SDG8000A 系列
函数/任意波形发生器

编程手册

CN01A

深圳市鼎阳科技股份有限公司
SIGLENT TECHNOLOGIES CO., LTD.

目录

1 编程概述

用户可以通过使用信号源的 USB、LAN 或 GPIB 端口，并结合 NI-VISA 和程序语言，远程控制信号源。基于 LAN 端口，SDG 支持 VXI-11、Sockets 和 Telnet 通信协议。本节介绍了如何建立 SDG 系列函数/任意波形发生器和计算机之间的通信，同时介绍如何远程控制信号源。

1.1 建立通信

1.1.1 NI-VISA 的安装

在编程之前，请确保正确安装 NI-VISA 软件的最新版本。

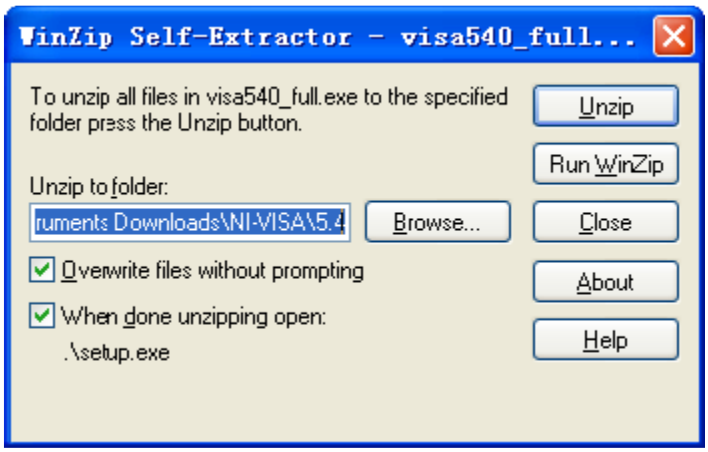
NI-VISA 是用于计算机与设备之间通信的通信库。NI 软件有两种有效 VISA 安装包：完整版和运行引擎版（Run-Time Engine）。完整版包括 NI 设备驱动和 NI MAX 工具，其中 NI MAX 是用于控制设备的用户界面。虽然驱动和 NI MAX 都很有用，但是它们不用于远程控制。运行引擎版（Run-Time Engine）是一个比完整版更小的文件，它主要用于远程控制。

你可以在 NI 官网上下载最新的 NI-VISA 运行引擎或完整版。它们的安装步骤基本相同。

按照下列步骤安装 NI-VISA（示例使用 NI-VISA5.4 完整版）：

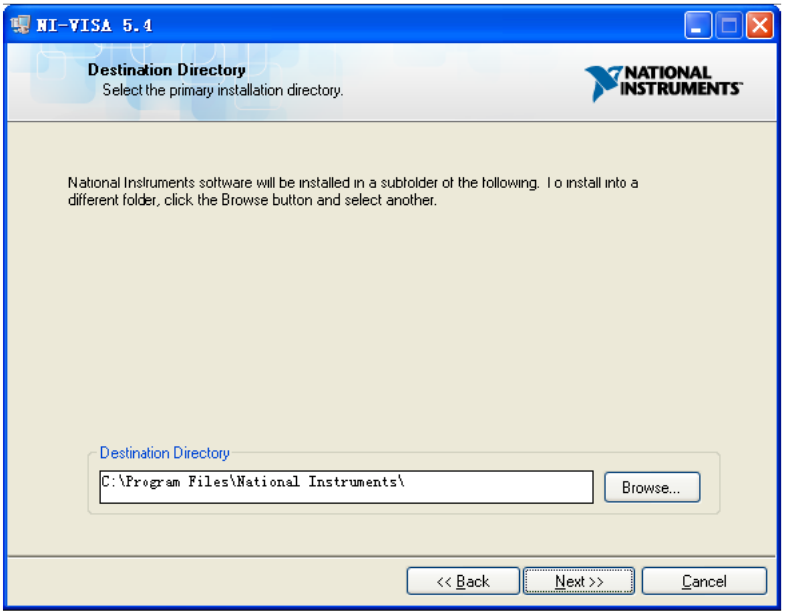
a. 下载合适版本的 NI-VISA（推荐运行引擎版）

b. 双击 visa540_full.exe，弹出对话框如下：

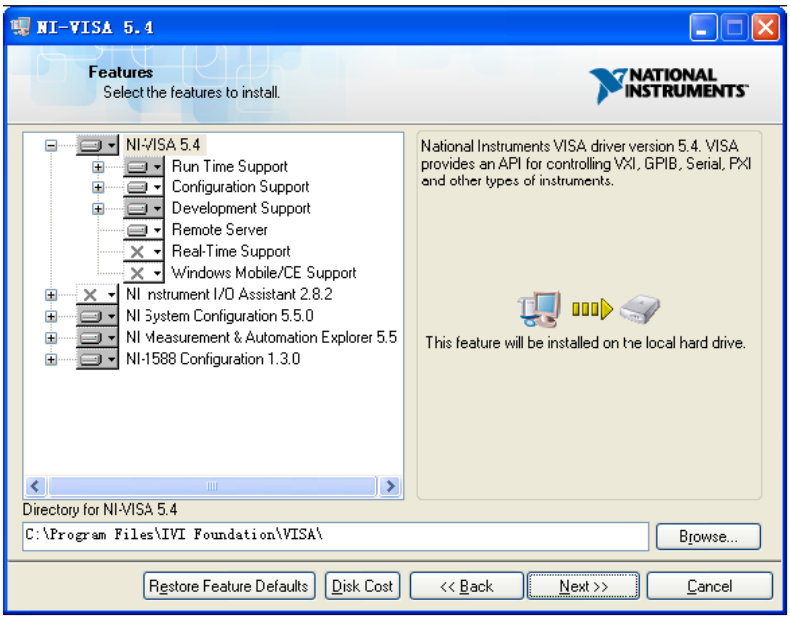


c. 点击 Unzip 解压文件，当解压完成后，安装程序将自动执行。若你的计算机需要安装.NET Framework4，则在安装过程会自动安装.NET Framework4。

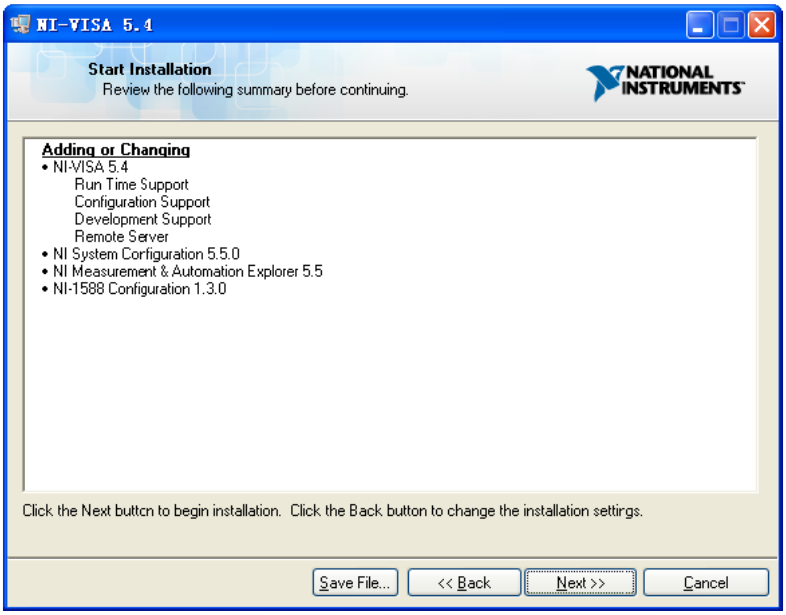
d. NI-VISA 安装对话框如上图所示，点击 Next 开始安装过程。



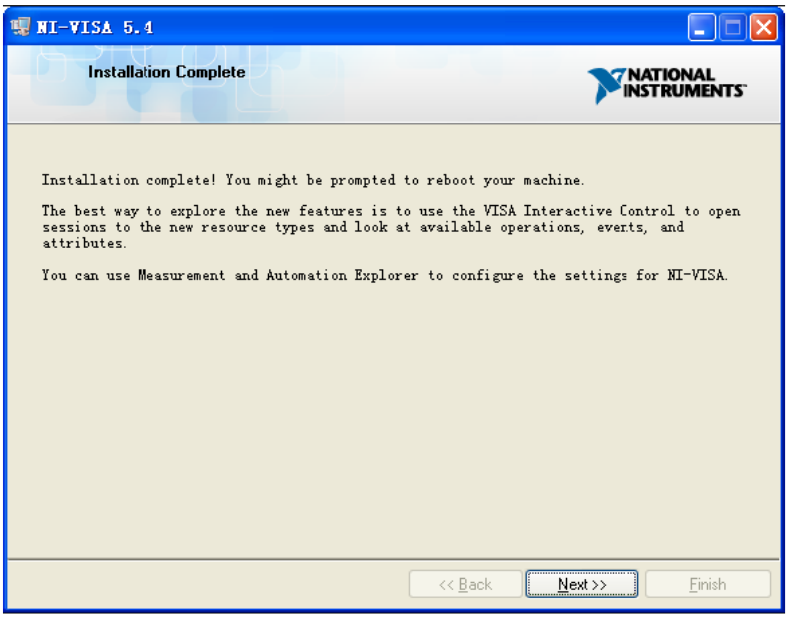
e. 设置安装路径，默认路径为“C:\Program Files\National Instruments\”。你也可以修改安装路径。点击 Next，对话框如下图所示。



f. 点击 Next 两次，在许可协议对话框下，选择“I accept the above 2 License Agreement(s).”并点击 Next，对话框如下图显示：



g. 点击 Next 开始安装：

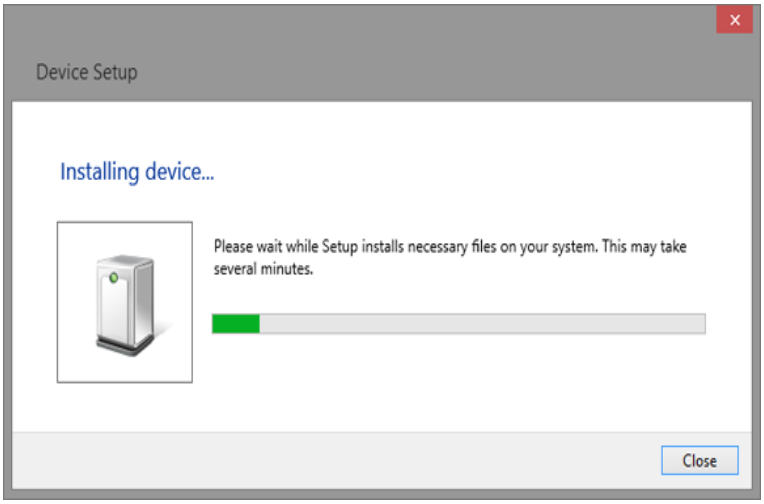


h. 安装完成后，重启电脑。

1.1.2 连接仪器

根据具体信号源机型，函数/任意波形发生器能够通过 USB、LAN 端口或 GPIB（选配）接口连接计算机。

使用 USB 线将函数/任意波形发生器的 USB Device 端口和计算机的 USBHost 端口连接起来。假设你的计算机已经启动，打开信号源后，桌面将弹出“设备安装”界面，并自动安装设备驱动，如下图所示：



等待安装完成，然后进行下一步。

1.2 远程控制的实现

1.2.1 用户自定义程序

用户可通过计算机发送 SCPI 命令实现编程和控制任意波形发生器。相关内容，请查阅“编程示例”中的介绍。

1.2.2 通过 NI-MAX 发送 SCPI 命令

NI-MAX 是由 NI 公司创建和维护的程序。它为 VXI、LAN、USB、GPIB 和串行通信提供基础的远程控制接口。用户可以通过 NI-MAX 发送 SCPI 命令远程控制信号源。

1.2.3 通过 Telnet 发送 SCPI 命令

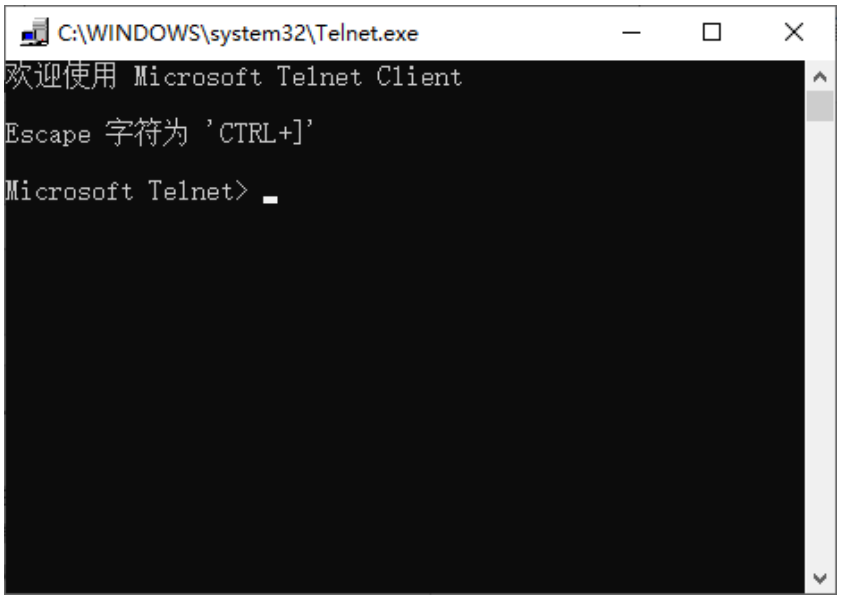
Telnet 提供一种通过 LAN 端口与信号源通信的方式。Telnet 协议支持从计算机向信号源发送 SCPI 命令，该方式类似于通过 USB 与信号源通信。发送和接受信息是交互的：一次只能发送一个命令。Windows 操作系统使用命令提示符样式接口作为 Telnet 客户端。

步骤如下：

1 在计算机桌面，点击开始>所有程序>附件>命令提示符

2 在命令提示符窗口，输入 Telnet

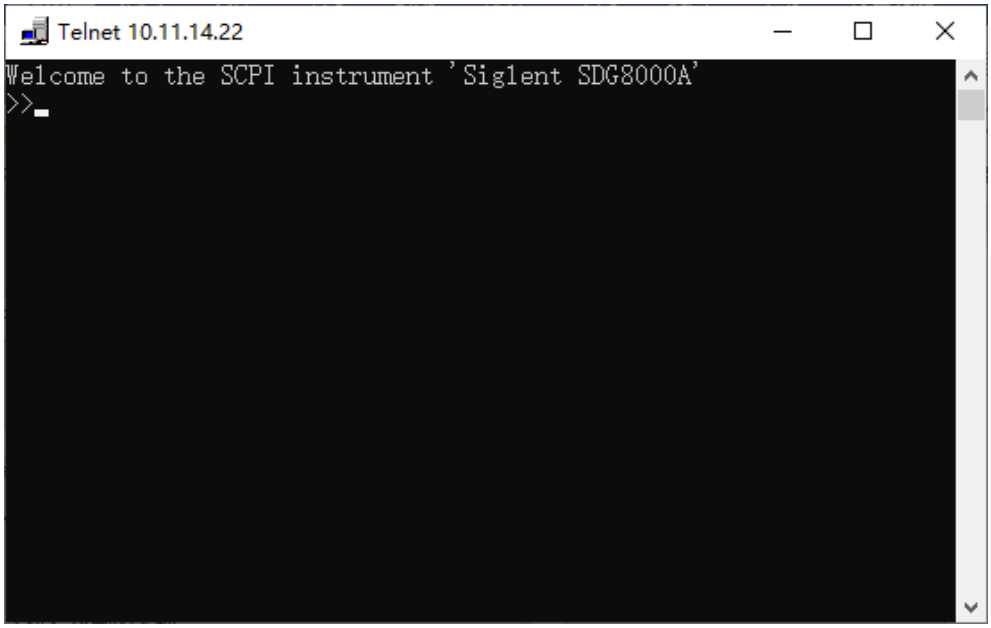
3 按下 Enter 键。将弹出 Telnet 显示窗



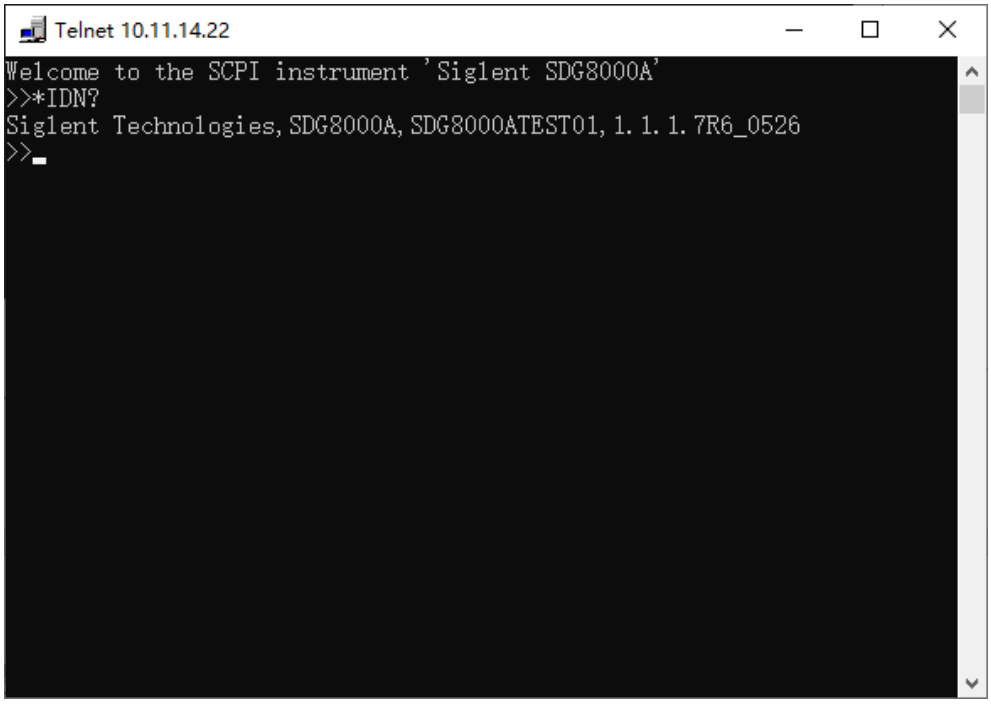
4　在 Telnet 命令行，键入：

open XXX.XXX.XXX.XXX 5024

其中 XXX.XXX.XXX.XXX 是指设备的 IP 地址；5024 是端口。通信成功后你会看到和下面类似的响应内容：



5　在“>>”提示符后，输入 SCPI 命令例如 **IDN?*。该命令将返回公司名、机器型号、序列号和固件版本号。



6 同时按下 Ctrl+] 键将退出 SCPI 会话。

7 通过在提示符出键入 *quit* 或关闭 Telnet 窗口来关闭设备的连接并退出 Telnet。

1.2.4 通过 Socket 发送 SCPI

Socket 接口可以在不安装其他库的情况下通过 LAN 端口控制 SDG 系列产品。它可以减少编程的复杂度。

SOCKET 地址	IP 地址+端口号
IP 地址	SDG IP 地址
端口号	5025

相关内容，请查阅第 5.2 节“Sockets 应用示例”。

2 SCPI 语言简介

2.1 有关命令和查询

本节列出并描述仪器识别的远程控制命令和查询语句。所有命令和查询都可以在本地或远程状态下执行。

本文对每个包含有语法以及其他信息的命令和查询都列举出一些例子。在“命令语法”和“查询语法”中，本文给出了该命令的长格式和短格式。 SCPI 命令头部可分为表示命令、查询或命令和查询。可根据 SCPI 命令头部后面跟着问号（？）来识别查询操作，例如获取信息。

2.2 描述

在描述中给出了执行功能的简要说明。接下来是命令的正式语法的表示，其中命令头部使用大小写字符和从大写字符派生出来的缩写形式。在适用的情况下，查询的语法以其响应的格式给出。

2.3 用法

本手册列出的命令和查询可以用在 SDG8000A 系列的函数/任意波形发生器上。

2.4 命令符号

下面是用在命令中的符号：

<> 角括号包含用于占位的字符，其中分为两种类型：标题路径和命令参数。

:= 冒号后面跟着等号，它用于分隔占位符和占位符的描述，占位符的描述是指用于替换命令中占位符的参数类型和范围。

{} 花括号列出了可供选择的参数，至少要选择一个参数

[] 方括号包含可选项。

… 省略号表示其左侧和右侧可以重复多次

3 命令与查询

3.1 IEEE488.2 通用命令介绍

IEEE 标准定义的通用命令适用于查询设备的基本信息和执行基本操作。这些命令通常以“*”开头以及命令的关键字长度为 3 个字符。

3.1.1 *IDN

描述	*IND?是用于查询设备的 id。响应包含厂商、机型、序列号、软件版本和硬件版本。
查询语法	*IDN?
响应格式	格式 1：*IDN,<设备 id>,<机型>,<序列号>,<软件版本>,<硬件版本> 格式 2：<设备商>,<机型>,<序列号>,<软件版本>,<硬件版本> <设备 id>:= “SDG” 被用于识别仪器。 <设备商>:= “Siglent Technologies”。 <机型>:= 机器型号。 <序列号>:= 每个产品有自己的编号，此序列号标注产品的唯一性。 <软件版本>:= 与软件版本信息相关的编号。 <硬件版本>:= 固件级字段应该包含所有可单独修改子系统的信息。这些信息包含在单个或多个修订版本代码中。
示例	读取版本信息。 **IDN?* 返回值： *Siglent\sTechnologies,SDG8000A,SDG8000ATEST01,1.1.1.7R6_0526\n* (每个版本可能不同)

3.1.2 *OPC

描述	*OPC（操作完成）命令设置在标准事件状态寄存器 [ESR] 的 OPC 位（位 0）。此命令对设备操作无其他影响。因为仪器是在处理完上一条设置或查询语句之后，才开始解析设置或查询命令。 *OPC?查询命令总是返回 ASCII 码的 1。因为在前一个命令已经完全执行完后设备才响应该查询命令。
命令语法	*OPC
查询语法	*OPC?
响应格式	格式 1：*OPC 1 格式 2：1

3.1.3 *RST

描述	*RST 命令启动仪器重置并调用默认设置。
命令语法	*RST
示例	该例子将信号发生器重置成默认设置： **RST*

3.2 输出设置

3.2.1 噪声叠加命令

描述	开启噪声叠加，并以指定信噪比向通道添加噪声。
命令语法	<通道>:NOISE_ADD <参数>,<值> <通道>:= { C1, C2, C3, C4}。 <参数>:= 见下表参数。 <值>:= 见下表参数的值。

参数	值	描述
STATE	<state>	:= {ON, OFF}。噪声叠加开关。
RATIO_DB	<value>	:= 信噪比（dB），值详见数据手册
RATIO	<value>	:= 信噪比，值详见数据手册

查询语法	<通道>:NOISE_ADD?
响应格式	<通道>:NOISE_ADD STATE,ON|OFF,RATIO,<值>,RATIO_DB,<值(dB)>
示例	打开通道一噪声叠加并设置信噪比为 120： *C1:NOISE_ADD STATE,ON,RATIO,120*

3.2.2 通道极性、开关、负载设置命令

描述	启用和禁用通道对应前面板[OUTPUT]接口的输出。 查询结果返回“ON”、“OFF”、LOAD 和 PLRT 参数。
命令语法	<通道>:OUTPut ON|OFF,LOAD,<负载>,PLRT,<极性> <通道>:= {C1, C2, C3, C4}。 <负载>:= {50, HZ}. 默认单位为 ohm。 <极性翻转>:= {NOR, INVT}，其中 NOR 代表正常（关闭），INVT 代表反相（打开）。
查询语法	<通道>:OUTPut?
响应格式	<通道>:OUTP ON|OFF,LOAD,<负载>,PLRT,<极性>
示例	打开 CH1： *C1:OUTP ON* 读取 CH1 输出状态： *C1:OUTP?* 返回值：

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

设置 CH1 的负载为 50ohm：
C1:OUTP LOAD,50

设置 CH1 的负载为高阻：
C1:OUTP LOAD,HZ

设置 CH1 的极性为正常(关闭)：
C1:OUTP PLRT,NOR

设置 CH1 的负载为自定义阻值：
C1:OUTP LOAD,80

3.2.3 幅度限制命令

描述	设置通道最大幅度限制。
命令语法	<通道>:BaSic_WaVe MAX_OUTPUT_AMP,<值> <通道>:= {C1, C2, C3, C4}。 <值>:= {可在数据手册查阅参数值的有效范围}。
查询语法	<通道>:BaSic_WaVe?
示例	设置通道 1 最大输出幅度为 1V： *C1:BSWV MAX_OUTPUT_AMP,1*

3.2.4 数字滤波器命令

描述	此命令用于设置数字滤波器的开关和截止频率。
命令语法	<通道>:FILTer<参数>,<值> <通道>:= {C1, C2, C3, C4}。 <参数>:= {下表的参数}。 <值>:= {相关参数的值}。

参数	值	描述
STATE	<state>	:= {ON, OFF}，数字滤波器状态。
COFF_FRQ	<cutoff_freq>	:= {1 uHz 至 2 GHz}，单位为赫兹，用于设置数字滤波器的截止频率。

查询语法	<通道>:FILTer? <通道>:= {C1, C2, C3, C4}。
响应格式	<通道>:FILT <参数>,<值>
示例	设置 CH1 的数字滤波器开启： *C1:FILT STATE,ON* 设置 CH1 的数字滤波器截止频率 200 MHz： *C1:FILT COFF_FRQ,200000000* 查询 CH1 的数字滤波器信息： *C1:FILT?* 返回值： *STATE,OFF,COFF_FRQ,200000000HZ*

3.2.5 模拟滤波器命令

描述	此命令用于设置模拟滤波器的切换。
命令语法	<通道>:FILT5G <参数> <通道>:= {C1, C2, C3, C4}。 <参数>:= {ON, OFF}.ON:使用 5G 模拟滤波器；OFF:使用 2G 模拟滤波器。
查询语法	<通道>:FILT5G? <通道>:= {C1, C2, C3, C4}。
响应格式	<参数>
示例	设置 CH1 的 5G 模拟滤波器开启： *C1:FILT5G ON* 查询 CH1 的 5G 模拟滤波器状态： *C1:FILT5G?* 返回值： *ON\n*

3.2.6 过压保护命令

描述	此命令用于设置或者获取过压保护状态。
命令语法	<通道>:VOLTPRT<状态> <通道>:= {C1, C2, C3, C4}。 <状态>:= {ON, OFF}。
查询语法	<通道>:VOLTPRT? <通道>:= { C1, C2, C3, C4}。
示例	打开 CH1 的过压保护开关： *C1:VOLTPRT ON* 获取 CH1 的过压状态： *C1:VOLTPRT?* 返回值： *ON\n*

3.2.7 OUTPUT 时滞命令

描述	此命令用于设置或查询 output 时滞。
命令语法	<通道>:OUTPut:SKEW <值> <通道>:= {C1, C2, C3, C4}。 <值>:= {-500 至 500}，单位“ns”。
查询语法	<通道>:OUTPut:SKEW? <通道>:= {C1, C2, C3, C4}。
响应格式	<值> <值>:= {当前通道设置的值}。
示例	设置 C1 通道 Output 时滞为 0.2 ns： *C1:OUTPut:SKEW 0.2e-9* 读取 C1 通道 Output 时滞： *C1:OUTPut:SKEW?* 返回值： *2e-10*

3.2.8 输出模式命令

描述	设置或者获取基本波参数。
命令语法	<通道>:BaSic_WaVe PATHMODE,<值> <通道>:= {C1, C2, C3, C4}。 <值>:= {DC_DIRECT（直流高带宽），DC_AMPLIFY（直流放大），AC_DIRECT（交流），AC_AMPLIFY（交流放大）}。
查询语法	<通道>:BaSic_WaVe?
响应格式	<通道>:BSWV<参数> <参数>:= {当前波形的所有参数}。
示例	设置 C1 输出模式为交流： *C1:BSWV PATHMODE,AC_DIRECT* 从设备读取 C1 的参数： *C1:BSWV?* 返回值： *C1:BSWV\sWVTP,SINE,FRQ,1000HZ,PERI,0.001S,AMP,0.7112V,AMPVRMS,0.251447Vrms,MAX_OUTPUT_AMP,1.5V,PATHMODE,AC_DIRECT,OFST,0V,DLY,-0S,DIFF_OFST,0V,HLEV,0.3556V,LLEV,-0.3556V,PHSE,0\n*

3.2.9 标记设置命令

描述	该命令用于设置或者获取 Marker 参数、状态。仅在 AWG 或 IQ 时，该命令有效。
命令语法	<通道>:MARKer<参数 1> <参数>,<值> <通道>:= {C1, C2, C3, C4}。 <参数 1>:= {1, 2}，1 表示 MarKer1，2 表示 MarKer2。 <值>:= {相关参数的值}。

参数	值	描述
STATE	<STATE>	:= {ON, OFF}打开或关闭 Marker。
VOLTE	<VALUE>	:= {0.2, 0.4, 0.6, 0.8, 1, 1.2, 1.4, 1.6, 1.8, 2}。单位是伏特，峰峰值“Vpp”,。
OFFSET	<VALUE>	:= {0 到 2 的浮点数}。单位是伏特，峰峰值“Vpp”。
SKEW	<VALUE>	:= {-500ns~500ns}。单位是 s。

查询语法	<通道>: MARKer<参数 1>? <通道>:= { C1, C2, C3, C4}。 <参数 1>:= {1, 2}。1 表示 MarKer1，2 表示 MarKer2。
示例	打开 CH1 的 MarkerA: *C1:MARKer1 STATE,ON* 设置 CH1 MarkerA 的幅度为 0.6 V: *C1:MARKer1 VOLTE,0.6* 获取 CH1 的 MarkerA 参数: *C1:MARKer1?* 返回值: *CH1:MARKERA,STATE,ON,MARKERA,HLEVEL,0.300000,LLEVEL,-0.300000,SKEW,0S\n*

3.3 多通道设置

3.3.1 通道模式命令

描述	设置或查询通道之间的关系，通道两两一组或四个通道为一组。
命令语法	GMOD <参数> <参数>:= {PAIR(两两一组), ALL(四通道为一组)}。
查询语法	GMOD?
响应格式	MODE<参数>
示例	设置通道之间两两为一组： *GMOD PAIR* 设置通道之间四个通道为一组： *GMOD ALL*

备注：仅四通道机型可用。

3.3.2 相位模式命令

描述	该参数设置或者获取相位模式。
命令语法	<分组>:MODE <参数> <分组>:= {G1, G2, G3}。 <参数>:= {PHASELOCKED（相位锁定）, INDEPENDENT（相位独立）}。
查询语法	<分组>:MODE?
响应格式	MODE<参数>
示例	设置 CH1-CH2 相位模式为相位独立模式： *G1:MODE INDEPENDENT*

备注：

G1：通道模式选择 PAIR 时，设置 CH1/CH2 两通道

G2：通道模式选择 PAIR 时，设置 CH3/CH4 两通道（四通道机型）

G3：通道模式选择 ALL 时，设置所有通道

3.3.3 通道合并命令

描述	此命令设置或者获取通道合并状态。
命令语法	<通道>:CoMBiNe <状态> <通道>:= {C1, C2, C3, C4}。 <状态>:= {ON, OFF}。
查询语法	<通道>:CoMBiNe? <通道>:= {C1, C2, C3, C4}。
响应格式	<通道>:CMBN <状态>
示例	打开 CH1 的波形合并： *C1:CoMBiNe ON* 查询 CH2 的波形合并状态： *C2:CMBN?* 返回值： *C2:CMBN OFF*

3.3.4 参数复制命令

描述	该命令将一个通道的参数复制到另一通道上。
命令语法	ParaCoPy <目标通道>,<通道源> <目标通道>:= {C1, C2, C3, C4}。 <通道源>:= {C1, C2, C3, C4}。
示例	通道 1 的参数复制到通道 2 上： *PACP C2,C1*

3.3.5 同相位命令

描述	此命令用于设置两个通道的相位同步。
命令语法	<分组>:EQPHASE <分组>:= {G1, G2, G3}。
示例	PAIR 模式，CH1-CH2 分组设置同相位： *G1:EQPHASE*

备注：

G1：通道模式选择 PAIR 时，设置 CH1/CH2 两通道

G2：通道模式选择 PAIR 时，设置 CH3/CH4 两通道（四通道机型）

G3：通道模式选择 ALL 时，设置所有通道

3.3.6 通道跟踪、耦合命令

描述

设置或者获取通道耦合参数。只有在追踪功能关闭时才能设置耦合值。

命令语法

<组别>:COUPling <参数>,<值>
<组别>:= {G1, G2, G3}。
<参数>:= {下表的参数}。
<值>:= {相关参数的值}。

参数	值	描述
TRACE	<track_enble>	:= {ON, OFF}；通道跟踪状态
TPHASE	<track_pha_dev>	:=跟踪相位差。单位是‘度’
STATE	<state>	:= {ON, OFF}；通道耦合状态
BSCH	<bsch>	:= {CH1, CH2, CH3, CH4}；基本通道
FCOUP	<fcoup>	:= {ON, OFF}；频率耦合状态
FDEV	<frq_dev>	:= 两通道间的频率差值。单位是赫兹‘Hz’
FRAT	<frat>	:= 两通道间的频率比值
PCOUP	<pcoup>	:= {ON, OFF}；相位耦合状态
PDEV	<pha_dev>	:=两通道间的相位差值。单位是‘度’
PRAT	<prat>	:=两通道间的相位比值
ACOUP	<acoup>	:= {ON, OFF}；幅度耦合的状态
ARAT	<arat>	:= 两通道间的幅度比值
ADEV	<adev>	:= 两通道间的幅度偏差。单位为伏特，峰峰值‘Vpp’

查询语法

<组别>:COUPling?

响应格式

COUP <参数>
<参数>:= {所有耦合参数值}。

示例

设置 CH1-CH2 分组耦合状态为打开：
G1:COUP STATE,ON

设置频率偏差为 5Hz：
G1:COUP FDEV,5

设置幅度比例为 2：
G1:COUP ARAT,2

查询 CH1-CH2 耦合信息：
G1:COUP?
返回值：
COUPTRACE,OFF,FCOUP,ON,PCOUP,ON,ACOUP,ON,FDEV,5HZ,PRAT,1,ARAT,2

备注：

G1：通道模式选择 PAIR 时，设置 CH1/CH2 两通道

G2：通道模式选择 PAIR 时，设置 CH3/CH4 两通道（四通道机型）

G3：通道模式选择 ALL 时，设置所有通道

3.4 通道模式选择命令

描述	该命令用于设置或者读取某通道的当前模式，选择通道的模式为 AFG/AWG/IQ。
命令语法	格式:<通道>:CPARam MODE,<参数> <通道>:= {C1, C2, C3, C4}。 <参数>:= {AFG, AWG, IQ}。
查询语法	<通道>:CPARam? <通道>:= {C1, C2, C3, C4}。
响应格式	MODE,<参数> <参数>:= {AFG, AWG, IQ}。
示例	设置通道 1 的当前输出模式为 AFG: *C1:CPARam MODE,AFG* 查询当前通道的输出模式: *C1:CPARam?* 返回值: *MODE,AFG*

3.5 AFG 命令

3.5.1 基本波形命令

描述	设置或者获取基本波参数。
命令语法	<通道>:BaSic_WaVe <参数>,<值> <通道>:= {C1, C2}。 <参数>:= MAX_OUTPUT_AMP。 <值>:= {相关参数的值}。

参数	值	描述
WVTP	<type>	:= {SINE, SQUARE, RAMP, PULSE, NOISE, ARB, DC, PRBS}。如果命令未设置基本波形类型，WVPT 默认设置为当前波形。
FRQ	<frequency>	:= 频率。单位是赫兹“Hz”，可在数据手册查阅参数值的有效范围。当 WVTP 为噪声和直流，该值无效。
PERI	<period>	:= 周期。单位是秒“s”。可在数据手册查阅参数值的有效范围。当 WVTP 为噪声和直流，该值无效。
AMP	<amplitude>	:= 幅值。单位是伏特，峰峰值"Vpp"。可在数据手册查阅参数值的有效范围。当 WVTP 为噪声和直流，该参数无效。
OFST	<offset>	:= 偏置。单位是伏特“V”。可在数据手册查阅参数值的有效范围。当 WVTP 为噪声，该值无效。
DIFF_OFST	<common offset>	:= {详见数据手册}。差模偏置电压。单位是伏特“V”。仅输出模式为 DC 放大下支持设置。
SYM	<symmetry>	:= {0 至 100}。三角波的对称度。单位是“%”。仅当 WVTP 为三角波才能设置该参数。
DUTY	<duty>	:={0 至 100}。占空比。单位是“%”。该参数的值取决于频率。仅当 WVTP 是方波和脉冲才能设置该参数。
PHSE	<phase>	:= {0 至 360}。单位是“°”，当 WVTP 是

		噪声、脉冲波、直流时，该参数无效。
STDEV	<stdev>	:= 噪声的标准差。单位是伏特“V”。可在数据手册查阅参数值的有效范围。仅当 WVTP 是噪声时，才能设置。
MEAN	<mean>	:= 噪声的均值。单位是伏特“V”。可在数据手册查阅参数值的有效范围。仅当 WVTP 是噪声时，才能设置该参数。
WIDTH	<width>	:= 正脉宽。单位是秒“s”。可在数据手册查阅参数值的有效范围。仅当 WVTP 是脉冲波时，才能设置该参数。
RISE	<rise>	:= 上升时间（10%~90%）。单位是秒“s”。可在数据手册查阅参数值的有效范围。仅当 WVTP 是脉冲波时，才能设置该参数。
FALL	<fall>	:= 下降时间（10%~90%）。单位是秒“s”。可在数据手册查阅参数值的有效范围。仅当 WVTP 是脉冲波时，才能设置该参数。
DLY	<delay>	:= 波形延迟。则可在数据手册查阅参数值的有效范围
HLEV	<high level>	:= 高电平。单位是伏特“V”。当 WVTP 为噪声，该值无效。
LLEV	<low level>	:= 低电平。单位是伏特“V”。当 WVTP 为噪声，该值无效。
BANDSTATE	<bandwidth switch>	:= {ON（打开），OFF（关闭）}。当 WVTP 为噪声，该参数才能设置
BANDWIDTH	<bandwidth value>	:= 噪声带宽。单位为赫兹“Hz”。可在数据手册查阅参数值的有效范围。仅当 WVTP 是噪声时，才能设置该参数。
LENGTH	<prbs length>	:= {3~32}。PRBS 实际长度= $2^{长度}-1$。仅当 WVTP 是 PRBS 时，才能设置该参数。
EDGE	<prbs rise/fall>	:= PRBS 的上升/下降时间。单位是秒

		“s”。可在数据手册查阅参数值的有效范围。仅当 WVTP 是 PRBS 时，才能设置该参数。
PATHMODE	<output pathmode>	:= {DC_DIRECT, DC_AMPLIFY, AC_DIRECT, AC_AMPLIFY} 通道的输出链路选择
BITRATE	<prbs bit rate>	:= PRBS 比特率。单位是位每秒“bps”。可在数据手册查阅参数值的有效范围。仅当 WVTP 是 PRBS 时，才能设置该参数。
AMPVRMS	<amplitude>	:= 幅度。设置幅度单位为 Vrm。
AMPDBM	<amplitude>	:= 幅度。设置幅度单位为 dBm。
MAX_OUTPUT_AMP	<amplitude>	:= 通道最大幅度限制，可在数据手册查阅参数值的有效范围。

查询语法

<通道>:BaSic_WaVe?
<通道>:= {C1, C2}。

响应格式

<通道>:BSWV<参数>
<参数>:= {当前波形的所有参数}。

示例

改变 C1 波形为三角波：
C1:BSWV WVTP,RAMP

改变 C1 频率为 2000 Hz：
C1:BSWV FRQ,2000

设置 C1 的幅度为 1 Vpp：
C1:BSWV AMP,1

从设备读取 C1 的参数：
C1:BSWV?
返回值：
C1:BSWV WVTP,RAMP,FRQ,2000HZ,PERI,0.01S,AMP,1V,OFST,0V,HLEV,0.5V,LLEV,-0.5V,PHSE,0

设置 C1 的噪声带宽为 100MHz：
C1:BSWV BANDWIDTH,100E6
或
C1:BSWV BANDWIDTH,100000000

3.5.2 PRBS 多项式命令

描述	该命令用于设置或者获取 PRBS 自定义多项式参数。仅当基本波形为 PRBS 时，该命令有效。
命令语法	<通道>:PRBS:<参数> <值> <通道>:= {C1, C2}。 <值>:= {相关参数的值}。 参数 | 值 | 描述 FSTAte | <STATE> | := {ON, OFF}，打开或关闭 PRBS 自定义多项式开关。 FORMula | <value> | := 设置 PRBS 多项式，需要加双引号。 SEED | <value> | := {1, 2, ..., M}，设置 PRBS 种子数。
查询语法	<通道>:PRBS:<参数>? <通道>:= { C1, C2, C3, C4}。
示例	启用 CH1 的 PRBS 自定义多项式功能： *C1:PRBS:FSTAte ON* 设置 CH1 的 PRBS 多项式为 X7+X5+1： *C1:PRBS:FORMula "X7+X5"* 获取 CH1 的 PRBS 自定义多项式： *C1:PRBS:FORMula?* 返回值： *X7+X5+1*

3.5.3 AFG 任意波切换波形命令

描述	该命令用于设置或者读取任意波。
命令语法	格式 1：<通道>:ArbWaVe INDEX,<索引号> 格式 2：<通道>:ArbWaVe NAME,<名称> 格式 3：<通道>:ArbWaVe NAME,<路径> <通道>:= { C1, C2, C3, C4}。 <索引号>:= 下表中任意波的索引号。 <名称>:= 下表的任意波的名称。 <路径>:= 波形路径。
查询语法	<通道>:ARbWaVe?

	<通道>:= {C1, C2, C3, C4}。
响应格式	<通道>:ARWV INDEX,<索引号>,NAME,<名称>
示例	通过索引号 2 设置 CH1 的当前波形： *C1:ARWV INDEX,2* 读取 CH1 的当前波形： *C1:ARWV?* 返回值： *C1:ARWV INDEX,2,NAME,StairUp* 通过波形名称设置 CH1 的波形为心波： *C1:ARWV NAME,"Cardiac"* 通过波形路径设置 CH1 的波形： *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"*

相关命令

索引号	名称	索引号	名称	索引号	名称	索引号	名称
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	Erfclnv	163	Duty38
11	ExpRise	62	Chebyshev1	113	Erflnv	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	NA	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
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-Hann	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		

3.5.4 任意波小点数波形命令

描述

此命令用于设置或者获取任意波形数据。

命令语法

<通道>:WVDT <参数>,<值>
<通道>:= {C1, C2, C3, C4}。
<参数>:= {下表的参数}。
<值>:= {相关参数的值}。

参数	值	描述
WVNM	<波形名>	:= 波形名称。
FRQ	<频率>	:= 频率。单位为赫兹“Hz”。
AMP	<幅值>	:= 幅值。单位是伏特，峰峰值“Vpp”。
OFST	<偏置>	:= 偏置。单位是伏特“V”。
PHASE	<相位>	:= 相位。单位是“度”。
WAVEDATA	<波形数据>	:= 波形数据。写入到波形中的数据。数据为二进制，16bit 为一个波形点。每个波形点的码字范围为-32767~32767。

查询语法

格式 1：WVDT? Mn
格式 2：WVDT? USER,<波形名称>
格式 3：WVDT? USER,<路径>,<波形名称>
<路径>:= 指定存储路径。
<波形名称>:= {用户定义的波形名称}。

示例

将数据 0x6000c0006000 写到‘wave1’波形中并发送到通道 1：

C1:WVDT WVNM,wave1,FRQ,2500,AMP,2.5,OFST,0.2,
WAVEDATA, b'0x6000c0006000'

注：该指令推荐使用 5.1.5 章节的方式进行编程。

3.5.5 任意波大点数波形命令

描述	此命令可分段发送大数据的波形到指定路径下指定的 bin 文件。
命令语法	<通道>:WVDT:SEGMENT <参数>,<data> <通道>:= {C1, C2, C3, C4}。 <参数>:= {下表的参数}。 {data}:= 波形数据。

参数	值	描述
WVNM	<波形名>	:= 波形名称。
FRQ	<频率>	:= 频率。单位为赫兹“Hz”。只在写起始数据时生效。
AMP	<幅值>	:= 幅值。单位是伏特，峰峰值“Vpp”。只在写起始数据时生效。
OFST	<偏置>	:= 偏置。单位是伏特“V”。只在写起始数据时生效。
PHASE	<相位>	:= 相位。单位是“度”。只在写起始数据时生效。
BEGIN,WAVEDATA	<波形数据>	:= 波形起始数据。数据为二进制，16bit 为一个波形点。每个波形点的码字范围为-32767~32767。
WAVEDATA	<波形数据>	:= 波形中间数据。数据为二进制，16bit 为一个波形点。每个波形点的码字范围为-32767~32767。中间数据至少需要写一段。
END,WAVEDATA	<波形数据>	:= 波形结束数据。数据为二进制，16bit 为一个波形点。每个波形点的码字范围为-32767~32767。

示例	分段发送一组大数据的波形到本地下的 wave1.bin： *C1:WVDT:SEGMENT WVNM,"wave1",FRQ,2000,AMP,2,OFST,0.2,BEGIN,WAVEDATA,b'0x6000c0006'* *C1:WVDT:SEGMENT WVNM,"wave1",WAVEDATA,b'0x0006000'* *C1:WVDT:SEGMENT WVNM,"wave1",END,WAVEDATA,b'0x6000'*

注：该命令适用于波形数据较大的情况，波形数据较小时，推荐用 3.15.9 章节的指令生成。该指令推荐使用 5.1.6 章节的方式进行编程。

3.5.6 非线性补偿命令

描述	该命令用于设置或查询非线性补偿开关状态。仅当基本波形为正弦波时，该命令有效。
命令语法	<通道>:NONLINECOMP STATE,<值> <通道>:= {C1, C2, C3, C4}。 <值>:= {ON, OFF}。
查询语法	<通道>:NONLINECOMP? <通道>:= { C1, C2, C3, C4}。
示例	启用 CH1 的非线性补偿功能： *C1:NONLINECOMP STATE,ON* 获取 CH1 的非线性补偿开关状态： *C1:NONLINECOMP?* 返回值： *C1:NonLineComp STATE,ON\n*

3.5.7 谐波命令

描述	该命令用于设置或者获取谐波参数。仅当基本波形为正弦波时，该命令有效。
命令语法	<通道>:HARMonic <参数>,<值> <通道>:= {C1, C2, C3, C4}。 <值>:= {相关参数的值}。

参数	值	描述
HARMSTATE	<STATE>	:= {ON, OFF}打开或关闭谐波。
HARMTYPE	<TYPE>	:= { EVEN（偶级数）, ODD（奇级数）, ALL（所有级数）}谐波类型。
HARMORDER	<NUM>	:= {1, 2,... , M}，其中 M 是支持的最大级数。
HARMPHASE	<PHASE>	:= {0~360}。单位是°。
HARMAMP	<VALUE>	:= 指定谐波的幅值。值的范围取决于机型。单位是伏特，峰峰值“Vpp”。
HARMDBC	<VALUE>	:= 指定谐波的幅值。值的范围取决于机型。单位是"dBc"。

查询语法	<通道>:HARMonic? <通道>:={ C1, C2, C3, C4}。

示例	启用 CH1 的谐波功能： *C1:HARM HARMSTATE,ON* 设置 CH1 的谐波级数为 2，幅度为-6dBc： *C1:HARM HARMORDER,2,HARMDBC,-6* 获取 CH1 的谐波参数： *C1:HARM?* 返回值： *C1:HARM ,HARMSTATE,ON,HARMTYPE,EVEN,HARMORDER,2,HARMAMP,2.004748935V,HARMDBC,-6dBc,HARMPHASE,0*

3.6 调制波形命令

描述	该命令可以设置或者获取调制波形参数。
命令语法	<通道>:MoDulateWaVe <类型> <通道>:MoDulateWaVe<参数>，<值> <通道>:= { C1, C2, C3, C4}。 <类型>:= {AM, DSBSC, FM, PM, PWM, ASK, FSK, PSK}。 <参数>:= {下表的值}。 <值>:= {相关参数的值}。

参数	值	描述
STATE	<state>	:= {ON（打开）, OFF（关闭）}。启用和禁用调制。 如果你想设置或者读取调制的其他参数，必须先将 STATE 设置为“ON”。
AM,SRC	<src>	:= {INT（内调制）, EXT（外调制），CH（通道调制）}。AM 调制源。
AM,MDSP	<mod wave shape>	:= {SINE, SQUARE, TRIANGLE, UPRAMP, DNRAMP, NOISE, ARB}。AM 调制波形。仅当调制源设置为内调制时，该参数才能设置。
AM,FRQ	<AM frequency>	:= AM 频率。单位是赫兹“Hz”，可在数据手册查阅参数值的有效范围。调制源设置为内调制时，该参数才能设置。
AM,DEPTH	<depth>	:= {0 至 120}。AM 深度。单位是“%”。触发源设置为内触发时，该参数才能设置。
DSBSC,SRC	<src>	:= {INT（内调制），EXT（外调制），CH（通道调制）}。DSBSC 调制源。
DSBSC,MDSP	<mod wave shape>	:= {SINE, SQUARE, TRIANGLE, UPRAMP, DNRAMP, NOISE, ARB}。DSB-SC 调制波形。调制源设置为内调制时，该参数才能设置。
DSBSC,FRQ	<DSB-AM frequency>	:= DSB SC 频率。单位是赫兹“Hz”，可在数据手册查阅参数值的有效范围。设置为内调制时，该参数才能设置。
FM,SRC	<src>	:= {INT（内调制），EXT（外调制），CH（通道调制）}。FM 调制源

FM,MDSP	<mod wave shape>	:= {SINE, SQUARE, TRIANGLE, UPRAMP, DNRAMP, NOISE，ARB}。FM 调制波形。调制源设置为内调制时，该参数才能设置。
FM,FRQ	<FM frequency>	:= FM 频率。单位是赫兹“Hz”，可在数据手册查阅参数值的有效范围。调制源设置为内调制时，该参数才能设置。
FM,DEVI	<FM frequency deviation>	:= {0 至当前载波频率}。FM 频率偏差。该值取决于载波频率和带宽频率的差值。调制源设置为内调制时，该参数才能设置。
PM,SRC	<src>	:= {INT（内调制），EXT（外调制），CH（通道调制）}。PM 调制源。
PM,MDSP	<mod wave shape>	:= {SINE, SQUARE, TRIANGLE, UPRAMP, DNRAMP, NOISE, ARB}。PM 调制波形。调制源设置为内调制时，该参数才能设置。
PM,FRQ	<PM frequency>	:= PM 频率。单位是赫兹“Hz”，可在数据手册查阅参数值的有效范围。调制源设置为内调制时，该参数才能设置。
PM,DEVI	<PM phase offset>	:= {0 至 360}。PM 相位偏差。单位是“°”， 调制源设置为内调制时，该参数才能设置。
PWM,SRC	<src>	:= {INT（内调制），EXT（外调制），CH（通道调制）}。PWM 调制源。
PWM,FRQ	<PWM frequency>	:= PWM 频率。单位是赫兹“Hz”，可在数据手册查阅参数值的有效范围。调制源设置为内调制时，该参数才能设置。
PWM,DEVI	<PWM dev>	:=占空比偏差。单位是“s”。值取决于载波的脉宽。
PWM,MDSP	<mod wave shape>	:= {SINE, SQUARE, TRIANGLE, UPRAMP, DNRAMP, NOISE, ARB}。PWM 调制波形。调制源设置为内调制时，该参数才能设置。
ASK,SRC	<src>	:= {INT（内调制）, EXT_A, EXT_B（外调制）}。ASK 调制源。
ASK,KFRQ	< key frequency>	:= ASK 键控频率。单位是赫兹“Hz”，可在数据手册查阅参数值的有效范

		围。调制源设置为内调制时，该参数才能设置。
FSK,SRC	<src>	:={INT（内调制）, EXT_A,EXT_B（外调制）}。FSK 调制源。
FSK,KFRQ	< key frequency>	:= FSK 键控频率。单位是赫兹“Hz”，可在数据手册查阅参数值的有效范围。调制源设置为内调制时，该参数才能设置。
FSK,HFRQ	<FSK_hop_freq>	:= FSK 跳频频率。与基础波形频率相同。单位是赫兹“Hz”，可在数据手册查阅参数值的有效范围。
PSK,SRC	<src>	:= {INT, EXT_A, EXT_B}。PSK 调制源。
PSK,KFRQ	< key frequency>	:= PSK 键控频率。单位是赫兹“Hz”，可在数据手册查阅参数值的有效范围。调制源设置为内调制时，该参数才能设置。
CARR,WVTP	<wave type>	:= {SINE, SQUARE, RAMP, ARB, PULSE}。载波频率类型。
CARR,FRQ	<frequency>	:= 载波频率。单位是赫兹“Hz”，可在数据手册查阅参数值的有效范围。
CARR,PHSE	<phase>	:= {0 至 360}。载波相位。单位为“度”。
CARR,AMP	<amplitude>	:= 载波幅值。单位是伏特，峰峰值"Vpp"。可在数据手册查阅参数值的有效范围。
CARR,OFST	<offset>	:=载波偏置。单位是伏特。可在数据手册查阅参数值的有效范围。
CARR,SYM	<symmetry>	:= {0 至 100}。当载波为 RAMP 时，载波对称度。单位是“%”。
CARR,DUTY	<duty>	:= {0 至 100}。当载波为方波或者脉冲波时，载波占空比。单位是“%”。
CARR,RISE	<rise>	:=当载波为方波或者脉冲波时，载波上升时间。单位是秒“s”。可在数据手册查阅参数值的有效范围。
CARR,FALL	<fall>	:=当载波为方波或者脉冲波时，载波下降时间。单位是秒“s”。可在数据手册查阅参数值的有效范围。
CARR,DLY	<delay>	:= 当载波为方波或者脉冲波时，载波延时。单位是秒“s”。可在数据手册查阅参数值的有效范围。

查询语法	<通道>:MoDulateWaVe? <通道>:= {C1, C2, C3, C4}。
响应格式	<通道>:MDWV<参数> <参数>:= {当前调制的所有参数}。
示例	设置 CH1 调制状态为打开： *C1:MDWV STATE,ON* 设置 CH1 调制类型为 AM： *C1:MDWV AM* 设置 AM 调制，且调制波形设为正弦波： *C1:MDWV AM,MDSP,SINE* 读取状态值为 ON 的 C1 的调制参数： *C1:MDWV?* 返回值： *C1:MDWVAM,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* 读取状态为 OFF 的 CH1 的调制参数： *C1:MDWV?* 返回值： *C1:MDWV STATE,OFF* 设置 CH1 的 FM 调制频率为 1000Hz： *C1:MDWV FM,FRQ,1000* 设置 CH1 的载波为正弦波： *C1:MDWV CARR,WVTP,SINE* 设置 CH1 载波频率为 10000Hz： *C1:MDWV CARR,FRQ,10000*

3.7 扫描波形命令

描述	该命令用于设置或者获取扫描波形的参数。
命令语法	<通道>:SWEep <state> <通道>:= {C1, C2, C3, C4}。 <state>:= {ON,OFF}。 <通道>:SWEep:<参数> <值> <通道>:= {C1, C2, C3, C4}。 <值>:= {相关参数的值}。

参数	值	描述
TYPE	<type>	:= {FREQ, AMP, BOTH}。扫描的类型。
SOURce	<src>	:= {INT, EXT_A, EXT_B, MAN_A, MAN_B}。扫描的触发源。
FMODe	<mode>	:= {LINE, LOG, STEP}。频率扫描的类型。
AMODe	<mode>	:= {LINE, STEP}。幅度扫描的类型。
FSNumber	<value>	:= {2 到 1024 之间的整数}。步进频率扫描的步进数。
ASNumber	<value>	:= {2 到 1024 之间的整数}。步进幅度扫描的步进数。
TIME	<value>	:= 浮点类型的值，单位秒。扫描的扫描时间。
SHTime	<value>	:= 浮点类型的值，单位秒。扫描的起始保持时间。
EHTime	<value>	:= 浮点类型的值，单位秒。扫描的结束保持时间。
RTIMe	<value>	:= 浮点类型的值，单位秒。扫描结束后的返回时间。
SFRequency	<value>	:= 浮点类型的值，单位赫兹（Hz）。频率扫描的起始频率
EFRequency	<value>	:= 浮点类型的值，单位赫兹（Hz）。频率扫描的终止频率
CFRequency	<value>	:= 浮点类型的值，单位赫兹（Hz）。频率扫描的中心频率
FSPan	<value>	:= 浮点类型的值，单位赫兹（Hz）。频率扫描的频率范围
SAMPlitude	<value>	:= 浮点类型的值，单位伏特（Vpp）。幅度扫描的起始幅度。
EAMPlitude	<value>	:= 浮点类型的值，单位伏特（Vpp）。幅度

		扫描的终止幅度。
CAMPlitude	<value>	:= 浮点类型的值，单位伏特（Vpp）。幅度扫描的中心幅度。
ASPan	<value>	:= 浮点类型的值，单位伏特（Vpp）。幅度扫描的幅度范围。
FDIRection	<direction>	:= {UP, DOWN, UP_DOWN}。频率扫描的扫描方向。UP_DOWN 仅在线性扫描（LINE）下支持。
ADIRection	<direction>	:= {UP, DOWN, UP_DOWN}。幅度扫描的扫描方向。UP_DOWN 仅在线性扫描(LINE)下支持。
FSYMmetry	<value>	:= {0 至 100 的浮点数}。线性频率扫描。设置上下扫描额的对称性。
ASYMmetry	<value>	:= {0 至 100 的浮点数}。线性幅度扫描。设置上下扫描额的对称性。
TOUT	<state>	:= {ON, OFF}。扫描波形的触发输出开关。触发源为内部或手动下可设置。
EDGe	<polarity>	:= {RISE, FALL}。扫描波形触发源为外部时，设置源的沿。
MTRigger		触发源为手动触发时，手动触发一次扫描。
MARKer<num>:FMARker	<state>	num = {1,2}，分别表示标记 1 和标记 2。state = {ON, OFF}。频率扫描的标记开关状态。
MARKer<num>:MFRequency	<value>	num = {1,2}，分别表示标记 1 和标记 2。value = 浮点类型的值，单位赫兹（Hz）。频率扫描的标记频率。
MARKer<num>:MSNumber	<value>	num = {1,2}，分别表示标记 1 和标记 2。value= {1~1024}。步进频率扫描的标记步进数。

查询语法

<通道>:SWEep<参数>?
<通道>:= {C1, C2, C3, C4}。

示例

启用 CH1 的 sweep 功能：
C1:SWEep ON

设置 CH1 的扫描类型为频率扫描：
C1:SWEep:TYPE FREQ

获取 CH1sweep 的扫描类型：
C1:SWEep:TYPE?

返回值：
FREQ\n

3.8 脉冲串命令

描述	此命令用于设置或者读取脉冲串波形参数。
命令语法	<通道>:BursTWaVe <参数>,<值> <通道>:= {C1, C2, C3, C4}。 <参数>:= {下表的参数}。 <值>:= {相关参数的值}。

参数	值	描述
STATE	<state>	:= {ON（打开），OFF（关闭）}。启用和禁用脉冲串。 如果你想设置或者读取脉冲串的其他参数，必须先将 STATE 设置为“ON”。
PRD	<period>	:= 脉冲串周期。单位是秒“s”。可在数据手册查阅参数值的有效范围。在下列情况下，该值无效： TRSR 是外触发
STPS	<start_phase>	:= {0 至 360}。载波的起始相位。单位是“°”。当载波为噪声或者脉冲波时，该值无效。
GATE_NCYC	<burst_mode>	:={GATE（门控模式），NCYC（N 循环模式）}。脉冲串模式。当载波为噪声时，该值无效。
TRSR	<trig_src>	:= {EXT_A, EXT_B, INT, MAN_A, MAN_B}。触发源。EXT_A、EXT_B 代表可使用的两个外部触发源，。 INT 代表内部触发；MAN_A、MAN_B 代表可使用的两个手动触发。
MTRIG		:= 发送一个手动触发信号。仅当 TRSR 是 MAN 时，该参数有效。
DLAY	<delay>	:= 触发延时。单位是秒“s”。可在数据手册查阅参数值的有效范围。当 GATE_NCYC 为 N 循环时，该值有效。当在载波为噪声时该值无效。
PLRT	<polarity>	:= {NEG（负极性），POS（正极性）}。门控极性，负极性或者正极性。

TRMD	<trig_mode>	:= {RISE（向上），FALL（向下），OFF（关闭）}。触发输出模式。当 GATE_NCYC 为 N 循环且触发源为内触发或者手动触发时，该参数有效。当载波为噪声时该值无效。
EDGE	<edge>	:= {RISE, FALL}。有效触发边沿。当 TRST 为手动触发或者外触发时，该值有效。当载波为噪声时该值无效。
TIME	<circle_ time>	:={INF, 1, 2, ..., M}，M 值支持的最大 N 循环数取决于机型。INF 设置脉冲串为无限模式。当 GATE_NCYC 为 N 循环时有效。当载波为噪声时，该值无效。
COUNT	<counter>	:= 脉冲串数，当触发源为外部或手动时有效，可在数据手册查阅参数的有效值范围。
HOLD	<type>	:={MIDDLE, START, END}。空闲电平。
CARR, WVTP	<wave type>	:={SINE, SQUARE, RAMP, ARB, PULSE, NOISE}。载波波形类型。
CARR, FRQ	<frequency>	:= 载波频率。单位是赫兹“Hz”，可在数据手册查阅参数值的有效范围。
CARR, PHSE	<phase>	:={0 至 360}。载波相位。单位是“度”。
CARR, AMP	<amplitude>	:= 载波幅值。单位是伏特，峰峰值“Vpp”。可在数据手册查阅参数值的有效范围。
CARR, OFST	<offset>	:= 载波偏置。单位是幅度“V”。可在数据手册查阅参数值的有效范围。
CARR, SYM	<symmetry>	:= {0 至 100}。载波对称度，当载波为三角波时，该值有效。单位是“%”。
CARR, DUTY	<duty>	:= {0 至 100}。载波占空比，当载波为方波和脉冲波，该值有效。单位是“%”。
CARR, RISE	<rise>	:= 上升时间。当载波为脉冲波时该值有效。单位是“s”。可在数据手册查阅参数值的有效范围。
CARR, FALL	<fall>	:= 下降时间。当载波为脉冲波时该值有效。单位是“s”。可在数据手册查阅参数值的有效范围。
CARR, DLY	<delay>	:= 脉冲波延时。当载波为脉冲波时该值有效。单位是“s”。可在数据手册查阅参数值的有效范围。
CARR, STDEV	<stdev>	:= 噪声的标准差。单位是伏特“V”。可在数据手册查阅参数值的有效范围。

	CARR, MEAN	<mean>	:= 噪声的均值。单位是伏特“V”。可在数据手册查阅参数值的有效范围。
查询语法	<通道>:BTWV(BursTWaVe)? <通道>:= {C1, C2, C3, C4}。		
响应格式	<通道>:BTWV <参数> <参数>:= {当前脉冲串的所有参数}。		
示例	设置 CH1 脉冲串状态为 ON: *C1:BTWV STATE,ON* 设置 CH1 脉冲串周期为 1s: *C1:BTWV PRD,1* 设置脉冲串延时为 1s: *C1:BTWV DLAY,1* 设置 CH1 脉冲串为无限: *C1:BTWV TIME,INF* 读取状态为 ON 的 CH2 脉冲串参数: *C2:BTWV?* 返回值: *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* 读取状态为 OFF 的 CH2 脉冲串参数: *C2:BTWV?* 返回值: *C2:BTWV STATE,OFF*		

3.9 AWG 命令

层级为单层下，Scenario 和 Sequence 的段序号都设置为 1。

3.9.1 <channel>:AWG:STATE <state>

描述	该命令用于设置或查询 AWG 的运行状态。
命令语法	<channel>:AWG:STATe <state> <channel>:= {C1, C2, C3, C4}。 <state>:= {STOP, RUN}。
查询语法	<channel>:AWG:STATe? <channel>:= {C1, C2, C3, C4}。
示例	设置通道 1 的运行状态为 RUN： *C1:AWG:STATE RUN* 查询通道 1 的运行状态： *C1:AWG:STATE?* 返回值： *RUN\n*

3.9.2 <channel>:AWG:DEFAult

描述	该命令用于设置 AWG 的默认设置。
命令语法	<channel>:AWG:DEFAult <channel>:= {C1, C2, C3, C4}。

3.9.3 <channel>:AWG:SRATE <value>

描述	该命令用于设置或查询 AWG 的采样率。
命令语法	<channel>:AWG:SRATE <value> <channel>:= {C1, C2, C3, C4}。 <value>:= 采样率。
查询语法	<channel>:AWG:SRATe? <channel>:= {C1, C2, C3, C4}。

示例	设置通道 1 的采样率为 200M： *C1:AWG:SRATE 2e8* 查询通道 1 的采样率 *C1:AWG:SRATE?* 返回值： *200000000\n*

3.9.4 <channel>:AWG:SCALe <value>

描述	该命令用于设置或查询 AWG 的输出幅值比例。
命令语法	<channel>:AWG:SCALe <value> <channel>:= {C1, C2, C3, C4}。 <value>:={0, 1}。
查询语法	<channel>: AWG:SCALe? <channel>:= {C1, C2, C3, C4}。
示例	设置通道 1 的幅值比例为 88%： *C1:AWG:SCALe 0.88* 查询通道 1 的幅值比例： *C1:AWG:SCALe?* 返回值： *0.88\n*

3.9.5 <channel>:AWG:OFFset <value>

描述	该命令用于设置(查询)AWG 的偏移量。
命令语法	<channel>:AWG:OFFset <value> <channel>:= {C1, C2, C3, C4}。 <value>:= {不同输出模式下取值范围不同}。
查询语法	<channel>:AWG:OFFset? <channel>:= {C1, C2, C3, C4}。
示例	设置通道 1 的偏移量为 0.01： *C1:AWG:OFFset 0.01* 查询通道 1 的偏移量：

	C1:AWG:OFFset? 返回值： *0.01\n*

3.9.6 <channel>:AWG:DIFFset <value>

描述	该命令用于设置（查询）AWG 的差分偏置。
命令语法	<channel>:AWG:DIFFset <value> <channel>:= {C1, C2, C3, C4}。 <value>:= {不同输出模式下取值范围不同}。
查询语法	<channel>:AWG:DIFFset? <channel>:= {C1, C2, C3, C4}。
示例	设置通道 1 的偏移量为 0.01： *C1:AWG:DIFFset 0.01* 查询通道 1 的偏移量： *C1:AWG:DIFFset?* 返回值： *0.01\n*

3.9.7 <channel>:AWG:INTPtype <type>

描述	该命令用于设置（查询）AWG 的插值类型。
命令语法	<channel>:AWG:INTPtype <type> <channel>:= {C1, C2, C3, C4}。 <type>:= {ZERO, LINEAR, SINC, SINC13, SINC27}。
查询语法	<channel>:AWG:INTPtype? <channel>:= {C1, C2, C3, C4}。
示例	设置通道 1 的插值类型为 SINC： *C1:AWG:INTPtype SINC* 查询通道 1 的采样率： *C1:AWG:INTPtype?* 返回值： *SINC\n*

3.9.8 <channel>:AWG:INCReasing <type>

描述	该命令用于设置（查询）AWG 的插值策略。
命令语法	<channel>:AWG:INCReasing <type> <channel>:= {C1, C2, C3, C4}。 <type>:= {INT, ZERo, HLAS, DUPL}。
查询语法	<channel>: AWG:INCReasing? <channel>:= {C1, C2, C3, C4}。
示例	设置通道 1 的插值方式为保持： *C1:AWG:INCReasing HLAS* 查询通道 1 的插值方式： *C1:AWG:INCReasing?* 返回值： *HLAS\n*

3.9.9 <channel>:AWG:DECReasing <type>

描述	该命令用于设置（查询）AWG 的抽值策略。
命令语法	<channel>:AWG:DECReasing <type> <channel>:= {C1, C2, C3, C4}。 < type >:={ DECi, CTAi, CHEa }。
查询语法	<channel>: AWG:DECReasing? <channel>:= {C1, C2, C3, C4}。
示例	设置通道 1 的抽值方式为截去头部： *C1:AWG:DECReasing CHEa* 查询通道 1 的抽值方式： *C1:AWG:DECReasing?* 返回值： *CHEa\n*

3.9.10 <channel>:AWG:TRIGger:TIMer <value>

描述	该命令用于设置（查询）AWG 的定时器时间。
命令语法	<channel>:AWG:TRIGger:TIMer <value>

	<channel>:= {C1, C2, C3, C4}。
查询语法	<channel>:AWG:TRIGger:TIMer? <channel>:= {C1, C2, C3, C4}。
示例	设置通道 1 触发定时的时间为 1 秒： *C1:AWG:TRIGger:TIMer 1* 查询通道 1 触发定时的时间： *C1:AWG:TRIGger:TIMer?* 返回值： *1\n*

3.9.11 <channel>:AWG:TRIGAger:SLOPe <type >

描述	该命令用于设置（查询）AWG 外部触发 1 的边沿触发方式。
命令语法	<channel>:AWG:TRIGAger:SLOPe <type> <channel>:= {C1, C2, C3, C4}。 <type>:={ RISe, FALL, BOTH}。
查询语法	<channel>:AWG:TRIGAger:SLOPe? <channel>:= {C1, C2, C3, C4}。
示例	设置通道 1 外部触发 1 的触发沿为上升沿触发： *C1:AWG:TRIGAger:SLOPe RISe* 查询通道 1 外部触发 1 的触发沿： *C1:AWG:TRIGAger:SLOPe?* 返回值： *RISe\n*

3.9.12 <channel>:AWG:TRIGBger:SLOPe <type >

描述	该命令用于设置（查询）AWG 的外部触发 2 的边沿触发方式。
命令语法	<channel>:AWG:TRIGBger:SLOPe <type> <channel>:= {C1, C2, C3, C4}。 <type>:={ RISe, FALL, BOTH}。
查询语法	<channel>: AWG: TRIGBger:SLOPe? <channel>:= {C1, C2, C3, C4}。
示例	设置通道 1 外部触发 2 的触发沿为上升沿触发：

	C1:AWG:TRIGBger:SLOPe RISe 查询通道 1 外部触发 2 的触发沿： *C1:AWG:TRIGBger:SLOPe?* 返回值： *RISe\n*

3.9.13 <channel>:AWG:TRIGger:DELAy <value>

描述	该命令用于设置（查询）AWG 的触发延时。
命令语法	<channel>:AWG:TRIGger:DELAy <value> <channel>:= {C1, C2, C3, C4}。
查询语法	<channel>:AWG:TRIGger:DELAy? <channel>:= {C1, C2, C3, C4}。
示例	设置通道 1 触发触发延时为 20 ns： *C1:AWG:TRIGger:DELAy 2e-08* 查询通道 1 触发模式的触发方式： *C1:AWG:TRIGger:DELAy?* 返回值： *2e-08\n*

3.9.14 <channel>:AWG:HOLDtype <type>

描述	该命令用于设置（查询）AWG 空闲保持电平的类型。
命令语法	<channel>:AWG:HOLDtype <type> <channel>:= {C1, C2, C3, C4}。 <type>:= {MID, START, END, USER}。
查询语法	<channel>:AWG:INTPtype? <channel>:= {C1, C2, C3, C4}。
示例	设置通道 1 的保持电平类型为 END： *C1:AWG:HOLDtype END* 查询通道 1 的保持电平： *C1:AWG:HOLDtype?* 返回值：

	END\n

3.9.15 <channel>:AWG:USRHOLD <value>

描述	该命令用于设置（查询）AWG 自定义空闲保持电平类型下的电平。
命令语法	<channel>:AWG:USRHOLD <value> <channel>:= {C1, C2, C3, C4}。 <value>:= 自定义保持电平的电平值。
查询语法	<channel>:AWG:USRHOLD? <channel>:= {C1, C2, C3, C4}。
示例	设置通道 1 的自定义保持电平为 0.3： *C1:AWG:USRHOLD 0.3* 查询通道 1 的自定义保持电平： *C1:AWG:USRHOLD?* 返回值： *0.3\n*

3.9.16 <channel>:AWG:DYNA:TYPE <type>

描述	该命令用于设置（查询）AWG 动态模式的类型。
命令语法	<channel>:AWG:DYNA:TYPE <type> <channel>:= {C1, C2, C3, C4}。 <type>:= {JUMP_TYPE, CONTROL_TYPE}。
查询语法	<channel>:AWG:DYNA:TYPE? <channel>:= {C1, C2, C3, C4}。
示例	设置通道 1 的动态跳转类型为动态控制： *C1:AWG:DYNA:TYPE CONTROL_TYPE* 查询通道 1 的动态跳转类型： *C1:AWG:DYNA:TYPE?* 返回值： *CONTROL_TYPE\n*

3.9.17 <channel>:AWG:GATELevel <type>

描述	该命令用于设置（查询）AWG 的门控有效电平。
命令语法	<channel>:AWG:GATELevel <type> <channel>:= {C1, C2, C3, C4}。 <type>:= {NEGATIVE, POSITIVE}。
查询语法	<channel>: AWG:GATELevel? <channel>:= {C1, C2, C3, C4}。
示例	设置通道 1 的门控有效电平为正： *C1:AWG:GATELevel POSITIVE* 查询通道 1 的门控有效电平： *C1:AWG:GATELevel?* 返回值： *POSITIVE\n*

3.9.18 <channel>:AWG:SCENario:TIMer <value>

描述	该命令用于设置（查询）AWG 的 Scenario 定时时间。
命令语法	<channel>:AWG:SCENario:TIMer <value> <channel>:= {C1, C2, C3, C4}。 <value>:= 定时时间。
查询语法	<channel>:AWG:SCENario:TIMer? <channel>:= {C1, C2, C3, C4}。
示例	设置通道 1 的 Scenario 定时时间为 0.01： *C1:AWG:SCENario:TIMer 0.01* 查询通道 1 的采样率： *C1:AWG:SCENario:TIMer?* 返回值： *0.01\n*

3.9.19 <channel>:AWG:SEQUence:TIMer <value>

描述	该命令用于设置（查询）AWG 的 SEQUence 定时时间。
命令语法	<channel>:AWG:SEQUence:TIMer <value>

	<channel>:= {C1, C2, C3, C4}。 <value>:= 定时时间。
查询语法	<channel>:AWG:SEQUence:TIMer? <channel>:= {C1, C2, C3, C4}。
示例	设置通道 1 的 SEQUence 定时时间为 0.01： *C1:AWG:SEQUence:TIMer 0.01* 查询通道 1 的采样率： *C1:AWG:SEQUence:TIMer?* 返回值： *0.01\n*

3.9.20 <channel>:AWG:SEGMent:TIMer <value>

描述	该命令用于设置（查询）AWG 的 SEGMent 定时时间。
命令语法	<channel>:AWG:SEGMent:TIMer <value> <channel>:= {C1, C2, C3, C4}。 <value>:= 定时时间。
查询语法	<channel>:AWG:SEGMent:TIMer? <channel>:= {C1, C2, C3, C4}。
示例	设置通道 1 的 SEGMent 定时时间为 0.01： *C1:AWG:SEGMent:TIMer 0.01* 查询通道 1 的采样率： *C1:AWG:SEGMent:TIMer?* 返回值： *0.01\n*

3.9.21 <channel>:AWG:COMpensation <state>

描述	该命令用于设置（查询）AWG 的补偿开关状态。
命令语法	<channel>:AWG:COMpensation <state> <channel>:= {C1, C2, C3, C4}。 <state>:= {ON, OFF}。
查询语法	<channel>:AWG:COMpensation? <channel>:= {C1, C2, C3, C4}。

示例	设置通道 1 的补偿状态为开： *C1:AWG:COMpensation ON* 查询通道 1 的补偿开关状态： *C1:AWG:COMpensation?* 返回值： *ON\n*

3.9.22 <channel>:AWG:RMODe <type>

描述	该命令用于设置（查询）AWG 的运行模式。
命令语法	<channel>:AWG:RMODe <type> <channel>:= {C1，C2，C3，C4}。 <type>:= {CONT, TCON, GATE_EXTA, GATE_EXTB, ADV}。
查询语法	<channel>: AWG:RMODe? <channel>:= {C1, C2, C3, C4}。
示例	设置通道 1 的运行模式为高级： *C1:AWG:RMODe ADV* 查询通道 1 的运行模式： *C1:AWG:RMODe?* 返回值： *ADV\n*

3.9.23 <channel>:AWG:WMODe <type>

描述	该命令用于设置（查询）AWG 的层级模式。
命令语法	<channel>:AWG:WMODe <type> <channel>:= {C1, C2, C3, C4}。 <type>:= {SIMPLE, ADVANCED}。
查询语法	<channel>:AWG:WMODe? <channel>:= {C1, C2, C3, C4}。
示例	设置通道 1 的层级模式为多层： *C1:AWG:WMODe ADVANCED* 查询通道 1 的层级模式：

	C1:AWG:WMODe? 返回值： *ADVANCED\n*

3.9.24 <channel>:AWG:DYNA:TABLe:ADD PATT,<value1>,SCEN,<value2>,SEQU,<value3>,SEGM,<value4>

描述	该命令用于 AWG 的动态跳转表增加项。
命令语法	<channel>:AWG:DYNA:TABLe:ADD PATT,<value1>,SCEN,<value2>,SEQU,<value3>,SEGM,<value4> <channel>:= {C1, C2, C3, C4}。 <value1>:= [0, 255]。 <value2>:= [1, 当前设置的最大 scenario 段数]。 <value3>:= [1, 当前设置的最大 sequence 段数]。 <value4>:= [1, 当前设置的最大 segment 段数]。
查询语法	<channel>: AWG:DYNA:TABLe? <channel>:= {C1, C2, C3, C4}。
示例	设置通道 1 的动态跳转表增加一项： *C1:AWG:DYNA:TABLe:ADD PATT,99,SCEN,1,SEQU,1,SEGM,1* 查询通道 1 的动态跳转表： *C1:AWG:DYNA:TABLe?* 返回值： *pattern:99,go_secn:0,go_sequ:0,go_segm:0\n"\n*

3.9.25 <channel>:AWG:DYNA:TABLe:DELEte <value>

描述	该命令用于 AWG 的动态跳转表删除项。
命令语法	<channel>:AWG:DYNA:TABLe:DELEte <value> <channel>:= {C1, C2, C3, C4}。 <value>:= [0, 255]。
示例	设置通道 1 的动态跳转表删除一项： *C1:AWG:DYNA:TABLe:DELEte 0*

3.9.26 <channel>:AWG:DYNA:TABLe:CLEAR

描述	该命令用于 AWG 的动态跳转表清空操作。
命令语法	<channel>:AWG:DYNA:TABLe:CLEAR <channel>:= {C1, C2, C3, C4}。
示例	设置通道 1 的动态跳转表清空： *C1:AWG:DYNA:TABLe:CLEAR*

3.9.27 <channel>:AWG:DYNA:TABLe?

描述	该命令用于 AWG 的动态跳转表查询。
查询语法	<channel>:AWG:DYNA:TABLe? <channel>:= {C1, C2, C3, C4}。
示例	查询通道 1 的动态跳转表： *C1:AWG:DYNA:TABLe?* 返回值： *pattern:99,go_secn:0,go_sequ:0,go_segm:0\n"\n*

3.9.28 <channel>:AWG:TRIGger:SOURce <type>

描述	该命令用于设置（查询）AWG 触发模式下的触发方式。
命令语法	<channel>:AWG:AWG:TRIGger:SOURce <type> <channel>:= {C1, C2, C3, C4}。 <type>:={MANA, MANB, TIMe, EXTA, EXTB}。
查询语法	<channel>:AWG:TRIGger:SOURce? <channel>:= {C1, C2, C3, C4}。
示例	设置通道 1 触发模式的触发方式为手动触发 B： *C1:AWG:TRIGger:SOURce MANB* 查询通道 1 触发模式的触发方式： *C1:AWG:TRIGger:SOURce?* 返回值： *MANB\n*

3.9.29 <channel>:AWG:TRIGgerA

描述	该命令用于触发一次 AWG 的手动触发按钮 A。
命令语法	<channel>:AWG:TRIGgerA <channel>:= {C1, C2, C3, C4}。
示例	设置一次 AWG 手动触发 A： *C1:AWG:TRIGgerA*

3.9.30 <channel>:AWG:TRIGgerB

描述	该命令用于触发一次 AWG 的手动触发按钮 B。
命令语法	<channel>:AWG:TRIGgerB <channel>:= {C1, C2, C3, C4}。
示例	设置一次 AWG 手动触发 B： *C1:AWG:TRIGgerB*

3.9.31 <channel>:AWG:SAVE PATH,<path>,SWFS,<swfs>

描述	该命令用于保存通道的 AWG 配置、波形数据。
命令语法	<channel>:AWG:SAVE PATH,<path>,SWFS,<swfs> <channel>:= {C1, C2, C3, C4}。 <path>:= 保存的路径和文件名，文件名需带后缀“.awgx”。 <swfs>:= {"TRUE", "FALSE"}，用于确定是否保存当前段的波形数据。
示例	通道 3 保存 AWG 配置及波形数据： *C3:AWG:SAVE PATH,"Local/aaa.awgx",SWFS,"TRUE"*

3.9.32 <channel>:AWG:LOAD PATH,<path>

描述	该命令用于加载通道的 AWG 配置、波形数据。
命令语法	<channel>:AWG:SAVE PATH,<path>,SWFS,<swfs> <channel>:= {C1, C2, C3, C4}。 <path>:= 加载的路径和文件名，文件名需带后缀“.awgx”。
示例	通道 3 加载波形： *C3:AWG:LOAD "Local/aaa.awgx"*

3.9.33 <channel>:AWG:SCENario:CLEAR

描述	该命令用于清除 AWG 指定通道 scenario 所有段。
命令语法	<channel>:AWG:SCENario:CLEAR <channel>:= {C1, C2, C3, C4}。
示例	通道 3 清空所有 Scenario 段： *C3:AWG:SCENario:CLEAR*

3.9.34 <channel>:AWG:SCENario:COUNt?

描述	该命令用于查询 AWG 的通道 scenario 个数。
查询语法	<channel>:AWG:SCENario:COUNt? <channel>:= {C1, C2, C3, C4}。
示例	查询通道 3 的 Scenario 段数： *C3:AWG:SCENario:COUNt?* 返回值： *3\n*

3.9.35 <channel>:AWG:SCENario:INSErt <pos>

描述	该命令用于 AWG 指定段前插入一个 Scenario。
命令语法	<channel>:AWG:SCENario:INSErt <pos> <channel>:= {C1, C2, C3, C4}。 <pos>:= [1, 当前设置的最大 scenario 段数]。
示例	通道 3 的第 1 段 Scenario 前插入一段 Scenario： *C3:AWG:SCENario:INSErt 2*

3.9.36 <channel>:AWG:SCENario:DELEte <pos>

描述	该命令用于 AWG 指定删除一个 Scenario 段。
命令语法	<channel>:AWG:SCENario:DELEte <pos> <channel>:= {C1, C2, C3, C4}。 <pos>:= [1, 当前设置的最大 scenario 段数]。
示例	删除通道 3 的第 2 段 Scenario：

	C3:AWG:SCENario:DELEte 2

3.9.37 <channel>:AWG:SCENario:MULTIDelete <pos1, pos2, pos3...>

描述	该命令用于 AWG 指定删除多个 Scenario 段。
命令语法	<channel>:AWG:SCENario:MULTIDelete <pos1, pos2, pos3...> <channel>:= {C1, C2, C3, C4}。 <pos1, pos2, pos3...>:= [1, 当前设置的最大 scenario 段数]用于指定需要删除的段数。
示例	删除通道 3 的第 2、3、6 段 Scenario： *C3:AWG:SCENario:MULTIDelete 2,3,6*

3.9.38 <channel>:AWG:SCENario<x>:LOOP <value>

描述	该命令用于设置或查询 AWG 中指定 Scenario 的循环次数。
命令语法	<channel>:AWG:SCENario<x>:LOOP<value> <x>:= 段编号。 <channel>:= {C1, C2, C3, C4}。 <value>:= 整数。
查询语法	<channel>:AWG:SCENario<x>:LOOP? <channel>:= {C1, C2, C3, C4}。
示例	设置通道 1 的第一段 Scenario 的 loop 为 5： *C1:AWG:SCENario1:LOOP 5* 查询通道 1 第一段 Scenario 的 loop： *C1:AWG:SCENario1:LOOP?* 返回值： *5\n*

3.9.39 <channel>:AWG:SCENario:STARTNumb <value>

描述	该命令用于设置起始播放 Scenario。
命令语法	<channel>:AWG:SCENario:STARTNumb <value> <channel>:= {C1, C2, C3, C4}。 <value>:= 整数。

查询语法	<channel>:AWG:SCENario:STARTNumb? <channel>:= {C1, C2, C3, C4}。
示例	设置通道 1 的起始 Scenario 为 2： *C1:AWG:SCENario:STARTNumb 2* 查询通道 1 的起始 Scenario 段号： *C1:AWG:SCENario:STARTNumb?* 返回值： *2\n*

3.9.40 <channel>:AWG:SCENario<x>:WAITEvent <type>

描述	该命令用于设置或查询 Scenario 的等待事件。
命令语法	<channel>:AWG:SCENario<x>:WAITEvent <type> <x>:= 段编号。 <channel>:= {C1, C2, C3, C4}。 <type>:= {AUTO, MANA, MANB, EXTA, EXTB, TIMe}。
查询语法	<channel>: AWG:SCENario<x>:WAITEvent? <x>:= 段编号。
示例	设置通道 1 第一段 Scenario 等待事件为手动触发 A： *C1:AWG:SCENario1:WAITEvent MANA* 查询通道 1 第一段 Scenario 的等待事件： *C1:AWG:SCENario1:WAITEvent?* 返回值： *MANA\n*

3.9.41 <channel>:AWG:SCENario<x>:GOTO <value>

描述	该命令用于设置或查询指定 Scenario 的下一个。
命令语法	<channel>:AWG:SCENario<x>:GOTO <value> <x>:= 段编号。 <channel>:= {C1, C2, C3, C4}。 <value>:= [1, 当前设置的最大 scenario 段数]。
查询语法	<channel>:AWG:SCENario<x>:GOTO? <x>:= 段编号。

示例	设置通道 1 第一段 Scenario 的下一个为 3： *C1:AWG:SCENario1:GOTO 3* 查询通道 1 第一段 Scenario 的下一个： *C1:AWG:SCENario1:GOTO?* 返回值： *3\n*

3.9.42 <channel>:AWG:SCENario<x>:PLAYBack <type>

描述	该命令用于设置或查询指定 Scenario 的重放模式。
命令语法	<channel>:AWG:SCENario<x>:PLAYBack <type> <x>:= 段编号。 <channel>:= {C1, C2, C3, C4}。 <type>:= { AUTO, SINGLE, CONDITIONAL}。
查询语法	<channel>:AWG:SCENario<x>:PLAYBack? <x>:= 段编号。
示例	设置通道 1 第一段 Scenario 的重放模式为单次： *C1:AWG:SCENario1:PLAYBack SINGLE* 查询通道 1 第一段 Scenario 的下一个： *C1:AWG:SCENario1:PLAYBack?* 返回值： *SINGLE\n*

3.9.43 <channel>:AWG:SCENario<x>:OUTEvent <type>

描述	该命令用于设置或查询指定 Scenario 的跳出事件。
命令语法	<channel>:AWG:SCENario<x>:OUTEvent <type> <x>:= 段编号。 <channel>:= {C1, C2, C3, C4}。 <type>:= {MANA, MANB, EXTA, EXTB, TIMe}。
查询语法	<channel>:AWG:SCENario<x>:OUTEvent? <x>:= 段编号。
示例	设置通道 1 第一段 Scenario 的跳出事件为 Scenario 定时器： *C1:AWG:SCENario1:OUTEvent TIMe*

	查询通道 1 第一段 Scenario 的跳出事件： *C1:AWG:SCENario1:OUTEven?* 返回值： *TIMe\n*

备注：重放模式为条件跳出时有效。

3.9.44 <channel>:AWG:SCENario<x>:SEQUence:STARTNumb <value>

描述	该命令用于设置或查询指定 Scenario 的起始 sequence 序号。
命令语法	<channel>:AWG:SCENario<x>:SEQUence:STARTNumb <value> <x>:= 段编号。 <channel>:= {C1, C2, C3, C4}。 <value>:= [1, 当前 scenario 设置的最大 sequence 段序号]。
查询语法	<channel>:AWG:SCENario<x>:SEQUence:STARTNumb? <x>:= 段编号。
示例	设置通道 1 第一段 Scenario 的起始 sequence 序号为 1： *C1:AWG:SCENario1:SEQUence:STARTNumb 1* 查询通道 1 第一段 Scenario 的起始 sequence 序号： *C1:AWG:SCENario1:SEQUence:STARTNumb?* 返回值： *1\n*

3.9.45 <channel>:AWG:SCENario<x>:SAVE PATH,<path>, SWFS,<swfs>

描述	该命令用于保存通道指定的 Scenario 配置、波形数据。
保存命令	<channel>:AWG:SCENario<x>:SAVE ATH,<path>,SWFS,<swfs> <channel>:= {C1, C2, C3, C4}。 <x>:= 段编号。 <path>:= 保存的路径和文件名，文件名后缀为.scen。 <swfs>:= {"TRUE", "FALSE"}，确定是否保存波形数据。
示例	通道 3 保存第一段 SCENario 的配置及波形数据： *C3:AWG:SCENario1:SAVE PATH,"Local/aaa.scen",SWFS,"TRUE"*

3.9.46 <channel>:AWG:SCENario:LOAD,<path>

描述	该命令用于通道加载 Scenario 配置、波形数据。
命令	<channel>:AWG:SCENario:LOAD <path> <channel>:= {C1, C2, C3, C4}。 <path>:= 保存的路径和文件名，文件名后缀是.scen。
示例	通道 3 加载波形： *C3:AWG:SCENario:LOAD "Local/aaa.scen"*

3.9.47 <channel>:AWG:SCENario<x>:SEQUence:CLEAR

描述	该命令用于清空指定 Scenario 的 sequence 的列表。
命令语法	<channel>:AWG:SCENario<x>:SEQUence:CLEAR <x>:= 段编号。 <channel>:= {C1, C2, C3, C4}。
示例	设置清空通道 1 第一段 Scenario 的所有 sequence： *C1:AWG:SCENario1:SEQUence:CLEAR*

3.9.48 <channel>:AWG:SCENario<x>:SEQUence:COUNt?

描述	该命令用于查询指定 Scenario 的 sequence 个数。
查询语法	<channel>:AWG:SCENario<x>:SEQUence:COUNt? <x>:= 段编号。 <channel>:= {C1, C2, C3, C4}。
示例	查询通道 1 第一段 Scenario 的 sequence 段数： *C1:AWG:SCENario1:SEQUence:COUNT?* 返回值 *3\n*

3.9.49 <channel>:AWG:SCENario<x>:SEQUence:INSErt <pos>

描述	该命令用于设置指定 Scenario 插入一段 sequence。
命令语法	<channel>:AWG:SCENario<x>:SEQUence:INSErt <pos> <x>:= 段编号。

	<channel>:= {C1, C2, C3, C4}。 <pos>:= [0, 当前设置的最大 sequence 段数]，插入段的位置。
示例	通道 1 第一段 Scenario 插一段 sequence 到第二段： *C1:AWG:SCENario1:SEQUence:INSErt 2*

3.9.50 <channel>:AWG:SCENario<x>:SEQUence:DELEte <pos>

描述	该命令用于删除指定 Scenario 中的指定 sequence 段。
命令语法	<channel>:AWG:SCENario<x>: SEQUence:DELEte <pos> <x>:= 段编号。 <channel>:= {C1, C2, C3, C4}。 <pos>:= [0, 当前设置的最大 sequence 段数]。
示例	删除通道 1 第一段 Scenario 中第二段 sequence： *C1:AWG:SCENario1:SEQUence:DELEte 2*

3.9.51 <channel>:AWG:SCENario<x>:SEQUence:MULTIDelete <pos1,pos2,pos3...>

描述	该命令用于 AWG 删除指定 Scenario 中指定的多个 sequence 段。
命令语法	<channel>:AWG:SCENario<x>:MULTIDelete <pos1, pos2, pos3...> <x>:= scenario 段编号。 <channel>:= {C1, C2, C3, C4}。 <pos1, pos2, pos3...>:= [1, 当前设置的最大 sequence 段数]用于指定需要删除的段数。
示例	删除通道 3 的第 2 段 Scenario 的第 2，3，6 段 sequence： *C3:AWG:SCENario2:SEQUence:MULTIDelete 2,3,6*

3.9.52 <channel>:AWG:SCENario<x>:SEQUence<y>:LOOP <value>

描述	该命令用于设置或查询 Scenario 中 sequence 的循环次数。
命令语法	<channel>:AWG:SCENario<x>:SEQUence<y>:LOOP <value> <x>:= 段编号　　<y>:= 段编号 <channel>:= {C1, C2, C3, C4}。 <value>:= 整数。

查询语法	<channel>:AWG:SCENario<x>:SEQUence<y>:LOOP? <x>:=段编号　　<y>:= 段编号。 <channel>:= {C1, C2, C3, C4}。
示例	设置通道 1 第一段 Scenario 的第一段 sequence 循环次数为 3： *C1:AWG:SCENario1:SEQUence1:LOOP 3* 查询通道 1 第一段 Scenario 的第一段 sequence 循环次数： *C1:AWG:SCENario1:SEQUence1:LOOP?* 返回值： *3\n*

3.9.53 <channel>:AWG:SCENario<x>:SEQUence<y>: WAITEvent <type>

描述	该命令用于设置或查询 Scenario 中 sequence 的等待事件。
命令语法	<channel>:AWG:SCENario<x>: SEQUence<y>: WAITEvent <type> <x>:= 段编号　　<y>:= 段编号 <channel>:= {C1, C2, C3, C4}。 <type>:= {AUTO, MANA, MANB, EXTA, EXTB, TIMe}。
查询语法	<channel>:AWG:SCENario<x>:SEQUence<y>:WAITEvent? <x>:= 段编号　　<y>:= 段编号 <channel>:= {C1, C2, C3, C4}。
示例	设置通道 1 第一段 Scenario 第一段 sequence 等待事件为手动触发 B： *C1:AWG:SCENario1:SEQUence1:WAITEvent MANB* 查询通道 1 第一段 Scenario 的第一段 sequence 循环次数： *C1:AWG:SCENario1:SEQUence1:WAITEvent?* 返回值： *MANB\n*

3.9.54 <channel>:AWG:SCENario<x>:GOTO <value>

描述	该命令用于设置或查询 Scenario 中 sequence 的下一个 sequence 播放段序号。
命令语法	<channel>:AWG:SCENario<x>:SEQUence<y>:GOTO <value> <x>:= 段编号　　<y>:= 段编号 <channel>:= {C1, C2, C3, C4}。 <value>:= [0, 当前设置的最大 sequence 段数]。

查询语法	<channel>:AWG:SCENario<x>:SEQUence<y>:GOTO? <x>:=段编号　　<y>:=段编号 <channel>:= {C1, C2, C3, C4}。
示例	设置通道 1 第一段 Scenario 第一段 sequence 播放完下一个播放序号 3 的 sequence： *C1:AWG:SCENario1:SEQUence1:GOTO 3* 查询通道 1 第一段 Scenario 的第一段 sequence 循环次数： *C1:AWG:SCENario1:SEQUence1:GOTO?* 返回值： *3\n*

3.9.55 <channel>:AWG:SCENario<x>: SEQUence<y>:PLAYBack <type>

描述	该命令用于设置或查询 Scenario 中 sequence 的重放模式。
命令语法	<channel>:AWG:SCENario<x>:SEQUence<y>:PLAYBack <type> <x>:= 段编号　　<y>:= 段编号 <channel>:= {C1, C2, C3, C4}。 <type>:= {AUTO, SINGLE, CONDITIONAL}。
查询语法	<channel>:AWG:SCENario<x>:SEQUence<y>:PLAYBack? <x>:= 段编号　　<y>:= 段编号 <channel>:= {C1, C2, C3, C4}。
示例	设置通道 1 第一段 Scenario 第一段 sequence 的重放模式为单次： *C1:AWG:SCENario1:SEQUence1:PLAYBack SINGLE* 查询通道 1 第一段 Scenario 的第一段 sequence 的重放模式： *C1:AWG:SCENario1:SEQUence1:PLAYBack?* 返回值： *SINGLE\n*

3.9.56 <channel>:AWG:SCENario<x>:SEQUence<y>:OUTEvent <type>

描述	该命令用于设置或查询 Scenario 中 sequence 的跳出事件。
命令语法	<channel>:AWG:SCENario<x>:SEQUence<y>:OUTEvent <type> <x>:= 段编号　　<y>:= 段编号 <channel>:= {C1, C2, C3, C4}。 <type>:= {MANA, MANB, EXTA, EXTB, TIMe}。

查询语法	<channel>:AWG:SCENario<x>:SEQUence<y>:OUTEvent? <x>:= 段编号　　<y>:= 段编号 <channel>:= {C1, C2, C3, C4}。
示例	设置通道 1 第一段 Scenario 第一段 sequence 跳出事件为手动触发 A: *C1:AWG:SCENario1:SEQUence1:OUTEvent MANA* 查询通道 1 第一段 Scenario 的第一段 sequence 的跳出事件: *C1:AWG:SCENario1:SEQUence1:OUTEvent?* 返回值: *MANA\n*

3.9.57 <channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent:CLEAR

描述	该命令用于清除某个 Scenario 中某个 sequence 的 segment 列表。
命令语法	<channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent:CLEAR <x>:= 段编号　　<y>:= 段编号 <channel>:= {C1, C2, C3, C4}。
示例	清空通道 1 第一段 Scenario 中第一段 sequence 所有 Segment 段: *C1:AWG:SCENario1:SEQUence1:SEGMent:CLEAR*

3.9.58 <channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent:STARTNumb <value>

描述	该命令用于设置 Scenario 中 sequence 的起始 segment 段序号。
命令语法	<channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent:STARTNumb <value> <x>:= 段编号　　<y>:= 段编号 <channel>:= {C1, C2, C3, C4}。 <value>:= segment 段编号。
查询语法	<channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent: STARTNumb? <x>:= 段编号　　<y>:= 段编号 <channel>:= {C1, C2, C3, C4}。
示例	设置通道 1 第一段 Scenario 的第一段 sequence 的起始 Segment 序号为 3: *C1:AWG:SCENario1:SEQUence1:SEGMent:STARTNumb 3* 查询通道 1 第一段 Scenario 中第一段 sequence 的起始 Segment 段序号:

	C1:AWG:SCENario1:SEQUence1:SEGMent:STARTNumb? 返回值： *3\n*

3.9.59 <channel>:AWG:SCENario<x>:SEQUence<y>:SAVE PATH,<path>,SWFS,<swfs>

描述	该命令用于保存通道指定的 Scenario 中指定的 sequence 配置、波形数据。
命令	<channel>:AWG:SCENario<x>:SEQUence<y>:SAVE PATH,<path>,SWFS,<swfs> <channel>:= {C1, C2, C3, C4}。 <x>:= 段编号　　<y>:= 段编号 <path>:= 保存的路径和文件名，文件名需要带后缀.seq。 <swfs>:= {"TRUE", "FALSE"}。
示例	通道 3 保存第一段 SCENario 中第一段 sequence 的配置，不包含波形数据： *C3:AWG:SCENario1:SEQUence1:SAVE PATH,"Local/aaa.seq",SWFS,"FALSE"*

3.9.60 <channel>:AWG:SCENario<x>:SEQUence:LOAD PATH,<path>

描述	该命令用于通道指定 Scenario 下加载 sequence 配置、是波形数据文件。
命令	<channel>:AWG: SCENario <x>: SEQUence:LOAD <path> <channel>:= {C1, C2, C3, C4}。 <x>:= 段编号。 <path>:= 保存的路径和文件名，文件名包括后缀.seq。
示例	通道 3 第一段 Scenario 加载文件： *C3:AWG: SCENario1:SEQUence:LOAD "Local/aaa.seq"*

3.9.61 <channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent:COUNt?

描述	该命令用于查询某个 Scenario 中某个 sequence 的 segment 段数。
查询语法	<channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent:COUNt? <channel>:= {C1, C2, C3, C4}。 <x>:= 段编号　　<y>:= 段编号
示例	查询通道 3 第一段 Scenario 中第一段 sequence 的段数：

	C3:AWG:SCENario1:SEQUence1:SEGMent:COUNT? 返回值： *3\n*

3.9.62 <channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent:INSErt <pos>

描述	该命令用于某个 Scenario 中的某个 sequence 的某个 segment 后插入一个 segment。
命令语法	<channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent:INSErt <pos> <channel>:= {C1, C2, C3, C4}。 <x>:=段编号　　<y>:=段编号 <pos>:= segment 段编号。
示例	通道 3 第一段 Scenario 中第一段 sequence 插入一段波形到序号 5： *C3:AWG:SCENario1:SEQUence1:SEGMent:INSErt 5*

3.9.63 <channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent: DELEte <pos>

描述	该命令用于删除某个 Scenario 的某个 sequence 的某个 segment。
命令语法	<channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent:DELEte <pos> <channel>:= {C1, C2, C3, C4}。 <x>:= 段编号　　<y>:= 段编号 <pos>:= segment 段编号。
示例	删除通道 3 第一段 Scenario 中第一段 sequence 的第 5 段 segment： *C3:AWG:SCENario1:SEQUence1:SEGMent:DELEte 5*

3.9.64 <channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent:MULTIDelete <pos1,pos2,pos3...>

描述	该命令用于 AWG 删除指定 Scenario、指定 sequence 中指定的多个 sequence 段。
命令语法	<channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent:MULTIDelete <pos1, pos2, pos3...> <x>:= scenario 段编号。 <y>:= sequence 段编号。 <channel>:= {C1, C2, C3, C4}。

	<pos1, pos2, pos3...>:= [1, 当前设置的最大 segment 段数]用于指定需要删除的段数。
示例	删除通道 3 的第 2 段 Scenario、第 2 段 sequence 的第 2、3、6 段 segment: *C3:AWG:SCENario2:SEQUence2:SEGMent:MULTIDelete 2,3,6*

3.9.65 <channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>:LOOP <value>

描述	该命令用于设置或查询某个 Scenario 的某个 sequence 的某个 segment 的循环次数。
命令语法	<channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>:LOOP <value> <channel>:= {C1, C2, C3, C4}。 <x>:= 段编号　　<y>:= 段编号　　<z>:= 段编号 <value>:= 整数。
查询语法	<channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>: LOOP? <channel>:= {C1, C2, C3, C4}。 <x>:= 段编号　　<y>:= 段编号
示例	设置通道 1 第一段 Scenario 的第一段 sequence 的第一段 segment 循环次数为 5: *C1:AWG:SCENario1:SEQUence1:SEGMent1:LOOP 5* 查询通道 1 第一段 Scenario 的第一段 sequence 的第一段 segment 循环次数: *C1:AWG:SCENario1:SEQUence1:SEGMent1:LOOP?* 返回值: *5\n*

3.9.66 <channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>: WAITEvent <type>

描述	该命令用于设置或查询某个 Scenario 的某个 sequence 的某个 segment 的等待事件。
命令语法	<channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>: WAITEvent <type> <channel>:= {C1, C2, C3, C4}。 <x>:= 段编号　　<y>:= 段编号　　<z>:= 段编号

	<type>:= {AUTO, MANA, MANB, EXTA, EXTB, TIMe}。
查询语法	<channel>:AWG:SCENario<x>:SEQUence<y>: SEGMent<z>:WAITEvent? <channel>:= {C1, C2, C3, C4}。 <x>:= 段编号　　<y>:= 段编号
示例	设置通道 1 第一段 Scenario 的第一段 sequence 的第一段 segment 的等待事件为手动触发 B: *C1:AWG:SCENario1:SEQUence1:SEGMent1:WAITEvent MANB* 查询通道 1 第一段 Scenario 的第一段 sequence 的第一段 segment 的等待事件: *C1:AWG:SCENario1:SEQUence1:SEGMent1:WAITEvent?* 返回值: *MANB\n*

3.9.67 <channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>: GOTO <value>

描述	该命令用于设置或查询某个 Scenario 的某个 sequence 的某个 segment 的下一个。
命令语法	<channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>: GOTO <value> <channel>:= {C1, C2, C3, C4}。 <x>:= 段编号　　<y>:= 段编号　　<z>:= 段编号 <value>:= {整数, 不超过 segment 最大段数}。
查询语法	<channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>:GOTO? <channel>:= {C1, C2, C3, C4}。 <x>:= 段编号　　<y>:= 段编号　　<z>:= 段编号
示例	设置通道 1 第一段 Scenario 的第一段 sequence 的第一段 segment 的下一个为段序号为 3 的 Segment: *C1:AWG:SCENario1:SEQUence1:SEGMent1:GOTO 3* 查询通道 1 第一段 Scenario 的第一段 sequence 的第一段 segment 的下一个: *C1:AWG:SCENario1:SEQUence1:SEGMent1:GOTO?* 返回值: *3\n*

3.9.68 <channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>: PLAYBack <type>

描述	该命令用于设置或查询某个 Scenario 的某个 sequence 的某个 segment 的重放模式。
命令语法	<channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>: PLAYBack <type> <channel>:= {C1, C2, C3, C4}。 <x>:= 段编号 <y>:= 段编号 <z>:= 段编号 <type>:= {AUTO, SINGLE, CONDITIONAL}。
查询语法	<channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>:PLAYBack? <channel>:= {C1, C2, C3, C4}。 <x>:= 段编号 <y>:= 段编号 <z>:= 段编号
示例	设置通道 1 第一段 Scenario 的第一段 sequence 的第一段 segment 的重放模式为单次： *C1:AWG:SCENario1:SEQUence1:SEGMent1:PLAYBack SINGLE* 查询通道 1 第一段 Scenario 的第一段 sequence 的第一段 segment 的重放模式： *C1:AWG:SCENario1:SEQUence1:SEGMent1:PLAYBack?* 返回值： *SINGLE\n*

3.9.69 <channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>: OUTEvent <type>

描述	该命令用于设置或查询某个 Scenario 的某个 sequence 的某个 segment 的跳出事件。
命令语法	<channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>:OUTEvent <type> <channel>:= {C1, C2, C3, C4}。 <x>:= 段编号 <y>:= 段编号 <z>:= 段编号 <type>:= {MANA, MANB, EXTA, EXTB, TIMe}。
查询语法	<channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>:OUTEvent? <channel>:= {C1, C2, C3, C4}。 <x>:= 段编号 <y>:= 段编号 <z>:= 段编号
示例	设置通道 1 第一段 Scenario 的第一段 sequence 的第一段 segment 的跳出

	事件为 segment 定时器： *C1:AWG:SCENario1:SEQUence1:SEGMent1:OUTEvent TIMe* 查询通道 1 第一段 Scenario 的第一段 sequence 的第一段 segment 的跳出事件： *C1:AWG:SCENario1:SEQUence1:SEGMent1:OUTEvent?* 返回值： *TIMe\n*

3.9.70 <channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>: WAVeform <wavename>

描述	该命令用于设置或查询某个 Scenario 的某个 sequence 的某个 segment 的波形。
命令语法	<channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>:WAVeform <wavename> <channel>:= {C1, C2, C3, C4}。 <x>:= 段编号　　<y>:= 段编号　　<z>:= 段编号 <wavename >:= {内建波表名，详见 3.5.3AFG 任意波切换波形命令} 或 <wavename >:= {文件路径（包含波形名及后缀）} 注：波形名字或波形路径需要加双引号“”。
查询语法	<channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>: WAVeform? <channel>:= {C1, C2, C3, C4}。 <x>:= 段编号　　<y>:= 段编号　　<z>:= 段编号
示例	设置通道 1 第一段 Scenario 的第一段 sequence 的第一段 segment 的波形为内建波 Noise： *C1:AWG:SCENario1:SEQUence1:SEGMent1:WAVeform “noise”* 设置通道 1 第一段 Scenario 的第一段 sequence 的第一段 segment 的波形为本地路径 Local 下的 wave.bin： *C1:AWG:SCENario1:SEQUence1:SEGMent1:WAVeform "Local/wave1.bin"* 查询通道 1 第一段 Scenario 的第一段 sequence 的第一段 segment 的波形名字： *C1:AWG:SCENario1:SEQUence1:SEGMent1:WAVeform?* 返回值： *wave1\n*

3.9.71 <channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>: AMPlitude <value>

描述	该命令用于设置某个 Scenario 的某个 sequence 的某个 segment 波形的幅度。
命令语法	<channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>:AMPlitude <value> <channel>:= {C1, C2, C3, C4}。 <x>:= 段编号　　<y>:= 段编号　　<z>:= 段编号 <value>:= 幅度，单位 Vpp。
查询语法	<channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>:AMPlitude? <channel>:= {C1, C2, C3, C4}。 <x>:= 段编号　　<y>:= 段编号　　<z>:= 段编号
示例	设置通道 1 第一段 Scenario 的第一段 sequence 的第一段 segment 的波形幅度为 0.3 Vpp： *C1:AWG:SCENario1:SEQUence1:SEGMent1:AMPlitude 0.3* 查询通道 1 第一段 Scenario 的第一段 sequence 的第一段 segment 的波形幅度： *C1:AWG:SCENario1:SEQUence1:SEGMent1:AMPlitude?* 返回值： *0.3\n*

3.9.72 <channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>: OFFset <value>

描述	该命令用于设置或查询某个 Scenario 的某个 sequence 的某个 segment 波形的偏移量。
命令语法	<channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>:OFFset <value> <channel>:= {C1, C2, C3, C4}。 <x>:= 段编号　　<y>:= 段编号　　<z>:= 段编号 <value>:= 偏移量，单位 Vdc。
查询语法	<channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>:OFFset? <channel>:= {C1, C2, C3, C4}。 <x>:= 段编号　　<y>:= 段编号　　<z>:= 段编号
示例	设置通道 1 第一段 Scenario 的第一段 sequence 的第一段 segment 的波形

	偏移量为 0.3 Vdc： *C1:AWG:SCENario1:SEQUence1:SEGMent1:OFFset 0.3* 查询通道 1 第一段 Scenario 的第一段 sequence 的第一段 segment 的波形偏移量： *C1:AWG:SCENario1:SEQUence1:SEGMent1:OFFset?* 返回值： *0.3\n*

3.9.73 <channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>: VOLTage:HIGH <value>

描述	该命令用于设置或查询某个 Scenario 的某个 sequence 的某个 segment 波形的高电平。
命令语法	<channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>:VOLTage:HIGH <value> <channel>:= {C1, C2, C3, C4}。 <x>:= 段编号　　<y>:= 段编号　　<z>:= 段编号 <value>:= 高电平，单位 V。
查询语法	<channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>:VOLTage:HIGH? <channel>:= {C1, C2, C3, C4}。 <x>:= 段编号　　<y>:= 段编号　　<z>:= 段编号
示例	设置通道 1 第一段 Scenario 的第一段 sequence 的第一段 segment 的波形高电平为 0.3 V： *C1:AWG:SCENario1:SEQUence1:SEGMent1:VOLTage:HIGH 0.3* 查询通道 1 第一段 Scenario 的第一段 sequence 的第一段 segment 的波形高电平： *C1:AWG:SCENario1:SEQUence1:SEGMent1:VOLTage:HIGH?* 返回值： *0.3\n*

3.9.74 <channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>: VOLTage: LOW <value>

描述	该命令用于设置或查询某个 Scenario 的某个 sequence 的某个 segment 波形的低电平。

命令语法	<channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>:VOLTage:LOW <value> <channel>:= {C1, C2, C3, C4}。 <x>:= 段编号　　<y>:= 段编号　　<z>:= 段编号 <value>:= 低电平，单位 V。
查询语法	<channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>:VOLTage:LOW? <channel>:= {C1, C2, C3, C4}。 <x>:= 段编号　　<y>:= 段编号　　<z>:= 段编号
示例	设置通道 1 第一段 Scenario 的第一段 sequence 的第一段 segment 的波形低电平为-0.3 V： *C1:AWG:SCENario1:SEQUence1:SEGMent1:VOLTage:LOW -0.3* 查询通道 1 第一段 Scenario 的第一段 sequence 的第一段 segment 的波形低电平： *C1:AWG:SCENario1:SEQUence1:SEGMent1:VOLTage:LOW?* 返回值： *-0.3\n*

3.9.75 <channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>:LENGth <value>

描述	该命令用于设置或查询某个 Scenario 的某个 sequence 的某个 segment 波形的长度。
命令语法	<channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>:LENGth <value> <channel>:= {C1, C2, C3, C4}。 <x>:= 段编号　　<y>:= 段编号　　<z>:= 段编号 <value>:= 波形长度。
查询语法	<channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>:LENGth? <channel>:= {C1, C2, C3, C4}。 <x>:= 段编号　　<y>:= 段编号　　<z>:= 段编号
示例	设置通道 1 第一段 Scenario 的第一段 sequence 的第一段 segment 的波形长度为 5000000： *C1:AWG:SCENario1:SEQUence1:SEGMent1:LENGth 5000000* 查询通道 1 第一段 Scenario 的第一段 sequence 的第一段 segment 的波形低电平： *C1:AWG:SCENario1:SEQUence1:SEGMent1:LENGth?*

	返回值： *5000000\n*

3.9.76 <channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>:MARK1er:POS<K> <state>

描述	该命令用于设置或查询 segment 的标记 1 开关状态。
命令语法	<channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>:MARK1er:POS<K> <state> <channel>:= {C1, C2, C3, C4}。 <x>:= 段编号　　<y>:= 段编号　　<z>:= 段编号 < state >:= {ON, OFF}。
查询语法	<channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>:MARK1er? <channel>:= {C1, C2, C3, C4}。 <x>:= 段编号　　<y>:= 段编号　　<z>:= 段编号 <K>:= 标记序号，波形数据长度。
示例	设置通道 1 第一段 Scenario 的第一段 sequence 的第一段 segment 的标记 1 的数据 1 打开： *C1:AWG:SCENario1:SEQUence1:SEGMent1:MARK1er:POS1 ON* 查询通道 1 第一段 Scenario 的第一段 sequence 的第一段 segment 的标记 1 状态： *C1:AWG:SCENario1:SEQUence1:SEGMent1:MARK1er?* 返回值： *"0,1,2,4"\n* #表示数据 0，1，2，4 数据位的标记打开

3.9.77 <channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>:MARK2er:POS<K> <state>

描述	该命令用于设置 segment 的标记 2。
命令语法	<channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>:MARK2er:POS<K> <state> <channel>:= {C1, C2, C3, C4}。 <x>:= 段编号　　<y>:= 段编号　　<z>:= 段编号 < state >:= {ON, OFF}。

查询语法	<channel>:AWG:SCENario<x>:SEQUence<y>:SEGMent<z>:MARK2er? <channel>:= {C1, C2, C3, C4}。 <x>:= 段编号　　<y>:= 段编号　　<z>:= 段编号 <K>:= 标记序号，波形数据长度。
示例	设置通道 1 第一段 Scenario 的第一段 sequence 的第一段 segment 的标记 2 的数据 1 打开： *C1:AWG:SCENario1:SEQUence1:SEGMent1:MARK2er:POS1 ON* 查询通道 1 第一段 Scenario 的第一段 sequence 的第一段 segment 的标记 2 状态： *C1:AWG:SCENario1:SEQUence1:SEGMent1:MARK2er?* 返回值： *"0,1,2,4"\n* #表示数据 0，1，2，4 数据位的标记打开

3.10 跳频命令

描述	该命令用于设置或者获取跳频功能的参数。
命令语法	<通道>:FHOP<参数> <值> <通道>:= {C1, C2, C3, C4}。 <值>:= {相关参数的值}。

参数	值	描述
SWITch	<state>	:= {ON, OFF}。设置跳频输出状态。
TYPE	<type>	:= {MANUAL, RHOP, RLIST}。设置跳频模式。
TIME	<time>	:= {0 到 1000 的浮点数}。单位秒。设置跳频时间。
SFREquency	<value>	:= {Min_Freq ~ 2 GHz 的浮点数}，设置随机跳频的起始频率。
EFREquency	<value>	:= {Min_Freq ~ 2 GHz 的浮点数}，设置随机跳频的终止频率。
FSTep	<value>	:= {Min_Freq ~ 2 GHz 的浮点数}，设置随机跳频的频率步进。
RPATtern	<value>	:= {3 到 32 的整数}。设置随机跳频的伪随机码型。
RLPATtern	<value>	:= {3 到 32 的整数}。设置随机列表跳频的伪随机码型。
ALSTate	<state>	:= {ON, OFF}。设置随机跳频频率过滤表的使能状态。
AFLIst <index>	<value>	:<index>= {1 到 4096 的整数}。 :<value>= {Min_Freq ~ 2 GHz 的浮点数}。在频率表指定项之前插入新的项，并指定频率值。
MFLIst <index>	<value>	:<index>= {1 到 4096 的整数}。 :<value>= {Min_Freq ~ 2 GHz 的浮点数}。修改频率表指定项之的频率值。
DFLIst <index>		:<index>= {1 到 4096 的整数}。 删除频率表指定项。
CFLIst		清空频率表，恢复为默认设置。
AOLIst <index>	<freq_num>	:<index>= {1 到 4096 的整数}。 :<freq_num>:= {1 到 4096 的整数，不超过频率表的长度}。在频率跳转表指定项之前插入新的项，并指定频率值。
MOLIst <index>	<freq_num>	:<index> ={1 到 4096 的整数}。

		:<freq_num>:= {1 到 4096 的整数，不超过频率表的长度}。修改频率跳转表指定项的频率值。
DOLIst <index>		:<index>= {1 到 4096 的整数}。 删除频率跳转表指定项。
COLIst		清空跳转频率表，恢复为默认设置。
AALIst	<start_freq, end_freq>	:<start_freq>= {Min_Freq ~ 2 GHz 的浮点数，小于 end_freq} :<end_freq>:= {Min_Freq ~ 2 GHz 的浮点数，大于 end_freq }。在频率过滤表插入新的项，并指定频率值。
MALIst <index>	<start_freq, end_freq>	:<index>= {1 到 4096 的整数}。 :<start_freq>={Min_Freq ~ 2 GHz 的浮点数，小于 end_freq }。 :<end_freq>:= {Min_Freq ~ 2 GHz 的浮点数，大于 end_freq}。修改频率过滤表指定项的频率值。
DALIst <index>		:<index>= {1 到 4096 的整数}。 删除频率过滤表指定项。
CALIst		清空跳转过滤表，恢复为默认设置。
LFLIst	<file>	:= 文件名（包含路径）。路径及文件名需要用双引号。
SFLIst	<file>	:= 文件名（包含路径）。路径及文件名需要用双引号，路径需要在文件系统中存在。
LOLIst	<file>	:= 文件名（包含路径）。路径及文件名需要用双引号。
SOLIst	<file>	:= 文件名（包含路径）。路径及文件名需要用双引号，路径需要在文件系统中存在。
LALIst	<file>	:= 文件名（包含路径）。路径及文件名需要用双引号。
SALIst	<file>	:= 文件名（包含路径）。路径及文件名需要用双引号，路径需要在文件系统中存在。

查询语法

<通道>:SWEep <参数>?
<通道>:= {C1, C2, C3, C4}。

示例

通道 1 的频率过滤表保存到文件 avoid.hop：

C1:FHOP:SOLIst "order.hop"

修改通道 1 的频率过滤表第一项的频率为 1K 到 10K：
C1:FHOP:MALIst 1,1000,10000

查询通道 1 跳频开关状态：
C1:FHOP:SWITch?
返回值：
"ON"

3.11 多音命令

描述	该命令用于设置或者获取多音功能的参数。
命令语法	<通道>:MTONE:<参数> <值> <通道>:= {C1, C2, C3, C4}。 <值>:= {相关参数的值}。

参数	值	描述
STAte	<state>	:= {ON, OFF}。设置切换到多音功能。前提是设置 AWG 模式，多音所有参数需要在此指令为"ON"状态下才可以设置。
RSTAte	<state>	:= {ON, OFF}。设置多音功能运行状态。前提是设置多音功能，多音所有参数需要在此指令为"OFF"状态下才可以设置。
SRATe	<value>	:= {100 Sa/s 到 5 GSa/s}。设置多音输出采样率。采样率>=2.5*输出的最大多音频率。
AMPlitude	<value>	:= 幅度。单位是伏特，峰峰值"Vpp"。不同输出模式设置范围不一样，可在数据手册查阅参数值的有效范围。
SFREquency	<value>	:= {Min 到 2 GHz}。设置起始频率。
EFREquency	<value>	:= {Min 到 2 GHz }。设置终止频率。
TNUMber	<value>	:= {2 到 1000 的整数}。设置多音数。
AMTList		根据起始频率、终止频率和多音数以及 notch 表，添加频点到多音表。
ATLIst	<value>	:= {Min_Freq ~ 2 GHz}，单位 Hz。多音表添加一项频点。
DTLIst	<value>	:= {整数}。删除多音表指定序号的频点。
CTLIst		清空多音表的所有项。
STLIst		对多音表所有项的频点合并重复项，从小到大排序。
MTLIst	<index,	:<index>= {整数}，多音表的频点序

	value>	号。:<value>= {Min_Freq ~ 2 GHz 的浮点数}，多音表频点的频率值，单位 Hz。根据序号设置多音表频点的频率。
TLISt		查询多音表所有项。
NSTAte	<state>	:= {ON, OFF}。设置 Notch 表的开、关状态。
ANLIst	<value1, value2>	Value1/2:= {Min_Freq ~ 2 GHz}，单位 Hz。Notch 表添加一项。
DNLIst	<value>	:= {整数}。删除 Notch 表指定序号项。
CNLIst		清空 Notch 表，恢复默认值。
MNLIst	<index, value1, value2>	:<index>= {整数}，Notch 表的频点序号。:<value1>= {Min_Freq~2GHz 的浮点数}，Notch 表起始值的频率值，单位 Hz。:<value2>= {Min_Freq~2GHz 的浮点数}，Notch 表结束值的频率值，单位 Hz。指定序号设置 Notch 表的频率。
NLIst		查询 Notch 表所有项。

查询语法

<通道>:MTONE:<参数>?

示例

通道 1 设置多音数为 3:

C1:MTONE:TNUMber 3

通道 1 设置多音表序号 1 的频点为 6 MHz:

C1:MTONE:MTLIst 1,6e6

查询通道 1 的多音数:

C1:MTONE:TNUMber?

返回值:

"3\n"

查询通道 1 多音表所有项:

C1:MTONE:TLIst?

返回值:

"1,6000000.000000;2,10000.000000\n"

3.12 线性调频命令

描述	该命令用于设置或者获取线性调频功能的参数。
命令语法	<通道>:CHIRp:<参数> <值> <通道>:= {C1, C2, C3, C4}。 <值>:= {相关参数的值}。

参数	值	描述
STAte	<state>	:= {ON, OFF}。设置切换到线性调频功能。线性调频所有参数需要在此指令为“ON”状态下才可以设置。
RSTAte	<state>	:= {ON, OFF}。设置线性调频功能运行状态。前提是设置多音功能，线性调频所有参数需要在此指令为“OFF”状态下才可以设置。
SRATe	<value>	:= {100 Sa/s 到 5 GSa/s}。设置线性调频输出采样率。采样率>=2.5*输出的最大调频频率。
AMPlitude	<value>	:=幅度。单位是伏特，峰峰值"Vpp"。不同输出模式设置范围不一样，可在数据手册查阅参数值的有效范围。
OFFSet	<value>	:=偏移量。单位是"Vdc"。不同输出模式设置范围不一样，可在数据手册查阅参数值的有效范围。
HVTYpe	<type>	设置不输出线性调频时的输出电平。 = {LOW, MIDDLE, HIGH, ZERO}
TRIGger:SOURce	<type>	设置线性调频的触发源。 = {INT, MANA, MANB, EXTA, EXTB, TIMER}
TRIGger:TIMer	<value>	触发源为定时器下，设置定时器的时间。单位是秒。= {最小触发延时~10s}。
TRIGger:DELAy	<value>	触发源为手动或外部触发下，设置触发延时时间。单位是秒。= {最小触发延时~10s}。
TRIGger:SLOPe	<type>	触发源为外部触发下，设置触发沿。 = {RISe，FALL}。
ATList	<start_freq, end_freq, duration>	在线性调频表末尾增加一行。start_freq：起始频率；end_freq：结束频率；duration：扫描时间。

		start_freq 和 end_freq 单位是 Hz，duration 单位是秒。
ILIst	<index, start_freq, end_freq, duration>	在线性调频表选中的行（index）之前插入一行。index：选中的行；start_freq：起始频率；end_freq：结束频率；duration：扫描时间。index 从 1 开始，start_freq 和 end_freq 单位是 Hz，duration 单位是秒。
DLIst	<index>	:= {整数}。index 从 1 开始，删除线性调频表指定序号的行。
CLIst		清空线性调频表的所有行，只保留一行，并设为默认值
MLIst	<index, start_freq, end_freq, duration>	修改线性调频表选中的行（index）参数。index：选中的行；start_freq：起始频率；end_freq：结束频率；duration：扫描时间。index 从 1 开始，start_freq 和 end_freq 单位是 Hz，duration 单位是秒。
LIst		查询线性调频表所有项。

查询语法 <通道>:CHIRp:<参数>?

示例

通道 1 设置线性调频采样率为 3 Gsa/s：

C1:CHIRp:SRATe 3e9

通道 1 设置线性调频表序号 2 的起始频率为 20 kHz，结束频率为 1 kHz，扫描时间为 100 us：

C1:CHIRp:MLIst 2,2e4,1e3,1e-4

查询通道 1 线性调频的采样率：

C1:CHIRp:SRATe?

返回值：

"3000000000\n"

查询通道 1 线性调频表所有项：

C1:CHIRp:LIst?

返回值：

"1,10000000.000000,100000000.000000,0.000100;2,20000.000000,1000.000000,0.000100\n"

备注：手动触发指令复用 AWG 的指令，详见<channel>:AWG:TRIGgerA3.9.29 和<channel>:AWG:TRIGgerB3.9.30

3.13 多脉冲命令

描述 该命令用于设置或者获取多脉冲功能的参数。

命令语法 <通道>:MPULse:<参数> <值>

<通道>:= {C1, C2, C3, C4}。

<值>:= {相关参数的值}。

参数	值	描述
STAte	<state>	:= {ON, OFF}。设置切换到多脉冲功能。前提是设置 AWG 模式，多脉冲所有参数需要在此指令为“ON”状态下才可以设置。
SRate:TYPe	<type>	: = {AUTO, CUSTOM}。设置多脉冲的采样率类型。
SRate	<value>	: ={100Sa/S 到 5GSa/s}。设置采样率类型为“自定义（CUSTOM）”时，设置采样率。
RSTAte	<state>	:= {ON, OFF}。设置多脉冲功能运行状态。前提是设置多脉冲功能，多脉冲所有参数需要在此指令为“OFF”状态下才可以设置。
PNUMber	<value>	:= {2 到 30 的整数}。设置脉冲个数。
HLEVel	<value>	:= {浮点数}。设置脉冲的高电平。不同输出模式的设置范围不一样，具体见产品数据手册。
LLEVel	<value>	:= {浮点数}。设置脉冲的低电平。不同输出模式的设置范围不一样，具体见产品数据手册。
HVTYpe	<type>	:={LOW, MIDDLE, HIGH, ZERO}，设置多脉冲的空闲电平。
TRIGger:SOURce	<str>	:= {INT, MANA, MANB, EXTA, EXTB, TIMER}，设置多脉冲的触发源。
TRIGger:TIMer	<value>	:= {min~10 秒的浮点数}，设置多脉冲定时器的时间。
TRIGger:DELAy	<value>	:= {min~10 秒的浮点数}，触发源设置为手动或外部时，设置触发延时。
TRIGger:SLOPe	<value>	:= {RISe, FALL}。触发源设置外部时，设置触发沿。
MPULse	<num,width, dap>	:num= {脉冲序号}。:width= {脉冲宽度}。:gap= {脉冲间隙}。 根据脉冲序号设置每个脉冲的宽度和间隙。

查询语法	<通道>:MPULse:<参数>? 注：查询所有脉冲的宽度和间隙用下面指令 <通道>:MPULse:PULse?
示例	通道 1 设置脉冲数为 3： *C1:MPULse:PNUMber 3* 通道 1 设置序号 2 的脉冲宽度为 6 ms，间隙为 9 ms： *C1:MPULse MPULse 2,0.006,0.009* 查询通道 1 的脉冲数： *C1:MPULse:PNUMber?* 返回值： *“3\n”* 查询通道 1 的所有脉冲： *C1:MPULse:PULse?* 返回值： *“1,0.000001,0.000004;2,0.006,0.009\n”*

备注：手动触发指令复用 AWG 的指令，详见 <channel>:AWG:TRIGgerA 3.9.29 和 <channel>:AWG:TRIGgerB 3.9.30。

3.14 高速串行命令

描述	该命令用于设置或者获取高速串行功能的参数。
命令语法	<通道>:HSS:<参数> <值> <通道>:= {C1, C2, C3, C4}。 <值>:= {相关参数的值}。

参数	值	描述
SSTAte	<state>	:= {ON, OFF}。设置切换到高速串行功能。所有参数需要在此指令为“ON”状态下才可以设置。
SRAte	<value>	: ={100Sa/S 到 5GSa/s}。设置采样率。
DRATe	<value>	:= {500Kbps 到 2.5Gbps }。设置信号输出数据率。
AMPL	<value>	设置信号幅度。
RSTAte	<state>	:= {ON, OFF}。设置高速串行的运行状态，其他所有指令需要在运行状态为 OFF 下才能设置。
PATTern	<type>	:={CLOCK,ZERO,ONE,PRBS,CUSTOM_PRBS,CUSTOM}。设置数据模式。
PATTern:CUSTom	<type,value>	数据模式为 CUSTOM 下设置数据。type= {BIN,HEX}，value 为 type 类型下对应的数据。BIN 代表二进制数据；HEX 代表十六进制数据。
PRBS:LENGth	<value>	: = {3 到 31}，整数。数据模式为 PRBS 下设置 PRBS 的长度。
PRBS:CUSTom	<”xa+xb”,c>	数据模式为 CUSTOM_PRBS 下设置 PRBS 的多项式和种子，多项式需要加双引号。a 为多项式的高项系数，b 为低项系数，c 为种子。
BINVert	<state>	:= {ON, OFF}。bit 位翻转的使能开关。
B810:STATe	<state>	:= {ON, OFF}。8b/10b 使能开关。
B810:TYPE	<type>	:= {MINUS, PLUS}。设置 8b/10b 的平衡方式，MINUS 是 RD-，PLUS 是 RD+。
SCRAmbling:STATe	<state>	:= {ON, OFF}。加扰的使能开关。
SCRAmbling:FORMula	<”xa+xb”,c>	使能加扰后，设置加扰的多项式。多项式需要加双引号。a 为多项式的高项系数，b 为低项系数，c 为种子。
SCRAmbling:	<type>	:= {ADD, MULTI}。设置加扰方式。ADD

TYPE		是相加，MULTI 是相乘。
SIGNal	<type>	:={NRZ, NRZ_I, PAM4, PAM8, PAM16}。设置编码/调制方式。
PAMData	<type,value>	设置 PAM 调制方式下的数据。type={PAM4, PAM8, PAM16}，value= 4 个或 8 个或 16 个介于±1 之间的值，中间以逗号分隔，如-1,-0.333,0.333,1。
PWM:STATe	<state>	:={ON, OFF}。设置 PWM 状态。
PWM:WIDTh	<value>	设置 PWM 脉宽。单位是 UI。
EDGE:STATe	<state>	:={ON,OFF}。设置沿开关状态。
EDGE:RTIme	<value>	设置沿的上升时间。单位是 UI。
EDGE:FTIme	<value>	设置沿的下降时间。单位是 UI。
EDGE:RSHape	<type>	:= {LINEAR, FIRST_ORDER}。设置上升沿的类型。
EDGE:FSHape	<type>	:= {LINEAR, FIRST_ORDER}。设置下降沿的类型。
EDGE:RANGe	<type>	:={RANGE_20_80, RANGE_10_90}。设置沿的范围类型
DCD:STATe	<state>	:= {ON, OFF}。设置占空比失真的开关。
DCD:DATA	<value>	:={0.01 到 0.5}。设置占空比失真值，单位是 UI。
PJITter:STATe	<state>	:={ON, OFF}。设置周期抖动的开关。
PJITter:ADD		增加一个周期抖动
PJITter:DEL	<value>	:= {1 到 4 的整数}。Value 是周期抖动的 ID 号。删除指定的一个周期抖动。
PJITter:SET	ID,index,MAG,value1,PHASE,value2,FREQ,value3	Index= {1 到 4 的整数}，是周期抖动的 ID 号；value1= {0.001~50}，抖动范围，单位 UI； value2= {0~360}，相位，单位度°； value3= {10 KHz~500 MHz}，频率，单位 Hz。
RJITter:STATe	<state>	:={ON, OFF}。设置随机抖动的开关。
RJITter:ADD		增加一个随机抖动。
RJITter:DEL	<value>	:={1 到 4}。Value 是随机抖动的 ID 号。删除指定的一个随机抖动。
RJITter:SET	ID,index,MAG,value1,BW,value2,CF,value3	Index= {1 到 4 的整数}，是随机抖动的 ID 号；value1={0.001~0.5}，抖动范围，单位 UI； value2= {1 mHz~2 GHz}，带宽，单位 Hz；

		value3= {0~12}，峰均比，单位 dB。
NOISe:STATe	<state>	:= {ON, OFF}。设置噪声叠加的开关。
NOISe:MAG	<value>	:= {0 到 1}。Value 是噪声幅度的范围，设置的是百分比。
NOISe:BW	<value>	:= {10 Hz 到 2 GHz}。Value 是噪声带宽，单位是 Hz。
NOISe:CF	<value>	:={-100 到 100}。Value 是噪声峰均比，单位是 dB。
EQUAlization:STATe	<state>	:= {ON, OFF}。设置均衡的开关。
EQUAlization:TAP<value1>	<value2>	Value1={1 到 5 的整数}；value2={-1 到 1 的浮点数}；设置的是均衡系数，五个 value1 设置的 value2 的绝对值相加不能大于 1。
SSC:STATe	<state>	:= {ON, OFF}。设置扩频时钟的开关。
SSC:SHAPe	<type>	:= {SINE, SQUARE, TRIANGLE}。设置扩频时钟的波形类型。
SSC:FREQ	<value>	:= {1 Hz 到 5 GHz}。设置扩频时钟频率，单位是 Hz。
SSC:DEV	<value>	:= {0 到 1000000 的整数}。设置扩频时钟的偏差，单位是 PPM。

查询语法

<通道>:HSS:<参数>?

<通道>:HSS:PAMData? <type>
type= {PAM4, PAM8, PAM16}。

<通道>:HSS:PJITter:SET? <IDx>
<通道>:HSS:RJITter:SET? <IDx>
x= {1 到 4 的整数}。

示例

通道 1 设置高速串行信号采样率为 1 GSa/s：
C1:HSS:SRAte 1e9

查询通道 1 的高速串行信号采样率：
C1:HSS:SRAte?
返回值：
“1000000000\n”

通道 1 设置 CUSTOM 数据模式下为二进制数据：
C1:HSS:PATTern:CUSTom BIN,01010011

查询通道 1 CUSTOM 数据模式下的数据：
C1:HSS:PATTern:CUSTom?
返回值：
“BIN,01010011\n”

通道 1 设置 CUSTOM_PRBS 数据模式下的多项式：
C1:HSS:PRBS:CUSTom “x5+x3”,4

查询通道 1CUSTOM_PRBS 数据模式下的多项式：
C1:HSS:PRBS:CUSTom?
返回值：
“Formula=X5+X3+1,Seed=4\n”

通道 1 设置 PAM8 调制下的数据：
C1:HSS:PAMData PAM8,-0.9,-0.6,-0.4,-0.1,0.1,0.4,0.6,0.9

查询通道 1 PAM8 调制下的数据：
C1:HSS:PAMData? PAM8
返回值：
“-0.900000;-0.600000;-0.400000;-0.100000;0.100000;0.400000;0.600000;0.900000;\n”

通道 1 设置序号 1 的周期抖动：
C1:HSS:PJITter:SET ID,1,MAG,0.003,PHASE,66,FREQ,66e6

查询通道 1 序号 1 的周期抖动：
C1:HSS:PJITter:SET? ID,1
返回值：
“MAG:0.003000,PHASE:66.000000,FREQ:66000000.000000\n”

通道 1 设置均衡的抽头 2 系数：
C1:HSS:EQUAlization:TAP2 0.3

查询通道 1 均衡抽头 2 的系数：
C1:HSS:EQUAlization:TAP2?
返回值：
“0.3\n”

3.15 IQ 命令

3.15.1 IQ#:WAVeinfo?

描述	该命令用于查询 IQ 的波形信息。
命令语法	IQ<index>:WAVeinfo? <index>:= {1, 2}。
查询语法	IQ<index>:WAVeinfo? <index>:= {1, 2}。
示例	查询 IQ1 的波形信息： *IQ1:WAVeinfo?* 返回值： *WAVE_INFO,SYMBOL_LENTH,1024,OVER_SAMPLING,4,MODULATION,2ASK,FILTER_TYPE,RootCosine,FILTER_ALPHA,0.35*

3.15.2 IQ#:CENTerfreq

描述	该命令设置或查询 I/Q 调制的中心频率。
命令语法	<通道>:CENTerfreq <中心频率><单位> <通道>:= {IQ1, IQ2}。 <中心频率>:= 中心频率。该参数的有效范围请查阅数据手册。 <单位>:= {Hz, kHz, MHz, GHz}。默认单位是赫兹“Hz”。
查询语法	<通道>:CENTerfreq? <通道>:={IQ1, IQ2}。
响应格式	<中心频率> (用 Hz 表示)
示例	设置第一路 IQ 的中心频率为 1kHz： *IQ1:CENTerfreq 1000Hz*

3.15.3 IQ#:SAMPlerate

描述	该命令设置或查询 I/Q 的采样率。
命令语法	<通道>:SAMPlerate <采样率><单位> <通道>:= {IQ1, IQ2}。 <采样率>:= 采样率。该参数的有效范围请查阅数据手册。 <单位>:= {Hz, kHz, MHz, GHz}。默认单位是赫兹“Hz”。

查询语法	<通道>:SAMPlerate? <通道>:= {IQ1, IQ2}。
响应格式	<采样率> (用 Hz 表示)
示例	设置第一路 IQ 的采样率为 100 kHz: *IQ1:SAMPlerate 100000* 或: *IQ1:SAMP 100kHz*

3.15.4 IQ#:SYMBolrate

描述	该命令用于设置 I/Q 的符号率。
命令语法	<通道>:SYMBolrate <符号率><单位> <通道>:= {IQ1, IQ2}。 <符号率>:= 符号率。该参数的有效范围请查阅数据手册。 <单位>:= {S/s, kS/s, MS/s}。默认单位是符号每秒“S/s”。
查询语法	<通道>:=SYMBolrate? <通道>:= {IQ1, IQ2}。
响应格式	<符号率> (使用 S/s 表示)
示例	设置第一路 IQ 的符号率为 1MS/s: *IQ1:SYMB 1MS/s*

3.15.5 IQ#:AMPLitude

描述	该命令设置 I/Q 幅度。
命令语法	<通道>:AMPLitude <幅值><单位> <通道>:= {IQ1, IQ2}。 <幅值>:= 幅度。该参数的有效范围请查阅数据手册。 <单位>:= {Vrms, mVrms, dBm}。默认单位是均方根“Vrms”。
查询语法	<通道>:AMPLitude? <通道>:= {IQ1, IQ2}。
响应格式	<幅值> (表示为 Vrms.)
示例	设置第一路 IQ 的幅值(sqrt(I2+Q2))为 0.2 Vrms: *IQ1:AMPL 0.2*

3.15.6 IQ#:IQADjustment:GAIN

描述	此命令在保持 IQ 复合状态下调节 I 与 Q 的比例。
命令语法	<通道>:IQADjustment:GAIN <增益比><单位> <通道>:= {IQ1, IQ2}。 <增益比>:= I 与 Q 的增益比。 <单位>:= {dB}。
查询语法	<通道>:IQADjustment:GAIN? <通道>:= {IQ1, IQ2}。
响应格式	<增益比> (用单位 dB 表示)
示例	设置第一路 IQ 的增益比例为 0.1dB: *IQ1:IQADjustment:GAIN 0.1*

3.15.7 IQ#:IQADjustment:IOFFset

描述	此命令调节 I 通道的偏移量。
命令语法	<通道>:IQADjustment:IOFFset <偏置><单位> <通道>: ={IQ1, IQ2}。 <偏置>:= I 的偏移量。 <单位>:= {V, mV, uV}。默认单位是伏特“V”。
查询语法	<通道>:IQADjustment:IOFFset? <通道>:= {IQ1, IQ2}。
响应格式	<偏置> (表示为 V.)
示例	设置第一路 IQ 的 I 路偏置为 1mV: *IQ1:IQADjustment:IOFFset 1mV*

3.15.8 IQ#:IQADjustment:QOFFset

描述	此命令调节 Q 通道的偏移量。
命令语法	<通道>:IQADjustment:QOFFset <偏置><单位> <通道>:={IQ1, IQ2}。 <偏置>:= Q 的偏移量。 <单位>:= {V, mV, uV}。默认单位是伏特“V”。
查询语法	<通道>:IQADjustment:QOFFset? <通道>:= {IQ1, IQ2}。

响应格式	<偏置>(用 V 表示)
示例	设置第一路 IQ 的 Q 路偏置为-1mV: *IQ1:IQAD:QOFF -0.001*

3.15.9 IQ#:IQADjustment:QSKew

描述	该命令通过增加或者减少 Q 的相位角来调节 I 和 Q 向量的相位角。
命令语法	<通道>:IQADjustment:QSKew <角度> <通道>:= {IQ1, IQ2}。 <角度>:= 角度。单位是度。
查询语法	<通道>:IQADjustment:QSKew? <通道>:= {IQ1, IQ2}。
响应格式	<角度>(使用度表示)
示例	设置第一路 IQ 的 Q 角度为 1°: *IQ1:IQADjustment:QSKew 1.0*

3.15.10 IQ#:TRIGger:SOURce

描述	该命令设置 I/Q 的触发源。
命令语法	<通道>:TRIGger:SOURce <触发源> <通道>:= {IQ1, IQ2}。 <触发源>:= {INT, EXTernalA, EXTernalB, MANualA, MANualB, TIMER}。
查询语法	<通道>:TRIGger:SOURce? <通道>:= {IQ1, IQ2}。
响应格式	<触发源>
示例	设置第一路 IQ 触发源为手动触发 A: *IQ1:TRIGger:SOURce MANA*

3.15.11 IQ#:TRIGger:SLOPe <type>

描述	该命令设置或查询 I/Q 的外部触发源的沿。
命令语法	<通道>:TRIGger:SLOPe <type> <通道>:= {IQ1, IQ2}。 <type>:= {RISE, FALL, BOTH}。
查询语法	<通道>:TRIGger:SLOPe? <通道>:= {IQ1, IQ2}。
响应格式	<触发源>
示例	设置第一路 IQ 外部触发源沿为下降沿： *IQ1:TRIGger:SLOPe FALL* 查询第一路 IQ 外部触发源的沿： *IQ1:TRIGger:SLOPe?* 返回值： *FALL\n*

3.15.12 IQ#:MANTriger<type>

描述	该命令用于设置 IQ 手动触发时触发。
命令语法	<通道>:MANTriger<type> <通道>:= {IQ1, IQ2}。 <type>:= {A, B}。
示例	设置 IQ1 手动触发 A： *IQ1:MANTrigerA*

3.15.13 IQ#:TRIGTimer <value>

描述	该命令设置或查询 I/Q 定时触发时的定时时间。
命令语法	<通道>:TRIGTimer <value> <通道>:= {IQ1, IQ2}。 <value>:= 定时时间。
查询语法	<通道>:TRIGTimer? <通道>:= {IQ1, IQ2}。
响应格式	<value>

示例	设置第一路 IQ 定时触发源时间为 0.01： *IQ1:TRIGTimer 0.01* 查询第一路 IQ 定时触发源时间： *IQ1:TRIGTimer?* 返回值： *0.01\n*

3.15.14 IQ#:WAVEload:BUILtin

描述	此命令用于从内建波形列表中选择 I/Q 波形。
命令语法	<通道>:WAVEload:BUILtin <波形名称> <通道>:= {IQ1, IQ2}。 <波形名称>:= {下表的波形名}。
查询语法	<通道>:WAVEload? <通道>:= {IQ1, IQ2}。
响应格式	BUILtin|USERstored <波形名称>
示例	设置第一路 IQ 的波形为内建波形 2ASK： *IQ1:WAVE:BUIL "2ASK"*

2ASK	4ASK	8ASK	BPSK	4PSK
8PSK	DBPSK	4DPSK	8DPSK	8QAM
16QAM	32QAM	64QAM	128QAM	256QAM
16QAM_2				

3.15.15 IQ#:WAVEload:USERstored

描述	此命令用于在用户存储波形中选择 I/Q 波形。
命令语法	<通道>:WAVEload:USERstored <路径> <通道>:= {IQ1, IQ2}。 <路径>:= {来自用户存储（本地，网络存储，U 盘）中的波形路径，包括文件名和后缀}。
查询语法	<通道>:WAVEload? <通道>:= {IQ1, IQ2}。

响应格式	BUILtin|USERstored <波形名称>
示例 1	设置第二路 IQ 的波形为用户本地存储波形 wave1.arb： *IQ2:WAVEload:USERstored "Local/EasyIQ_arb/DQPSK_UserIQ_1.arb"*
示例 2	设置第一路 IQ 的波形为网络存储波形 wave1.arb： *IQ1:WAVEload:USERstored "net_storage/wave/wave1.arb"*
示例 3	设置第一路 IQ 的波形为 U 盘存储波形 wave1.arb： *IQ1:WAVEload:USERstored "U-disk0/wave/wave1.arb"*

3.15.16 IQ#:MARKer1:POS# <state>

描述	此命令设置 I/Q 的标记开关状态。
命令语法	<通道>:MARKer<value1>:POS<value2> <state> <通道>:= {IQ1, IQ2}。 <value1>:={1, 2}。 <value2>= 标记序号，波形数据长度。 <state>:= {ON, OFF}。
查询语法	
响应格式	<state>
示例	设置标记 1 的第一个数据点打开： *IQ1:MARKer1:POS1 ON*

3.15.17 DACTYPe <type>

描述	此命令设置 I/Q 的时钟类型。
命令语法	DACTYPe <type> <type>:= {5G, 6G}。
查询语法	DACType?
响应格式	<type>
示例	设置时钟类型为 6G： *DACTYPe 6G*

备注：四通道机型需要四个通道都是 IQ 模式才可以切到 6G 时钟。

3.15.18 IQ#:COMpensation <state>

描述	该命令用于设置或查询 IQ 补偿的状态。
命令语法	<channel>:COMpensation <state> <channel>:= {IQ1, IQ2}。 <state>:= {ON, OFF}。
查询语法	<channel>:COMpensation? <channel>:= { IQ1, IQ2}。
示例	设置 IQ1 的补偿状态为开： *IQ1:COMpensation ON* 查询 IQ1 的补偿状态： *IQ1:COMpensation?* 返回值： *ON\n*

3.15.19 IQ#:MODE <type>

描述	该命令用于设置 IQ 与 IQ Sequence 的切换及当前模式的查询。
命令语法	<channel>:MODE <type> <channel>:= {IQ1, IQ2}。 <type>:= {NORMAL, AWG}。
查询语法	<channel>:MODE? <channel>:= { IQ1, IQ2}。
示例	设置 IQ1 的模式为 NORMAL： *IQ1:MODE NORMAL* 查询 IQ1 的模式： *IQ1:MODE?* 返回值： *IQ_NORMAL\n*

3.15.20 IQ Sequence

IQ Sequence 复用 AWG 的命令，见 3.9AWG 命令，设置相应参数前确保是在 IQ Sequence 模式。

例如，在 IQ Sequence 下，设置和查询 IQ 的采样率如下

描述	该命令用于设置(查询)AWG 的采样率。
命令语法	<channel>:AWG:SRATE <value> <channel>:= {C1, C2, C3, C4}。 <value>:= 采样率。
查询语法	<channel>:AWG:SRATe? <channel>:= {C1, C2, C3, C4}。
示例	设置通道 1 的采样率为 200M: *C1:AWG:SRATE 2e8* 查询通道 1 的采样率: *C1:AWG:SRATE?* 返回值: *200000000\n*

3.16 时钟命令

3.16.1 时钟源切换

描述	该命令设置或获取时钟源。
命令语法	ROSCillator <src> <src>:= {INT (内部时钟), EXT (外部时钟), SAMPEXT (外部采样时钟)}。
查询语法	ROSCillator?
示例	设置时钟源为内部时钟： *ROSCillator INT* 获取时钟源配置： *ROSCillator?* 返回值： *ROSC\sINT,10MOUT,OFF,AMPL,31dBm,SAMPOUT,OFF,AMPL,31dBm,SYNCOUT,OFF,AMPL,31dBm\n*

3.16.2 时钟输出设置

描述	该命令用于设置或者获取 10M 时钟、采样时钟、同步时钟的输出、输出幅度。
命令语法	ROSCillator <参数>,<值> <参数>:= {下表的参数}。 <值>:= {相关参数的值}。
查询语法	ROSCillator?
示例	打开 10M 时钟输出开关： *ROSCillator 10MOUT,ON* 设置 10M 时钟输出幅度为 20 dBm：

参数	值	描述
10MOUT	<state>	:= {ON, OFF}，打开或关闭 10M 时钟输出开关。
SAMPOUT	<state>	:= {ON, OFF}，打开或关闭采样时钟输出开关。
SYNCOUT	<state>	:= {ON, OFF}，打开或关闭同步时钟输出开关。
10MAMPL	<value>	:= {3 ~ 10}。单位是 dBm。10M 时钟输出幅度。
SAMPAMPL	<value>	:= {3 ~ 10}。单位是 dBm。采样时钟输出幅度。
SYNCAMPL	<value>	:= {3 ~ 10}。单位是 dBm。同步时钟输出幅度。

	ROSCillator 10MAMPL,20 获取时钟输出状态： *ROSCillator?* 返回值： *ROSC\sINT,10MOUT,ON,AMPL,20dBm,SAMPOUT,OFF,AMPL,31dBm,SYNCOUT,OFF,AMPL,31dBm\n*

3.17 蜂鸣器命令

描述	该命令用于打开或者关闭蜂鸣器。
命令语法	BUZZer <状态> <状态>:= {ON, OFF}。
查询语法	BUZZer?
响应格式	BUZZ <状态>
示例	打开蜂鸣器： *BUZZ ON*

3.18 屏幕保护命令

描述	该命令用于关闭或者设置屏幕保护时间（单位为分钟）。
命令语法	SCreen_SaVe <参数> <参数>:= {OFF, 1, 5, 15, 30, 60, 120, 300}。
查询语法	SCreen_SaVe?
响应格式	SCSV <参数>
示例	设置屏幕保护时间为 5 分钟： *SCSV 5* 读取当前的屏幕保护时间： *SCreen_SaVe?* 返回值： *SCSV 5MIN*

3.19 按键开关命令

描述	该命令用于打开或者关闭前面板按键。
命令语法	KEY <状态> <状态>:= {ON, OFF}。
查询语法	KEY?
响应格式	KEY <状态>
示例	打开前面板按键： *KEY ON*

3.20 语言切换命令

描述	该命令用于设置或者获取系统语言。
命令语法	LAnGuaGe <语言> <语言>:= {EN, CH}，其中 EN 是英语，CH 是中文。
查询语法	LAnGuaGe?
响应格式	LAGG <语言>
示例	设置语言为英文： *LAGG EN* 读取当前的系统语言： *LAGG?* 返回值： *LAGG EN\n*

3.21 日期和时间命令

描述	该命令用于设置设备的日期和时间。
命令语法	SYST:DATE <日期> <日期>:= {要设置的日期，格式：yyyy/mm/dd}。 SYST:TIME <时间> <时间>:= {要设置的时间，格式：hh/mm/ss}。
查询语法	SYST:DATE?

	SYST:TIME?
示例	设置日期为 2021/01/10： *SYST:DATE 20210110* 设置时间为 10:06:32： *SYST:TIME 100632*

3.22 屏幕截图

描述	该命令用于设置屏幕截图及格式设置。
命令语法 1	设置屏幕截图 SCREEN_SHOT
命令语法 2	设置或查询屏幕截图格式 SCREEN_SHOT_TYPE <type> <type>:= {BMP, PNG}。
查询语法	SCREEN_SHOT_TYPE?
示例	截当前 UI 界面图： *SCREEN_SHOT* 设置屏幕截图格式为 BMP： *SCREEN_SHOT_TYPE BMP* 查询屏幕截图格式： *SCREEN_SHOT_TYPE?* 返回值： *BMP\n*

3.23 文件管理命令

3.23.1 MMEMory:DELete

描述	该命令用于删除指定文件。
命令语法	MMEMory:DELete <参数> <参数>:= 文件路径（包含操作文件名称）。
示例	删除路径为”Local/1000pts.bin”的文件： *MMEMory:DELete “Local/1000pts.bin”*

3.23.2 MMEMory:RDIRectory

描述	该命令用于删除指定文件夹。
命令语法	MMEMory:RDIRectory <参数> <参数>:= 文件路径（包含操作文件夹名称）。
示例	删除文件夹路径为“Local/”下名为 test 的文件夹： *MMEMory:RDIRectory “Local/test”*

3.23.3 MMEMory:MDIRectory

描述	该命令用于新建指定路径的文件夹。
命令语法	MMEMory:MDIRectory <参数> <参数>:= 文件路径（包含操作文件夹名称）。
示例	新建一个“Local”路径下的名为 test 的文件夹： *MMEMory:MDIRectory “Local/test”*

3.23.4 MMEMory:CATalog

描述	该命令用于查询指定路径下的全部文件和文件夹或查看指定格式文件。
查询语法	查看路径下的所有文件和文件夹 MMEMory:CATalog? <参数> <参数>:=文件夹路径 MMEMory:CATalog: <参数 1>?<参数 2>

	<参数 1>:= {(DATA:ARBitrary), (STATe:XMLanguage)}。 <参数 2>:= 文件夹路径。
响应格式	剩余内存大小，已用内存大小 “文件名，文件类型，文件大小”
示例	查看“Local/”路径下的所有文件和文件夹： *MMEMory:CATalog? “Local”* 查看“Local/”路径下的“.arb”或“.ARB”后缀的文件： *MMEMory:CATalog:DATA:ARBitrary? “Local”* 查看“Local/”路径下的“.xml”或“.XML”后缀文件： *MMEMory:CATalog:STATe:XMLanguage? “Local”*

3.23.5 MMEMory:COPY

描述	此命令复制一个文件或文件夹。
命令格式	MMEMory:COPY <参数> <参数>:= “源文件的路径”，“目标路径”。 复制文件：源文件路径和目标路径均包含操作文件名称。 复制文件夹：源文件路径包含操作文件夹和目标路径不包含操作文件名称。
示例	复制路径为“Local/test/1000pts.bin”的文件并粘贴至 “Local/1000pts.bin”： *MMEMory:COPY “Local/test/1000pts.bin”,”Local/1000pts.bin”* 复制路径为”Local/src”的文件/文件夹并粘贴至“Local/copy/”文件夹中： *MMEMory:COPY “Local/src”, ”Local/copy”*

3.23.6 MMEMory:MOVE

描述	此命令移动一个文件或文件夹到新的路径下。
命令语法	MMEMory:MOVE <参数> <参数>:= “源文件/文件夹的路径”，“目标路径”。 移动文件：源文件路径和目标路径均包含操作文件名称 移动文件夹：源文件路径包含操作文件夹和目标路径不包含操作文件名称
示例	移动文件路径为”Local/test/1000pts.bin” 的文件到路径“Local/1000pts.bin”： *MMEMory:MOVE “Local/test/1000pts.bin”,”Local/1000pts.bin”*

	移动文件夹路径为"Local/src" 到文件夹路径为 "Local/copy/"下： *MMEMory:MOVE "Local/src", "Local/copy/"*

3.23.7 MMEMory:SAVE:XML

描述	此命令保存一个 xml 配置文件到默认路径或指定路径。
命令语法	MMEMory:SAVE:XML <参数> <参数>:= {保存路径，包括文件名和后缀}。
示例	保存 test.xml 文件到本地： *MMEMory:SAVE:XML "Local/test.xml"*或 *MMEMory:SAVE:XML "test.xml"* 保存 test.xml 文件到网路存储盘： *MMEMory:SAVE:XML "net_storage/test.xml"* 保存 test.xml 文件到 U 盘： *MMEMory:SAVE:XML "U-disk0/test.xml"*

3.23.8 MMEMory:LOAD:XML

描述	此命令从默认路径或指定路径加载一个 xml 配置文件。
命令语法	MMEMory:LOAD:XML <参数> <参数>:= {路径，包括文件名和后缀}。
示例	从本地加载 test.xml 文件： *MMEMory:LOAD:XML "Local/test.xml"*或 *MMEMory:LOAD:XML "test.xml"* 从网路存储盘加载 test.xml 文件： *MMEMory:LOAD:XML "net_storage/test.xml"* 从 U 盘加载 test.xml 文件： *MMEMory:LOAD:XML "U-disk0/test.xml"*

3.23.9 MMEMory:TRANsfer

描述	此命令可发送自定义的数据到指定路径下指定的 bin 文件。
命令语法	MMEMory:TRANsfer <参数>,#{data} <参数>:= {保存路径，包括路径、文件名及后缀}。 {data}:= 数据的长度的长度+数据长度+二进制数据。
示例	发送数据长度的长度为 1，数据长度为 4 的数据到本地下的 wave1.bin： *MMEMory:TRANsfer "Local/wave1.bin",#14ABCD* 发送数据长度的长度为 1，数据长度为 4 的数据到 U 盘下的 wave1.bin： *MMEMory:TRANsfer "U-disk0/wave1.bin",#14ABCD* 发送数据长度的长度为 1，数据长度为 4 的数据到网络存储盘下的 wave1.bin： *MMEMory:TRANsfer "net_storage/wave1.bin",#14ABCD*

3.24 IP 命令

描述	该参数用于设置或者读取设备的 IP 地址。
命令语法	SYSTem:COMMunicate:LAN<网口>:IPADdress <参数 1>.<参数 2>.<参数 3>.<参数 4> <网口>:= [1, 2]。 <参数 1>:= {在 1 至 223 之间的整数值}。 <参数 2>:= {在 0 至 255 之间的整数值}。 <参数 3>:= {在 0 至 255 之间的整数值}。 <参数 4>:= {在 0 至 255 之间的整数值}。
查询语法	SYSTem:COMMunicate:LAN1:IPADdress?
示例	设置 IP 地址为 10.11.13.203： *SYST:COMM:LAN1:IPAD "10.11.13.203"* 读取 IP 地址： *SYST:COMM:LAN1:IPAD?* 返回值： *"10.11.13.203"*

3.25 子网掩码命令

描述	该命令用于设置或者获取设备的子网掩码。
命令语法	SYSTem:COMMunicate:LAN<网口>:SMASk <参数 1>.<参数 2>.<参数 3>.<参数 4> <网口>:= [1, 2]。 <参数 1>:= {在 0 至 255 之间的整数值}。 <参数 2>:= {在 0 至 255 之间的整数值}。 <参数 3>:= {在 0 至 255 之间的整数值}。 <参数 4>:= {在 0 至 255 之间的整数值}。
查询语法	SYSTem:COMMunicate:LAN1:SMASk?
示例	设置子网掩码为 255.0.0.0： *SYST:COMM:LAN1:SMAS "255.0.0.0"* 获取子网掩码： *SYST:COMM:LAN1:SMAS?*

	返回值： *“255.0.0.0”*

3.26 网关命令

描述	该命令设置并获取设备的网关。
命令语法	SYSTem:COMMunicate:LAN<网口>:GATeway <参数 1>.<参数 2>.<参数 3>.<参数 4> <网口>:= [1, 2]。 <参数 1>:= {在 0 至 223 之间的整数值}。 <参数 2>:= {在 0 至 255 之间的整数值}。 <参数 3>:= {在 0 至 255 之间的整数值}。 <参数 4>:= {在 0 至 255 之间的整数值}。
查询语法	SYSTem:COMMunicate:LAN1:GATeway?
示例	设置网关为 10.11.13.5： *SYSTem:COMMunicate:LAN1:GATeway “10.11.13.1”* 获取网关： *SYSTem:COMMunicate:LAN1:GATeway?* 返回值： *“10.11.13.1”*

3.27 同步命令

描述	该命令用于设置或者获取通道的同步输出。
命令语法	<channel>:SYNC <state> <channel>:= {C1, C2, C3, C4}。 <state>:= {ON, OFF}。 <channel>:SYNC TYPE, <type> <channel>:= {C1, C2, C3, C4}。 <type>:= {CH, MOD_CH}。
查询语法	<channel>:SYNC? <channel>:= {C1, C2, C3, C4}。
示例	打开通道 1 同步信号输出功能：

C1:SYNC ON

通道 1 输出 CH1 同步信号：
C1:SYNC TYPE, CH

读取通道 1 同步功能状态：
C1:SYNC?
返回值：
C1:SYNC ON,TYPE,CH

3.28 输入信号设置命令

描述	该命令用于设置或者输入信号
命令语法	格式 1：SYSTEM:<参数 1> <值> 格式 2：SYSTEM:<参数 1> <参数 2>,<值> <参数 1>:= {下表参数 1}。 <参数 2>:= {下表参数 2}。 <值>:= {下表相关的值}。

参数 1	参数 2	值	描述
SYNCLEVEL		<value>	:= {-5 ~ 5}，单位伏特“V”。同步信号电平。格式 1
SYNCLOAD		<value>	:= {50, 10K}，单位欧姆“Ω”。同步信号负载。格式 1
TRIGLEVEL	EXTA	<value>	:= {-5 ~ 5}，单位伏特“V”。外部触发 A 信号电平。格式 2
TRIGLEVEL	EXTB	<value>	:= {-5 ~ 5}，单位伏特“V”。外部触发 B 信号电平。格式 2
TRIGLOAD	EXTA	<value>	:= {50, 10K}，单位欧姆“Ω”。外部触发 A 信号负载。格式 2
TRIGLOAD	EXTB	<value>	:= {50, 10K}，单位欧姆“Ω”。外部触发 B 信号负载。格式 2

查询语法	SYSTEM:<参数 1>? <参数 1>:= {上表参数 1}。
示例	设置同步信号负载为 10KΩ： *SYSTEM:SYNCLOAD 10K* 设置外部触发 B 电平为 1V：

	SYSTEM:TRIGLEVEL EXTB,1 读取同步信号电平： *SYSTEM:SYNCLEVEL?* 返回值： *0\n* 读取外部触发信号电平： *SYSTEM:TRIGLEVEL?* 返回值： *EXTA_LEVEL:0.000000,EXTB_LEVEL:1.200000\n*

3.29 动态跳转有效沿命令

描述	该命令用于设置或者获取动态跳转有效沿。
命令语法	SYSTEM:PATTERNEDGE <type> <type>:= {RISE, FALL}。
查询语法	SYSTEM:PATTERNEDGE?
示例	设置动态跳转有效沿为下降沿： *SYSTEM:PATTERNEDGE FALL* 读取动态跳转有效沿状态： *SYSTEM:PATTERNEDGE?* 返回值： *FALL\n*

3.30 GPIB 命令

描述	该命令设置或查询 GPIB 端口号。
命令语法	SYSTem:COMMunicate:GPIB:ADDRess <value> <value>:= {1-30}。
查询语法	SYSTem:COMMunicate:GPIB:ADDRess?
示例	设置当前 GPIB 端口号： *SYSTem:COMMunicate:GPIB:ADDRess 20* 查询当前 GPIB 端口号：

	SYSTem:COMMunicate:GPIB:ADDRess? 返回值： *18*

3.31 多设备同步命令

描述	该命令设置两个或多个仪器之间的同步，并实现同相输出。
命令语法	CASCADE STATE,ON|OFF,MODE,<MODE>,DELAY,<DELAY> <MODE>:= {MASTER, SLAVE}。 <DELAY>:= {0-0.000025}，单位= s，此参数只能在从机模式时设置。
查询语法	CASCADE?
响应格式	从机模式返回值： CASCADE STATE,ON,MODE,SLAVE,DELAY,<DELAY> 主机模式返回值： CASCADE STATE,ON,MODE,MASTER
示例	设置设备为从机模式且延迟为 0.0000001s： *CASCADE STATE,ON,MODE,SLAVE,DELAY,0.0000001*

3.32 开机预设命令

描述	设置或者获取上电开机模式。
命令语法	格式 1：Sys_CFG <模式> <模式>:= {DEFAULT（默认），LAST（上次），USER（用户）}。 格式 2：Sys_CFG <配置>,<路径> <配置>:= {USER（选择开机恢复文件），PRESET（选择恢复文件）}。 <路径>:= {用户存储（本地，网络存储，U 盘）的配置文件的路径，包括文件名和后缀}。
查询语法	Sys_CFG?
响应格式	SCFG <模式>
示例 1	设置上电开机系统为上次： *SCFG LAST*
示例 2	设置开机恢复文件： *SCFG USER,"net_storage/config/state.xml"*

	或：*SCFG USER,“U-disk0/config/state.xml”* 或：*SCFG USER,“Local/state.xml”*
示例 3	设置恢复文件 *SCFG PRESET,” net_storage/config/state.xml”* 或：*SCFG PRESET,”U-disk0/config/state.xml”* 或：*SCFG PRESET,"Local/ state.xml"*

备注 1：路径必须使用双引号，英文包括且必须添加后缀名 .xml。具体可用路径请参考文件管理器。

4 波形格式

本章节给出信号源使用到的波形文件格式的说明。通过这些说明，你可以了解如何自定义编辑波形文件，并结合上个章节列出的命令实现对信号源控制。

4.1 bin

bin 文件内容为二进制，文件内容就是各个点的码字值（码字范围-32768~32767），不支持手动编辑，信号源导入文件时，保持当前的幅度，频率、偏移信息，直接将各码字值转换为电压输出。

4.2 csv/dat

csv 与 dat 文件内容为 text。CSV 文件分为头部信息段与波形数据段两部分。

1. csv 头部信息段内容包含以下四项，可以有多的信息项，但必须包含以下四项，多的项会被忽略。每项一行，以换行符结束。

信息项	描述
amp,value	波形的幅度
offset,value	波形的偏移
frequency,value	波形的频率
data length,value	波形的长度

2. 数据段数据每一组占一行。可用以下四种字符串之一作为起始标识（标识符占一行），放在正式数据之前。

   Second,Volt

   xpos,value

   Time,Ampl

   Second,Value

   csv 格式数据示例图如下：

1	data length	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	

csv 文件去掉头部信息段，只保留数据段，就是.dat 格式

dat 格式数据示例图如下：

```
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
```

4.3 mat

1. mat 文件格式说明

MAT 文件由一个格式固定的 128 字节的文件头部信息（mat_head），和两个数据单元组成

a) 文件头部用如下结构体表示，此处只需关注 endian_indicator：

```
typedef struct
{
  char descriotive[116];
  char data_offset[8];
  uint16 version;              //导出 mat 文件的软件版本
    char endian_indicator[2];      //值为 IM 或者 MI，表示大小端模式
```

```
}cfg_mat_header_t;
```

b) 每个数据单元起始处有一个数据头部信息（data_head，为 88 字节），用于说明数据单元的占用的字节数、数据类型，数据块名字等信息。

数据块头部用如下结构体表示：

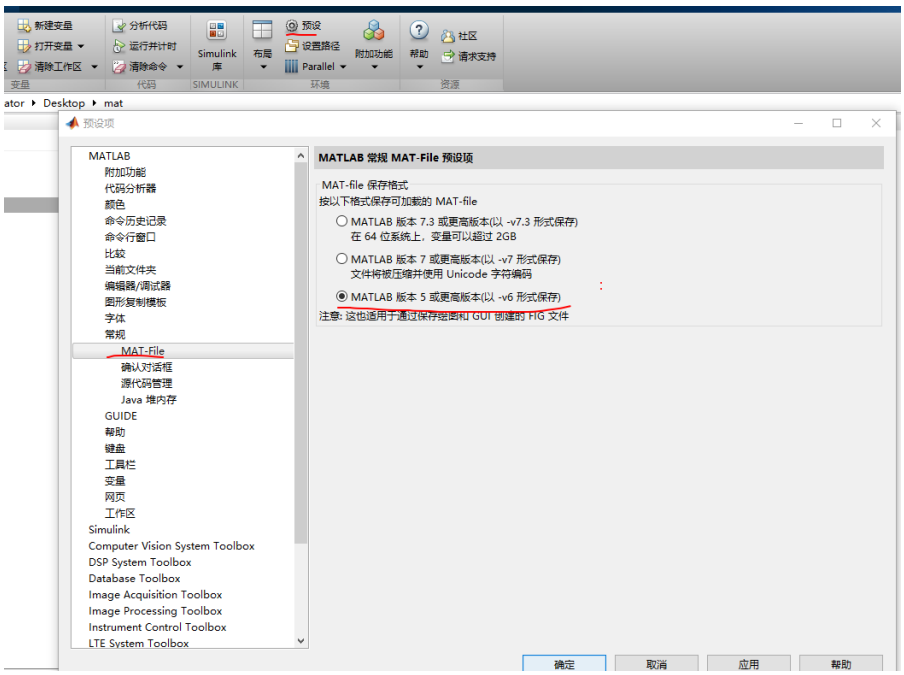
```
typedef struct
{
    u32 data_type;          //值必须为 CFG_MI_MATRIX
    u32 array_len;
    u32 array_flag_data_type;
    u32 array_flag_data_len;
    u32 array_flag_data_1;
    u32 array_flag_data_2;
    u32 dimensions_array_data_type;
    u32 dimensions_array_data_len;
    u32 dimensions_array_data_1;
    u32 dimensions_array_data_2;
    u32 array_name_data_type;
    u32 array_name_data_len;
    char array_name_data[32];  //示波器导出文件此处为 XX_time 或者 XX_data
    u32 data_tag_data_type;    //后边数据区数据的类型
    u32 data_tag_data_len;     //后边数据区数据的大小
}matlab_data_head_t;
```

其中 XX_data_type 表示数据类型，对应到如下枚举值：

```
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;
```

2. matlab 导出的.mat 文件默认为压缩格式，需按以下方式设置为非压缩方式保存。并且只支持两个数据段的文件。时间数据需用 double，波形数据需用 single。



4.4 hop

hop 文件是用于存储跳频序列的文件。文件内分为三个数据块存储三个表格，分别存储：跳频频率表，频率跳转表，频率过滤表。文件内容为 text，可以手动编辑。

Hop 文件有三个数据块，分行存储，每一行存储一个参数或数据。

第一行为文件版本号：**Ver:1.0**

第二行为跳频频率表版本号：**FreqListVer:1.0**

第三行为频率表的数据起始标记：**freq_list_start**

下来是频率表的数据

最后频率表以结束标记结束：**freq_list_end**

接下来是频率跳转表、频率过滤表，结构和频率表一样，只是关键字不同，如下：

OrderListVer:1.0 -- 频率跳转表版本号

order_list_start -- 频率跳转表数据起始标记

order_list_end -- 频率跳转表数据结束标记

AvoidListVer:1.0 -- 频率过滤表版本号

avoid_list_start -- 频率过滤表数据起始标记

avoid_list_end -- 频率过滤表数据结束标记

数据块中每行两个数据以逗号分割。

数据示例：

```
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
```

4.5 wav/arb

wav/arb 文件为 IQ 波形文件。wav/arb 文件由头部信息（header）与波形数据（data）两个部分组成。header 是 text 数据。波形数据是二进制数据。

header 必须以字符串“IQData,”结尾。从文件起始到字符串“IQData,”，是 text 格式的文件信息。字符串“IQData,”之后就是二进制的波形数据。

加载 wav/arb 文件时，会从 heade 中读取以下关键字（key_str）进行解析。如果下面某些关键字在 header 中不存在，对应信息会被设为默认值或置空。如果有多余的信息，会被忽略。”key_str,value”的形式构成一组描述信息，每组描述信息之间以逗号分割。以下是一个 wav/arb 文件的 header 实例，及解析的各关键字含义。

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,

FileType 文件类型（IQ）

Version 版本号

DataSourceType

SymbolLength 符号长度

SymbolRate 符号率

ModulationType 调制类型

FilterType 滤波器类型

FilterBandwidth

FilterAlpha 滚降系数

FilterLength 滤波器长度

OverSampling 过采样点

ActualSampleLength

SampleRate 采样率

DataLength

RMS

IQData

5 编程示例

本章节给出了一些编程示例。通过这些例子，你可以了解如何使用 VISA 或者 sockets，并结合上节列出的命令实现对信号源控制。通过下面的例子，你可以开发更多应用。

5.1 VISA 应用示例 VC++示例

环境： Win7 32 位系统, Visual Studio

描述： 分别通过 USBTMC 和 TCP/IP 访问信号源，并在 NI-VISA 上发送“*IDN?”命令来查询设备信息。

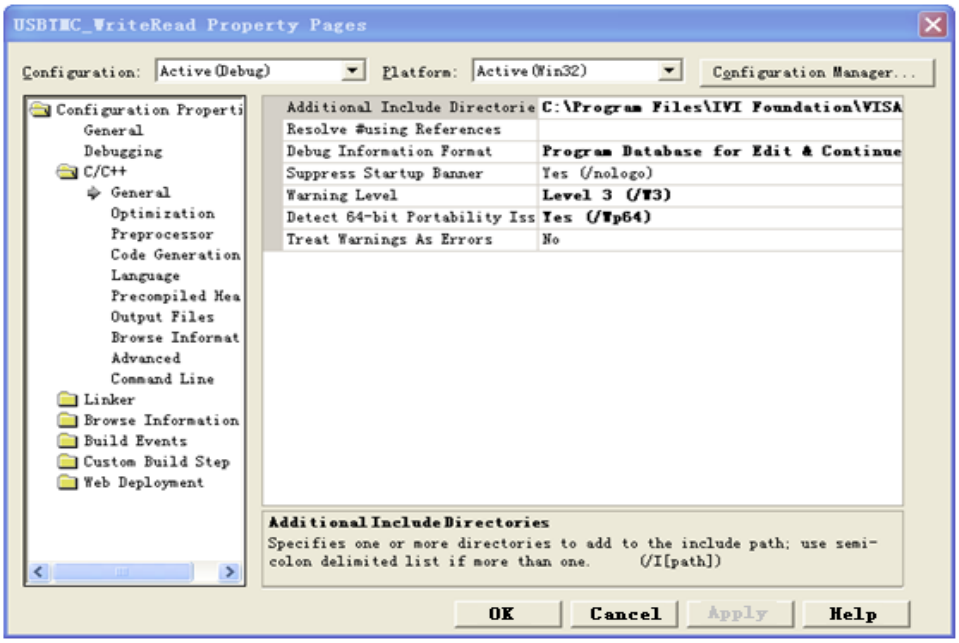
步骤：

1. 打开 Visual Studio 软件, 新建一个 VC++ win32 console project。
2. 设置调用 NI-VISA 库的项目环境。此处给出两种设置方法，分别为静态和动态：
   a) 静态：
   b) 在 NI-VISA 安装路径找: visa.h、visatype.h、 visa32.lib 文件，将它们复制到 VC++项目的根路径下并添加到项目中。在 projectname.cpp 文件上添加下列两行代码：

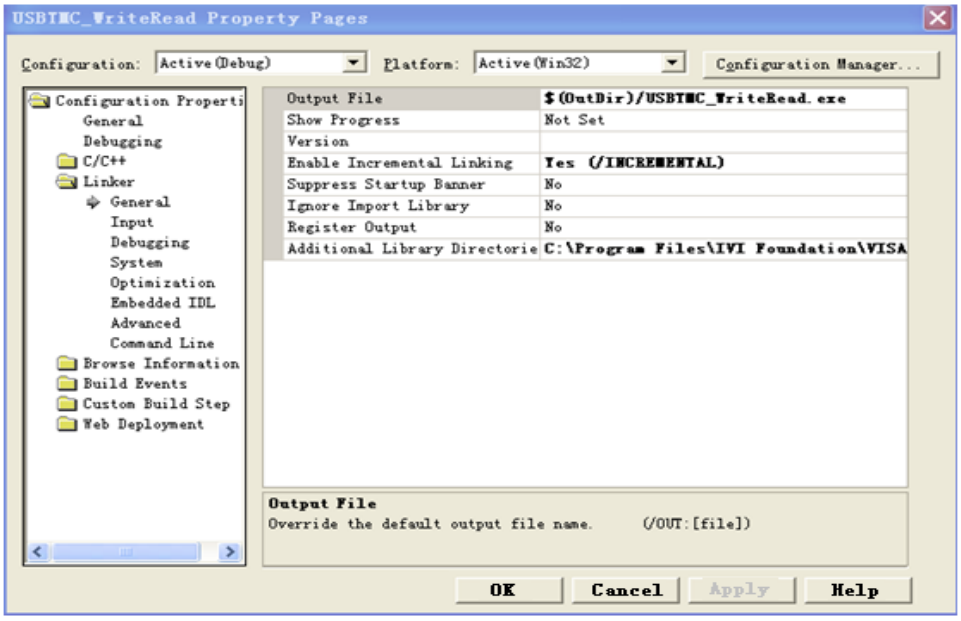
```
#include "visa.h"
    #pragma comment(lib,"visa32.lib")
```

   c) 动态：

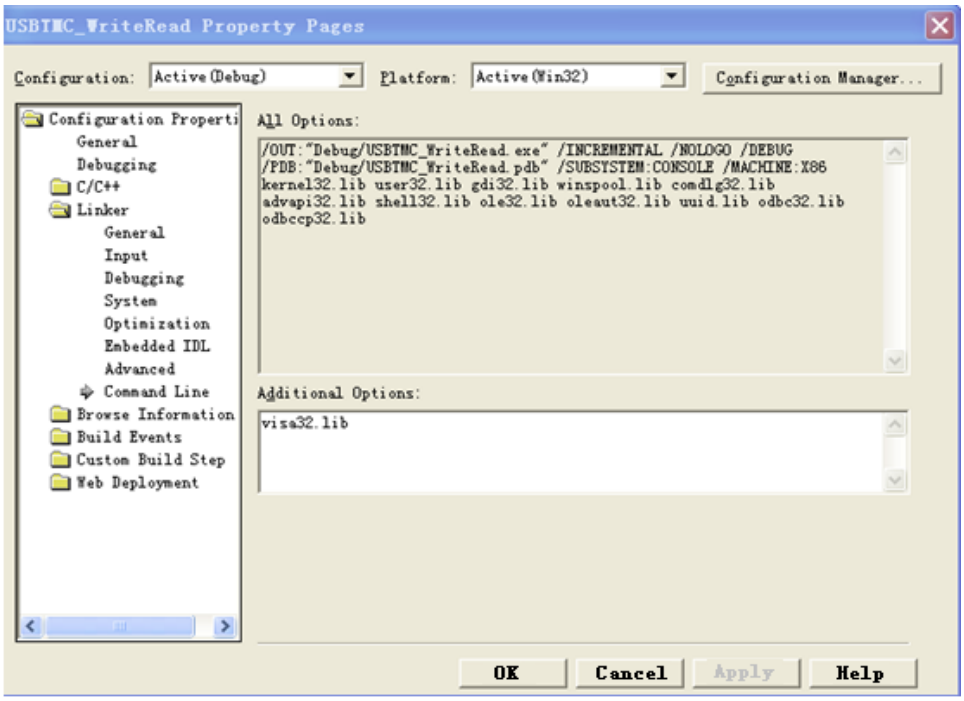
点击"project>>properties"，在属性对话框左侧选择”c/c++---General”中，将“Additional Include Directories”项的值设置为 NI-VISA 的安装路径（例如：C:\Program Files\IVI Foundation\VISA\WinNT\include），如下图所示：



在属性对话框左侧选择" Linker---General " ,并将“Additional Library Directories”项的值设置为 NI-VISA 的安装路径。(例如：C:\Program Files\IVI Foundation\VISA\WinNT\include)，如下图所示：



在属性对话框左侧选择"Linker---Command Line" ， 将“Additional”项的值设置为 visa32.lib，如下图所示：



在 projectname.cpp 文件上添加 visa.h 文件：

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

3. 编码：

a) USBTMC:

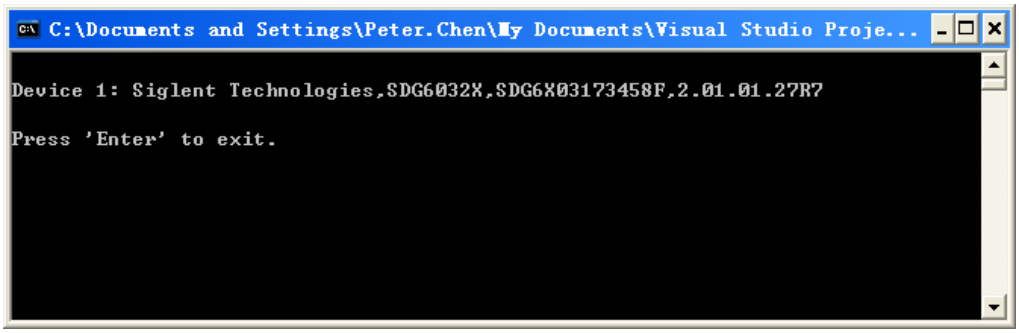
```
int Usbtmc_test()
{
    /* This code demonstrates sending synchronous read & write commands */
    /* to an USB Test & Measurement Class (USBTMC) instrument using    */
    /* NI-VISA                                                         */
    /* The example writes the "*IDN?\n" string to all 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;
        charinstrResourceString[VI_FIND_BUFLEN];
        unsigned      charbuffer[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;
}
```

运行结果：

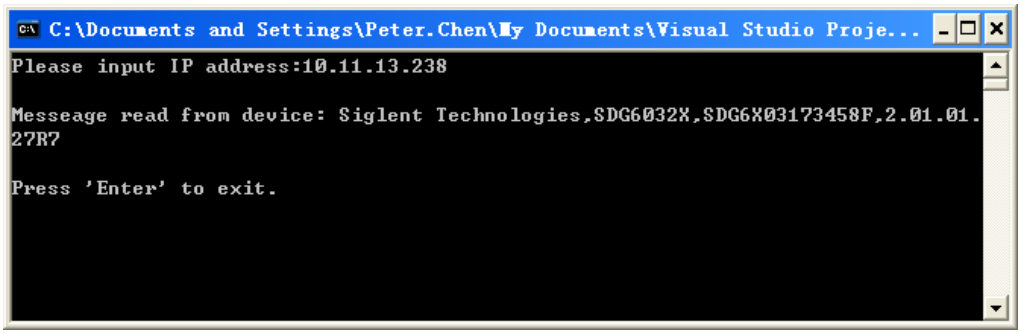


b) TCP/IP:

```
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;
}
```

运行结果：



5.1.2 VB 示例

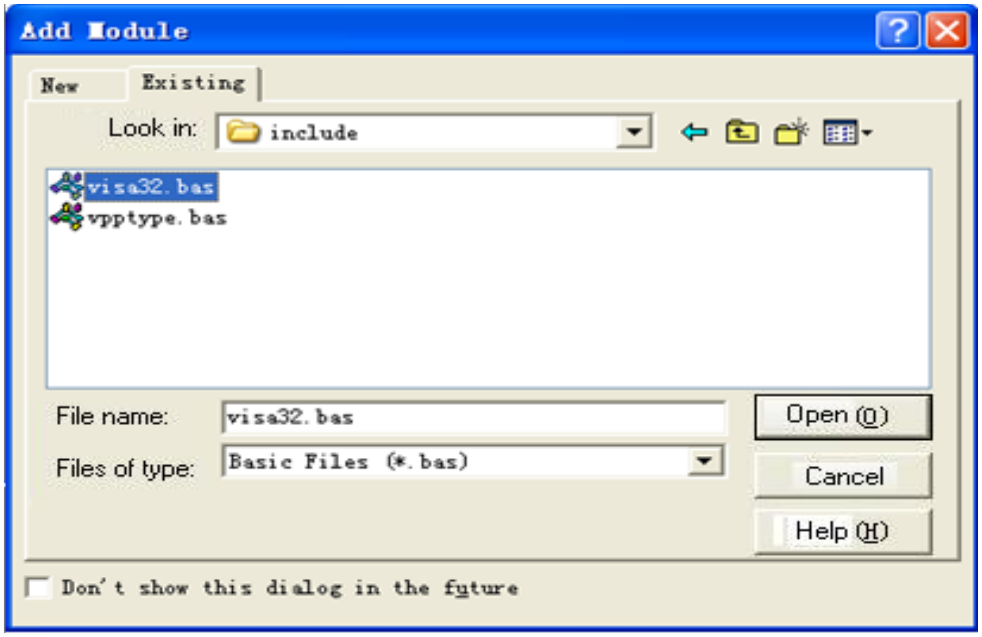
环境：Windows7 32 位系统, Microsoft Visual Basic 6.0

描述：分别通过 USBTMC 和 TCP/IP 访问信号源，并在 NI-VISA 上发送“*IDN?”命令来查询设备信息。

步骤：

1. 打开 Visual Basic 软件，并新建一个标准的应用程序项目。

设置调用 NI-VISA 库项目环境：点击 Existing tab of Project>>Add Existing Item，在 NI-VISA 安装路径下的"include"文件夹中查找 visa32.bas 文件并添加该文件。如下图所示：



2. 编码：

a) USBTMC:

```
PrivateFunction Usbtmc_test() AsLong
' 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 AsLong
Dim instrsesn AsLong
Dim numInstrs AsLong
Dim findList AsLong
Dim retCount AsLong
Dim status AsLong
Dim instrResourceString AsString * VI_FIND_BUFLEN
Dim Buffer AsString * MAX_CNT
Dim i AsInteger

' 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
ExitFunction
EndIf

' 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
    ExitFunction
    EndIf

    ' 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)
    EndIf
        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
    EndIf
```

```
' 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
EndIf

' 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
EndIf
      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
EndFunction
```

b) TCP/IP:

```
PrivateFunction TCP_IP_Test(ByVal ip AsString) AsLong
Dim outputBuffer AsString * VI_FIND_BUFLEN
Dim defaultRM AsLong
Dim instrsesn AsLong
Dim status AsLong
Dim count AsLong

' 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
ExitFunction
EndIf

' 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
ExitFunction
EndIf

   status = viWrite(instrsesn, "*IDN?", 5, count)
If (status < VI_SUCCESS) Then
```

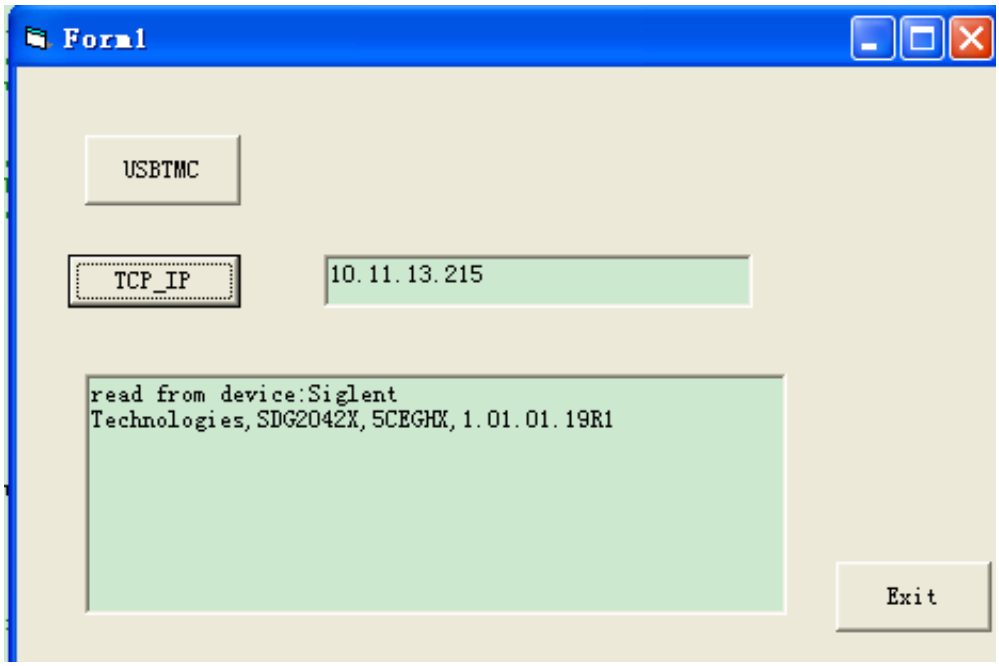
```
        resultTxt.Text = "Error writing to the device."
    EndIf
    status = viRead(instrsesn, outputBuffer, VI_FIND_BUFLEN, count)
    If (status < VI_SUCCESS) Then
        resultTxt.Text = "Error reading a response from the device." + CStr(i + 1)
    Else
        resultTxt.Text = "read from device:" + outputBuffer
    EndIf
    status = viClose(instrsesn)
    status = viClose(defaultRM)
    TCP_IP_Test = 0
EndFunction
```

c) Button control code:

```
PrivateSub exitBtn_Click()
End
EndSub
PrivateSub tcpipBtn_Click()
Dim stat AsLong
    stat = TCP_IP_Test(ipTxt.Text)
If (stat < VI_SUCCESS) Then
    resultTxt.Text = Hex(stat)
EndIf
EndSub
PrivateSub usbBtn_Click()
Dim stat AsLong
    stat = Usbtmc_test
If (stat < VI_SUCCESS) Then
```

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

运行结果：



5.1.3 MATLAB 示例

环境：Windows 7 32 位系统 MATLAB R2013a

描述：分别通过 USBTMC 和 TCP/IP 访问信号源，并在 NI-VISA 上发送“*IDN?”命令来查询设备信息。

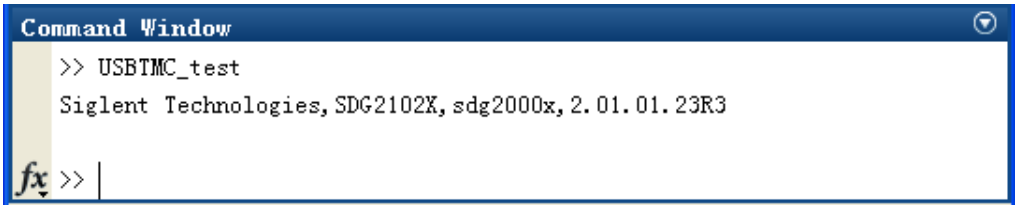
步骤：

1. 打开 MATLAB 软件，并修改当前目录。在本示例中，当前目录修改为："D:\USBTMC_TCPIP_Demo"。
2. 点击在 Matlab 界面的 File>>New>>Script 创建一个空的 M 文件。
3. 编码：
   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
```

运行结果：



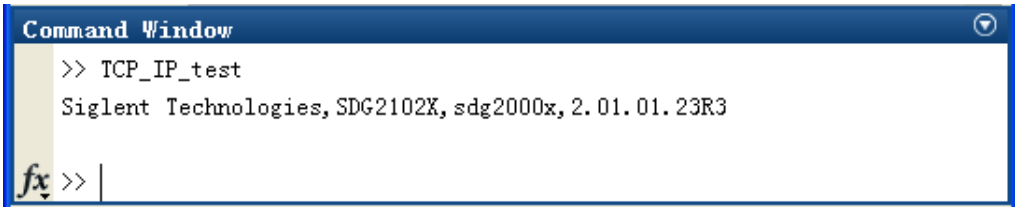
b) TCP/IP :

写一个名为 TCP_IP_Test 的函数：

```
function TCP_IP_test()
% 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::','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
```

运行结果：



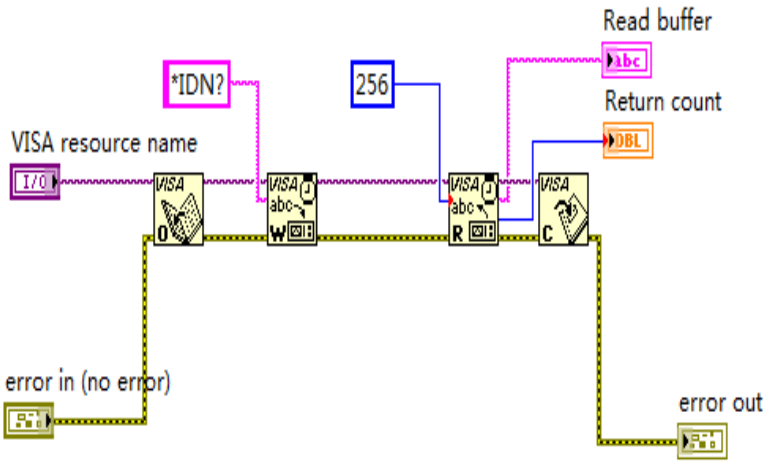
5.1.4 LabVIEW 示例

环境：Windows 7 32 位系统, LabVIEW 2011

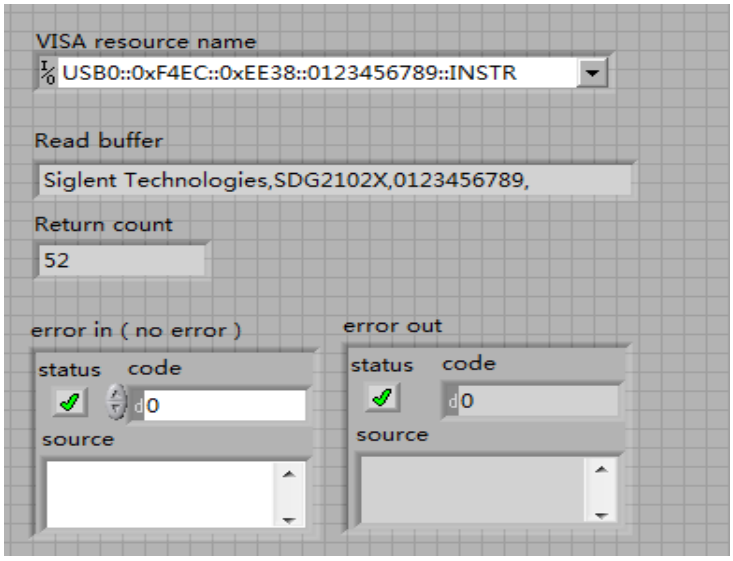
描述：分别通过 USBTMC 和 TCP/IP 访问信号源，并在 NI-VISA 上发送“*IDN?”命令来查询设备信息。

步骤：

1. 打开 LabVIEW 软件，并创建一个 VI 文件。
2. 添加控件。右击前面板界面，从控制列中选择并添加 VISA 资源名、错误输入、错误输出以及部分的指示符。
3. 打开框图界面。右击 VISA 资源名称，并在弹出菜单的 VISA Palette 中选择和添加下列功能：**VISA Write**、**VISA Read**、**VISA Open** 和 **VISA Close**。
4. 如下图连接它们：



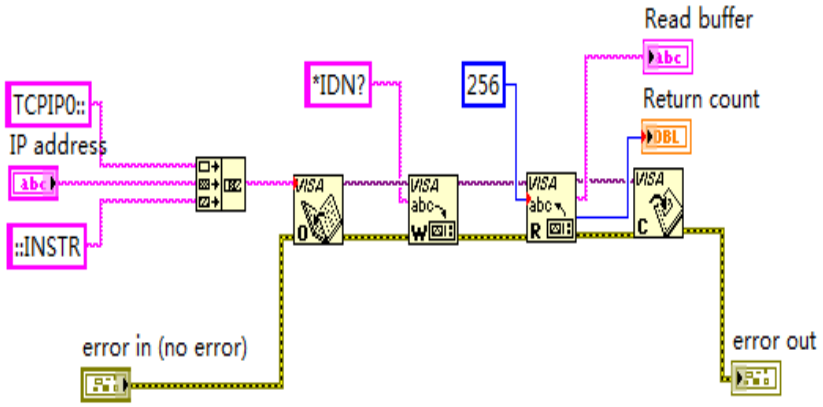
5. 从 VISA 资源名列表中选择设备资源并运行程序。



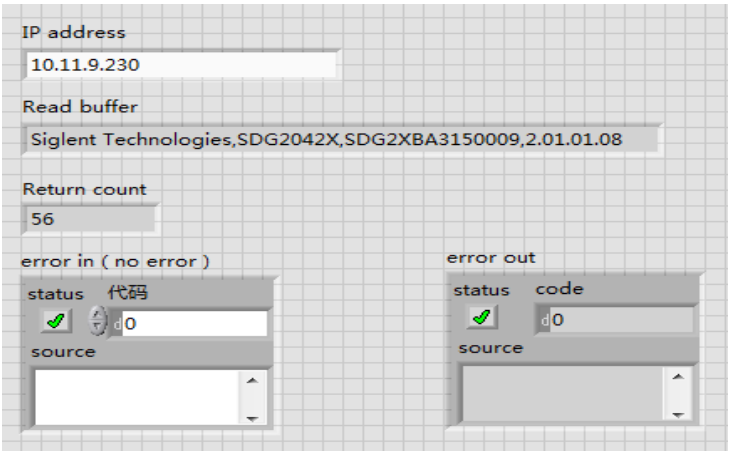
在此例中，VI 打开了一个 USBTMC 设备的 VISA 会话，并向设备写*IDN? 命令，以及回读的响应值。当所有通信完成时，VI 将关闭 VISA 会话。

6. 通过 TCP/IP 与设备通信类似于 USBTMC。但是你需要将 VISA 写函数和 VISA 读函数设置为同步 I/O。LabVIEW 默认设置为异步 IO。右键单击节点，然后从快捷菜单中选择“Synchronous I/O Mod >>Synchronous”以实现同步写入或读取数据。

7. 按照下图连接它们：



8. 输入 IP 地址并运行程序。



5.1.5 Python3 示例 1

环境：Python3.6.5，PyVISA 1.9

描述：示例中创建的波形点数，值代表的是每个波形点数的码字，用于确定波形点数的电平（每个波形点数的码字范围是-32767~32767）。适用于波形点数较小的场景，具体可见章节 3.5.2。

```
#!/usr/bin/env python3.6.5
# -*- coding: utf-8 -*-

import pyvisa as visa
import struct

#USB resource of Device
sdg=visa.ResourceManager().open_resource("USB0::0xF4EC::0x1105::SDG3000XABC002::INSTR")
data = [32767, 32767, 32767, 32767, 32767, 32767, 16384,16384,32767, 32767, 32767, 32767, 32767, 32767, -32767, -32767, -32767, -32767, -32767, -32767, -32767, -16384,-16384,-32767, -32767, -32767, -32767, -32767]

def int_to_two_byte_small_endian(n):
    return struct.pack('<h', n)  # '<h' 表示小端序的短整型（2 个字节）

final_byte_data = b''.join(int_to_two_byte_small_endian(num) for num in data)
print('write bytes:', len(final_byte_data))
print(final_byte_data)
cmd = bytes('C1:WVDT WVNM,"wave1",FRQ,2500,AMP,2.5,OFST,0.2,WAVEDATA,', 'utf-8') + final_byte_data
sdg.write_raw(cmd)

# 通过读取 bin 文件的数据，可以将已有波形写入设备中
f = open(r"C:\Users\Administrator\Desktop\wave1.bin", "rb")
```

```
data1 = f.read()
print ('write bytes:',len(data1))
cmd = bytes('C1:WVDT WVNM,"wave1",FRQ,2500,AMP,2.5,OFST,0.2,WAVEDATA,', 'utf-8') + data1
sdg.write_raw(cmd)
f.close()
```

5.1.6 Python3 示例 2

环境：Python3.6.5，PyVISA 1.9

描述：示例中创建的波形点数，值代表的是每个波形点数的码字，用于确定波形点数的电平（每个波形点数的码字范围是-32767~32767）。适用于波形点数较大的场景，具体可见章节 3.5.3。

```
#!/usr/bin/env python3.6.5
# -*- coding: utf-8 -*-

import pyvisa as visa
import struct

#USB resource of Device
sdg=visa.ResourceManager().open_resource("USB0::0xF4EC::0x1105::SDG3000XABC002::INSTR")
data = [32767, 32767, 32767, 32767, 32767, 32767, 16384,16384,32767, 32767, 32767, 32767, 32767, 32767, -32767, -32767, -32767, -32767, -32767, -32767, -32767, -16384,-16384,-32767, -32767, -32767, -32767, -32767]

def int_to_two_byte_small_endian(n):
    return struct.pack('<h', n)  # '<h' 表示小端序的短整型（2 个字节）

byte_data1 = b''.join(int_to_two_byte_small_endian(num) for num in data[:8])
print('write bytes:', len(byte_data1))
print(byte_data1)
byte_data2 = b''.join(int_to_two_byte_small_endian(num) for num in data[8:16])
print('write bytes:', len(byte_data2))
print(byte_data2)
byte_data3 = b''.join(int_to_two_byte_small_endian(num) for num in data[16:])
print('write bytes:', len(byte_data3))
```

```
        print(byte_data3)
        cmd1=bytes('C1:WVDT:SEGMENT
WVNM,"wave6",FRQ,3500,AMP,1.5,OFST,0.01,BEGIN,WAVEDATA,', 'utf-8') + byte_data1
        cmd2 = bytes('C1:WVDT:SEGMENT WVNM,"wave6",WAVEDATA,', 'utf-8') + byte_data2
        cmd3 = bytes('C1:WVDT:SEGMENT WVNM,"wave6",END,WAVEDATA,', 'utf-8') + byte_data3
        sdg.write_raw(cmd1)
        sdg.write_raw(cmd2)
        sdg.write_raw(cmd3)

        # 通过读取 bin 文件的数据，可以将已有波形写入设备中
        f = open(r"C:\Users\Administrator\Desktop\wave1.bin", "rb")
        data1 = f.read()
        cmd1=bytes('C1:WVDT:SEGMENT
WVNM,"wave6",FRQ,3500,AMP,1.5,OFST,0.01,BEGIN,WAVEDATA,', 'utf-8') + data1[:8]
        cmd2 = bytes('C1:WVDT:SEGMENT WVNM,"wave6",WAVEDATA,', 'utf-8')  + data1[8:16]
        cmd3 = bytes('C1:WVDT:SEGMENT WVNM,"wave6",END,WAVEDATA,', 'utf-8') + data1[16:]
        sdg.write_raw(cmd1)
        sdg.write_raw(cmd2)
        sdg.write_raw(cmd3)
        f.close()
```

5.2 Sockets 应用示例

5.2.1 Python 示例

Python 有一个用于访问 socket 接口的低级的网络模块。基于 sockets 编写的 Python 脚本可用于用于执行各种测试和测量任务。

环境：Windows 7 32 位系统，Python v2.7.5。

Description：打开一个 socket,发送一个查询操作并循环执行 10 次后关闭 socket。注意：程序中的 SCPI 命令的字符串必须以“\n”字符（换行）作为结尾。

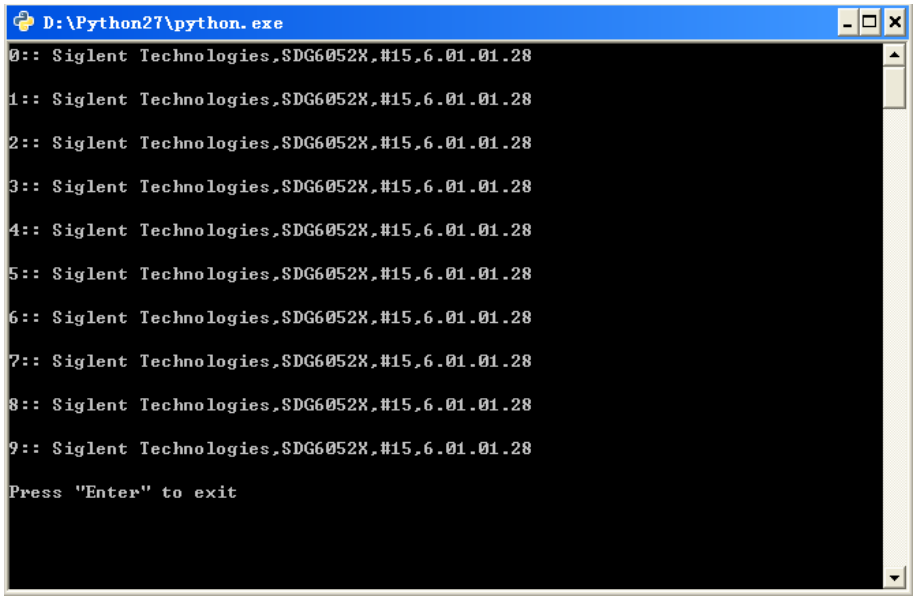
下面是脚本的代码：

```
#!/usr/bin/env python
#-*- coding:utf-8 -*-
#------------------------------------------------------------------------
# The short script is a example that open 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()
```

运行结果：



关于鼎阳

鼎阳科技（SIGLENT）是通用电子测试测量仪器领域的行业领军企业，A股上市公司。

2002 年，鼎阳科技创始人开始专注于示波器研发，2005 年成功研制出鼎阳第一款数字示波器。历经多年发展，鼎阳产品已扩展到数字示波器、手持示波表、函数/任意波形发生器、频谱分析仪、矢量网络分析仪、射频/微波信号源、台式万用表、直流电源、电子负载、精密源表等基础测试测量仪器产品，是全球极少数能够同时研发、生产、销售数字示波器、信号发生器、频谱分析仪和矢量网络分析仪四大通用电子测试测量仪器主力产品的厂家之一，国家重点“小巨人”企业。同时也是国内主要竞争对手中极少数同时拥有这四大主力产品并且四大主力产品全线进入高端领域的厂家。公司总部位于深圳，在马来西亚槟城州设有生产基地，在美国克利夫兰、德国奥格斯堡、日本东京成立了子公司，在成都成立了分公司，产品远销全球 80 多个国家和地区，SIGLENT 已经成为全球知名的测试测量仪器品牌。

联系我们

深圳市鼎阳科技股份有限公司

全国免费服务热线：400-878-0807

网址：www.siglent.com