASK signal source.
ASK, KFRQ	<ASK key frequency>	ASK key frequency. Only when ASK signal source is set to INT, you can set the parameter.
FSK, KFRQ	<FSK key frequency>	FSK key frequency. Only when FSK signal source is set to INT, you can set the parameter.
FSK, HFRQ	<FSK hop frequency>	FSK hop frequency.
FSK, SRC	<src>	FSK signal source.
PSK, KFRQ	<FSK key frequency>	PSK key frequency. Only when PSK signal source is set to INT, you can set the parameter.
PSK, SRC	<src>	PSK signal source.
CARR, WVTP	<wave type>	Carrier wave type.

CARR, FRQ	<frequency>	Value of carrier frequency.
CARR, AMP	<amplitude>	Value of carrier amplitude.
CARR, OFST	<offset>	Value of carrier offset.
CARR, SYM	<symmetry>	Value of carrier symmetry. Only ramp can set this parameter.
CARR, DUTY	<duty>	Value of duty cycle. Only square and pulse can set this parameter.
CARR, PHSE	<phase>	Value of carrier phase.
CARR, RISE	<rise>	Value of rise time. Only Pulse can set this parameter.
CARR, FALL	<fall>	Value of fall time. Only Pulse can set this parameter.
CARR, DLY	<delay>	Value of carrier delay. Only PULSE can set this parameter.

Note: If carrier wave is Noise you can't set to turn on modulation.

If you want to set AM, FM, PM, CARR and STATE the first parameter have to be one of them.

where:

<state>:= {ON, OFF}
 <src>:= {INT, EXT}
 <mod wave shape>:= {SINE, SQUARE, TRIANGLE, UP RAMP, DNRAMP, NOISE, ARB}
 <am frequency>:= {Default unit is "Hz". Value depends on the model.}
 <depth>:= {0% to 120%}
 <fm frequency>:= {Default unit is "Hz". Value depends on the model.}
 <fm frequency deviation > := { 0 to carrier frequency, Value depends on the difference between carrier frequency and bandwidth frequency.}
 <pm frequency> := { Default unit is "Hz", Value depends on the model.}
 <pm phase deviation >:= {0 to 360.}
 <pwm frequency>:= {Default unit is "Hz", Value depends on the model. }
 <pwm dev>:= { Default unit is "%",value depends on carrier duty cycle}
 <ask key frequency>:= {Default unit is "Hz", Value depends on the model.}
 <fsk frequency>:= { Default unit is "Hz", Value depends on the version.}
 <fsk jump frequency>:= { the same with basic wave frequency}
 <wave type>:= {SINE ,SQUARE, RAMP, ARB, PULSE }
 <frequency> := { Default unit is "Hz", Value depends on the model.}
 <amplitude> := { Default unit is "V", Value depends on the model.}
 <offset> := { Default unit is "V", Value depends on the model.}
 <duty>:= {0% to 100 %}.
 <symmetry>:= { 0% to 100%}
 <rise>:= {Value depends on the model.}
 <fall>:= {Value depends on the model.}
 <delay>:= {Default unit is "S".}

Note:

There are some parameters Value depends on the model, You can read version datasheet to get specific parameters

QUERY SYNTAX <channel>: MDWV (MoDulateWaVe)?
<channel>:={C1, C2}

RESPONSE <channel>:MDWV <parameter>
FORMAT <parameter> :={ Return all parameter of the current modulation parameters.}

EXAMPLE Set channel one modulation type to AM.
C1: MDWV AM

Set modulation shape to AM, and set AM modulating wave type to sine wave.

C1: MDWV AM, MDSP, SINE

Read channel one modulation parameters of which STATE is ON.

C1: MDWV?

Return:

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

Read channel one modulate wave parameters of which STATE is OFF.

C1: MDWV?

Return:

C1: MDWV STATE, OFF

Set channel one FM frequency to 1000Hz

C1: MDWV FM, FRQ, 1000

Set channel one carrier shape to SINE.

C1: MDWV CARR, WVTP, SINE

Set channel one carrier frequency to 1000 Hz.

C1: MDWV CARR, FRQ,1000

RELATED ARWV, BTWV, SWWV, BSWV

COMMANDS

Note:

Parameter/command	SDG800	SDG1000	SDG2000X	SDG5000	SDG1000x
<channel>	No(single channel)	yes	yes	yes	yes
[type], SRC	no(only internal source)	yes	yes	yes	yes
CARR, DLY	no	yes	yes	yes	yes
CARR, RISE	yes	no	yes	yes	yes
CARR, FALL	yes	no	yes	yes	yes

[type]:={AM, FM, PM, FSK, ASK, DSBAM, PWM}

3.6 Sweep Wave Command

DESCRIPTION Sets or gets sweep parameters.

COMMAND SYNTAX <channel>SWWV(SweepWaVe) <parameter>
 <channel>:={C1, C2}
 <parameter>:= {a parameter from the table below}

Parameters	Value	Description
STATE	<state>	Turn on or off sweep. Note: if you want to set or read other parameters you must set STATE to ON at first.
TIME	<time>	Value of sweep time.
STOP	<stop frequency>	Value of stop frequency.
START	<start frequency>	Value of start frequency.
TRSR	<trigger src>	Trigger source.
TRMD	<trigger mode>	State of trigger output. If TRSR is EXT, the parameter is invalid.
SWMD	<sweep mode>	Sweep style.
DIR	<direction>	Sweep direction.
EDGE	<edge>	Value of edge. Only when TRSR is EXT, the parameter is valid.
MTRIG	<manual trigger>	Make a manual trigger. Only when TRSR is MAN, the parameter is valid.
CARR, WVTP	<wave type>	Carrier type.
CARR, FRQ	<frequency>	Value of carrier frequency.
CARR, AMP	<amplitude>	Value of carrier amplitude.
CARR, OFST	<offset>	Value of carrier offset.
CARR, SYM	<symmetry>	Value of carrier symmetry, Only Ramp can set this parameter.
CARR, DUTY	<duty>	Value of carrier duty cycle. Only Square can set this parameter.
CARR,	<phase>	Value of carrier phase.

PHSE		
------	--	--

Note: If carrier is Pulse or Noise you can't turn on sweep.

If you want to set CARR and STATE, the first parameter has to be one of them.

where:

<state>:= {ON, OFF}
 <time>:= { Default unit is "S". Value depends on the model.}
 <stop frequency> := { the same with basic wave frequency}
 <start frequency> := { the same with basic wave frequency}
 <trigger src>:= {EXT, INT, MAN}
 <trigger mode>:= {ON, OFF}
 <sweep mod>:= {LINE, LOG}
 <direction>:= {UP, DOWN}
 <edge>:= {RISE, FALL}
 <wave type>:= {SINE, SQUARE, RAMP, ARB}
 <frequency> := { Default unit is "Hz". Value depends on the model.}
 <amplitude> := { Default unit is "V". Value depends on the model.}
 <offset> := { Default unit is "V", Value depends on the model.}
 <duty>:= {0% to 100 %}.
 <symmetry>:= { 0% to 100%}

Note:

There are some parameters Value depends on the model,
You can read version datasheet.

QUERY SYNTAX

<channel>: SWWV (SWEEPWaVe)? <channel>:= {C1, C2}

RESPONSE FORMAT

<parameter> := { Return all parameters of the current sweep wave.}

EXAMPLE

Set channel one sweep time to 1 S.

C1: SWWV TIME, 1

Set channel one sweep stop frequency to 1000 Hz.

C1: SWWV STOP, 1000

Read channel one sweep parameters of which STATE is ON.

C2: SWWV?

Return:

C2: SWWV STATE, ON, TIME, 1S, STOP, 100HZ, START, 100HZ, TRSR, MAN, TRMD, OFF, SWMD, LINE, DIR, UP, CARR, WVTP, SQUARE,

FRQ, 1000HZ, AMP, 4V, OFST, 0V, DUTY, 50, PHSE, 0

Read channel two sweep parameters of which STATE is OFF.

C2: SWWV?

Return:

C2: SWWV STATE, OFF

Note:

Parameter/command	SDG800	SDG1000	SDG2000X	SDG5000	SDG1000X
<channel>	no(single channel)	yes	yes	yes	yes
TRMD	no	yes	yes	yes	yes
EDGE	no	yes	yes	yes	yes

3.7 Burst Wave Command

DESCRIPTION Sets or gets burst wave parameters.

COMMAND SYNTAX <channel>BTWV(BursTWaVe) <parameter>
 <channel>:={C1, C2}
 <parameter>:= {a parameter from the table below}

Parameters	Value	Description
STATE	<state>	Turn on or off burst. Note: If you want to set or read other parameters of burst, you must set state to ON at first.
PRD	<period>	Value of burst period. When carrier is NOISE wave, you can't set it. When GATE was chosen, you can't set it (but in SDG2000X, you can). And when trigger source is EXT, you can't set it.
STPS	<start phase>	Start phase of carrier. When carrier is NOISE or PULSE wave, you can't set it.
GATE_NCYC	<gate Ncycle>	Set the burst mode to GATE or NCYC. When carrier is NOISE, you can't set it.
TRSR	<trigger source>	Set the trigger source.
DLAY	<delay>	Value of delay. When carrier is NOISE wave, you can't set it. When NCYC is chosen you can set it.
PLRT	<polarity>	Value of polarity. When GATE is chosen you can set it. When carrier is NOISE, it is the only parameter.
TRMD	<trig mode>	Value of trigger mode. When carrier is NOISE wave, you can't set it. When NCYC is chosen you can set it. When TRSR is set to EXT, you can't set it.

EDGE	<edge>	Value of edge. When carrier is NOISE wave, you can't set it. When NCYC is chosen and TRSR is set to EXT, you can set it.
TIME	<circle time>	Value of Ncycle number. When carrier is NOISE wave, you can't set it. When NCYC is chosen you can set it.
MTRIG	<manual trig>	Manual trigger. When TRSR is set to MAN, it can be set.
CARR, WVTP	<wave type>	Value of carrier type.
CARR, FRQ	<frequency>	Value of carrier frequency
CARR, AMP	<amplitude>	Value of carrier amplitude.
CARR, OFST	<offset>	Value of carrier offset.
CARR, SYM	<symmetry>	Value of symmetry. Only Ramp can set this parameter.
CARR, DUTY	<duty>	Value of duty cycle. Only Square or Pulse can set this parameter.
CARR, PHSE	<phase>	Value of carrier phase.
CARR, RISE	<rise>	Value of rise edge. Only when carrier is Pulse, the Value is valid.
CARR, FALL	<fall>	Value of fall edge. Only when carrier is Pulse, the Value is valid.
CARR, STDEV	<standard deviation >	Value of standard deviation. Only when carrier is Noise, the Value is valid.
CARR, MEAN	<mean>	Value of mean. Only when carrier wave is Noise, the Value is valid.
CARR, DLY	<delay>	Value of delay. Only when carrier is Pulse, the parameter is valid

Note: If you want to set CARR and STATE, the first parameter has to one of them

where:

<state>:= {ON, OFF}
 <period>:= {Default unit is "S". Value depends on the model.}
 <start phase>:= {0 to 360}
 <gate ncycle>:= {GATE, NCYC}
 <trigger source>:= {EXT, INT, MAN}
 <delay>:= {Default unit is "S", Value depends on the model.}
 <polarity>:= {NEG, POS}
 <trig mode >:= {RISE, FALL, OFF}
 <edge>:= { RISE, FALL}
 <circle time> :=:{ Value depends on the Model ("INF" means infinite).}

<wave type>:={SINE ,SQUARE, RAMP, PULSE, NOISE, ARB}
 <frequency> := { Default unit is "HZ". Value depends on the model.}
 <amplitude>:= {Default unit is "V". Value depends on the model.}
 <offset>:= {Default unit is "V". Value depends on the model.}
 <duty>:= {0% to 100%.}
 <symmetry> := { 0% to 100%}
 <phase>:= {0 to 360}
 < standard deviation >:= {Default unit is "V". Value depends on the model.}
 <mean>:= {Default unit is "V". Value depends on the model.}
 <width> := { Max_width < (Max_duty * 0.01) * period and Min_width > (Min_duty * 0.01) * period.}
 <rise>:= {Value depends on the model.}
 <fall>:= {Value depends on the model.}
 <delay>:= {Default unit is "S".}

Note:

There are some parameters Value depends on the model,
You can read version datasheet to get specific parameters.

QUERY SYNTAX

<channel>: BTWV (BursTWaVe)? <parameter>
 <channel>:={C1, C2}
 <parameter>:=<period>.....

RESPONSE FORMAT

<channel>:BTWV <type>,<state>,<period>.....

EXAMPLE

Set channel one burst period to 1S.

C1: BTWV PRD, 1

Set channel one burst delay to 1s

C1: BTWV DLAY, 1

Set channel one burst to infinite

C1: BTWV TIME, INF

Read channel two burst parameters of which 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 channel two burst parameters of which STATE is OFF.

C2: BTWV?

Return:

C2: BTWV STATE, OFF

Note:

Parameter/command	SDG800	SDG1000	SDG2000X	SDG5000	SDG1000X
<channel>	no(single channel)	yes	yes	yes	yes
TRMD	no	yes	yes	yes	yes
EDGE	no	yes	yes	yes	yes
CARR, DLY	yes	yes	yes	yes	yes
CARR, RISE	yes	no	yes	yes	yes
CARR, FALL	yes	no	yes	yes	yes

3.8 Parameter Copy Command

DESCRIPTION

Copies parameters from one channel to another.

COMMAND SYNTAX

PACP(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 channel one to channel two.

PACP C2, C1

RELATED COMMANDS

ARWV, BTWV, MDWV, SWWV, BSWV

Note:

Parameter/command	SDG800	SDG1000	SDG2000X	SDG5000	SDG1000X
PACP	no	yes	yes	yes	yes

3.9 Arbitrary Wave Command

DESCRIPTION

Sets and gets arbitrary wave type.

COMMAND SYNTAX <channel> ARWV(ArbWaVe) INDEX,<value1>, NAME,<value2>
 <channel>:={C1, C2}
 < value1>: the table below shows what the index number mean.)
 < value2>: see table below.

QUERY SYNTAX <channel>: ARWV (ARbWaVe)?
 <channel>:={C1, C2}

RESPONSE FORMAT <channel>:ARWV <index>

EXAMPLE Set StairUp arbitrary wave output by index.
 C1:ARWV INDEX, 2

Read system current wave.
 ARWV?
 Return:
 ARWV INDEX,2,NAME,StairUp

Set Cardiac arbitrary wave output by name.
 ARWV NAME, Cardiac

<table>:

Index	Name	Index	Name	Index	Name	Index	Name
0	Sine	12	Logfall	24	Gmonopuls	36	Triang
1	Noise	13	Logrise	25	Tripuls	37	Harris
2	StairUp	14	Sqrt	26	Cardiac	38	Bartlett
3	StairDn	15	Root3	27	Quake	39	Tan
4	Stairud	16	X^2	28	Chirp	40	Cot
5	Ppulse	17	X^3	29	Twotone	41	Sec
6	Npulse	18	Sinc	30	Snr	42	Csc
7	Trapezia	19	Gaussian	31	Hamming	43	Asin
8	Upramp	20	Dlorentz	32	Hanning	44	Acos
9	Dnramp	21	Haversine	33	Kaiser	45	Atan
10	Exp_fall	22	Lorentz	34	Blackman	46	Acot
11	Exp_rise	23	Gauspuls	35	Gausswin	47	Square

About the table: This table is just an example, the index may depend on the model, you can execute “STL?” command to get them accurately.

Note:

Parameter/command	SDG800	SDG1000	SDG2000X	SDG5000	SDG1000X
<channel>	no(single channel)	yes	yes	yes	yes

INDEX	yes	yes	yes(only built-in wave)	yes	yes(only built-in wave)
NAME	yes	yes	yes(user define wave)	yes	yes(user define wave)

3.10 Sync Command

DESCRIPTION	Sets synchronization signal.
COMMAND SYNTAX	<channel>: SYNC <parameter> <channel>:={C1, C2} <parameter>:= {ON, OFF}
QUERY SYNTAX	<channel>: SYNC? <channel>:={C1, C2}
RESPONSE FORMAT	<channel>:SYNC <parameter>
EXAMPLE	Turn on sync function of channel one. C1: SYNC ON Read state of channel one sync. C1: SYNC? Return: C1: SYNC OFF

Note:

Parameter/command	SDG800	SDG1000	SDG2000X	SDG5000	SDG1000X
SYNC	no	yes	yes	yes	yes

3.11 Number Format Command

DESCRIPTION

Sets or gets number format.

COMMAND SYNTAX

NBFM(NumBer_ForMat) <parameter>

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

Parameters	Value	Description
PNT	<pnt>	Point format
SEPT	<sept>	Separator format

Where:

<pnt>:= {Dot, Comma}.

<sept> := { Space, Off, On}.

QUERY SYNTAX	NBFM (NumBer_FoRMat)?
RESPONSE FORMAT	NBFM <parameter>
EXAMPLE	Set point format to DOT. NBFM PNT, DOT Set Separator format to ON. NBFM SEPT,ON Read number format. NBFM? Return: NBFM PNT, DOT, SEPT, ON

3.12 Language Command

DESCRIPTION	Sets or gets system language.
COMMAND SYNTAX	LAGG(LAnGuaGe) <parameter> <parameter>:={EN, CH, RU}
QUERY SYNTAX	LAGG (LAnGuaGe)?
RESPONSE FORMAT	LAGG <parameter>
EXAMPLE	Set language to English. LAGG EN Read language LAGG? Return: LAGG EN

Note:

Parameter/command	SDG800	SDG1000	SDG2000X	SDG5000	SDG1000X
RU	no	yes	no	no	no

3.13 Configuration Command

DESCRIPTION	Sets or gets the power-on system setting..
COMMAND SYNTAX	SCFG(Sys_CFG)<parameter> <parameter>:= {DEFAULT, LAST}
QUERY SYNTAX	SCFG (Sys_CFG)?
RESPONSE FORMAT	SCFG <parameter>
EXAMPLE	Set the power-on system setting to LAST. SCFG LAST

3.14 Buzzer Command

DESCRIPTION	Turns on or off the buzzer.
COMMAND SYNTAX	BUZZ(BUZZer) <parameter> <parameter>:= {ON, OFF}
QUERY SYNTAX	BUZZ (BUZZer)?
RESPONSE FORMAT	BUZZ <parameter>
EXAMPLE	Turn on the buzzer. BUZZ ON

3.15 Screen Save Command

DESCRIPTION	Turns off or sets screen save time (default unit is minutes).
COMMAND SYNTAX	SCSV (SCreen_SaVe) <parameter> <parameter>:= {OFF, 1, 5, 15, 30, 60, 120, 300 }
QUERY SYNTAX	SCSV (SCreen_SaVe)?
RESPONSE FORMAT	SCreen_SaVe <parameter>

EXAMPLE Set screen save time to 5 minutes.
SCSV 5

Read the current screen save time.
SCreen_SaVe?
Return:
SCSV 5MIN

3.16 Clock Source Command

DESCRIPTION Sets or gets the clock source.

COMMAND SYNTAX ROSC (ROSCillator) <parameter>
<parameter>:= {INT, EXT}

QUERY SYNTAX ROSC (ROSCillator)?

RESPONSE FORMAT ROSC <parameter>

EXAMPLE Set internal time base as the clock source.
ROSC INT

Note:

Parameter/command	SDG800	SDG1000	SDG2000X	SDG5000	SDG1000X
ROSC	no	yes	yes	yes	yes

3.17 Frequency Counter Command

DESCRIPTION Sets or gets frequency counter parameters.

COMMAND SYNTAX FCNT(FreqCouNter) <parameter>
<parameter>:= {a parameter from the table below}

Parameters	Value	Description
STATE	<state>	State of frequency counter.
FRQ	<frequency>	Value of frequency. Can't be set.
PW	<position width>	Value of positive width. Can't be set.
NW	<negative width>	Value of negative width. Can't be set.
DUTY	<duty>	Value of duty cycle. Can't be set.
FRQDEV	<freq deviation>	Value of freq deviation. Can't be set.
REFQ	<ref freq>	Value of reference freq.

TRG	<triglev>	Value of trigger level.
MODE	<mode>	Value of mode.
HFR	<HFR>	State of HFR.

where:

- < state >:={ON, OFF}
- <frequency>:= {Default unit is "Hz". Value range depends on the model.}
- < mode >:={AC, DC}
- <HFR>:={ON, OFF}

QUERY SYNTAX FCNT (FreqCouNter)?

RESPONSE FORMAT FCNT < state ><frequency><duty><ref freq><triglev><position width><negative width>
<freq deviation><mode><HFR>

EXAMPLE

Turn frequency counter on:
FCNT STATE,ON
Set reference freq to 1000Hz:
FCNT REFQ,1000
Query frequency counter information:
FCNT?
Return:
FCNT STATE,ON,FRQ,100000000HZ,DUTY,59.8568,REFQ,
1e+07HZ,TRG,0V,PW,5.98568e-08S,NW,4.01432e-08S,FR
QDEV,0ppm,MODE,AC,HFR,OFF

Note:

Parameter/command	SDG800	SDG1000	SDG2000X	SDG5000	SDG1000X
FCNT	no	yes	yes	yes	yes

3.18 Invert Command

DESCRIPTION Sets or gets polarity of current channel.

COMMAND SYNTAX <channel>:INVT(INVerT) <parameter>
<channel>:={C1, C2}
<parameter>:= {ON, OFF}

QUERY SYNTAX <channel>: INVT (INVerT)?
<channel>:={C1, C2}

RESPONSE FORMAT <channel>:INVerT <parameter>

EXAMPLE

Set C1 ON:
C1: INVT ON

Read the polarity of channel one.
C1: INVT?
Return:
C1: INVT ON

Notes:

1.

Parameter/command	SDG800	SDG1000	SDG2000X	SDG5000	SDG1000X
<channel>	no(single channel)	yes	yes	yes	yes

2. The <channel> is a selectable parameter. If channel is not set, default is current channel.

3.19 Coupling Command

DESCRIPTION Sets or gets channel coupling parameters. You can only set coupling value when trace switch off.

COMMAND COUP (COUPling)<parameter>

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

Parameters	Value	Description
TRACE	<trace>	Trace switch
STATE	<state>	State of channel coupling.
BSCH	<bsch>	Value of base channel.
FDEV	<frq_dev>	Value of f frequency deviation.
PDEV	<pha_dev>	Value of position phase deviation.
FCOUP	<fcoup>	Value of frequency coupling switch
FRAT	<frat>	Value of frequency coupling ratio
PCOUP	<pcoup>	Value of phase coupling switch
PRAT	<prat>	Value of phase coupling ratio
ACOUP	<acoup>	Value of amplitude coupling switch
ARAT	<arat>	Value of amplitude coupling ratio
ADEV	<adev>	Value of amplitude coupling deviation

where:

<trace>:={ON, OFF}
< state >:={ON, OFF}

< bsch >:= {CH1, CH2}
 < frq_dev >:= { Default unit is “Hz”, value range depends on the model}
 < pha_dev >:= { Default unit is “°” value range depends on the model }
 <fcoup>,<acoup>,<pcoup>:= {ON, OFF}
 <frat>,<prat>,< arat >:= {a ratio value. value range depends on the model }
 <adev>:= { a deviation value. value range depends on the model }

QUERY SYNTAX COUP (COUPling)?

EXAMPLE Set SDG5000 coupling state on
COUP STATE,ON

Set SDG5000 frequency deviation value 5Hz
COUP FDEV,5

Set SDG2000x amplitude coupling ratio
COUP ARAT,2

Query SDG2000X coupling information.
COUP?

Return:
COUP\STATE,OFF,FCOUP,ON,PCOUP,ON,ACOUP,ON,FDEV,5HZ,
PRAT,1,ARAT,2\n

Note:

Parameter/command	SDG800	SDG1000	SDG2000X	SDG5000	SDG1000X
TRACE	no	no	yes	no	yes
STATE	yes	yes	no	yes	no
BSCH	yes	yes	no	yes	no
FCOUP	no	no	yes	no	yes
FRAT	no	no	yes	no	yes
PCOUP	no	no	yes	no	yes
PRAT	no	no	yes	no	yes
ACOUP	no	no	yes	no	yes
ARAT	no	no	yes	no	yes
ADEV	no	no	yes	no	yes

3.20 Voltage Overload Command

DESCRIPTION Sets or gets state of over-voltage protection.

COMMAND SYNTAX VOLTPRT<parameter>
 <parameter>:= {ON, OFF}

QUERY SYNTAX VOLTPRT?

RESPONSE FORMAT VOLTPRT<parameter>

Note:

Parameter/command	SDG800	SDG1000	SDG2000X	SDG5000	SDG1000X
VOLTPRT	no	yes	yes	no	yes

3.21 Store List Command

DESCRIPTION This command is used to read the stored wave data names if the store unit is empty; the command will return “EMPTY” string.

Note: M50~ M59 is user defined memory. The name will return what you defined. if you do not define an arbitrary name, it will return “EMPTY”(It is depends on the model).

QUERY SYNTAX STL (StoreList)? BUILDIN, USER

EXAMPLE Read all arbitrary data saved in the device.

STL?

Return:

STL M0, StairUp, M1, StairDn, M2, StairUD, M3, Trapezia, M4, ExpFall, M5, ExpRise, M6, LogFall, M7, LogRise, M8, Sqrt, M9, X^2, M10, Sinc, M11, Gaussian, M12, Dlorentz, M13, Haversine, M14, Lorentz, M15, Gauspuls, M16, Gmonopuls, M17, Cardiac, M18, Quake, M19, TwoTone, M20, SNR, M21, Hamming, M22, Hanning, M23, Kaiser, M24, Blackman, M25, GaussiWin, M26, Harris, M27, Bartlett, M28, Tan, M29, Cot, M30, Sec, M31, Csc, M32, Asin, M33, Acos, M34, Atan, M35, ACot, M36, EMPTY, M37

Read built-in wave data.

STL? BUILDIN

Return:

STL M0, Sine, M1, Noise, M10, ExpFal, M11, ExpRise, M12, LogFall, M13, LogRise, M14, Sqrt, M15, Root3, M16, X^2, M17, X^3, M18, Sinc, M19, Gussian, 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, Triang, M37, Harris, M38, Bartlett, M39, Tan, M4, StairUD, M40, Cot, M41, Sec, M42, Csc, M43, Asin, M44, Acos, M45, Atan, M46, Acot, M47, Square, M5, Ppulse, M6, Npulse, M7, Trapezia, M8, Upramp, M9, Dnramp

Read wave data defined by user.

STL? USER

Return:

STL

WVNM,sinec_8M,sinec_3000000,sinec_1664000,ramp_8M,
sinec_2000000,sinec_50000,square_8M,sinec_5000,wave1,
square_1M

Note:

Parameter/command	SDG800	SDG1000	SDG2000X	SDG5000	SDG1000X
BUILDIN	no	no	yes(get built-in wave)	no	yes(get built-in wave)
USER	no	no	yes(get user defined wave)	no	yes(get user defined wave)

3.22 Arbitrary Wave Data Command

DESCRIPTION Sets and gets arbitrary wave data.

COMMAND < channel>:WVDT <address>,<parameter>

SYNTAX <channel>:={C1, C2}

<address>:={Mn}(The “n” value is based on the model, but SDG2000X/SDG1000X User define wave don’t have it(see notes table below).)

<parameter>: see the table below.

Parameters	Value	Description
WVNM	<wave name>	Wave name.
TYPE	<type>	Wave type.
LENGTH	<length>	Wave length, the value depends on the product(SDG800 /SDG1000: 16kb;SDG5000:16kb,512kb;SDG2000X/SDG1000X: 8b~8M)

FREQ	<frequency>	Wave frequency.
AMPL	<amplifier>	Wave amplifier.
OFST	<offset>	Wave offset.
PHASE	<phase>	Wave phase.
WAVEDATA	<wave data>	Wave data.

QUERY SYNTAX

For all the arbitrary wave of SDG800/1000/5000 and the built-in wave of SDG2000X /SDG1000X:
WVDT? Mn

For SDG2000X / SDG1000X user define wave:
WVDT? USER,<wave name>
<wave name>:={The name of user define wave}

EXAMPLE

Send wave1 to the M55 address of SDG1000.
C1:WVDT M55,WVNM,wave1,TYPE,5,LENGTH,16KB,WAVED
ATA, xxxxxxxx

Send wave1 to SDG2000X.
C1:WVDT WVNМ,wave1,TYPE,5,LENGTH,16384B,FREQ,1000,
WAVEDATA, xxxxxxxx

[illegible]

Query SDG2000X user define wave (wave1) command.
WVDT? USER,wave1
Return:
WVDT\ sPOS,\ s/Local,\ sWVNM,\ swave1,\ sLENGTH,\ s1048576B,\ sTYPE,\ s6,
\ sWAVEDATA.\ 00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\FE\

```

FF\FE\FF\FE\FF\FE\FF\FE\FF\FE\FF\FE\FF\FD\FF\FD\FF\FD\FF\
FD\FF\FD\FF\FD\FF\FD\FF\FD\FF\FC\FF\FC\FF\FC\FF\FC\FF\FC\
FF\FC\FF\FC\FF\FA\FF\FA\FF\FA\FF\FA\FF\FA\FF\FA\FF\FA\FF\
F9\FF\F9\FF\F9\FF\F9\FF\F9\FF\F9\FF\F9\FF\F8\FF\F8\FF\F8\
FF\F8\FF\F8\FF\F8\FF\F8\FF\F7\FF\F7\FF\F7\FF\F7\FF\F7\FF\
F7\FF\F7\FF\F7\FF\F6\FF\F6\FF\F6\FF\F6\FF\F6\FF\F6\FF\F6\
FF\F6\FF\F6\FF\F6\FF\F6\FF\F6\FF\F6\FF\F5\FF\F5\FF\
.....

```

Note:

Parameter/comm and	SDG800	SDG1000	SDG2000X	SDG5000	SDG1000X
Mn	(0<=n<=59): M0~M49: build in wave (32KB). M50~M59: can store user defined wave (32KB)	(0<=n<=59): M0~M49: build in wave (32KB). M50~M59: can store user defined wave (32KB)	(0<=n<=196): M0~M196: all of them are building in waves (32KB). user defined waves have not this index.	(0<=n<=68): M0~M35: build in wave (32KB). M36~M59: User define wave (32KB). M60~M67: User defined wave(1024KB)	(0<=n<=196): M0~M196: all of them are building in waves (32KB). user defined waves have not this index.
USER	no	no	yes(get user defined wave)	no	yes(get user defined wave)

3.23 Virtual Key Command

DESCRIPTION

The Command is used to send simulate a operation of pressing key on front panel.

COMMAND SYNTAX

VKEY (VirtualKEY) VALUE,<value>,STATE,<sate>
 <value>:= {a parameter from the table below.}
 <state>:=<0,1>("1" is effective to virtual value, and "0" is useless)

EXAMPLE

VKEY VALUE,15, STATE,1
 VKEY VALUE,KB_SWEEP, STATE,1

Note:

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_NUMBER_1	49		
KB_NUMBER_2	50		
KB_NUMBER_3	51		

Note:

Parameter/command	SDG800	SDG1000	SDG2000X	SDG5000	SDG1000X
KB_FUNC1	no	no	yes	yes	yes
KB_STORE_RECALL	yes	yes	yes	no	yes
KB_HELP	yes	yes	no	no	no
KB_CHANNEL	no	yes	yes	no	yes
KB_SINE	yes	yes	no	no	no
KB_SQUARE	yes	yes	no	no	no
KB_RAMP	yes	yes	no	no	no
KB_PULSE	yes	yes	no	no	no
KB_NOISE	yes	yes	no	no	no
KB_ARB	yes	yes	no	no	no
KB_UP	yes	yes	no	no	no
KB_DOWN	yes	yes	no	no	no

3.24 IP Command

DESCRIPTION

The Command can set and get system IP address.

COMMAND SYNTAX

SYST:COMM:LAN:IPAD
 (SYSTEM:COMMunicate:LAN:IPADdress)
 <parameter1>.<parameter2>.<parameter3>.<parameter4>

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

QUERY SYNTAX

SYST:COMM:LAN:IPAD
 (SYSTem:COMMunicate:LAN:IPADdress)?

EXAMPLES

Set IP address to 10.11.13.203
 SYSTem: COMMunicate: LAN:IPADdress 10.11.13.203

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

Note:

Parameter/command	SDG800	SDG1000	SDG2000X	SDG5000	SDG1000X
SYST:COMM:LAN:IPAD	no	no	yes	no	yes

3.25 Subnet Mask Command

DESCRIPTION

The Command can set and get system subnet mask.

COMMAND SYNTAX

SYST:COMM:LAN:SMAS (SYSTem:COMMunicate:LAN:SMASk)
 <parameter1>.<parameter2>.<parameter3>.<parameter4>

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

QUERY SYNTAX

SYSTem:COMMunicate:LAN:SMASk?

EXAMPLES

Set subnet mask to 255.0.0.0
 SYSTem:COMMunicate:LAN:SMASk 255.0.0.0

Get subnet mask
 SYSTem:COMMunicate:LAN:SMASk?
 Return:
 "255.0.0.0"

Note:

Parameter/command	SDG800	SDG1000	SDG2000X	SDG5000	SDG1000X
SYST:COMM:LAN:SMAS	no	no	yes	no	yes

3.26 Gateway Command

DESCRIPTION	The Command can set and get system Gateway.
COMMAND SYNTAX	SYST:COMM:LAN:GAT(SYSTem:COMMunicate:LAN:GATeway) <parameter1>.<parameter2>.<parameter3>.<parameter4> <parameter1>:={a integer value between 0 and 223} <parameter2>:={a integer value between 0 and 255} <parameter3>:={a integer value between 0 and 255} <parameter4>:={a 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:

Parameter/command	SDG800	SDG1000	SDG2000X	SDG5000	SDG1000X
SYST:COMM:LAN:GAT	no	no	yes	no	yes

3.27 Sampling Rate Command

DESCRIPTION	Sets or gets sampling rate. You can only use it in TrueArb mode.
COMMAND SYNTAX	<channel>:SRATE(SampleRATE) MODE <parameter1>, VALUE, <parameter2> <channel> :=<C1, C2> <parameter1> :=< DDS, TARB> <parameter2> :={ a integer value between 1e-6 and 75000000, (default unit is Sa/s)}

QUERY SYNTAX <channel>: SRATE?

EXAMPLES

Get the channel one sample rate value

C1: SRATE?

Return:

C1: SRATE MODE, DDS

Set channel one to TureArb mode.

C1: SRATE MODE, TARB

Set channel one sample rate value to 1000000Sa/s.

C1: SRATE VALUE, 1000000

Note:

Parameter/command	SDG800	SDG1000	SDG2000X	SDG5000	SDG1000X
<channel>	No(single channel)	yes	yes	yes	yes
SRATE	no	no	yes	no	no

3.28 Harmonic Command

DESCRIPTION

Sets or gets harmonic information. The command can be used by SDG2000X/SDG1000X.and the channel current basic wave must be sine.

COMMAND SYNTAX

<channel>:HARM(HARMonic) HARMSTATE,<value1>, HARMTYPE, < value2>, HARMORDER,< value3>, <parameter>, <value4>, HARMPHASE, < value5>

< value1>:= <ON, OFF>

< value2>:= <EVEN, ODD, ALL>

< value3>:= {an integer value.}

<parameter> :=< HARMAMP, HARMDBC>

< value4>:= {an integer value.}

< value5>:= {an integer value.}

QUERY SYNTAX

<channel>: HARM (HARMonic)?

<channel>:={C1, C2}

EXAMPLES

Set the channel one harmonic switch on.

C1: HARMHARMSTATE, ON

Get the channel one harmonic information.

C1: HARM?

Return:

C1:HARM HARMSTATE, ON,HARMTYPE, EVEN,HARMORDER,
2, HARMAMP, 0V, HARMPHASE, 0

Note:

Parameter/command	SDG800	SDG1000	SDG2000X	SDG5000	SDG1000X
HARM	no	no	yes	no	yes

3.29 Waveform Combining Command

DESCRIPTION Sets or gets waveform combining information. The command can be used by SDG2000X/SDG1000X.

COMMAND SYNTAX <channel>:CMBN (CoMBiNe) <parameter>
 <channel>:={C1, C2}
 <parameter>:= {ON, OFF}

QUERY SYNTAX <channel>: CMBN (CoMBiNe)?
 <channel>:={C1, C2}

EXAMPLES Turn on the waveform combining of channel one.
 C1:CMBN ON
 Query the waveform combining state of channel two.
 C2:CMBN?
 Return:
 C2:CMBN OFF

Note:

Parameter/command	SDG800	SDG1000	SDG2000X	SDG5000	SDG1000X
CMBN	no	no	yes	no	yes

4 Programming Examples

This chapter gives some examples for the programmer. In these examples you can see how to use the NI-VISA lib and the commands which have been described before this chapter to control our devices. By the examples' guide, you can develop more functions application as you want. This example is developed by Visual Studio project.

Main topics of this part:

- Example of Vc++
- Example of VB
- Example of MATLAB
- Example of LabVIEW

4.1 Example of VC++

Environment: Win7 32bit system, Visual Studio

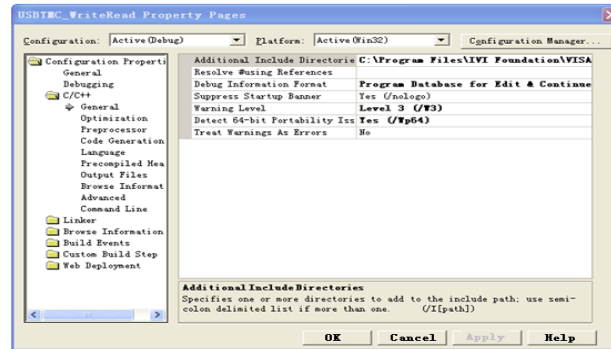
The functions of this example: use the NI-VISA, to control the device with USBTMC or TCP/IP access to do a write and read.

Follow the steps to finish the example:

- 1、Open Visual Studio, create a new VC++ win32 console project.
- 2、Set the project environment to use the NI-VISA lib, there are two ways to use NI-VISA, static or automatic:
 - 2.1 Static: find files: visa.h, visatype.h, visa32.lib in NI-VISA install path. Copy them to your project, and add them into project. In the projectname.cpp file, add the follow two lines:

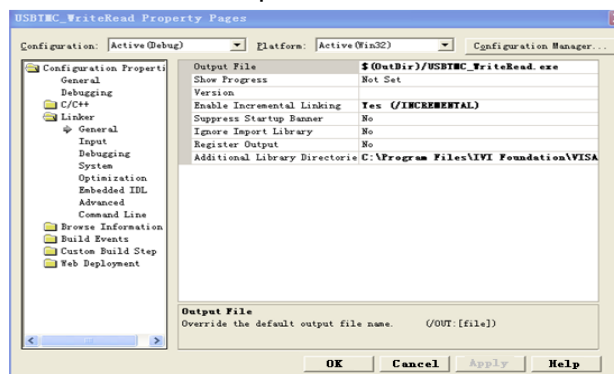
```
#include "visa.h"
#pragma comment(lib,"visa32.lib")
```
 - 2.2 Automatic:

Set the .h file include directory, the NI-VISA install path, in our computer we set the path is : C:\Program Files\IVI Foundation \VISA\WinNT\include. Set this path to project---properties---c/c++---General---Additional Include Directories: See the picture.

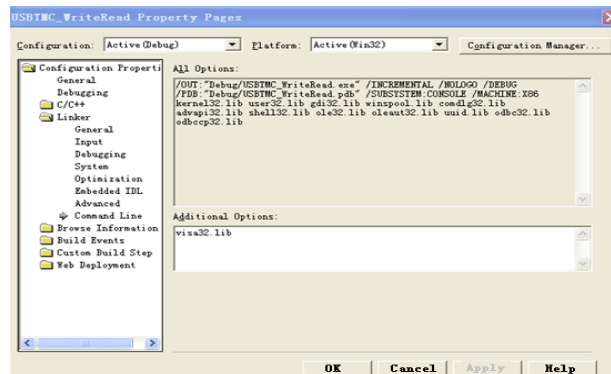


Set lib path set lib file:

Set lib path: the NI-Visa install path, in our computer we set the path is :
C:\Program Files\IVI Foundation\Visa\WinNT
lib\msc. Set this path to project---properties---Linker---General---Additional
Library Directories: as seen in the pictures below.



Set lib file:project---properties---Linker---Command Line---Additional Options:
visa32.lib



Include visa.h file: In the projectname.cpp file:

```
#include <visa.h>
```

3、 Add codes:

3.1 USBTMC access code.

Write a function Usbtmc_test:

```
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;

ViUInt32 retCount;

ViUInt32 writeCount;

ViStatus status;

char instrResourceString[VI_FIND_BUFLLEN];

unsigned char buffer[100];

char stringinput[512];

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.\nHit 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<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 * cmmmand = "*IDN?\n";

    status = viPrintf (instr, cmmmand);

    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);

```

```

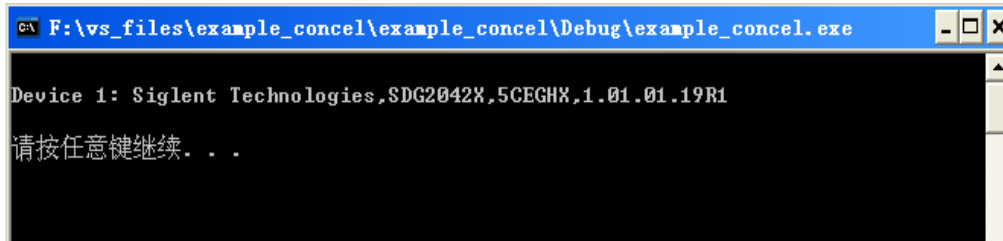
    system("pause");    // pause to keep off the console flashed.

    return 0;

}

```

Result:



3.2 TCP/IP access code.

Write a function TCP_IP_Test:

```

int TCP_IP_Test(char *pIP)
{
    char outputBuffer[VI_FIND_BUFLLEN];

    ViSession defaultRM, instr;

    ViStatus status;

    ViUInt32 count;

    ViUInt16 portNo;

    /* 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";

    char resource [256];

    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 ("\ndata read from device: %*s\n", 0,outputBuffer);

status = viClose (instr);

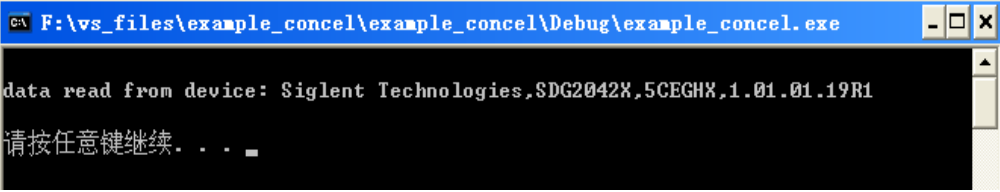
status = viClose (defaultRM);

system("pause");

return 0;

}
```

Run result.



The screenshot shows a Windows command prompt window with the title bar "F:\vs_files\example_concel\example_concel\Debug\example_concel.exe". The window has a black background and white text. The first line of output is "data read from device: Siglent Technologies,SDG2042X,5CEGHX,1.01.01.19R1". The second line is "请按任意键继续. . .", followed by a cursor. The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

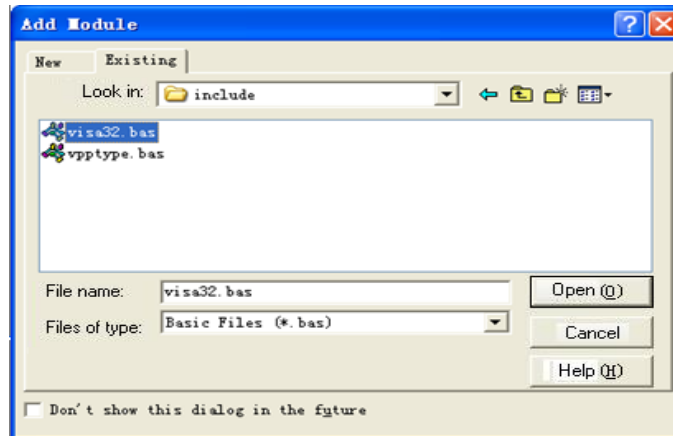
4.2 Example of VB

Environment: Win7 32bit system, Microsoft Visual Basic 6.0

The function of this example: Use the NI-VISA, to control the device with USBTMC and TCP/IP access to do a write and read.

Follow the steps to complete the example:

1. Open Visual Basic, build a standard application program project (Standard EXE)
2. Set the project environment to use the NI-VISA lib, Click the Existing tab of Project>>Add Existing Item. Search for the visa32.bas file in the include folder under the NI-VISA installation path and add the file.



This allows the VISA functions and VISA data types to be used in a program.

3. Add codes:

3.1、USBTMC access code.

Write a function Usbtmc_test:

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 writeCount 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?*INSTR", 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 device'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)

NextFind:

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

3.2、TCP/IP access code.

Write a function TCP_IP_Test:

Private Function TCP_IP_Test(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 + "::INSTR", VI_LOAD_CONFIG, VI_NULL, instrsesn)

If (status < VI_SUCCESS) Then

    resultTxt.Text = "An error occurred opening the session"

    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
```

3.3、 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
```

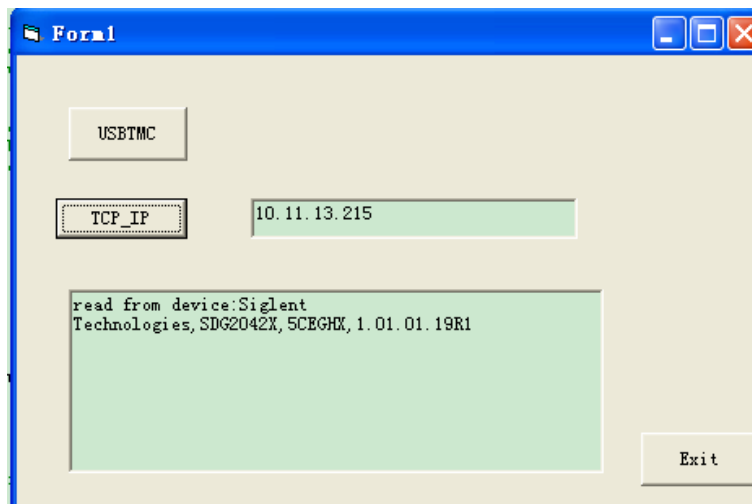
```
If (stat < VI_SUCCESS) Then
```

```
    resultTxt.Text = Hex(stat)
```

```
End If
```

```
End Sub
```

3.4、 Run result:



4.3 Example of MATLAB

Environment: Win7 32bit system, MATLAB R2010b

The function of this example: Use the NI-VISA, to control the device with USBTMC or TCP/IP access to do a write and read.

Follow the steps to complete the example:

Open MATLAB, modify the current directory. In this demo, the current directory is modified to D:\USBTMC_TCPIP_Demo.

Click File>>New>>Script in the Matlab interface to create an empty M file

Add codes:

1.1 Write a function Usbtmc_test.

```
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::0xF4EC::0xEE38::0123456789::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
```

```
11
12 %Send the string "*IDN?", asking for the device's identification.
13 - fprintf(vu, '*IDN?');
14
```

```
Siglent Technologies,SDG2042X,5CEGHX,1.01.01.19R1
Siglent Technologies,SDG2042X,5CEGHX,1.01.01.19R1
Siglent Technologies,SDG2042X,5CEGHX,1.01.01.19R1
Siglent Technologies,SDG2042X,5CEGHX,1.01.01.19R1
```

```
fx >>
```

1.2 TCP/IP access code.

Write a function TCP_IP_Test:

```
function TCP_IP_test( IPstr )
```

```
% This code demonstrates sending synchronous read & write commands
% to an TCP/IP instrument using NI-VISA
```

```
%Create a VISA-TCPIP object connected to an instrument
%configured with IP address.
vt = visa('ni',['TCPIP0::',IPstr,':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
```

```
11
12 %Send the string "*IDN?", asking for the device's identification.
13 - fprintf(vt, '*IDN?');

1 usage of "TCP_IP_test" found

>> TCP_IP_test('10.11.13.215')
Siglent Technologies, SDG2042X, 5CEGHX, 1.01.01.19R1

fx >>
```

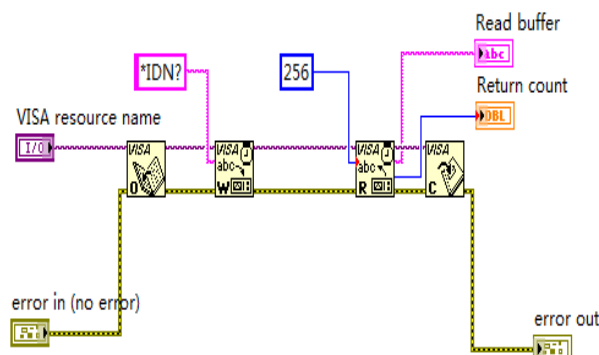
4.4 Example of LabVIEW

Environment: Win7 32bit system, LabVIEW 2011

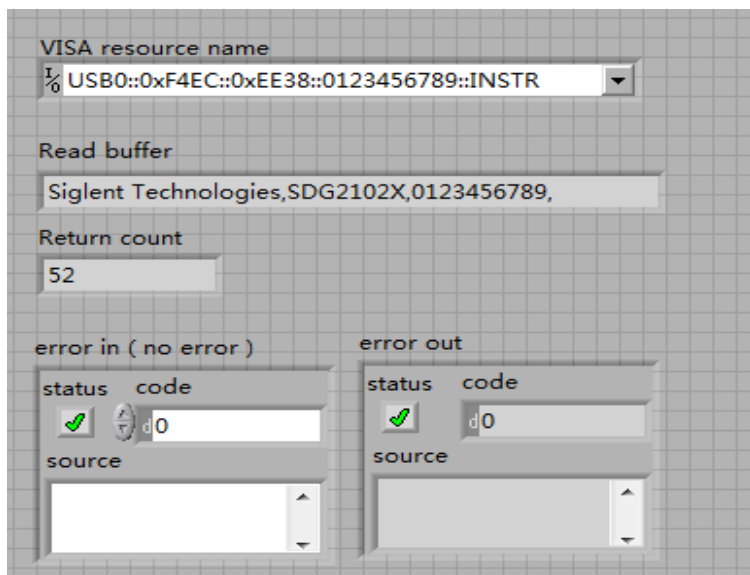
The functions of this example: use the NI-VISA, to control the device with USBTMC and TCP/IP access to do a write and read.

Follow the steps to complete the example:

- 1、Open LabVIEW, 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** and you can select and add the following functions from VISA Palette from the pop-up menu: **VISA Write**, **VISA Read**, **VISA Open** and **VISA Close**.
- 4、Connect them as shown in the figure below



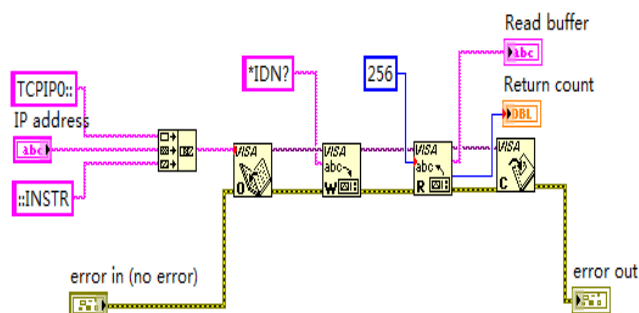
- 5、Select the device resource from the VISA Resource Name list box and run the program.



In this example, the VI opens a VISA session to a USBTMC device, writes a command to the device, and reads back the response. In this example, the specific command being sent is the device ID query. Check with your device manufacturer for the device command set. 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、Connect them 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	代碼
✓	0
source	
error out	
status	code
✓	0
source	

Note: you can obtain the source code of above examples, please visit SIGLENT website at www.siglent.com.

5 Index

*IDN

OPC

*CLS

*ESE

*ESR

*RST

*SRE

*STB

*TST

*WAI

DDR

CMR

A

ARWV ARBWAVE

B

BSWV BASIC_WAVE

BTWV BURSTWAVE

BUZZ BUZZER

C

CHDR COMM_HEADER

COUP COUPLING

CMBN COMBINE

F

FCNT FREQCOUNTER

H

HARM HARMONIC

I

IVNT INVERT

L

LAGG LANGUAGE

M

MDWV MODULATEWAVE

N

NBFM NUMBER_FORMAT

O

OUTP OUTPUT

P

PACP PARACOPY

R

ROSC ROSCILLATOR

S

SCFG Sys_CFG

SCSV SCREEN_SAVE

SWWV SWEEPWAVE

SYNC SYNC

STL STORELIST

SYST:COMM:LAN:IPAD SYSTEM:COMMUNICATE:LAN:IPADDRESS

SYST:COMM:LAN:SMAS SYSTem:COMMUnicate:LAN:SMASk

SYST: COMM: LAN:GAT SYSTem:COMMUnicate:LAN:GATeway

SRATE SAMPLERATE

W

WVDT WVDT

V

VOLTPRT VOLTPRT

VKEY VIRTUALKEY