用户手册

AWG4100 任意波形发生器

国仪量子
CIQTEK

版权所有

商标信息

- Intel Core 英特尔公司在美国和/或其他国家的商标或注册商标；
- Windows 10 是微软公司在美国和/或其他国家的注册商标或商标；
- Labview 是美国国家仪器有限公司在美国和/或其他国家的注册商标或商标；
- Matlab 是 Mathworks 在美国和/或其他国家的注册商标或商标；
- 本手册中出现的其他公司名和产品名均属于各自公司的商标或注册商标。

版本历史

本手册版本为 V3.0，全部产品使用手册的版本主要修订记录如下：

版本	更新时间	更新内容	备注
V1.0	2020.06	创建文档	
V2.0	2020.12	1、预览模块增加保存和重置功能 2、 代码编辑中新增 marker 函数、square_p、triangle_p 函数；增加代码编辑提示区域 3、 新增离线固件升级版本功能	
V2.1	2021.04	1、增加支持电平和下降沿触发模式 2、增加 SWEEP 扫频函数 3、SDK 接口增加 2 种下载方式和触发方式	
V2.2	2021.07	1、优化波形代码编辑的说明 2、增加用户常见使用问题	
V3.0	2021.11	1、增加部分基础波形函数 2、新增 DIO 接口 3、多设备控制的优化	

保证和声明

软件或固件版本

软件或固件升级可能更改或增加产品功能和性能，请关注国仪量子（合肥）技术有限公司官方网站获取最新版本手册或联系国仪量子（合肥）技术有限公司升级软件和/或固件。

声明

- 本公司产品受中国及其他国家和地区的专利（包括已取得和正在申请的专利）保护。
- 本公司拥有改变产品规格及价格的权利。
- 本手册提供的信息取代以往出版的任何资料。
- 本手册内容随着仪器性能与功能的升级而改变，恕不另行通知。
- 本手册中的产品或仪器软件图片可能与实际产品存在轻微差异，请以实物为准。
- 我们努力将本手册的内容做到完善，如有任何疑问或发现任何错误，请与国仪量子（合肥）技术有限公司联系。
- 未经我司事先书面许可，不得影印、复制、转载或改变本手册的任何部分。
- 用户一旦使用产品，即视为对本声明的全部内容认可和接受。

联系我们

- 电子邮箱：service@ciqtek.com
- 电　　话：4000606976-602
- 企业官网：www.ciqtek.com

有限担保和责任范围

国仪量子（合肥）技术有限公司保证销售的产品在质量和计量上都是完全合格的，此项保证不包括保险丝以及因疏忽、误用、污染、意外或非正常状况使用造成的损坏；本项保证仅适用于原购买者，且不可转让。

国仪量子（合肥）技术有限公司承诺本产品的主机及附件自发货之日起保修一年，在产品保修期内无任何材料和工艺缺陷等产品质量问题。

凡在保修期内因产品本身的质量引起的硬件或软件的故障，国仪量子（合肥）技术有限公司将按照完整的保修声明所述，免费提供修理或更换服务（国仪量子（合肥）技术有限公司可自行决定服务的方式）。

在质保期内若出现以下情况的一种，国仪量子（合肥）技术有限公司将不提供免费维修服务：

（1）运输过程中造成的意外损坏。

（2）因错误安装或在非产品规定的工作环境下使用造成的仪器故障或损坏。

（3）产品人为的外观损坏（如表面擦伤，变形等）。

（4）私自拆机修理，改造，更换器件及产品保修封条被撕毁。

（5）因不可抗拒因素（如雷击）造成的故障或损坏。

（6）因用户不恰当操作或与不兼容的设备连接造成的直接或间接损坏。

（7）维修已改动或者与其他产品集成的产品（如果这种改动或集成会增加产品维修的时间或难度）。

（8）若因客户操作不当引起仪器的测量不准或不能测量，仪器本身无问题的，返程费用由客户承担。

若产品已超过保修期，国仪量子（合肥）技术有限公司将为客户提供有偿维修服务；

本项保证不适用于由于意外、机器部件的正常磨损、在产品规定的范围之外使用、使用不当、维护保养不当或不足而造成的任何缺陷、故障或损坏。

安全注意事项

为避免可能的危险以及防止损坏本产品和与本产品连接的任何设备，在安装、使用或维修本产品之前，请务必仔细阅读、并完全理解“安全注意事项”章节的相关内容。

安全概要

为保证您能正确安全地使用本仪器，请务必遵守以下注意事项。如果未遵守本手册指定的方法操作本仪器，可能会造成仪器损坏或人员损伤。因违反以下注意事项操作仪器所引起的损伤，国仪量子（合肥）技术有限公司概不承担责任。

使用正确的供电电源

在连接电源线之前，请确认电源电压与仪器的额定电压相一致，并且小于电源线的最大额定电压。

使用正确的电源线

为避免对操作人员造成伤害或损坏产品，请使用本产品专用并经所在国家/地区认证的电源线。

保护接地

本产品通过电源电缆的保护接地线接地，为避免电击，在连接本产品的任何输入或输出端子之前，请确认本产品电源电缆的接地端子与保护接地端可靠连接。

请勿切断本仪器内部和外部的保护接地线、或拔出保护接地端子的电线，否则将有潜在的触电危险。

查看所有终端额定值

为避免起火和过大电流的冲击，请遵守产品上所有的额定值和标记说明；在连接产品之前，请先查看产品手册，了解额定值的详细信息。

使用合适的过压保护

确保没有过电压（如由雷电造成的电压）到达该产品，否则操作人员可能有遭受电击的危险。

请勿开盖操作

请勿在外盖/面板拆除或机壳打开的状态下操作本产品，可能有危险电压暴露。

防静电保护

静电会造成仪器损坏，应尽可能在防静电区进行测试，在连接电缆到仪器前，应将其内外导体短暂接地以释放静电。

断开电源

电源开关可以使产品断开电源，请参阅有关位置的说明，勿将设备放在难以断开电源开关的位置，确保产品需要快速断开电源连接时，用户可以随时操作电源开关。

避免接触裸露电路

产品接通电源时，请勿接触任何裸露的接点和部件。

保持良好的散热条件

为避免因电路板过热而损坏，在使用本产品的过程中请勿堵住通风口。

请勿在潮湿环境下操作仪器

为避免产品内部电路出现短路等危险情况，请勿在潮湿环境下操作仪器。

请勿靠近易燃易爆物品

为避免人身伤害或产品损坏，严禁易燃易爆物靠近本产品。

远离高温环境

为避免发生危险，严禁将本产品放置于高温环境中。

请保持产品表面的清洁和干燥

为避免灰尘或空气中的水分影响仪器性能，请保持产品表面的清洁和干燥。

注意搬运安全

为避免仪器在搬运过程中滑落，造成仪器面板上的按键、旋钮或接口等部件损坏，请注意搬运安全。

安全规则

怀疑产品出故障时，请勿进行操作，如果您怀疑本产品出现故障，请联络售后维修人员进行检测；任何维护、调整或零件更换必须由我公司维修人员执行；为防止触电，非本公司授权人员，严禁拆开机器。

为避免造成人身伤害或产品损坏，严禁未经过培训的人员使用本产品。

安全标识

以下术语和符号可能出现在本手册中：

警告

警告性声明指出可能会造成人身伤害或危及生命安全的情况或操作

注意

注意性声明指出可能导致本产品损坏或数据丢失的情况或操作

以下术语可能出现在产品上：

“危险”表示您看到该标记时可直接导致人身伤害的危险。

“警告”表示您看到该标记时不会直接导致人身伤害的危险。

“注意”表示会对本产品或其他财产造成损害的危险。

以下符号可能出现在产品上：

	注意请参阅手册 这将通知用户潜在的危险，并表明用户必须参考说明书。
	小心触电
	地端子
	PE 接线端
	壳体接地端
	处理静电敏感设备时请小心
	警告！ 小心烫伤
	警告！ 激光辐射
	警告！ 微波辐射

	直流 (DC)
	交流 (AC)
	交/直流 (DC/AC)
	表示电源开关的 ON 位置
	表示电源开关的 OFF 位置
	请勿将使用过的仪器丢入垃圾桶

合规性说明

此部分列出仪器遵循的 EMC（电磁兼容性）、安全和环境标准。本产品仅供专业人员和受过培训的人员使用。

安全合规性

本产品安全级别 1 级 - 接地产品。

污染度说明 对产品周围和产品内部环境中可能出现的污染的一种量度。通常认为产品的内部环境与外部环境相同。产品只应该在其规定环境中使用。

污染度 1。无污染或仅出现干燥、非导电性污染。此类别的产品通常 进行了封装、密封或被置于干净的房间中。

污染度 2。通常只发于非使用状态时，才会发生临时凝结。

污染度 3。导电性污染，或由于凝结会变成导电性污染的干燥、非导电性污染。此类场所为温度和湿度不受控制的建有遮盖设施的场所。此类区域不受阳光、雨水或自然风的直接侵害。

污染度 4。通过导电性的尘埃、雨水或雪而产生永久导电性的污染。户外场所通常属于这种情况。

本产品污染度 2。注意： 仅适合在室内使用。

安装（过压）类别说明

本产品的端子可能有不同的安装（过压）类别指定。 安装类别包括：

测量类别 IV。 用于在低压安装电源处进行的测量。

测量类别 III。 用于在建筑安装中进行的测量。

测量类别 II。 用于在与低压安装直接相连的电路上进行的测量。

测量类别 I。 用于在不直接连接到市电的电路上进行的测量。

本产品过压类别 II

环境注意事项

本部分提供有关产品对环境影响的信息。

产品报废处理

设备回收 生产本设备需要提取和使用自然资源。如果对本产品的报废处理不当，则该设备中包含的某些物质可能会对环境或人体健康有害。为避免将 有害物质释放到环境中，并减少对自然资源的使用，建议采用适当的方法回收本产品，以确保大部分材料可以得到恰当的重复使用或回收。

电池回收 本产品还装有小型锂金属纽扣电池。如果电量用尽，请根据当地政府法规正确处理或回收此电池。

目录

1.关于本手册

本手册包含了 AWG4100 的使用说明和指标参数。本手册内容随着仪器性能与功能的升级而改变，请及时更新软件、固件版本至最新版，并在官网下载最新版本手册。

2.使用前准备

本章主要概述收到仪器后必须进行的一些检查以及在安装使用仪器之前必须了解和具备的条件等。

2.1 拆包与检查

2.1.1 检查运输包装

AWG4100 任意波形发生器出厂前已进行完整测试和严格检查，但运输过程中仍可能出现损坏情况，请在签收产品前进行详细检查。

• 用户收到产品后，请先检查包装是否完整，如果发现包装纸箱严重破损，请保留被损坏的包装，直到整机和附件通过电性和机械性测试。因运输造成仪器损坏，由发货方和承运方联系赔偿事宜。国仪量子（合肥）技术有限公司恕不进行免费维修或更换。

• 如果仪器的包装完好,请您核对一下您所订购的仪器型号和包装箱上所注的型号是否一致;如果不一致,请与供应商或国仪量子（合肥）技术有限公司联系。

2.1.2 检查包装内容

根据装箱清单检查货品是否完整，是否与订单相符合，如有损坏或缺失，请与供应商或国仪量子（合肥）技术有限公司联系。

请您保存好原包装材料，以便在以后运输或存储使用。

包装内容：

本产品提供以下的随机配件：

表 2.1.1 包装清单表

包装内容	**数量**	**单位**
AWG4100 任意波形发生器	1	台
附件		
电源线	1	根

AWG4100 任意波形发生器用户手册	1	份
网线	1	根
合格证	1	份
检验报告	1	份

2.1.3 整机检查

如产品存在机械损坏，或者产品未通过性能测试，请及时与供应商或国仪量子（合肥）技术有限公司联系。并提供损坏处的照片，便于提供服务。

2.1.4 存储、重新包装、运输

仪器若需要长时间存储设备，需要将其存放在特定的环境条件下：

- 使用原包装箱重新打包，保持干燥。
- 存储温度范围-10℃～50℃，相对湿度范围 0～95%，无冷凝。
- 使用防尘帽将信号或者检测接口封堵防护。

仪器若需要重新包装运输，需要注意以下要求：

1、在重新包装运输时，使用足够强度和空间的纸箱放置仪器，并填充。

2、在运输过程中，请注意避免剧烈震动影响。

2.2 仪器的安装

2.2.1 一般注意事项

AWG4100 任意波形发生器可在实验室或生产线环境中使用，可采用桌面安装或 19 英寸 IEC 机架安装两种方式，机架安装的方式时，2 台设备并列安装，占用 2U 高度。

注意

1．本产品采用强迫风冷散热设计，使用时确保出风口与进风口通畅，不要用物体挡住进、出风口，否则可能使仪器内部温度过高，进而导致仪器的损坏（最小距离不小于 10 cm）。

2．桌面安装时，工作台表面必须平坦，且仪器需要水平放置，不得使用侧面或者背面放置，以免损坏设备接口或者跌落设备。

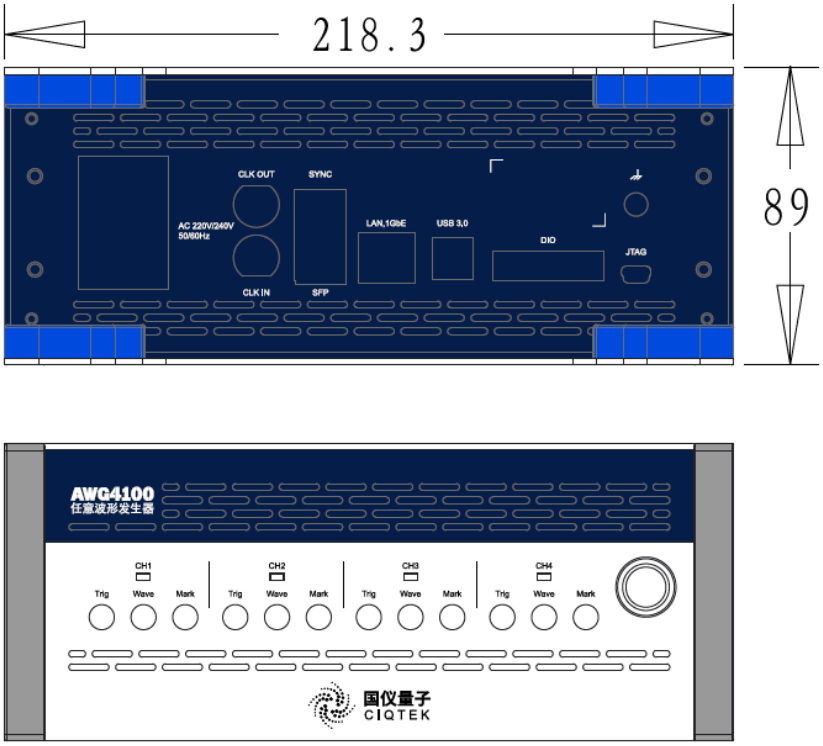

图 2.2.1 AWG4100 正面、背面尺寸

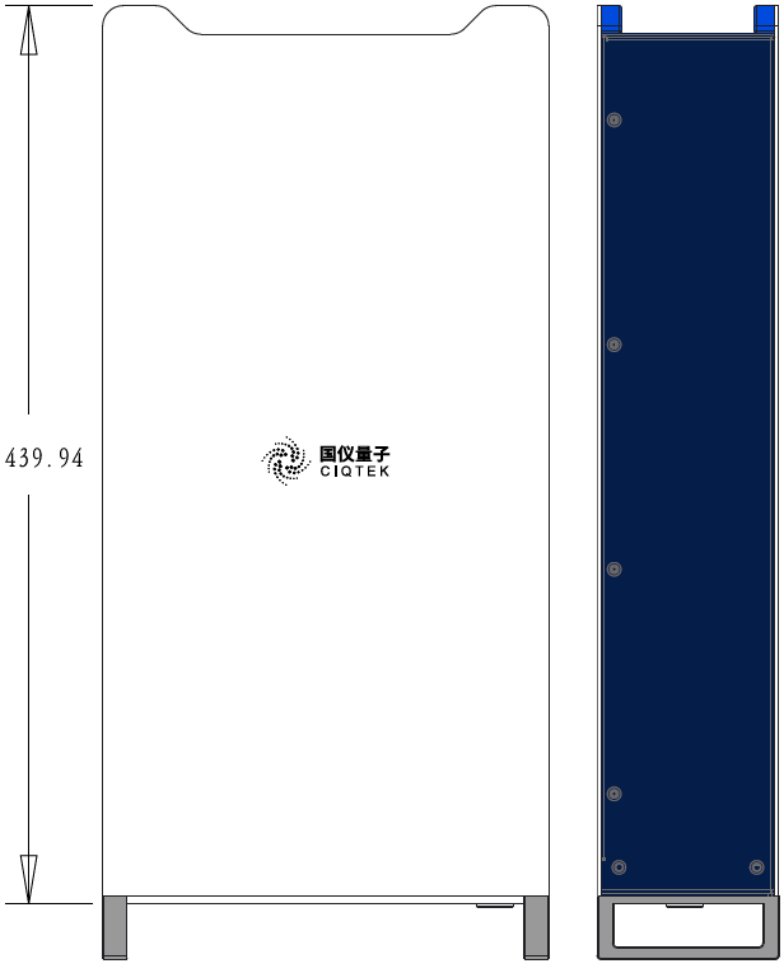

图 2.2.2 AWG4100 侧面尺寸

2.2.2 安装环境

本产品安装温度和湿度环境需要满足：

表 2.2.1 安装环境表

特性	说明
温度	工作温度：0℃～40℃ 存贮温度：-10℃～50℃
存储相对湿度	≤95%，无凝露
工作相对湿度	≤90%，无凝露
工作海拔	≤2000 米
安装（过电压）类别	本产品由符合安装（过电压）类别 II 的主电源供电。

其他	1.不得在有易燃易爆气体的环境安装或者使用仪器。 2.不得在室外、阳光直射的地方、靠近火源或热源的环境安装或者使用仪器。 3.远离油烟、蒸汽、灰尘、腐蚀性气体及其他污染物。 4.远离机械振动。 5.远离强电磁场、高压设备/电源线或脉冲噪声源。

2.3 连接电源

连接电源线前，请仔细阅读安全信息。疏忽下述警告可能会导致触电或仪器受损。

警告

1．确保插座电压与 AWG4100 的额定电压相吻合。

2．确保 AWG4100 的电源开关为 OFF 状态。

3．确保电源线是原厂提供 AWG4100 专用电线，使用不合适的电源线可能会引起触电或火灾。

4．确保将电源线插在标准的带保护地线的 3 脚插座中。

5．禁止使用不接地的延长线。

AWG4100 配有一个交流电源连接器，可以使用不同的交流电源电压，并能够自适应这些电压，请参阅技术规格指标章节说明，交流电源连接器位于仪器的后面板上。

1、确保后面板上的电源开关置为“O”。

2、确保前面板上的电源开关状态为“OFF”。

3、将随机的电源线插进 AWG4100 后面板上的电源接口。

4、将电源线的另一端插入带接地保护的 3 脚插座中，接地插座要求如下：

额定电源电压：220±10%VAC

允许电压范围：90～264 VAC

额定电源频率：50/60 Hz

允许电源频率范围：47～65 Hz

最大电源功耗：＜60 W

注意

1. 电源输入相线 L、零线 N、地线 E 应与本仪器电源插头相同。

2. 仪器内部采用高性能电源设计方案减少因 AC 电源端输入带来的杂波干扰，但仍应尽量在低噪声的环境下使用该仪器，如果无法避免，请安装电源滤波器。

2.4 打开/关闭仪器

打开仪器前，请确认机器正确安装且电源已正确连接。

打开仪器

1.将后面板上的交流电源开关拔到位置”I”。

2.按下前面板上 ON/OFF 键。

启动后，前面板上 ON/OF 键上的 LED 指示灯亮启，表示产品电源已接通。

注意

如果要得到高精度的输出结果，开机后 AWG4100 至少需要 10 分钟的预热时间。请保持环境温度为：23±2℃,湿度为：50±10%RH。

关闭仪器

1.按下前面板上 ON/OFF 键，关闭后，前面板上 ON/OF 键上的 LED 指示灯关闭。

2.将后面板上的交流电源开关切换到位置”O”，或将仪器从交流电源上断开。

AWG4100 将进入关机模式。

注意

存在数据丢失的风险

如果直接关闭后面板上的开关或通过断开电源线关闭正运行中的仪器，那么仪器的当前设置将会丢失且程序数据也可能丢失。

为了正确关闭应用程序，请先关闭 ON/OFF 键关机后再关闭后面板上的开关和断开电源线。

2.5 软件安装

用户可到我官网 https://www.ciqtek.com 产品中心→量子测控系列产品→任意波形发生器（AWG4100）产品介绍界面下方自行下载 AWG4100 的上位机软件安装包到本地后进行安装。具体安装步骤如下：

1、 如图 2.5.1，双击 AWG4100.exe 的图标，进入软件安装向导界面，见图 2.5.2。

图 2.5.1 安装包文件图标

图 2.5.2 安装向导页面

2、 在安装向导页面可以选择安装的软件语言，选好语言之后可以点击“快速安装”直接进入安装过程，直接安装到默认 C:\Program Files(X86)文件夹；也支持用户自己选择安装目录，点击“浏览”进行安装路径选择。点击“快速安装”后进入安装页面，如图 2.5.3。

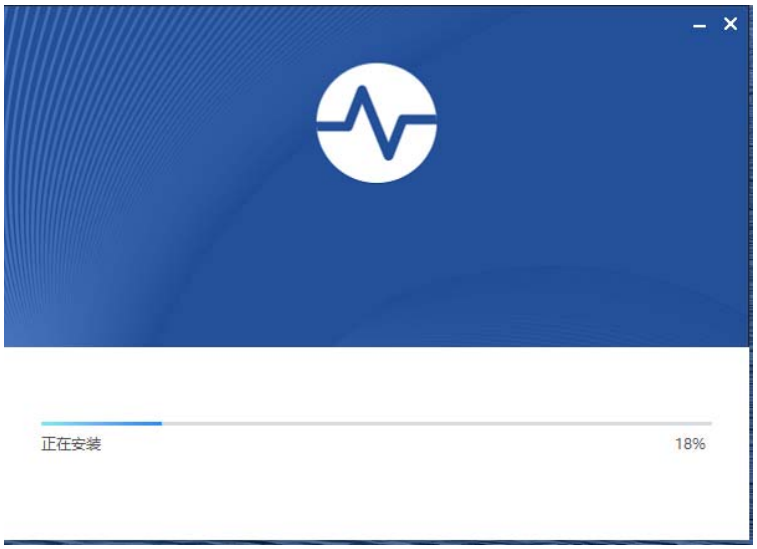

图 2.5.3 安装过程

3、显示安装的详细过程和信息，完成安装后，进入安装成功页面，可以点击“开始使用”直接进入软件页面。

2.6 设置网络(LAN)连接

AWG4100 配备网络接口，可连接到以太网 LAN（局域网）或者直接连接上位机。

AWG4100 提供两种方法建立仪器的局域网连接：

1、使用随机附赠的或者符合当地标准的 RJ45 网线将设备的网络接口连接至集线器、交换机或者网关等，建立仪器至现有网络的非专用网络（以太网）连接。仪器将被分配一个 IP 地址，可以与计算机和其它主机共处于同一个网络中。

2、使用随机附赠的的或者符合当地标准的 RJ45 网线将设备的网络直接连接至上位机 PC，建立仪器到计算机的专用网络连接（点对点连接），计算机必须配装网络适配器，并直接连接至仪器，这种模式无需使用集线器、交换机或者网关，AWG4100 操作软件支持跨网段连接，无需手动修改上位机静态 IP 地址，即可直接连接设备。

注意

虽然 AWG4100 操作软件支持跨网段连接，若上位机与设备的 IP 地址不在同一网段时，上位机操作软件将自动修改仪器的固定 IP 地址。

3、连接网络后，请检查网口出的连接指示灯是否亮起，如图 2.6.1 所示。

图 2.6.1 网络连接状态指示

3 软件和固件在线升级

AWG4100 采用远程升级技术，可满足客户远程在线升级设备操作软件和固件程序。

3.1 操作软件在线升级

AWG4100 操作软件支持在线实时检测更新信息，在电脑已连接互联网的情况，用户点击打开操作软件进入连接页面时将自动检测是否有可更新的操作软件，如图 3.1.1 所示，客户可点击“Remind next（下次提醒）”按钮忽略本次更新提醒或者点击“Update（更新）”进入软件升级流程。

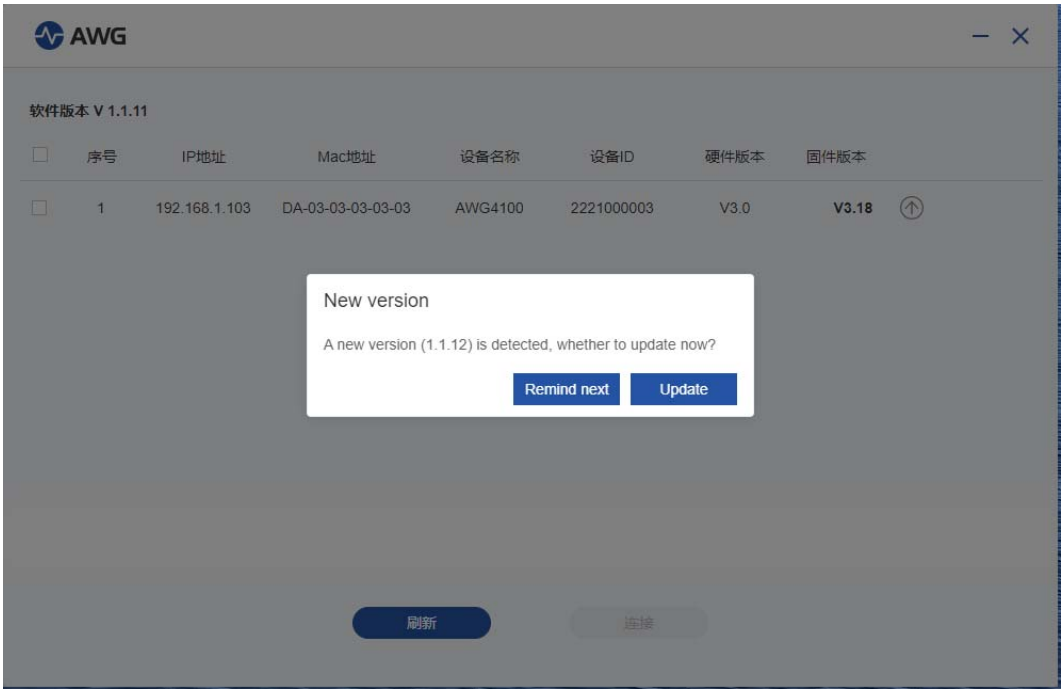

图 3.1.1 检测软件更新提示

1、 点击“Update（更新）”按钮后，进入软件更新文件下载流程，如图 3.1.2 所示。

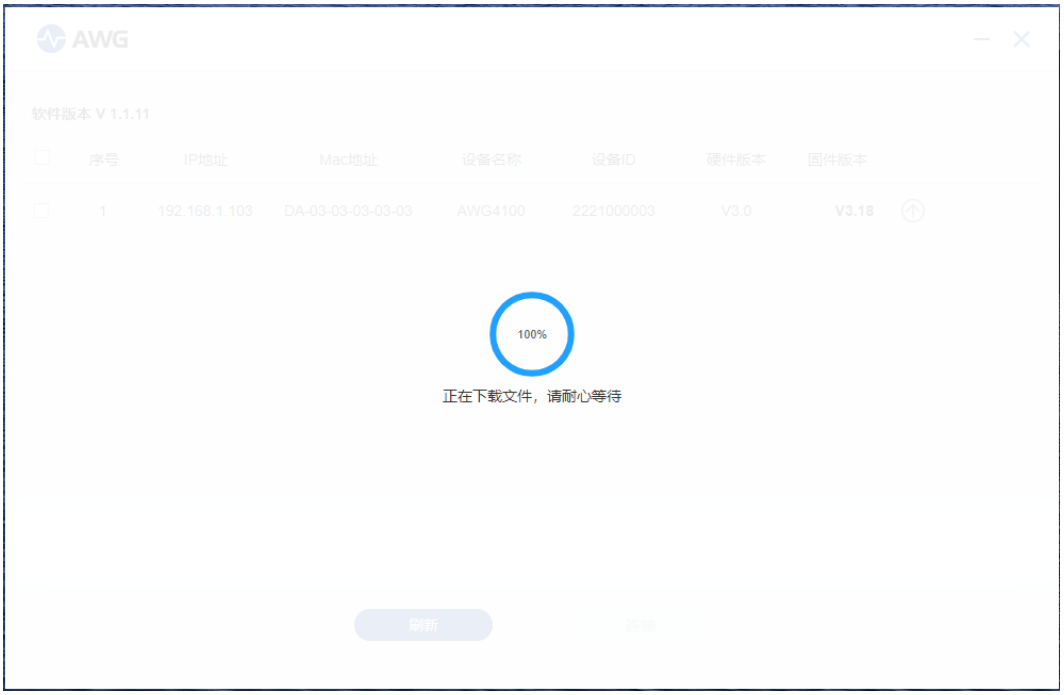

图 3.1.2 软件更新文件下载

2、更新文件下载完成后，将进入更新文件安装流程，其与软件安装流程一致，详情参见 2.5 章节介绍。

3、客户可点击“Remind next（下次提醒）”按钮忽略本次更新后，进入设备连接界面，设备连接界面左上角将显示当前软件版本号和需要更新的状态，如图 3.1.3 所示。

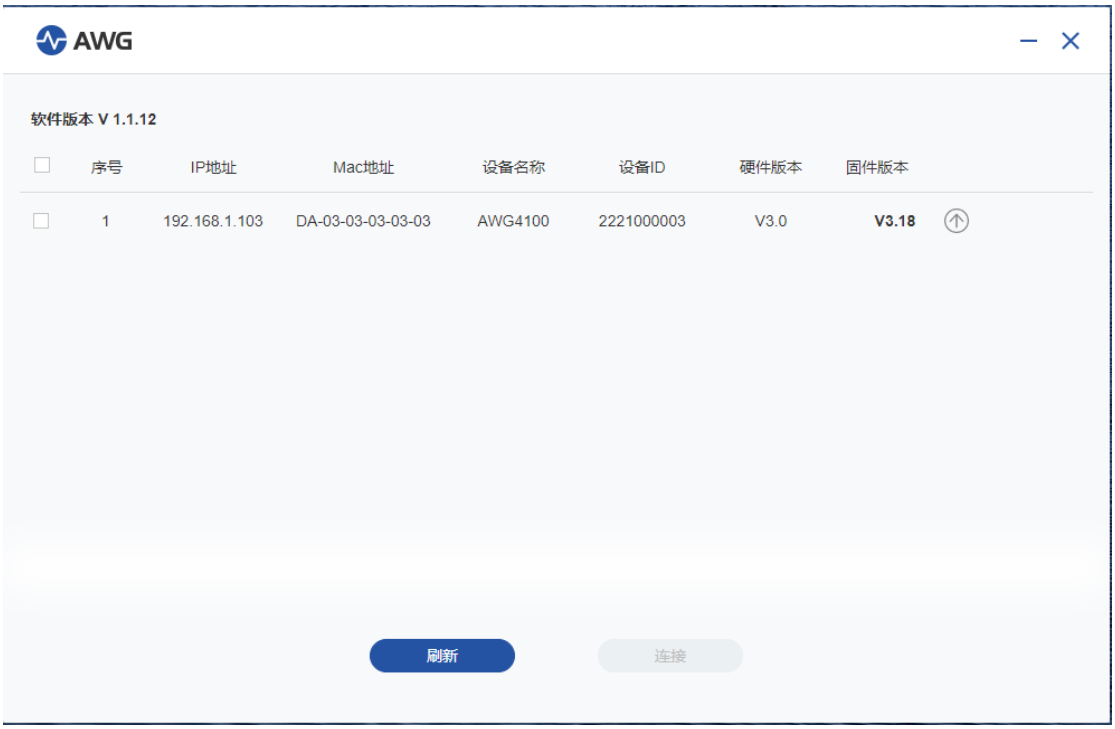

图 3.1.3 连接界面

3.2 固件在线升级

AWG4100 设备固件程序支持在线实时检测更新信息的功能，电脑已连接互联网的情况，用户点击打开操作软件进入连接界面后，将在局域网搜索到设备列表，在固件版本号右侧显示设备的固件程序是否需要更新，如图 3.2.1 所示。

图 3.2.1 连接界面设备固件可升级状态显示

若需要更新，可单击控件，进入固件更新流程，若不需要更新忽略即可。

注意：固件版本号过低，或是软件版本号过低，可能会产生不匹配的情况，请将软件、固件升级至最新版本使用。

1、点击控件，弹出固件更新提醒信息，如图 3.2.2 所示。

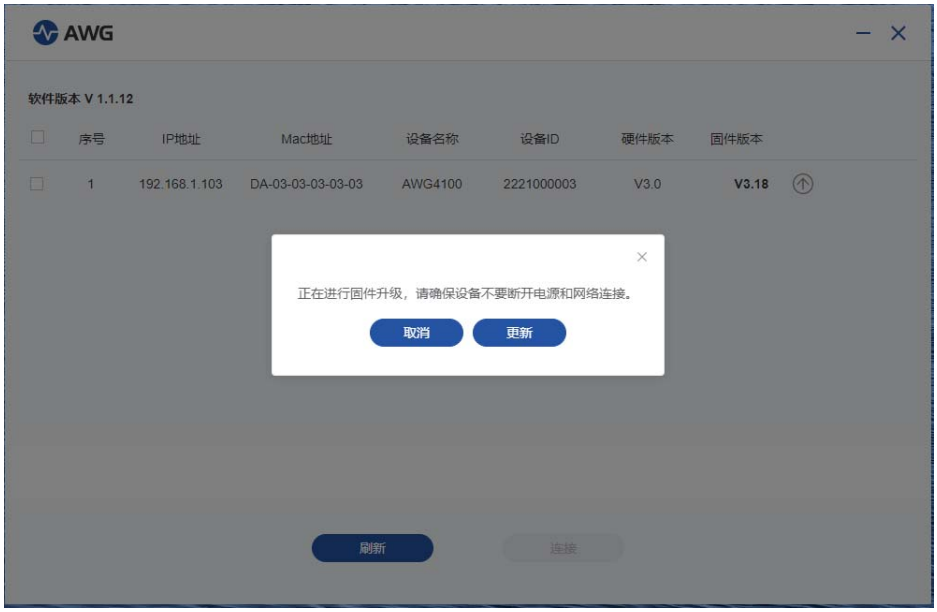

图 3.2.2 固件更新提醒信息

2、点击“更新”进入固件更新文件下载和更新流程，如图 3.2.3 所示；点击“取消”出固件更新流程返回设备连接界面。

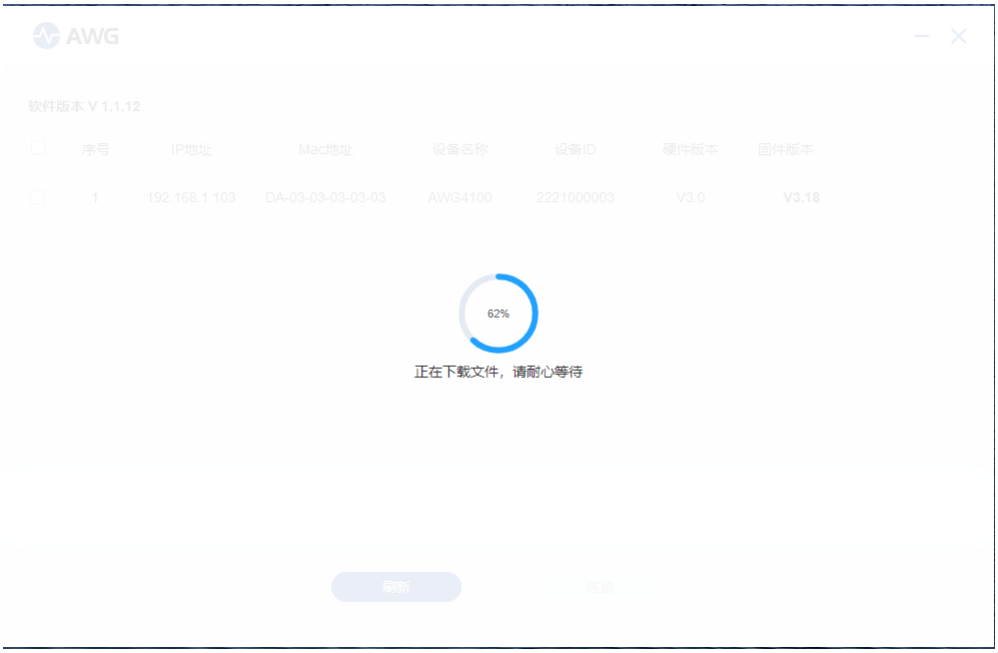

图 3.2.3 固件更新文件下载

3、等待设备固件自动更新完成后，将提示固件更新成功以及重启电源，如 3.2.4 所示，使用设备前面板上的电源按钮重启设备。

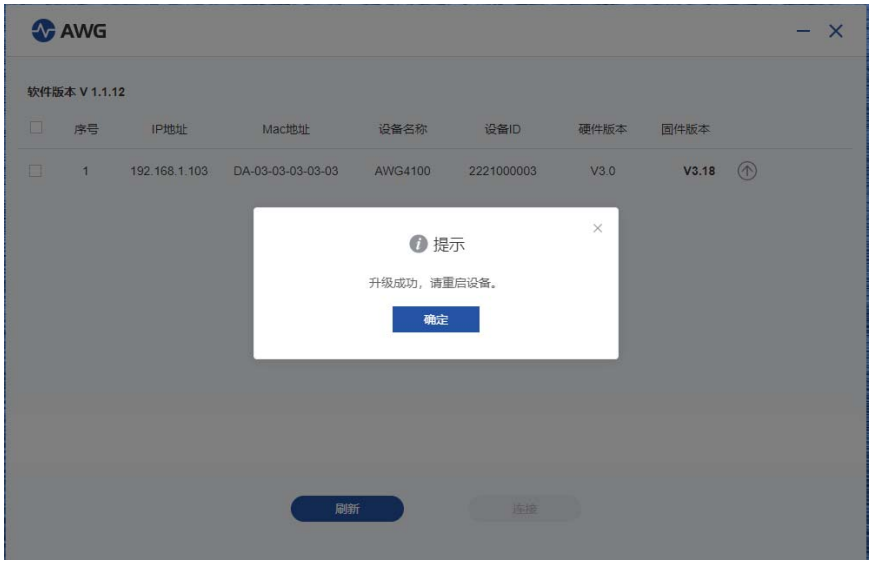

图 3.2.4 固件更新完成

设备重启，软件可以连接更新固件后的设备。

4.快速启动

本章节适用于初次拿到 AWG4100 的用户。我们将通过演示任意波形产生，来讲解如何快速启动和应用该仪器。

该实验过程需要的具体设备清单如下：

- 1 根网线
- 4 根 BNC 连接同轴电缆
- 1 台已经安装 AWG4100 上位机软件的 PC
- 1 台双通道示波器。

4.1 准备工作

实验开始之前，需要完成以下准备工作：

1. 将 AWG 和 PC 的电源线连接好，并上电开机；
2. AWG4100 后面板的 LAN,1GbE 接口通过网线直接连接到 PC。具体连接线路图见 4.1.1：

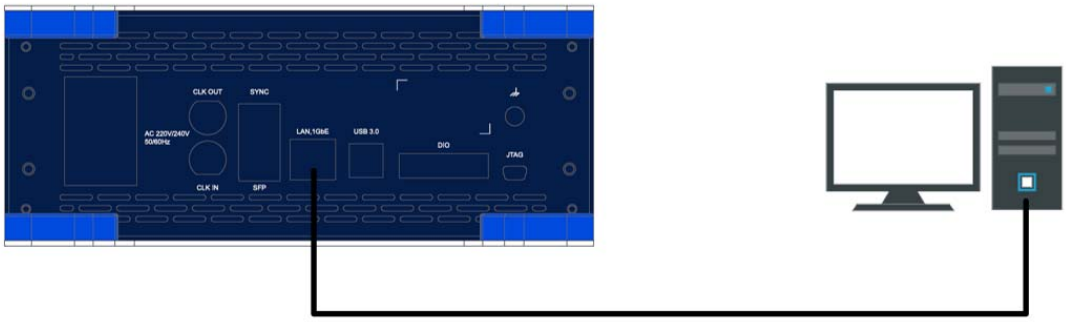

图 4.1.1 后面板线路连接图

3. 将 AWG4100 前面板 4 个 WAVE 通道通过 BNC 同轴电缆分别连接到示波器通道，具体连接线路图见 4.1.2：

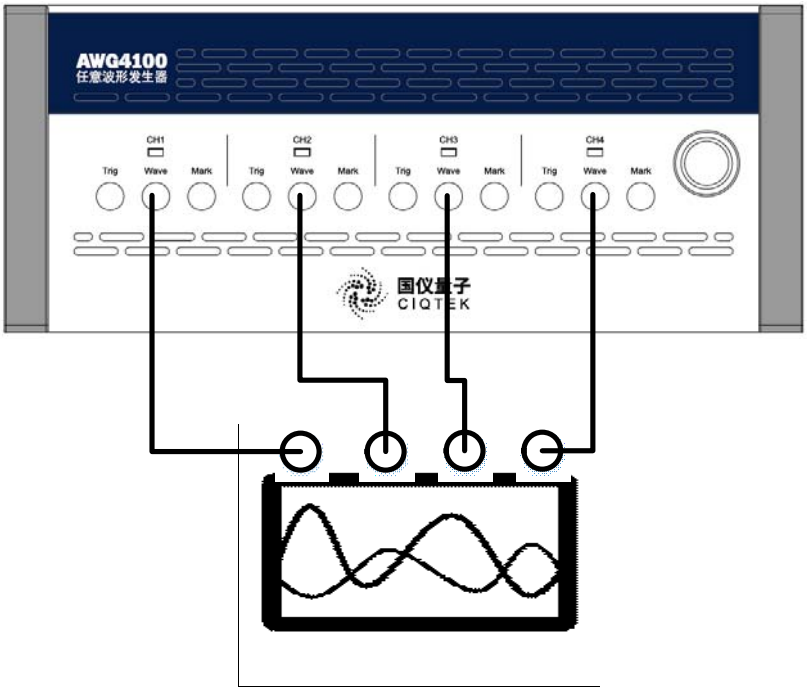

图 4.1.2 前面板线路连接图

4.2 打开操作软件

AWG4100 操作软件默认安装会在桌面建立快捷方式，双击桌面快捷方式或者在 Windows 开始菜单中找到 AWG4100 程序，打开软件，进入设备连接界面。

进入设备连接界面，选中需要连接的设备，如图 4.2.1 所示，点击连接。

图 4.2.1 操作软件连接界面

点击连接软件进入操作主界面。

4.3 基础波形产生实验

本章节将演示 2 个通道基础波形产生。打开上位机软件，并且连接设备，如下图所示，实验主页面进行详细的参数配置。

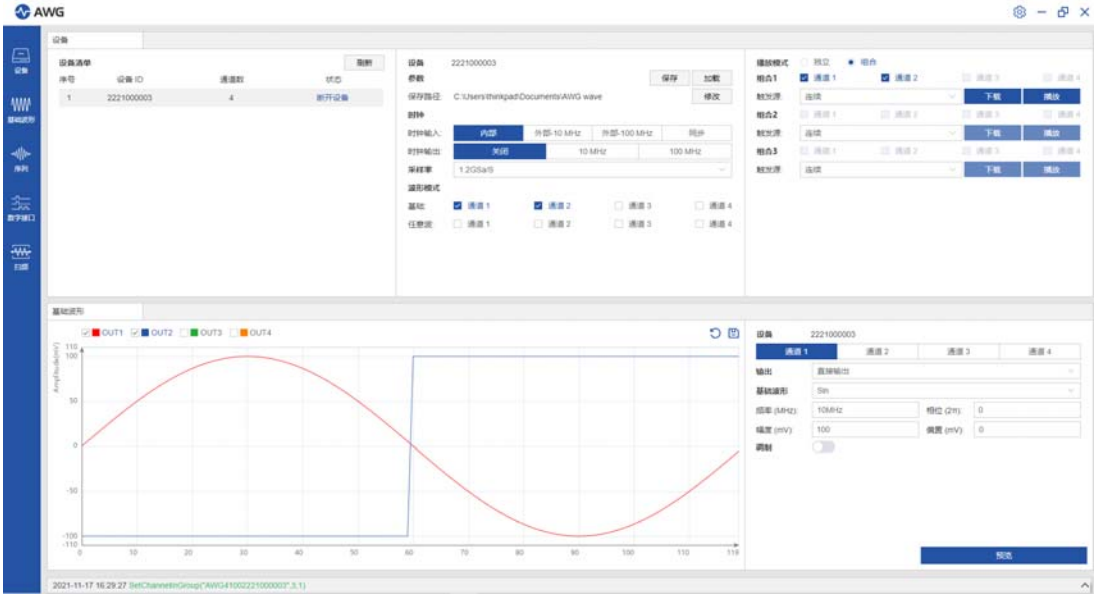

图 4.3.1 软件配置页面

1. 在设备页面下配置：

时钟	
时钟输入	内部
时钟输出	关闭
采样率	1.2GSa/s
波形模式	
基础	选中通道 1、2
任意波	不选
播放模式	
播放模式	组合
组合 1	通道 1，通道 2 选中
触发源	连续

2. 基础波形设置

通道 1	
输出	直接输出
基础波形	Sin
频率（MHz）	10MHz
相位（2pi）	0
幅度（mV）	100
偏置	0
调制	不选中
通道 2	
输出	直接输出
基础波形	Square
循环次数	选中无限
频率（MHz）	10MHz
相位（2pi）	0
幅度（mV）	100
偏置	0
占空比	0.5
模式	UP-DOWN

点击预览，可以看到配置的波形形状。

3. 配置好示波器的设置，在设备-播放模式-组合 1 下，点击下载和播放。在示波器上可以看到分别采集到了正弦波和方波。

4.4 任意波形产生实验

本章节将演示 4 个通道任意波形产生。打开上位机软件，并且连接设备，如下图所示，实验主页面进行详细的参数配置。

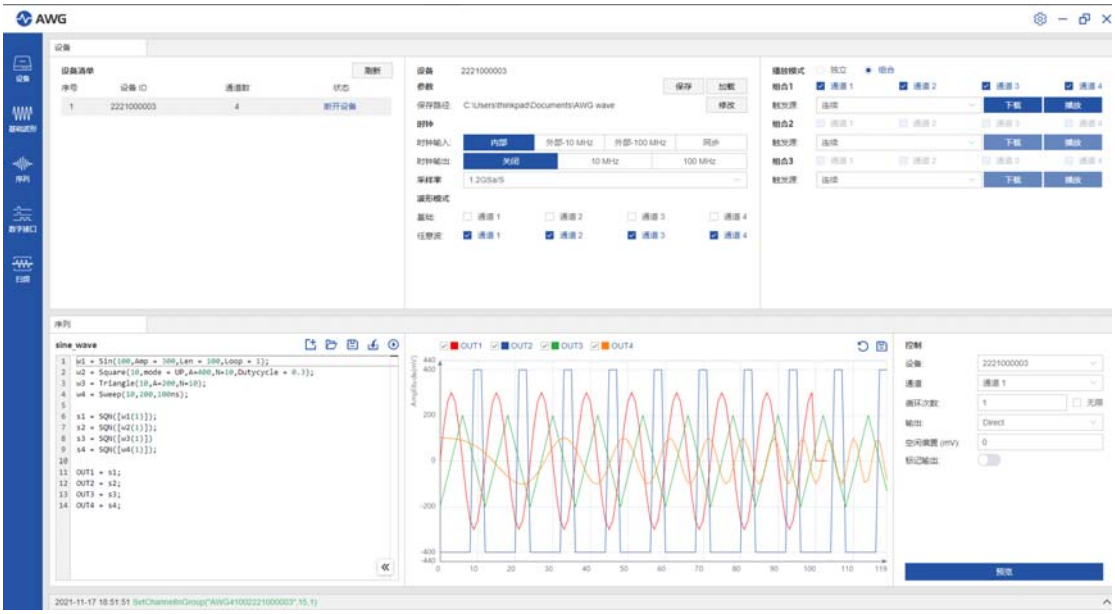

图 4.4.1 软件配置页面

4. 在设备页面下配置：

时钟	
时钟输入	内部
时钟输出	关闭
采样率	1.2GSa/s
波形模式	
基础	不选
任意波	选中通道 1、2、3、4
播放模式	
播放模式	组合
组合 1	通道 1、2、3、4 选中
触发源	连续

5. 序列设置

在代码编辑区编译以下代码

```
w1 = Sin(100,Amp = 300,Len = 100,Loop = 1);
w2 = Square(10,mode = UP,A=400,N=10,Dutycycle = 0.3);
```

```
w3 = Triangle(10,A=200,N=10);
w4 = Sweep(10,200,100ns);

s1 = SQN([w1(1)]);
s2 = SQN([w2(1)]);
s3 = SQN([w3(1)])
s4 = SQN([w4(1)]);

OUT1 = s1;
OUT2 = s2;
OUT3 = s3;
OUT4 = s4;
```

编辑完成后点击预览可以在绘图区看到编辑波形预览，本代码在 4 个通道分别产生正弦波、方波、三角波和扫频的波形。。

在控制栏下配置：

控制	
通道	分别配置通道 1、2、3、4
循环次数	选中无限
输出	Direct
空闲偏置（mV）	0
幅度（mV）	100
标记输出	不选中

6. 配置好示波器的设置，在设备-播放模式-组合 1 下，点击下载和播放。在示波器上可以看到 4 个通道分别采集到了正弦波、方波、三角波和扫频波形。

5.功能概述

本章节的目的是为了帮助您快速了解 AWG4100 的基本功能及面板介绍。

5.1 基本概述

AWG4100 是一款多通道宽带任意波形发生器，具有 4 个通道，可以选配 2 通道版，每个通道配备有单独的触发输入和 Mark 输出能力。AWG4100 具有 330MHz 模拟带宽,优于 1ns 的上升时间，较低的杂散、谐波水平，在科研、工业测试领域有广泛应用。

5.2 面板布局介绍

5.2.1 前面板

AWG4100 前面板布置参见如图 5.2.1 所示和表 5.2.1 说明。

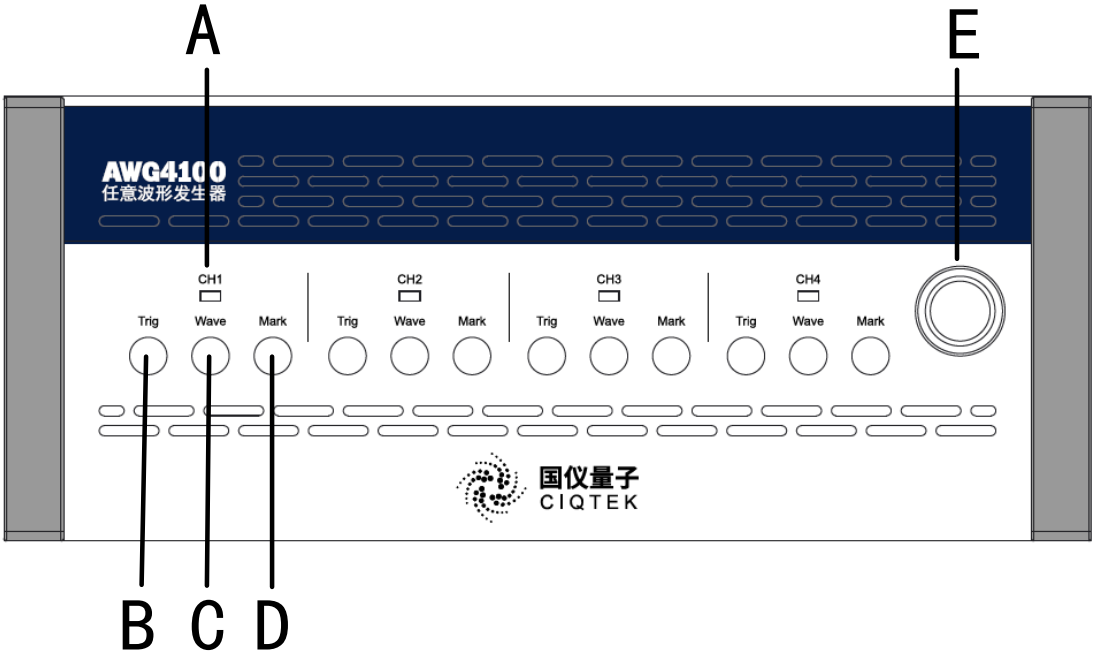

图 5.2.1 AWG4100 前面板示意图

表 5.2.1　AWG4100 前面板说明

编号	接口名称	描述
A	LED 指示灯	通道播放过程中,该 LED 灯会亮起
B	Trig	TTL 触发输入
C	Wave	单端波形信号输出
D	Mark	TTL Marker 信号输出
E	开关	开关、电源指示灯

5.2.2 后面板

AWG4100 后面板主要有电源，触发、时钟、通讯等相关的接口和功能，详细说明请参见如图 5.2.2 所示和表 5.2.2 说明。

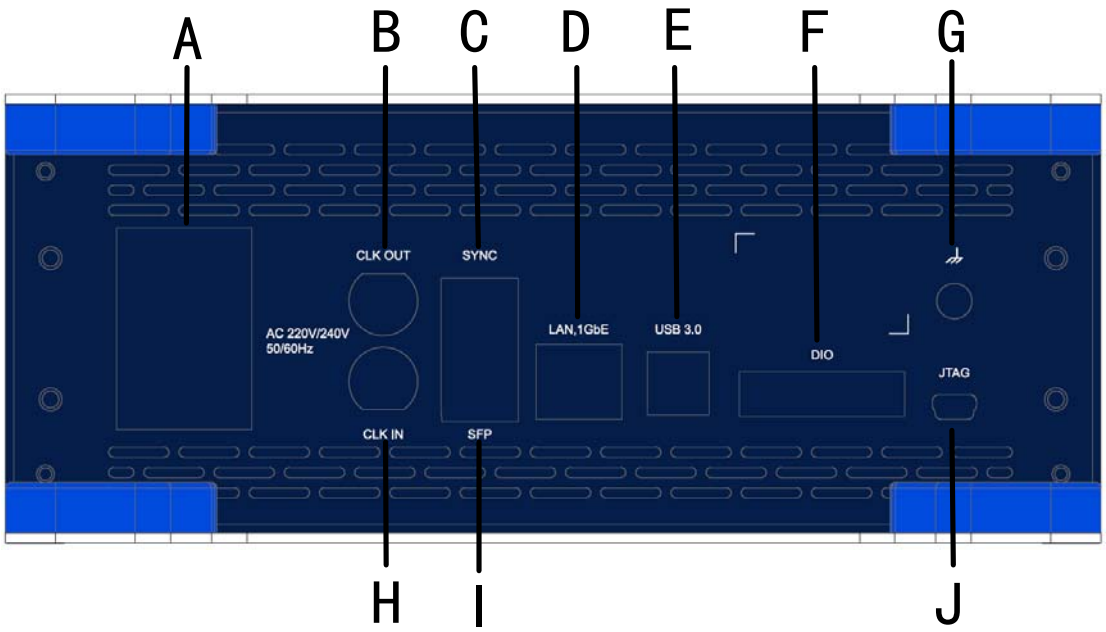

图 5.2.2 AWG4100 后面板示意图

表 5.2.2　AWG4100 后面板说明

位置	名称	描述
A	AC 220V	电源入口+电源开关
B	参考时钟输出	参考时钟输出(10MHz)与其他仪器同步
C	同步接口	仪器之间的同步总线连接器
D	LAN，1GBE	千兆网口(网线连接请用此端口)
E	USB 3.0	USB3.0 接口
F	DIO	8 位 DIO 数字输入端口
G	GND	接地端口
H	参考时钟输入	参考时钟输入(10/100 MHz)与其他仪器同步
I	SFP GBE	千兆光网络
J	JTAG	用于芯片内部测试以及对系统进行仿真、调试

注意

1．本产品采用强制风冷扇热设计，确保前后面板出风口与进风口通畅，不要用物体挡住进、出风口，否则可能使仪器内部温度过高，进而导致仪器的损坏。（最小距离不小于 10 cm。）

2．桌面安装时，工作台表面必须平坦，且仪器需要水平放置，不得使用侧面或者背面放置，以免损坏设备接口或者设备跌落的危险。

6.软件功能介绍

6.1 打开软件

6.1.1 设备和上位机连接

如果将设备和上位机通过网线直连，或者两者通过交换机进行连接时，都需要先确认自己的电脑 IP 配置情况，确保电脑通过 DHCP 方式或者手动配置方式配置了 IP 地址。通过打开上位机的“网络和 Internet“设置—”更改适配器选项“页面，找到对应的网络连接，鼠标右键打开”属性“—”Internet 协议版本 4（TCP/IPv4）“---”属性“下配置固定 IP 地址。

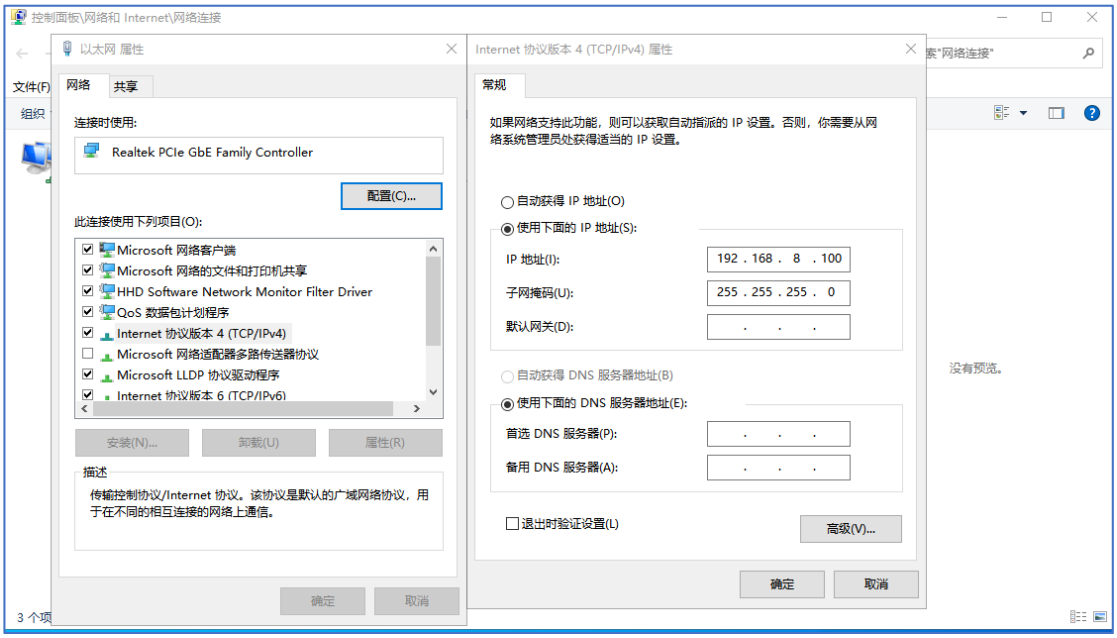

图 6.1.1 上位机配置 IP 地址

由于 AWG4100 和上位机软件之间可以通过跨网段连接，所以可以在保证 IP 地址不冲突的情况下，配置任何 IP，都可以搜索到设备。(设备出厂前会配置固定的 IP 地址，一般不可更改，可以通过上位机软件搜索到)。

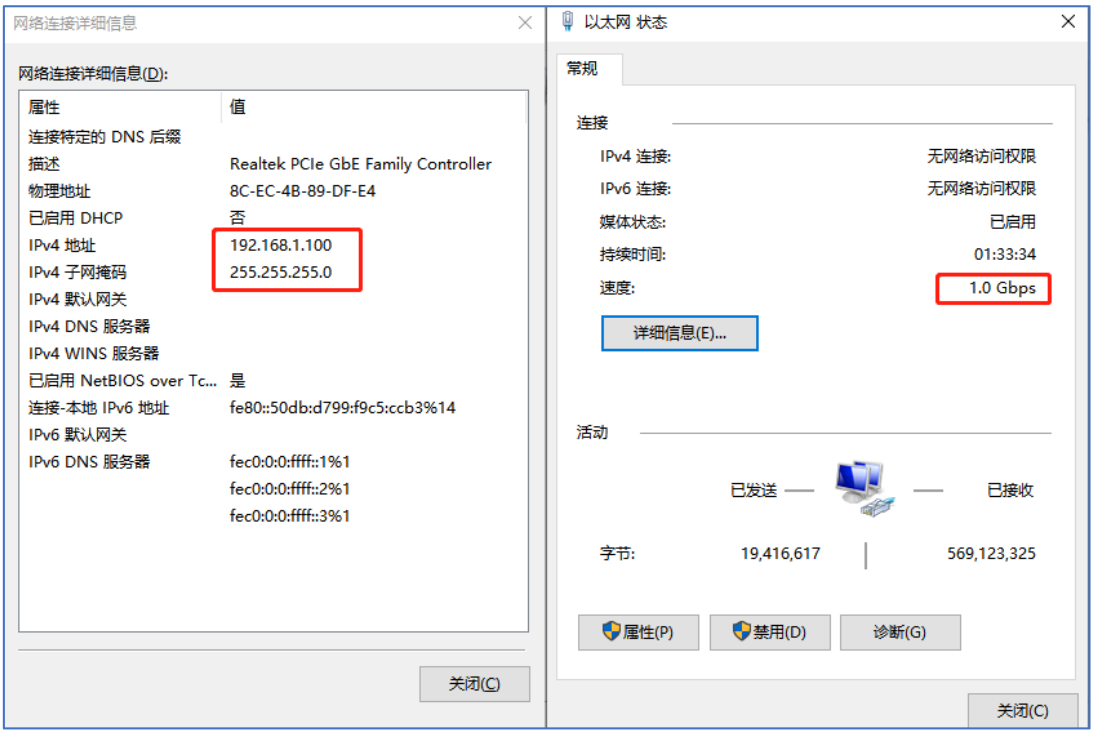

图 6.1.2 上位机软件安装 PC 的 IP 地址配置

如果设备直连的情况下，注意观察下图 6.1.2 设备的网速是否为 1.0 Gbps，为了保证更好的波形下载速度，建议使用 1.0 Gbps 网络环境。如果通过交换机连接的话，可以注意观察交换机和设备连接口千兆网连接指示灯是否亮起，如图 6.1.3。当然，AWG4100 设备目前同样也可以支持百兆网连接。

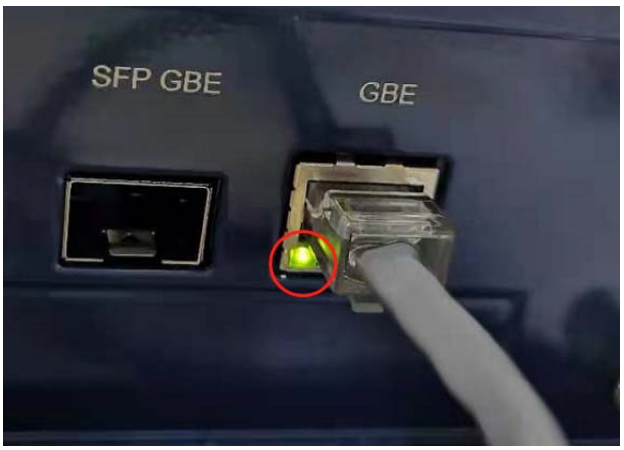

图 6.1.3 设备千兆网连接状态指示

如果上位机和设备之间是通过千兆路由器进行连接，则不需要进行 IP 地址配置，直接连接就可以使用。

6.1.2 打开软件

AWG4100 软件默认安装会在桌面建立快捷方式，可以双击桌面快捷方式打开软件，如下图 6.1.4 所示为 AWG4100 的上位机软件快捷图标。

图 6.1.4 设备快捷图标

此外还可以在开始菜单里面的软件列表里面找到 AWG4100 软件图标打开软件。打开软件后，软件首先进入设备的连接界面，界面如图 6.2.1 所示。

6.2 软件界面介绍

6.2.1 连接界面

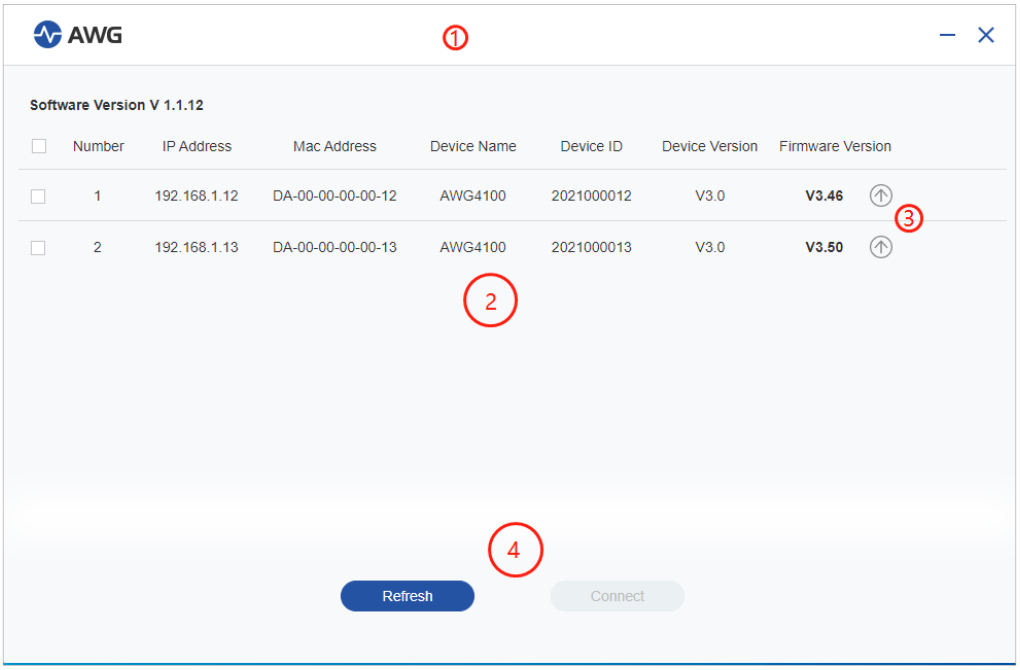

图 6.2.1 设备连接

1、窗口最小化和关闭区：

用户单击最小化按钮，将连接界面最小化，连接界面最小化后，用户可以单击 Windows 桌面下方的任务栏中 AWG4100 操作软件图标恢复连接界面。用户单击关闭按钮后，将退出 AWG4100 操作软。

2、当前操作软件版本号和需要更新状态显示区：

其中表示当前操作软件有可更新的软件版本（若没有可升级的软件版本则不显示），若需要更新，单击按钮即可进入软件更新流程，详情参见 1.8 章节介绍，若不需要更新忽略即可。

3、局域网搜索到设备列表：

该区域显示在局域中搜索到设备的 IP Address（IP 地址）、Mac Address（Mac 地址）、Device Name（设备名称）、Device ID（设备编码）、Device Version（硬件版本）、Firmware Version（固件版本号）、固件可更新状

态，用户可勾选需要连接的所有设备，支持一次选择一台或者多台设备，然后点击“Connect（连接）”按钮即可进入设备操作的软件界面。 其中表示有可更新的固件版本（若没有可升级的固件版本则不显示此控件），若需要更新，单击控件即可进入固件更新流程，详情参见 1.8 章节介绍，若不需要更新忽略即可。

4、设备搜索和设备连接区：

“Refresh（刷新）”按钮：用户单击该按钮，操作软件再次在局域网内搜索可连接的设备，搜索后更新设备展示列表中的设备信息。“Connect（连接）”按钮：用户单击该按钮，执行软件和对应硬件设备之间的连接（默认此按钮为禁用状态，当用户选中设备展示列表中的一台或者多台设备后启用该按钮），用户可通过移动鼠标单击选中所需要连接的设备，点击“Connect（连接）”按钮即可进入设备操作的软件界面。

6.2.2 操作主页面

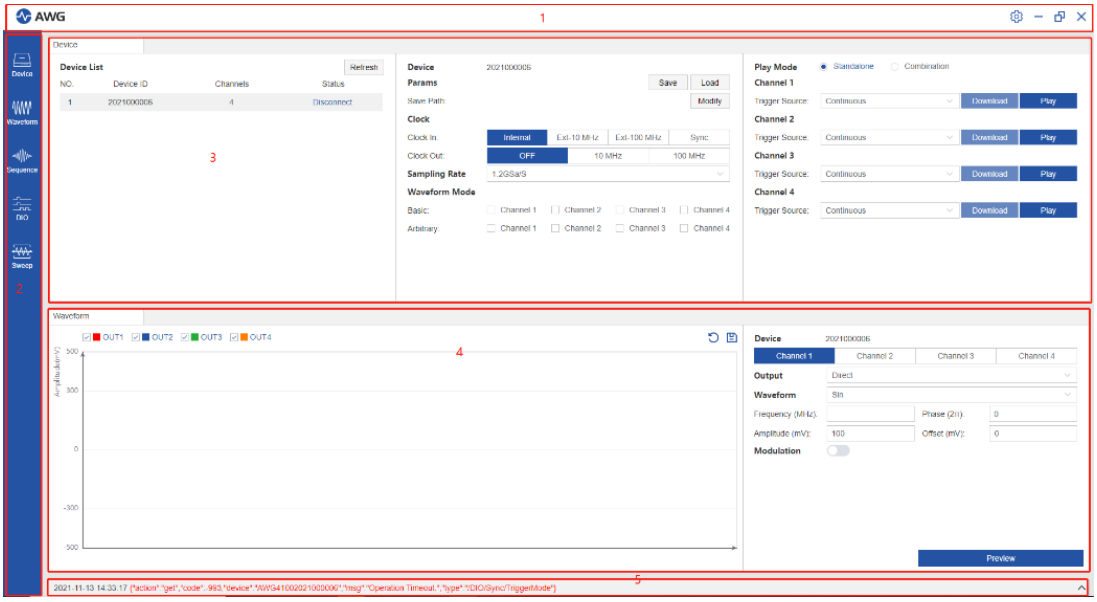

图 6.2.2 操作主页面

操作主页面主要分为 5 个基本功能区：

1、窗口操作区：主要有设备设置，软件窗口的最小化、最大化，关闭操作等功能。

2、设备功能选项区：主要有“Device”、“Waveform”、“Sequence”、“DIO”、“Sweep”等功能。

3、功能操作展示区（上半区）：“Device”、“DIO”、“Sweep”功能默认打开在此区域展示。

4、功能操作展示区（下半区）：“Waveform”、“Sequence”功能默认打开在此区域展示。

注：功能操作展示区（上半区）和（下半区）在有多个功能选项卡时，可以将鼠标移动对应的选项卡上，长按鼠标左键，拖动选项卡切换功能选项显示区域，另外功能操作展示区（上半区）和（下半区）的每个区域只有一个选项卡时无法拖动或者关闭相应的功能选项卡。

5、运行操作日志展示区：本区域主要用于展示运行操作的指令、反馈状态、错误告警等日志信息。

注意：在关闭软件或断开连接后重新来打开连接进入操作页面，软件会自动读取设备参数并显示（自定义代码除外）。

6.2.2.1 窗口操作区

图 6.2.3 窗口操作区

1、⚙为设备和软件状态设置按钮，单击该按钮进入设置界面，详细介绍请见 6.2.8 章节。

2、－ ⧉ × 软件窗口的最小化、切换小窗口显示、关闭操作

单击－按钮，将软件操作界面最小化，用户可以单击 Windows 桌面下方的任务栏中 AWG4100 操作软件图标恢复软件操作界面。单击按钮，将 AWG4100 操作软件主界面由最大尺寸显示（1920*1080）切换至最小尺寸显示（1600*900）。单击⧉按钮，将 AWG4100 操作软件主界面由最小尺寸显示（1600*900）切换至最大尺寸显示（1920*1080）。单击×按钮，将 AWG4100 操作软件断开与设备的连接并退出软件。

6.2.2.2 设备功能选项区

本章主要是设备功能选项区的功能选项简介，详情参见表 6.2.4 介绍，各功能选项的详细介绍下述各功能详细介绍章节。

表 6.2.4 设备功能选项区功能简介

功能	功能选项控件	描述
设备设置	Device	设备的时钟、采样率、波形模式和播放方式等功能的设置
基础波形设置	Waveform	对于每个通道，如果设置为基础波形，进行每种基础波形的参数详细设置
波形序列设置	Sequence	对于每个通道，如果设置为任意波形，则可以通过波形序列进行波形的详细设置

外部输入设置		设置外部输入接口相关参数，包括触发、数字输入和计数等功能
扫频波形设置		对于每个通道，如果设置了扫频波形，则可以进行详细的扫频波形参数设置

6.2.2.3 运行操作日志展示区

运行操作日志展示区默认只显示一行，进行操作日志和各种告警报错信息的展示，点击按钮展开日志信息，如下图所示。在展开状态下，可以再次收起该区域，在展开状态下，可以清空该区域的全部日志信息。支持将日志区域的操作日志拷贝到 Python 主程序中作为 SDK 调用，可以点击每条日志前面的“Copy”按钮拷贝当前语句。

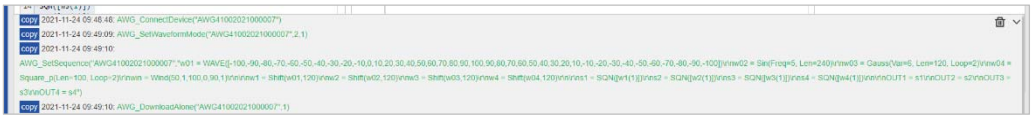

图 4.2.4 日志区域展开

6.2.3 “Device”功能介绍

“Device”功能主要完成是对已连接仪器的工作时钟、运行参数等功能的设置。用户可在操作软件的左侧功能列表中单击 

功能键，即可打开“Device”功能。此功能选项卡默认在功能操作展示区（上半区）打开此功能，如图 6.2.5 所示。

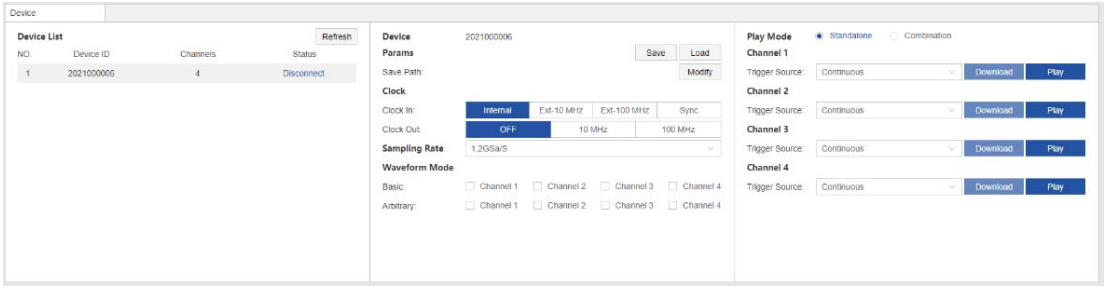

图 6.2.5 Device 功能

1、Device List

该区域展示当前搜索到的所有设备信息，默认优先显示已经选中的设备。每个设置显示的信息包括：Device ID、Channels 和 Status。所有设备中有一个设备为当前选中设备，右侧展示当前选中设备的相关信息。对于每个已经连接的设备，支持通过点击 Status 下面的“Disconnect”来断开连接，断开连接后，则 Status 变为“Connect”；对于没有连接的设备，支持通过点击 Status 下面的“Connect”来连接设备，连接成功后，则 Status 变为“Disconnect”。

对于列表区域，支持点击“Refresh”按钮，再次在局域网内进行设备的搜索，并且更新显示设备列表。

2、Device

该字段显示为当前选中的设备的设备编码。

3、Params

用户可在此区域通过单击“Modify”来修改参数保存的文件夹，选择好文件夹之后，可以通过单击“Save”将当前已经设置好的参数保存到指定文件夹。也可以通过单击“Load”来在设置好的文件夹下选择已有的参数文件进行加载。

4、Clock

用户可以在此功能区设置设备的工作时钟来源和时钟频率，默认采用的是内部时钟。也可以设置设备对外输出的时钟信号，默认不对外输出时钟信号。

如果“Clock In”选择了“Sync”选项，则“Trigger In”的“Source”支持选择“Sync”选项。另外在使用同步接口作为时钟输入的场景下，为了保证输出的时钟相位和同步接口源系统的时钟相位一致，则必须先切换好时钟输出，再切换时钟输入的值为“Sync”。

5、Sampling Rate

设备工作采样率为 1.2 GSa/s。

6、Waveform Mode

用户可以在此功能区为每个通道分别选择对应的波形模式，波形模式包括“Basic”和“Arbitrary”。对于某个通道，如果选择了基础波形“Basic”，则该通道的波形参数通过“Waveform”功能选项区进行设置；如果选择了任意波形“Arbitrary”，则该通道的波形参数通过“Sequence”功能选项区进行设置。

7、Play Mode

用户可以在该功能区域进行设备的波形播放方式的设置。播放方式包括：“Standalone”和“Combination”，默认为“Standalone”模式。

如果用户选择了“Standalone”模式，则该设备对应的每个通道都是需要单独设置“Trigger Source”，并进行下载“Download”和播放“Play”。“Trigger Source”包括：“Continuous”、 “Trigger Input1~2/4”，默认是“Continuous”连续模式，不需要外部触发即可进行工作。如果需要外触发模式，每个通道都可以分别选择自己的不同触发源。如果是两通道的设备，则可以选择的触发源也有两个；如果是四通道的设备，则可以选择的触发源也有四个。不同通道都可以同时进行下载和播放，不同通道也可以分别选择不同的“Trigger Source”。

如果某个通道处于播放状态，则这时候不支持对通道的波形参数进行修改，也不允许进行波形的再次下载。

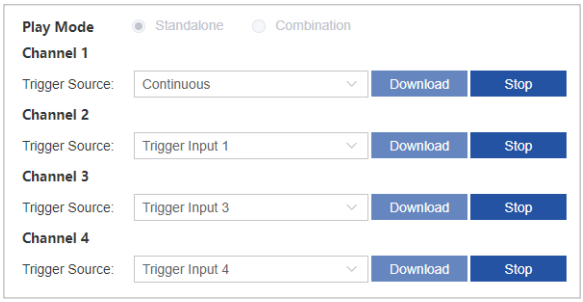

图 4.2.6 “Standalone”功能

如果用户选择了“Combination”模式，则可以将所有通道进行分组。如果当前设备为四通道设备，则可以至多分为三个组；如果当前设备为两通道设备，则可以将两个通道作为一个组。对于某个组进行“Download”或者“Play”，则对该组内所有通道的参数都进行下载或者波形播放。不同组可以同时进行下载和播放，不同组也可以分别选择不同的“Trigger Source”。

如果某个组处于播放状态，则这时候不支持对改组所有通道的波形参数进行修改，也不允许进行波形的再次下载。

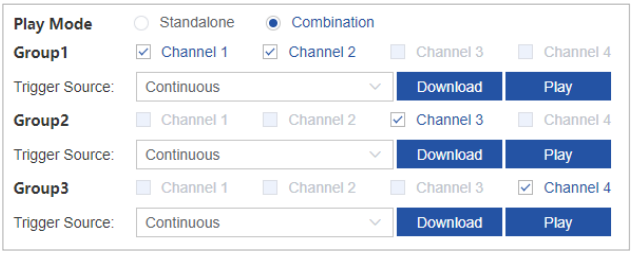

图 4.2.7“Combination”功能

6.2.4 “Waveform”功能介绍

“Waveform”功能主要完成每个通道基础波形的参数详细设置。用户可在操作软件的左侧功能列表中单击  功能键，即可打开“Waveform”功能。此功能选项卡默认在功能操作展示区（下半区）打开此功能，如图 6.2.8 所示。该功能分为两个区域，右侧是参数设置区域，左侧是波形预览区域。

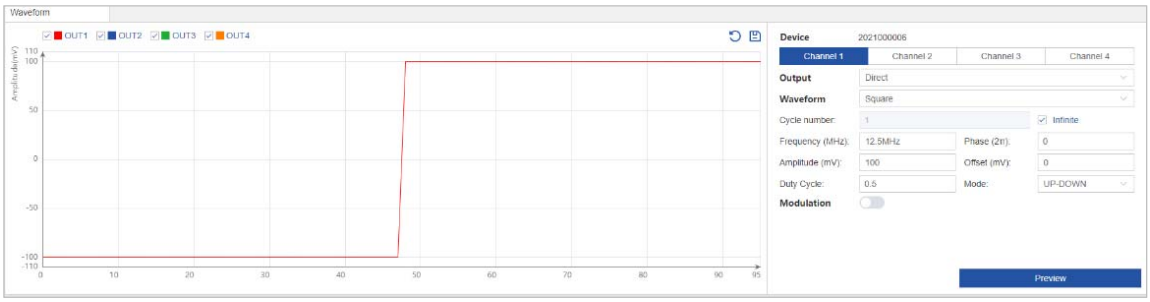

图 6.2.8“Combination”功能

6.2.4.1 参数设置区域

用户可以在该功能区域进行每个通道基础波形的参数设置，选择了某一个通道后，可以设置“Output”和“Waveform”。

“Output”可以设置为“Direct”或者“Amplifier(✕5)”，如果选择“Direct”,则输出的幅度范围为±500 mV；如果选择“Amplifier(✕5)”，则输出的幅度范围为±2.5V。

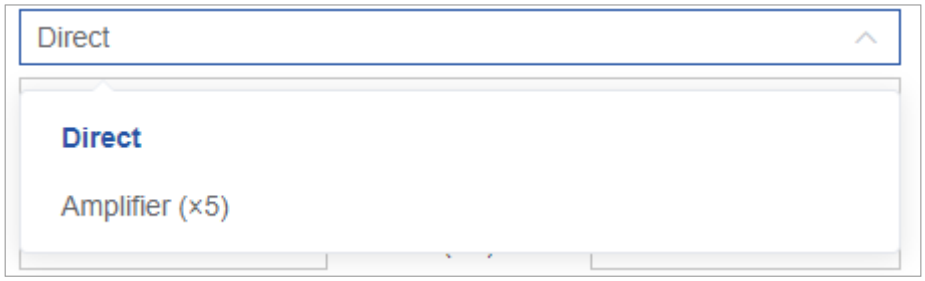

图 6.2.9 Output 功能下拉框

“Waveform”可以设置为不同的波形类型，包括：Sin（正弦）、Square（方波）、Gauss（高斯）、Triangle（三角波）、DC（直流）、PRBS（伪随机码）等几种波形。

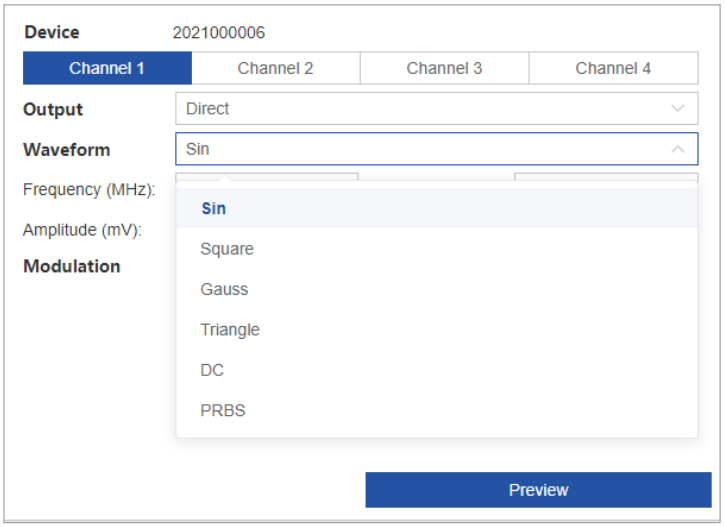

图 6.2.10 Waveform 下拉框清单

1、正弦波 Sin

如果选择的基础波形为正弦波 Sin，则可以通过设置“Frequency”、“Phase”、“Amplitude”和“Offset”四个参数来控制输出的波形。

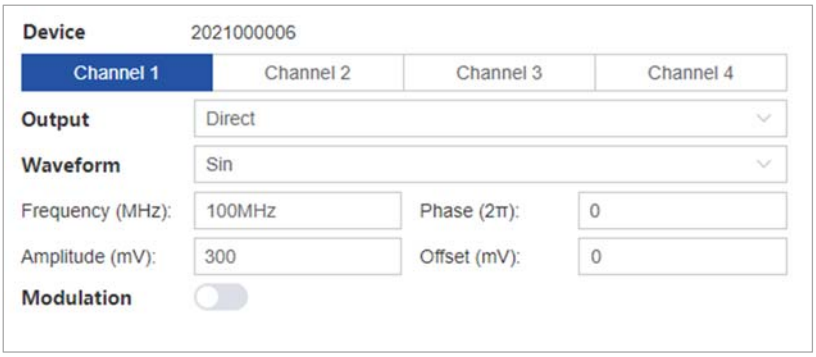

图 6.2.11 正弦波 Sin 参数设置

对于正弦波，支持进行调制，打开“Modulation”开关后，支持通过设置“Cycle Number”、“Mod Type”、“Mod Waveform”、“Mod Frequency”和另一个参数来进行设置。

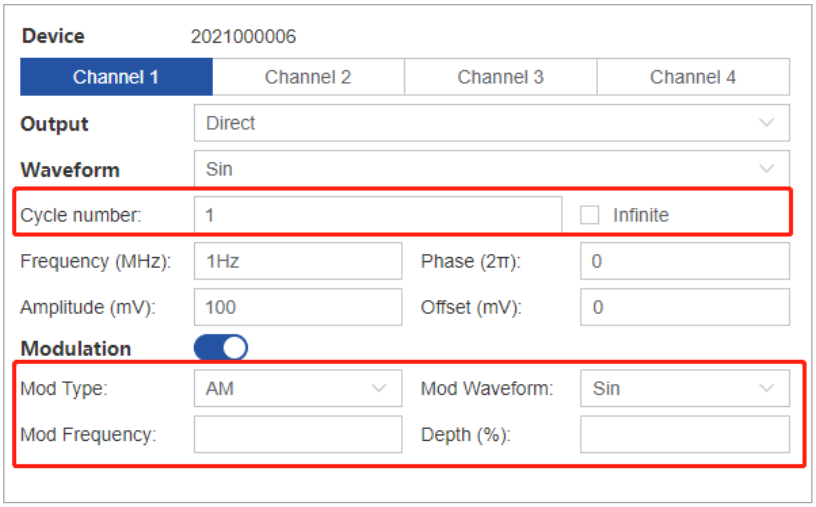

图 6.2.12 正弦波调试参数设置

2、方波 Square

如果选择的基础波形为方波 Square，则可以通过设置“Cycle Number”、“Frequency”、“Phase”、“Amplitude”、“Offset”、“Duty Cycle”和“Mode”七个参数进行波形的设置。

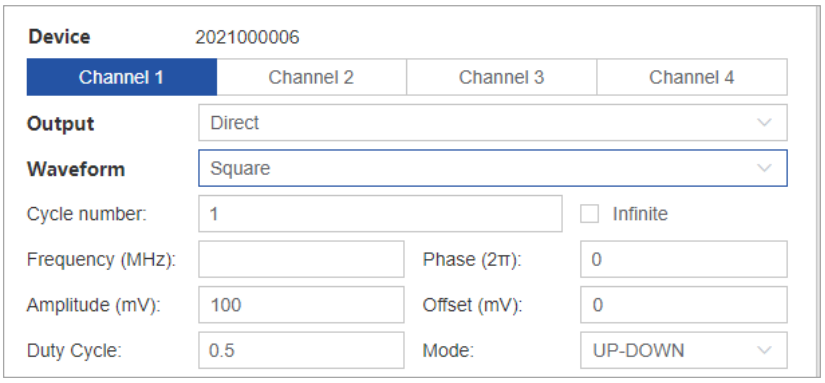

图 6.2.13 方波 Square 参数设置

3、三角波 Triangle

如果选择的基础为三角波 Triangle，则可以通过设置“Cycle Number”、“Time”、“Offset”、“Amplitude”、“Symmetry”和“Mode”六个参数进行波形的设置。

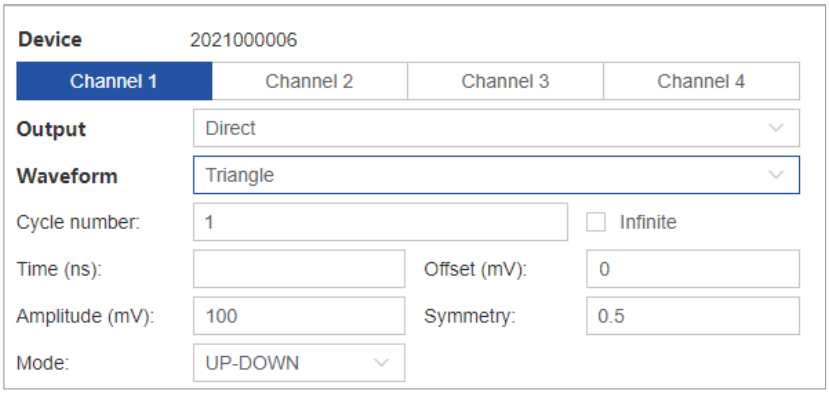

图 6.2.14 三角波 Triangle 参数设置

4、高斯 Gauss

如果选择的基础为高斯 Gauss，则可以通过设置“Cycle Number”、“Variance”、“Length”、“Amplitude”和“offset”五个参数进行波形的设置。

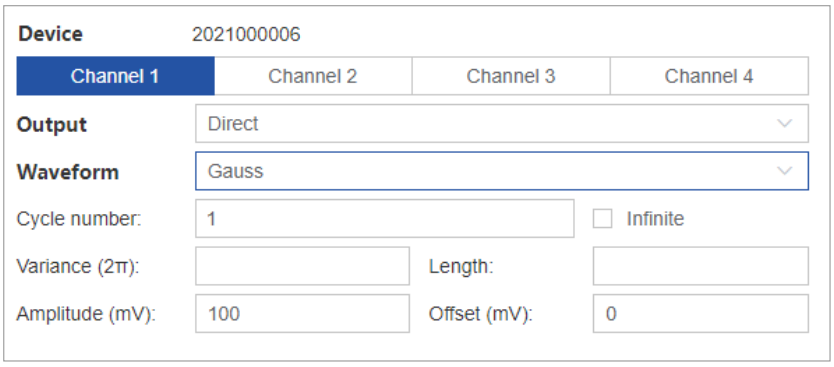

图 6.2.15 高斯 Gauss 参数设置

5、直流 DC

如果选择的基础为直流 DC，则可以通过设置“Cycle Number”、“Amplitude”和“Length”三个参数进行波形的设置。

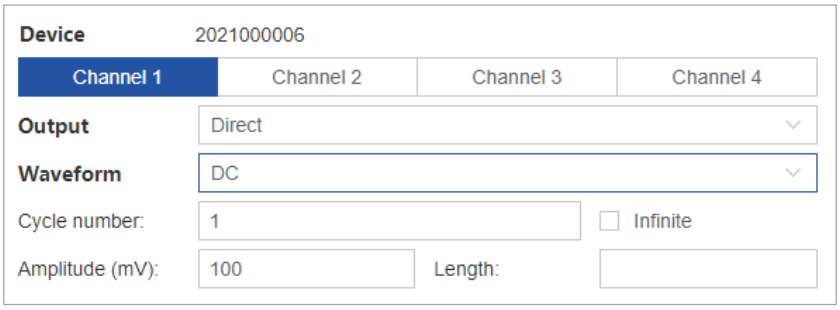

图 6.2.16 直流 DC 参数设置

6、伪随机码 PRBS

如果选择的基础为伪随机码 PRBS，则可以通过设置“Cycle Number”、“Bit Rate”和“Order”三个参数进行波形的设置。

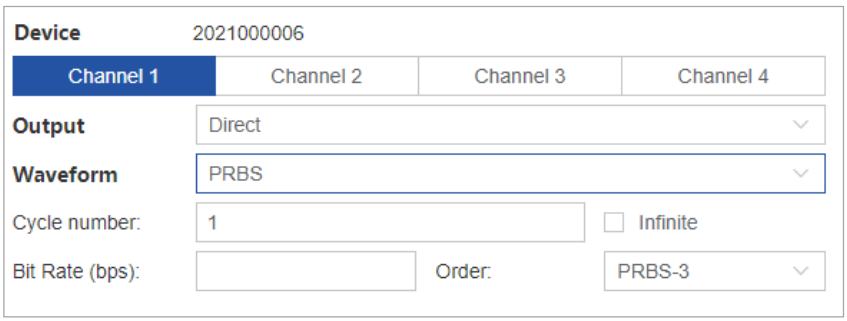

图 6.2.17 伪随机码 PRBS 参数设置

针对以上所有设置的参数的详细说明如下表所示：

表 6.2.5 Waveform 功能参数详细说明

功能	选项	描述
Sin	Frequency	频率，取值范围为(0,330 MHz]，最小值为 1 Hz
	Phase	相位，取值范围为[0,1]
	Amplitude	幅度，取值范围为(0,500 mV]
	Offset	偏置，取值范，，围为[-500 mV,500 mV]， |Amplitude|+|Offset|不大于 500 mV
	Modulation	调制开关，默认为关闭状态
	Cycle Number	循环次数，只是输入[1, 4294967295]，默认值为 1，支持勾选 Infinite，进行无限循环
	Mod Type	调制类型，支持选择调幅 AM、调频 FM 和调相 PM。 如果调制类型为 AM，则可以设置 Depth：调制深度，用百分比表示，取值范围为：0～120； 调制后，波形幅度不能超多 500 mV； 如果调制类型为 FM，则可以设置 Offset，频率偏置取值范围为：[0,330 MHz]，频率偏置+调制频率不大于 330 MHz，默认为 0 Hz； 如果调制类型为 PM，则可以设置 Offset，相位偏置取值范围为：[0,1]，默认值为 0。
	Mod Waveform	调制波形，当前支持的为正弦波
	Mod Frequency	调制频率，取值范围为：[0,330 MHz]，频率偏置+调制频率不大于 330 MHz
Square	Cycle Number	循环次数，只是输入[1, 4294967295]，默认值为 1，支持勾选 Infinite，进行无限循环
	Frequency	频率，取值范围为(0,330 MHz]，最小值为 1 Hz

	Phase	相位，取值范围为[0,1]
	Amplitude	幅度，取值范围为(0,500 mV]
	Offset	偏置，取值范围为[-500 mV,500 mV]，Amplitude+Offset 的绝对值不大于 500 mV
	Duty Cycle	占空比，取值范围为（0，1），默认值为 0.5
	Mode	方波模式，值为 UP/DOWN，UP 是先低后高，DOWN 是先高后低；默认值为 UP
Triangle	Cycle Number	循环次数，只是输入[1, 4294967295]，默认值为 1，支持勾选 Infinite，进行无限循环
	Time	三角波的周期，参数输入范围为：0-447392426 ns
	Offset	偏置，取值范围为[-500 mV,500 mV]，Amplitude+Offset 的绝对值不大于 500 mV
	Amplitude	幅度，取值范围为(0,500 mV]
	Symmetry	对称性，取值范围为（0，1），默认值为 0.5
	Mode	方波模式，值为 UP/DOWN，UP 是先低后高，DOWN 是先高后低；默认值为 UP
Gauss	Cycle Number	循环次数，只是输入[1, 4294967295]，默认值为 1，支持勾选 Infinite，进行无限循环
	Variance	高斯方差
	Length	采样点数，高斯函数的中心在 Len/2 处；默认值采样率范围为[0，5*Variance]
	Amplitude	幅度，取值范围为(0,500 mV]
	Offset	偏置，取值范围为[-500 mV,500 mV]，Amplitude+Offset 的绝对值不大于 500 mV

DC	Cycle Number	循环次数，只是输入[1, 4294967295]，默认值为1，支持勾选 Infinite，进行无限循环
	Amplitude	幅度，取值范围为(0,500 mV]
	Length	采样点数，如果采样点数不是 8 的倍数，后面不足的部分会自动补 0，最小点数为 56
PRBS	Cycle Number	循环次数，只是输入[1, 4294967295]，默认值为1，支持勾选 Infinite，进行无限循环
	Bit Rate	比特率，单位为 bps；参数输入范围为：[6.8 bps，300 Mbps]；
	Order	编码方式，可选值包括：PRBS-3、PRBS-7、PRBS-9、PRBS-11、PRBS-15、PRBS-23，默认值为 PRBS-3；（2^N-1）*SR/BRate 不能大于 512M

6.2.4.2 波形预览区域

用户在参数设置区域完成每个通道的波形选择和设置后，可以点击“Preview”按钮在该区域进行波形预览。预览时可以勾选不同的通道进行预览，横轴为点数，纵轴为幅度。滚动鼠标可以进行横轴的缩放显示；按住键盘“Shift”再进行鼠标滚动，可以进行纵轴的缩放显示。当图形缩放后，可以通过点击右上角的↻，回到波形预览的最初状态。

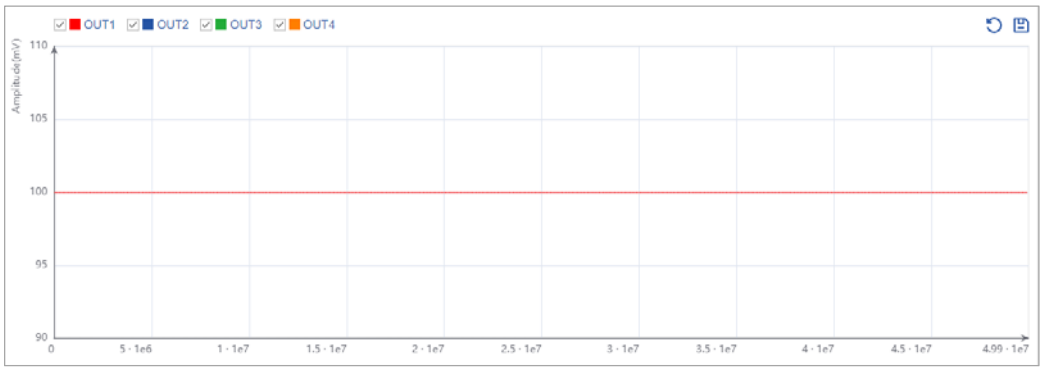

图 6.2.18 预览功能区域

在预览区域的右上角有保存按钮，点击后弹出保存对话框，如下图。在保存对话框中可以勾选需要保存的当前正在预览的通道的数据，可以勾选保存 wave 和 txt 格式，对于保存的文件名支持进行修改。

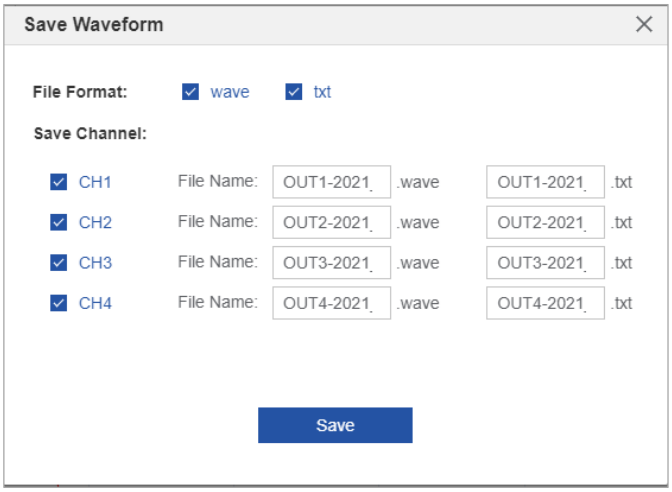

图 6.2.19 预览波形保存

4.2.5 “Sequence”功能介绍

“Sequence”功能主要完成任意波形模式的波形编辑和参数设置。用户可在操作软件的左侧功能列表中单击功能键，即可打开“Sequence”功能。此功能选项卡默认在功能操作展示区（下半区）打开此功能，如图 6.2.5-1 所示。该功能模块分为波形编辑、波形预览和参数设置三个区域。

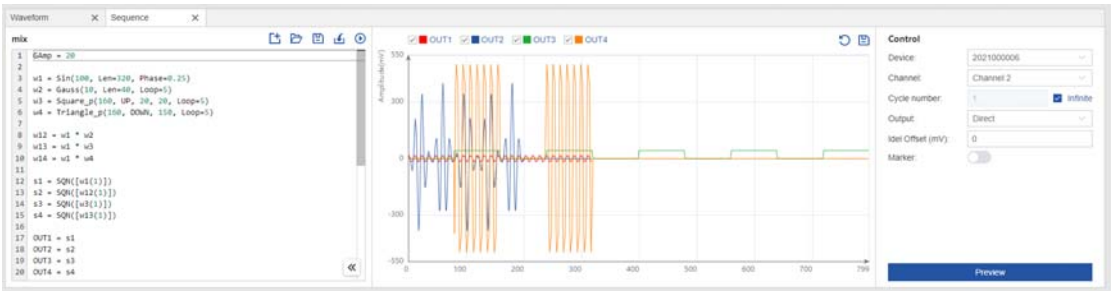

图 6.2.20 Squence 功能模块

4.2.5.1 参数设置区域

用户可以在该功能区域进行任意波形的参数设置，可以设置的参数包括：“Device”、“Channel”、“Cycle Number”、“Output”、“Idle Offset”和“Marker”。每个参数的详细说明如下表所示：

表 6.2.6 Sequence 功能参数详细说明

功能	选项	描述	备注
Control	Device	可以从已经连接的所有设备中进行选择	
	Channel	对于已经选中的 Device，这里可以选中其某个通道进行参数设置，下面的几个但是都是针对该选中的通道	
	Cycle Number	循环次数，只是输入[1, 4294967295]，默认值为 1，支持勾选 Infinite，进行无限循环	
	Output	输出包括 Direct 和 Amplifier(✕5)两个选择项，如果选中的是 Direct，则最大输出幅度为 500 mV；如果选中的是 Amplifier(✕5)，则最大输出幅度为 2.5 V。	
	Idle Offset	空闲偏置，对于没有波形输出时，可以输出的低电平的幅度。取值范围为[0,500 mV]。	

	Marker	Marker 信号开关，默认为关闭状态。如果是关闭状态，即使在波形代码中编辑了 Marker 波形，也不能输入；如果该状态为打开状态，则波形代码编辑的 Marker 波形可以正常输出。	

6.2.5.2 波形编辑区域

用户可以在该功能区域通过编写代码的方式，进行波形的编辑，并且进行编译、下载和播放。在进行代码编辑时，支持同时对多台已经连接的设备进行波形编辑。

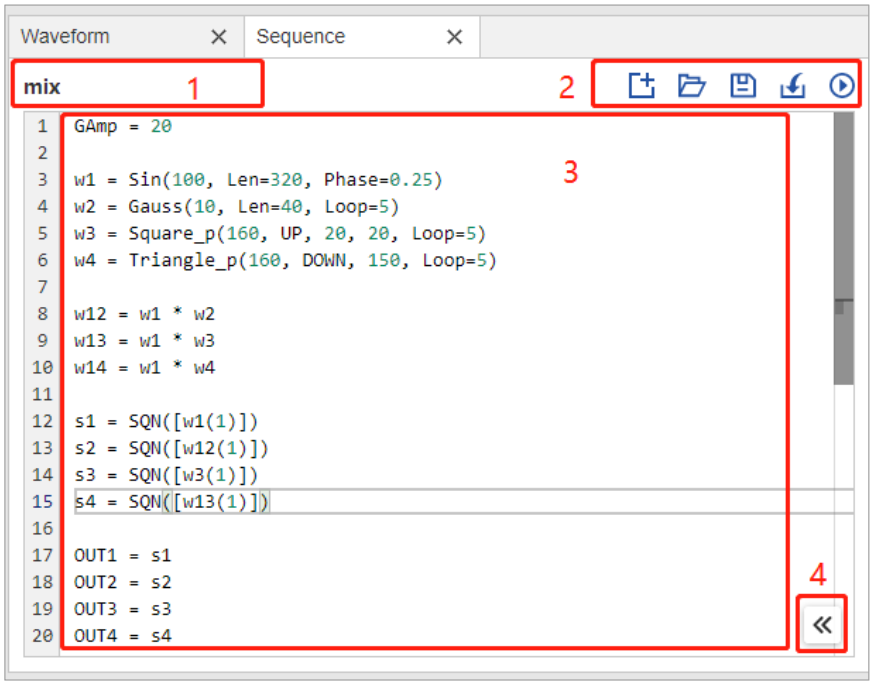

图 6.2.21 波形编辑区域

1、代码编辑区域 1 显示为当前显示的代码文件名称。

2、代码编辑区域 2 是操作区域，支持的操作包括新建文件、加载代码文件、保存代码文件、下载波形代码和播放。

点击，会将当前打开的文件清空，新建一个波形代码文件，默认文件名为 wave1.cw 文件，该文件进行保存到本地时，会自动在 wave1 后面增加上时间戳。

点击，会打开默认选中的文件夹，支持用户进行文件夹的变动，并在文件夹中选中某个波形代码文件，双击后在波形编辑区域打开该文件。

点击，将当前代码编辑区域的代码文件保存到默认文件夹，支持用户进行文件夹的修改，支持对保存文件名称的修改。

点击，将波形代码下载到设备，下载之前会先对波形代码进行编译，编译结果会显示在区域 4。如果代码中有多台设备，则会将每台设备对应的波形下载到对应的设备。

点击，将已经下载到设备的波形在设备进行输出播放，如果在代码中有多台设备，则多台设备同时进行播放。

3、代码编辑区域 3 为用户编写波形代码的区域，该区域支持 Python 语言，对于自定义的波形相关函数，可以按照波形代码定义章节描述进行波形的编辑。

4、代码编辑区域 4 为波形代码的编译结果，默认为收缩状态，编译后自动展示显示编译结果，支持点击后收起。

图 6.2.22 波形编译结果展示

6.2.5.3 波形预览区域

该区域的波形预览功能同 6.2.4.2 章节。

6.2.6 “Sweep”功能介绍

“Sweep”功能主要完成扫频 Sweep 的波形参数设置。用户可在操作软件的左侧功能列表中单击功能键，即可打开“Sweep”功能。此功能选项卡默认在功能操作展示区（上半区）打开此功能，如图 6.2.23 所示。

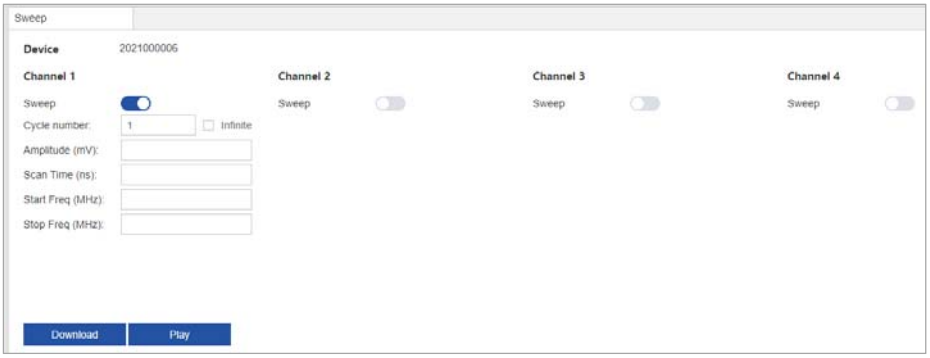

图 6.2.23 Sweep 功能区域

该区域对于已经选中的设备，支持对设备的所有通道进行扫频的开关设置，默认所有通道的 Sweep 功能都为关闭状态。打开某个通道 Sweep 开关后，可以进行设置的参数包括：Cycle Number、Amplitude、Scan Time、Start Freq 和 Stop Freq，具体每个参数的详细信息如下表所示：

表 6.2.7 Sweep 功能参数详细说明

功能	选项	描述
Sweep	Cycle Number	循环次数，只是输入[1, 4294967295]，默认值为 1，支持勾选 Infinite，进行无限循环
	Amplitude	幅度，取值范围为(0,500 mV]
	Scan Time	扫频时长，单位可以是 ns、us、ms 和 s，支持在输入框中输入数字+单位；取值范围为：74 ns~445 ms

	Start Freq	扫频开始频率，单位为 MHz；取值范围为：（0-330 MHz）
	Stop Freq	扫频开始频率，单位为 MHz；取值范围为：（0-330 MHz] Stop Freq 必须大于 Start Freq。

6.2.7 “DIO”功能介绍

“DIO”功能主要用于进行外部输出接口相关的参数设置。用户可在操作软件的左侧功能列表中单击功能键，即可打开“DIO”功能。此功能选项卡默认在功能操作展示区（上半区）打开此功能，如图 6.2.24 所示。

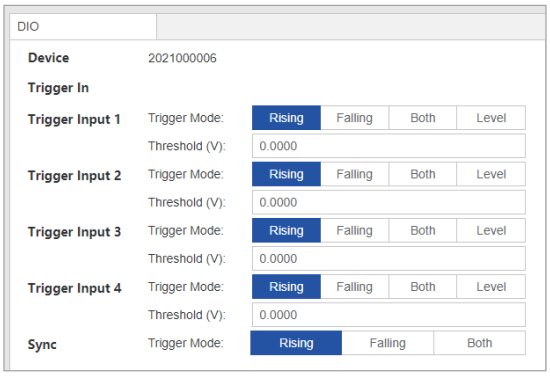

图 6.2.24 DIO 功能区域

DIO 功能区域可以对 Trigger In 口和 Sync 口的通道属性进行设置，对于四通道的设备，则有四个 Trigger In 通道，如果是两通道的设备，则有两个 Trigger In 通道。针对每个 Trigger In 通道可以设置的参数包括 Trigger Mode、Threshold。每个设备都有一个 Sync 接口，Sync 接口可以作为设备的时钟输入和触发输入，对于 Sync 作为触发功能时的 Trigger Mode 参数设置。每个参数的详细描述见下表：

表 6.2.8 DIO 功能参数详细说明

功能	选项	描述
Trigger In	Trigger Mode	触发模式，每个 Trigger In 通道支持的触发模块包括：Rising（上升沿触发）、Falling（下降沿触发）、Both（上升沿和下降沿同时触发）、Level（电平触发）
	Threshold	触发阈值，单位为 V，支持输入的参数范围为 0～5 V，输入精度为 0.1 mV
Sync	Trigger Mode	触发模式，Sync 接口支持的触发模块包括：Rising（上升沿触发）、Falling（下降沿触发）、Both（上升沿和下降沿同时触发）

6.2.8 “Setting”功能介绍

“Setting（设置）”功能主要显示用户当前操作软件和已连接设备信息等。用户单击按钮后，弹出“设置功能界面”，如图 6.2.25 所示，主要包括“Mointor”、“About”和“Language”三个功能项。

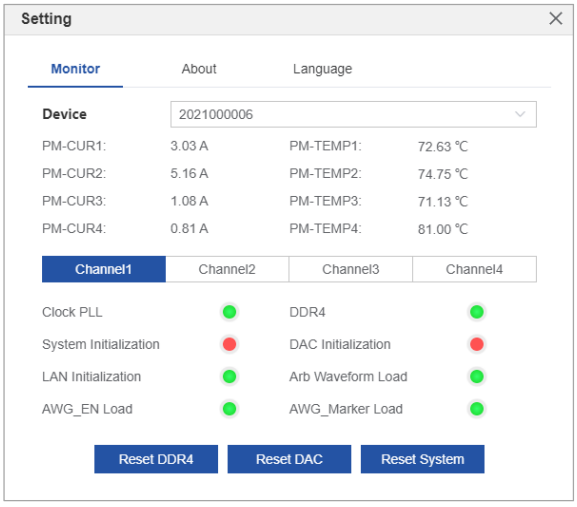

图 6.2.25 “Setting”功能界面

6.2.8.1“Monitor”功能

“Monitor”功能区域主要包括上下两部分，首先可以进行已经连接的所有设备的选择，选中某个设备之后，可以进行设备的温度、电流的监控；可以对设备的所有通道的状态进行监控。

1、温度监控

对于每台设备，进行如下表所示的四处关键温度的监控：

表 6.2.9 温度监控表

序号	温度监控处	含义
1	PM-TEMP1	母板 FPGA 环境温度
2	PM-TEMP2	母板电源温度
3	PM-TEMP3	子卡电源温度
4	PM-TEMP4	子卡放大电路温度

2、电流监控

对于每台设备，进行如下表所示的四处关键温度的监控：

表 6.2.11 电流监控表

序号	电流监控处	含义
1	PM-CUR1	总电流
2	PM-CUR2	内核电流 1.0
3	PM-CUR3	子卡电流 12
4	PM-CUR4	内存电流 1.2

3、通道状态监控

设备的状态显示在设置栏中，绿色表示正常，红色表示异常。

“Clock PLL”：表示该通道的时钟状态是否锁定，绿色表示锁定，红色表示未锁定。

“DDR4”：AWG4100 每个通道有一片独立编址的 DDR 颗粒，当相应通道 DDR4 正常工作时，显示为绿色，异常为红色。

“System Initialization”：系统在设备上电时，会加载相应的系统参数，当加载成功时候，显示绿色，失败为红色。

“DAC Initialization”：当设备上电时，对 DAC 数模转化芯片进行配置，当配置成功，显示绿色，配置失败显示红色。

“LAN Initialization”：当网络正常连接时，状态显示绿色，异常连接显示红色。

“Arb Waveform Load”、“AWG_EN Load”和“AWG_Marker Load”为 AWG 模式下载自定义波形的诊断状态，当所有的状态显示绿色的时候，才能播放正确的波形数据。当诊断自定义的波形状态时候，可以查看这三个状态提示。

“Reset DDR4”、“Reset DAC”和“Reset System”分别表示在 DDR4、DAC 或者系统出现故障时，支持通过点击对应的按钮进行复位操作。

6.2.8.2“About”功能

“About”界面显示设备的软件、硬件、固件的版本信息，同时本页面可以提供更新软件和设备固件的功能。具体的软件升级和固件升级功能可以参考第 3 章。

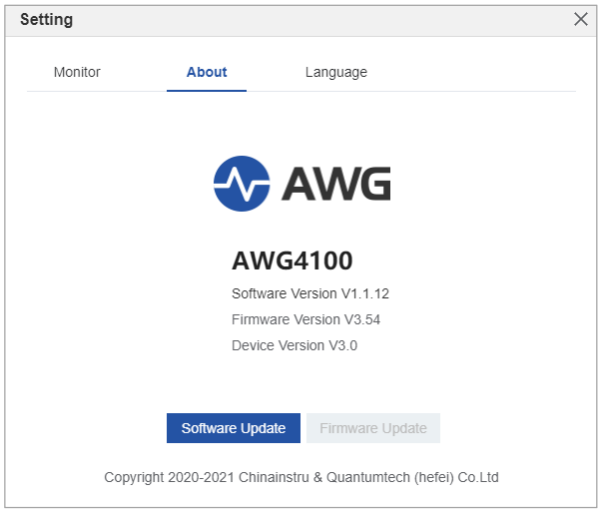

图 6.2.26 “About”功能界面

6.2.8.3“Language”功能

“Language”界面显示如下图所示，支持进行中文、英文的版本切换。

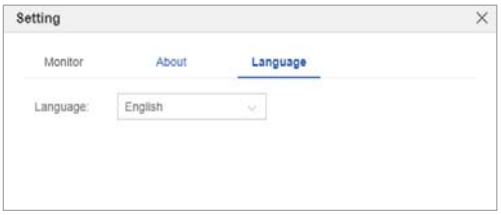

图 6.2.27 “Language”功能界面

7 波形自定义开发

自定义波形代码的基本规则如下：

1. 波形相关数据类型有 WAVE、SQN、ADVS。

2. 内置变量 OUT1、OUT2、OUT3、OUT4 表示仪器的四个输出，对于 2 通道的版本，OUT1、OUT2 表示仪器的两个输出。

3. 内置函数包括 Sin( )、Gauss( )、Triangle( )和 Square( )等分别用于生成正弦波序列、高斯脉冲序列、三角波序列和方波序列。

4. 全局变量 GAmp 用于统一指定所有波形的幅度，默认值为 100mV。

5. 注释以#开始。

6. 一条语句以英文分号“;”结尾。

7. 语句需要缩进，可以调用相关 Python 语句。

8. 函数支持可选参数和必选参数，可选参数按顺序依次给出时可以不明确指定参数名，否则必须指定参数名。

9. 内置常量，圆周率 PI。

10. 支持数值变量及其四则运算“+”、“*” 。

11. 字符串仅支持单引号。

12.波形长度需要大于 56 点，不足的点会进行补零；波形长度需是 8 的整倍数，不足的点会进行补零。

13. 设备的输出采样率为 1.2 GSa/s。

7.1 波形编辑

任意波形发生器波形编辑主要包括全局常量设置、基本波形和序列波形的产生。

7.1.1 WAVE 类型

WAVE 类型用来表示一个独立的波形，我们可以通过三种方法生成一个波形，一是直接指定每个采样点的电压值，二是通过文件，三是通过内置函数。

7.1.1.1 直接指定电压值

通过列表数据直接指定每个采样点的电压值生成一个波形，电压值的单位是 mV，范围 [-500.0, 500.0]。WAVE 会自动将电压值线性编码为[0, 65535]，其中 0 mV 对应 32768。通过数据列表构造 WAVE 时，mode 参数起作用，有 2 个可选值。当 Mode='Volt'时（默认值），将数据视为电压值；当 Mode='Code' 时将数据视为已经编码的值。

```
w1 = WAVE([0,1,2,3,4,5,6,7]);          #  电压数据, 0mV, 1mV...
w1 = WAVE([32768], mode='code');    #  一个 0 电平
```

7.1.1.2 加载文件生成 WAVE 类型

通过.wave 文件（本质上为 2 进制文件）加载 wave 数据。wave 文件有数据头和数据块组成，数据头和数据块的地址格式为下表所示：

表 7.1.1　wave 文件数据格式

地址	说明
0H00-0H03	'AWGw'波形文件标识
0H04-0H07	'\0u16' 或 '\0f32' 或 '\0f64'
0H08-0H0B	uint32, 波形长度，点数
0H0B-0H0F	(空字段, 占位， 无用)
0H10 - ...	数据块
u16 表示数据块为编码的，16 位无符号整型	
f32 表示 mV 为单位的电压值, 32 位浮点数	
f64 表示 mV 为单位的电压值, 64 位浮点数	

可以使用完整的绝对路径也可以指定一个全局的目录然后在该目录下查找文件。

注意：不支持相对路径。

```
wf = WAVE('E:\\data.wave');
WD = 'E:\\';
wf = WAVE('data.wave');
```

注意： data.wave 数据格式为数据头+具体的数据，以下是生成数据的 Python 代码示例：

```
import sys
import time
import struct
list_dec = [150, 200, 300, 400, 500,400, 300, 220, 244, 255]
dtype = b'\x00f64'#其中数据类型，还支持 b'\x00f32'，b'\x00u16'
with open('E:\data.wave', 'bw') as wf:
    wf.write(b'AWGw') #文件头标识符
    wf.write(dtype) #数据点的表示格式
    wf.write(struct.pack('I', len(list_dec))) #文件数据的长度
    wf.write(b'\x00\x00\x00\x00')#数据位的补齐，预留
    wf.write(struct.pack('{}d'.format(len(list_dec)), *list_dec))#写入的数据
```

注意：以上是提供的 Python 脚本输出自定义波形数据文件的方法，也可以用 Matlab 等脚本语言生成数据，只要注意数据文件的头部即可。

7.1.1.3 内置函数

基本函数库包含了一些常见波形的内置函数，例如正弦信号、高斯信号、方波、三角波等等，可以直接使用这些内置函数直接生成 WAVE 波形。

也支持用户使用 Python 语法自定义函数。

1、Sin

函数原型为 Sin(Freq, Amp, Len, Phase, Offset, Loop)

其相关参数解释如下：

表 7.1.2 Sin 函数参数解释

Freq	必选，实数，正弦波频率，单位 MH
Amp	可选，实数，正弦波振幅，单位 mV
Len	可选，正整数，采样点数
Phase	可选，实数，初始相位，单位2 π，默认值为 0
Offset	可选，实数，偏置，单位 mV，默认值为 0 mV
Loop	可选，正整数，循环次数，默认 1

注意：Len*Loop 长度要在 56 点到 512M 个点之间；|Amp|+|Offset|要在正负 500mV 之间，否则有溢出。

说明：按 Amp*Sin(2*pi*(Freq*t + Phase))+ Offset 计算，生成一个频率为 Freq MHz，振幅为 Amp mV，初始相位为 Phase，具有 Len 个采样点的正弦波序列。当 Amp 的值没有指定时取 GAmp 的值，Len 的值没有指定时自动取 Len=Fs/Freq，向下取整。频率 Freq 必须指定，后面参数可选。

示例：

```
w2 = Sin(10);  #10MHz，振幅为 GAmp m，初始相位为 0 的 1 个周期的正弦波
w3 = Sin(20, 100, 200); # 20MHz，100mV，200 个采样点的正弦波
w4 = Sin(20, Amp=100, Len=200); #  与 w3 相同
w5 = Sin(25, Len=200); # 25MHz, GAmp mV，200 个采样点
```

2、Gauss

函数原型为 Gauss(Var, Amp, Len, Offset, Loop)

其相关参数解释如下：

表 7.1.3 Gauss 函数参数解释

Var	必选，实数，高斯函数方差，按点数计算
Amp	可选，实数，高斯脉冲强度，即最大值，单位 mV
Len	可选，正整数，采样 L 个点，高斯函数的中心在 L/2 处
Offset	可选，实数，偏置，单位 mV，默认为 0 mV
Loop	可选，正整数，循环次数, 默认 1

注意：Len*Loop 长度要在 56 点到 512M 个点之间；|Amp|+|Offset|要在正负 500mv 之间，否则有溢出。

说明：按 Amp*exp(-t*t/(Var*Var))+Offset 计算，生成一个高斯脉冲序列，其中心位置在 Len/2 处。当 A 的值没有指定时取 GAmp 的值，Len 的值没有指定时默认采样范围是[0, 5*Var)。方差 Var 必须指定，后面参数可选。

示例：

```
w6 = Gauss(10); # 方差为 10，强度为 GAmp mV 的高斯脉冲
w7 = Gauss(10, 500, 100); # 方差为 10，强度为 500mV，100 个采样点的高斯脉冲
w8 = Gauss(10, Amp=500, Len=100); # 与 w7 相同
w9 = Gauss(10, Len=200); # 方差为 10，强度为 GAmp mV，200 个采样点的高斯脉冲
```

3、Square_p

函数原型为 Square_p(Len, Mode, Amp, Offset, Loop，Dutycycle)

其相关参数解释如下：

表 7.1.4 Square_p 函数参数解释

Len	必选，正整数，方波周期的点数
Mode	可选，方波模式，UP 是先低后高，DOWN 是先高后低，默认值是 UP
Amp	可选，实数，振幅，单位 mV
Offset	可选，实数，偏置，单位 mV，默认值是 0 mV
Loop	可选，正整数，方波循环次数, 默认 1
Dutycycle	可选，占空比，默认 0.5，范围 0-1

注意：Len*Loop 长度要在 56 点到 512M 个点之间；|Amp|+|Offset|要在正负 500mv 之间，否则有溢出。

示例：

```
w1 = Square_p(160, Mode=UP); #周期为 160 个点数，占空比为 0.5，先低后高的方波
```

4、Square

函数原型：Square(T, mode=UP, A=None, N=1，Dutycycle);

其相关参数解释如下：

表 7.1.5 Square 函数参数解释

T	必选，正实数，方波周期，单位 ns
mode	可选，方波模式，UP 是先低后高，DOWN 是先高后低，默认值为 UP
A	可选，实数，同前,范围在 0-500mv，默认值为 GAmp
N	可选，正整数，方波循环次数，默认为 1
Dutycycle	可选，占空比，默认 0.5，范围 0-1

注意：SR*T*N 长度要在 56 点到 512M 个点之间(注：SR 为采样率，支持 200MSa/s 至 1.2GSs/s 之间 10 挡位切换)。

```
w1 = Square (100, Mode=UP);  #周期为 100ns,一共 120 个点数，占空比为 0.5，先低后高的方波
```

说明：采样率为 1.2GSa/s，方波周期 100ns，则采样的点数为 120 个，占空比为 0.5，所以脉宽为 60 个点数。

5、Triangle_p

函数原型为 Triangle_p(Len, Mode, Amp, Offset，Loop，Symmetry)

其相关参数解释如下：

表 7.1.6 Triangel_p 函数参数解释

Len	必选，正整数，三角波周期的点数
Mode	可选，三角波模式，UP 是从低到高，DOWN 是从高到低，默认值为 UP
Amp	可选，振幅，单位 mV，默认值为 GAmp
Offset	可选，实数，偏置，单位 mV，默认值为 0 mV
Loop	可选，正整数，三角波循环次数, 默认 1
Symmetry	对称性，可选参数，实数，参数输入范围为：（0，1）；如果没有指定，则默认值为 0.5；Len*Symmetry 之后的值进行四舍五入取整。

注意：Len*Loop 长度要在 56 点到 512M 个点之间；|Amp|+|Offset|要在正负 500mv 之间，否则有溢出。

示例：

```
w2 = Triangle_p(160, Mode=DOWN); #周期为 160 个点数，振幅为 GAmp mV，先高后低的三角波；
```

6、Triangle

函数原型：Triangle(T, mode=UP, A=None, N=1， Symmetry);

其相关参数解释如下：

表 7.1.7 Triangle 函数参数解释

T	必选，正整数，三角波周期，单位 ns
mode	可选，三角波模式，UP 是从低到高，DOWN 是从高到低
A	可选，振幅，单位 mV，默认值为 GAmp
N	可选，正整数，三角波循环次数，默认值为 1
Symmetry	对称性，可选参数，实数，参数输入范围为：（0，1）；如果没有指定，则默认值为 0.5；Len*Symmetry 之后的值进行四舍五入取整。

示例：

```
w4 = Triangle (100, Mode=DOWN); #周期为 100ns,120 个点数，振幅为 GAmp mV，先高后低的三角波；
```

7、Wind

窗函数，其函数原型为 Wind(N1,Val1,N2,……,Nn,Valn)，只参与与其他波形进行乘法运算（不能作为波形进行下载），相关参数解释如下：

表 7.1.8 Wind 函数参数解释

Nn	必选，正整数，点数
Valn	必选，值 0 或 1

注意：不能作为基本波形，需要和别的波形绑定。

示例：

```
win = Wind (100,1,200,0,300,1);
w1 = Sin(25, Len=600);
w2 = win*w1;
```

8、Shift

对波形进行移位操作，其函数原型为 Shift(Wave,Offset)，相关参数解释如下：

表 7.1.9 Shift 函数参数解释

Wave	波形
Offset	偏移的点数

示例：

```
w1 = Shift (w1,1000);
```

注意：不能作为基本波形，需要和别的波形绑定或者操作。

9、Delay

对波形进行延迟，其函数原型为 Delay(Cycle)，相关参数解释如下：

Cycle：延迟周期数（采样时钟的周期*4），采样频率为 1.2GHz

示例：

```
Delay (1000);
```

10、Combine

函数原型为 Combine(Wave1, Wave2,..., WaveN)，其功能是可以将多个波形拼接在一起，组成一个新的序列。

示例：

```
W4 = Combine(W1,W2,W3); #将 W1,W2,W3 拼接组成一个新的序列
```

11、Marker

函数原型为 Marker (N1,Val1,N2,Val2)，其相关参数解释如下：

表 7.1.10　Marker 函数参数解释

N1/N2	点数（注意 N1 的点数要比 Marker 函数绑定的波形的总点数小 4 个以上）
VAL1/VAL2	值 0 或 1

通过 Marker 函数可以将 Marker 信号与相应的波形进行绑定。

示例：

```
W2_marker = Marker(100,1,200,0)
W2 = w2_wave + W2_marker; #将 Marker 信号与波形进行绑定
```

注意：要输出相关波形的 marker 信号，需要将相关波形通过 Marker 绑定，然后还要将相关通道的 Marker 开关打开。（N1+N2 长度最好和绑定波形一致，如果长度不一致的情况，会截取和波形长度一致）。

图 7.1.1 Marker 开关

基本波形是支持相加和相乘的运算，运算的时候，注意波形的长度（或者点数）需要一致。基本波形是不能够直接赋值给 OUT1-OUT4 的通道变量的。

12、Sweep

函数原型为 Sweep(Fstart,Fstop,Stime)，其相关参数解释如下：

表 7.1.11 Sweep 函数参数解释

Fstart	必选，正数，扫频开始频率，单位为 MHz；取值范围为：0-330 MHz
Fstop	必选，正数，扫频结束频率，单位为 MHz；取值范围为（0，330 MHz]，且值必须大于 Fstart
Stime	必选，正数加单位，扫频时长，单位可以是 ns、us、ms 和 s，取值范围为(74 ns,445 ms]

示例：

```
W1 = Sweep(1,20,20ms); #从 1 到 20 MHz 范围内进行扫频，扫频时长为 20 ms
```

13、DC

函数原型为 DC（Len，Amp），其相关参数解释如下：

表 7.1.12 DC 函数参数解释

Len	必选，正整数类型,采样点数

Amp	可选，正数，输出幅度，单位为 mV；参数输入范围为 [0,500mV]，如果没有指定值，则优先使用全局变量 GAmp 的值，如果全局变量 GAmp 也没有指定值，则使用 100mV 作为默认值

示例：

```
W1 = DC(200,100); #长度为 200 点，幅度为 100mV 的直流
```

14、PRBS

函数原型为 PRBS（BRate,Ord）， 伪随机二进制序列产生，其相关参数解释如下：

表 7.1.13 PRBS 函数参数解释

BRate	必选，正整数类型,比特率，单位 bps,参数输入范围<300 Mbps
Ord	可选，可选值包括：PRBS-3、PRBS-7、PRBS-9、PRBS-11、PRBS-15、PRBS-23，默认值为 PRBS-3；需保证（2^N-1）*SR/BRate <512M，否则波形长度过长，溢出而不能下发。

示例：

```
W1 = PRBS(BRate=100000,Ord='PRBS-9'); #比特率 100Kbps，PRBS-9 的伪随机序列。
```

15、GetDIO

函数原型为 GetDIO( ),没有参数输入，返回值为 DIO 端口获取的数据，正整数。

通过 DIO 接口可以实现反馈功能，支持反馈功能：支持根据 DIO 接口的输入来确认播放的波形序列；在播放波形过程中，如果切换了 DIO 输入，则实时变换波形播放；

支持的语法包括：if.…else；if…else if.…else；switch …… case……case…..；支持运算：>、<、=；具体示例如下：

示例 1：

```
if (GetDIO()>1):
```

```
    OUT1 = s1;
    OUT2 = s1;
  elif (GetDIO()==1):
    OUT1 = s1;
    OUT2 = s2;
  else:
    OUT1 = s2;
    OUT2 = s2;
```

16、GetCounter

函数原型为 GetCounter( ),没有参数输入，返回值为 Gounter 计数值，正整数。GetCounter 与 GetDIO 类似，也可通过该函数实现反馈。

7.1.2 SQN 类型

一个 SQN 包含一个或多个“子序列”，其按时间先后排列，在外部触发模式下，子序列播放受触发信号控制。

SQN([w1(重复次数), …, wn(重复次数)])

其中 w1,…,wn 是 n 个 WAVE，n >= 1。

重复次数为正整数。

多个子序列可以包含在独立的“[ ]”内以组成新的序列。

注意：各重复次数需小于 2^32，且各单次触发播放波形长度点数小于 2^50。

示例：

```
    # 写出所有波形
    s1=SQN([w1(2)]); #包含一个子序列，播放完全部波形需要 2 次触发
    s1=SQN([[w1(2)]]); #包含一个子序列，[w1(2)]组成一个新的子序列，播放完全部波形需要 1 次触发
    s1 = SQN([w2(4), w3(2)]);  # 包含 2 个子序列，播放完全部波形需要 6 次触发
    s2 = SQN([w1(5), w2(2), w3(1)]);# 包含 3 个子序列，播放完全部波形需要 8 次触发
    s3 = SQN([[w1(5), w2(2)], w3(1)]); # 包含 2 个子序列, [w1(5), w2(2)]组成一个新的子序列，播放完全部波形需要 2 次触发
    s4 = SQN([[w1(5),Delay(10), w2(2)], w3(1)])  # 包含 2 个子序列，第 1 个子序列又包含 2 个子序列，播放完全部波形需要 2 次触发
```

7.1.3 ADVS 类型

通过调用一个 SQN 类型可生成一个 ADVS，ADVS 支持“+”和“+=”运算。
SQN(循环次数)
多个 ADVS 可以拼接为一个 ADVS。

注意：各 SQN 的循环次数需小于 2^32。

示例：

```
a1 = ADVS( );
a1 += s1(3) + s2(2);#s1 和 S2 属于 SQN 类型变量
a2 = ADVS(s1(3) + s2(2)); #  与 a1 等效
a3 = s1(3) + s2(2);              #  自动处理类型，与 a1 等效
```

7.1.4 OUT 类型

OUT 类型主要有 OUT1、OUT2、OUT3 和 OUT4 四种，分别对应四个通道的输出。

7.2 编辑和下载

7.2.1 编辑

用户进行波形编辑可以分为以下三个步骤：

A. 使用函数库编辑 WAVE 集合

B. 使用 SQN 定义序列的播放顺序

C. 将 ADVS 和 SQN 定义的序列赋值给相应的通道

示例：

```
####使用函数库编辑 WAVE 集合
GAmp = 200;
WD = 'E:\\'
w0 = WAVE('data.wave')
f = 10;
w1 = Sin(10);
w2_wave = Gauss(10, Len=88);
w3 = Sin(f);
```

```
w2_maker = Marker(10,0,20,1);
w2 = w2_wave + w2_maker;

####使用 SQN 定义序列
s1 = SQN([w0(4),Delay(100), w1(2), w2(2)])
s2 = SQN([w3(4)]);
s3 = s1（1） + s2（1）;
s4 = SQN([[w2(6), w1(2)],[w3(2), w3(2)]]);
####将 SQN 定义的序列赋值给相应的通道
OUT1 = s3;
```

7.2.2 解析及下载

软件对用户编辑的语句进行编译的过程为：首先解析出用户定义的 WAVE 集合{Wave1，Wave2，…，WaveN}，然后根据 ADVS 和 SQN 语句，解析出子序列集合{Seq1，Seq2，…，SeqN}和每个序列波形的组成，最后根据 OUT 赋值，将相应的波形集合和序列播放方式下载到硬件进行执行。

7.2.2.1 波形集合解析

使用函数产生波形时，每个子序列波形的长度如果不满足 8 的倍数，则解析时自动补零使其长度满足 8 的倍数。当两个波形进行运算时，波形短的自动补零，使参与运算的波形长度一样。

Marker 函数是用来产生 Marker 标记信号的，其与波形集合中的波形一一对应，有多少个波形，就有多少个 Marker 信号。使用过程中会出现有些波形定义了 Marker 信号，有些没有定义，这时候波形没有定义的 Marker 信号默认输出 1。

软件解析出用户编辑的所有基本波形后，通过下表定义的格式下发给硬件进行存储，每个波形的开始地址和结束地址，软件建立表进行存储，以方便后续处理调用。

7.2.2.2 ADVS 和 SQN 序列集合解析

使用 ADVS 和 SQN 类型时，需要软件解析出序列的个数、序列的组成结构和序列播放方式，如下所示。

波形示例解析

```
####使用函数库编辑波形集合
GAmp = 200;
freq = 10;
w1 = Sin(freq,Len=40);
w2_wave = Gauss(10, Len=90);
w2_maker = Marker(45,1,45,0);
w2 = w2_wave + w2_maker;
w3_wave = Sin(2*freq,Len=200);
w3_maker = Marker(100,0,100,1);
w3 = w3_wave + w3_maker;
w4 = Square_p(96,Mode=UP);
####使用 SQN 定义序列
####s1 = SQN([[w1,w2],[w3,Delay(100),w2],[w4,w3,w1(3)],w3]);
####注意：以上区域错误，每种波形后面需要加上相应的次数如 w2(1)而不能####
写成为 w2
s1=SQN([[w1(1),w2(1)],[w3(1),Delay(100),w2(1)],[w4(1),w3(1),w1(3)],w3(1)]);
####将 SQN 定义的序列赋值给相应的通道
OUT1 = s1;
```

编辑的波形时序图如下图所示。

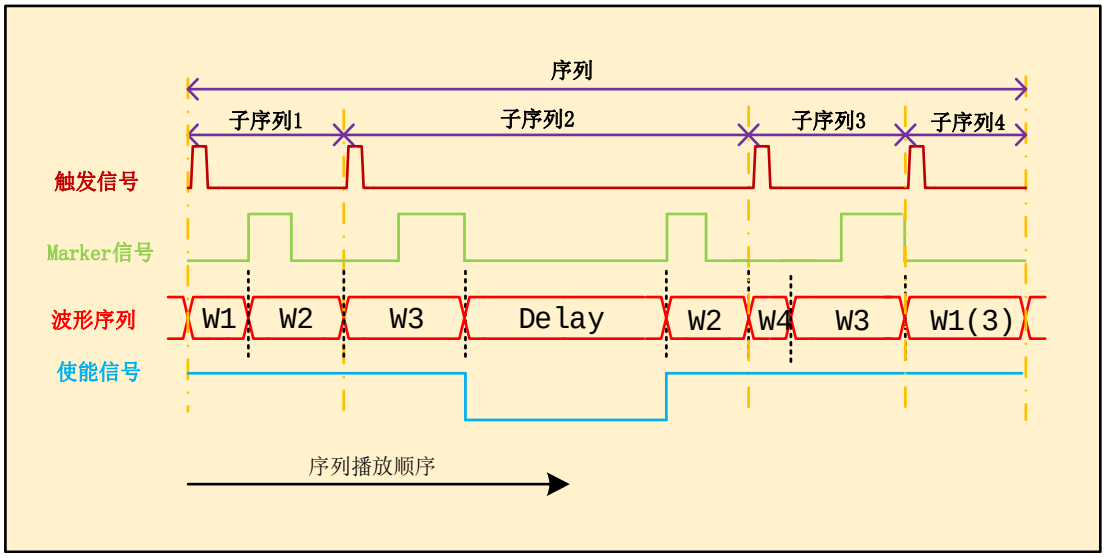

图 7.2.1 波形时序图

注意：单个基本波形长度必须大于等于 56 个点，如果单个波形的点数小于 56 个点的情况下，会自动补零。

8 接口程序调用

8.1 调用说明

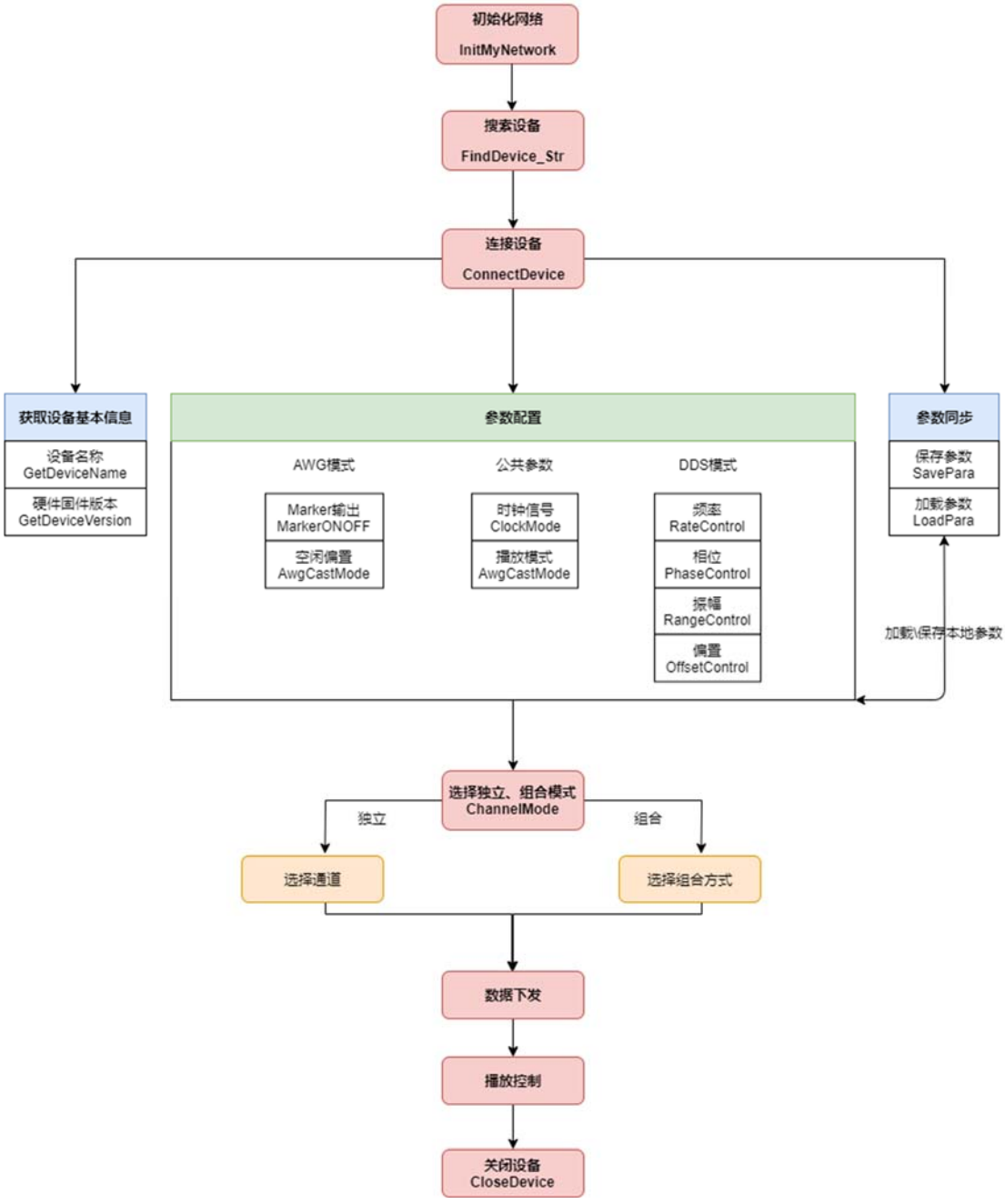

图 8.1.1 API 调用说明

8.2 C++ API 接口程序及调用注意事项

8.2.1 C++ API 接口

8.2.1.1 AWG_FindDevice

功能： 搜索设备

接口函数:

int AWG_FindDevice(AWGDeviceInfo*buffer, int buffer_len);

参数说明：

buffer:参数回填数组指针
buffer_len:内存大小

返回值：返回搜索到的设备个数

8.2.1.2 AWG_ConnectDevice

功能： 连接设备

接口函数:

QMCResult AWG_ConnectDevice(AWGDeviceInfo buffer);

参数说明：

buffer: 设备信息结构体

返回值：返回连接结果，参考返回值说明

8.2.1.3 AWG_ConnectDeviceEx

功能：连接设备，在 PC 端有多网卡时会默认选择网卡

接口函数:

QMCResult AWG_ConnectDeviceEx(const char* str_device_name);

参数说明：

str_device_name: 设备名

返回值：返回连接结果，参考返回值说明

8.2.1.4 AWG_DisconnectDevice

功能：断连设备

接口函数:

QMCResult AWG_DisconnectDevice(const char* str_device_name);

参数说明：

str_device_name: 设备名

返回值：返回断连结果，参考返回值说明

8.2.1.5 AWG_SetClockIn

功能：AWG_SetClockIn 设置时钟输入参数

接口函数:

QMCResult AWG_SetClockIn(const char* str_device_name, const unsigned int value);

参数说明：

str_device_name: 设备名
value: Internal:0 Ext-10MHz:1 Ext-100MHz:2 Sync:3

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.6 AWG_GetClockIn

功能：AWG_GetClockIn 获取输入时钟参数

接口函数:

QMCResult AWG_GetClockIn(const char* str_device_name, unsigned int* const pvalue);

参数说明：

str_device_name: 设备名
pvalue：回填参数指针

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.7 AWG_SetClockOut

功能： 设置输出时钟参数

接口函数:

QMCResult AWG_SetClockOut(const char* str_device_name, const unsigned int value);

参数说明：

str_device_name: 设备名
value：设备名 OFF:0 10MHz:1 100MHz:2

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.8 AWG_GetClockOut

功能： 获取输出时钟参数

接口函数:

QMCResult AWG_GetClockOut(const char* str_device_name, unsigned int* const pvalue);

参数说明：

str_device_name: 设备名
pvalue：回填参数指针

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.9 AWG_SetSampleRate

功能：设置采样率

接口函数:

QMCResult AWG_SetSampleRate(const char* str_device_name, const double value);

参数说明：

str_device_name: 设备名
value：采样率值 GSa/s

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.10 AWG_GetSampleRate

功能： 获取采样率

接口函数:

QMCResult AWG_GetSampleRate(const char* str_device_name, double* const pvalue);

参数说明：

str_device_name: 设备名
pvalue：回填参数指针

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.11 AWG_SetWaveformMode

功能： 设置通道波形模式

接口函数：

QMCResult AWG_SetWaveformMode(const char* str_device_name, const unsigned int value, int channel);

参数说明：

str_device_name: 设备名
value：None:0 Basic:1 Arbitrary:2
channel：通道号

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.12 AWG_GetWaveformMode

功能： 获取通道波形模式

接口函数:

QMCResult AWG_GetWaveformMode(const char* str_device_name, unsigned int* const pvalue, int channel);

参数说明：

str_device_name: 设备名
pvalue： 回填参数指针
channel：通道号

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.13 AWG_SetPlayMode

功能：设置播放模式

接口函数：

QMCResult AWG_SetPlayMode(const char* str_device_name, const unsigned int value);

参数说明：

str_device_name: 设备名
value：Standalone:0 Combination:1

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.14 AWG_GetPlayMode

功能： 获取播放模式

接口函数:

QMCResult AWG_GetPlayMode(const char* str_device_name, unsigned int* const pvalue);

参数说明：

str_device_name: 设备名
pvalue： 回填参数指针

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.15 AWG_SetChannelPlayStatus

功能： 设置独立模式下通道播放状态

接口函数：

QMCResult AWG_SetChannelPlayStatus(const char* str_device_name, const unsigned int value, int channel);

参数说明：

str_device_name: 设备名
value：None: Stop:0 Play:1
channel：通道号

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.16 AWG_GetChannelPlayStatus

功能： 获取独立模式下通道播放状态

接口函数:

QMCResult AWG_GetChannelPlayStatus(const char* str_device_name, unsigned int* const pvalue, int channel);

参数说明：

str_device_name: 设备名
pvalue： 回填参数指针
channel：通道号

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.18 AWG_SetChannelTriggerSource

功能：设置独立模式下通道触发源

接口函数：

QMCResult AWG_SetChannelTriggerSource(const char* str_device_name, const unsigned int value, int channel);

参数说明：

str_device_name: 设备名
value：Continuous:0 Trigger Input1:1 Trigger Input2:2 Trigger Input3:3 Trigger Input4:4 Sync:5
channel：通道号

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.19 AWG_GetChannelTriggerSource

功能： 获取独立模式下通道触发源

接口函数:

QMCResult AWG_GetChannelTriggerSource(const char* str_device_name, unsigned int* const pvalue, int channel);

参数说明：

str_device_name: 设备名
pvalue： 回填参数指针

channel：通道号

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.20 AWG_SetGroupPlayStatus

功能： 设置组合模式下通道播放状态

接口函数：

QMCResult AWG_SetGroupPlayStatus(const char* str_device_name, const unsigned int value, int group);

参数说明：

str_device_name: 设备名
value：None: Stop:0 Play:1
group：组合模式下组号

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.21 AWG_GetGroupPlayStatus

功能：获取组合模式下通道播放状态

接口函数:

QMCResult AWG_GetGroupPlayStatus(const char* str_device_name, unsigned int* const pvalue, int group);

参数说明：

str_device_name: 设备名
pvalue： 回填参数指针
group：组合模式下组号

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.22 AWG_SetGroupTriggerSource

功能： 设置组合模式下通道触发源

接口函数：

QMCResult AWG_SetGroupTriggerSource(const char* str_device_name, const unsigned int value, int group);

参数说明：

str_device_name: 设备名
value：Continuous:0 Trigger Input1:1 Trigger Input2:2 Trigger Input3:3 Trigger Input4:4 Sync:5
group：组合模式下组号

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.23 AWG_GetGroupTriggerSource

功能：获取组合模式下通道触发源

接口函数:

QMCResult AWG_GetGroupTriggerSource(const char* str_device_name, unsigned int* const pvalue, int group);

参数说明：

str_device_name: 设备名
pvalue： 回填参数指针
group：组合模式下组号

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.24 AWG_SetChannelInGroup

功能： 设置组合模式下Group的包含的通道

接口函数：

QMCResult AWG_SetChannelInGroup(const char* str_device_name, const char* value, int group);

参数说明：

str_device_name: 设备名
value：例如“1101”表示此组中包含了 1、2、4 三个通道
group：组合模式下组号

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.25 AWG_GetChannelInGroup

功能： 获取组合模式下 Group 的包含的通道

接口函数:

QMCResult AWG_GetChannelInGroup(const char* str_device_name, char* const pvalue,int size, int group);

参数说明：

str_device_name: 设备名
pvalue： 回填参数指针
group：组合模式下组号

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.26 AWG_SetChannelOutput

功能： 设置通道输出模式

接口函数：

QMCResult AWG_SetChannelOutput(const char* str_device_name, const unsigned int value, int channel);

参数说明：

str_device_name: 设备名

value：Direct:0 Amplifier:1
channel：通道号

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.27 AWG_GetChannelOutput

功能：获取通道输出模式

接口函数:

QMCResult AWG_GetChannelOutput(const char* str_device_name, unsigned int* const pvalue, int channel);

参数说明：

str_device_name: 设备名
pvalue： 回填参数指针
channel：通道号

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.28 AWG_SetWaveformCycleNumber

功能：设置基础波形的通道循环次数

接口函数：

QMCResult AWG_SetWaveformCycleNumber(const char* str_device_name, const unsigned int value, int channel);

参数说明：

str_device_name: 设备名
value：无线循环 1-(2^32-1)表示循环次数
channel：通道号

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.29 AWG_GetWaveformCycleNumber

功能：获取基础波形的通道循环次数

接口函数:

QMCResult AWG_GetWaveformCycleNumber(const char* str_device_name, unsigned int* const pvalue, int channel);

参数说明：

str_device_name: 设备名
pvalue： 回填参数指针
channel：通道号

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.30 AWG_SetSequenceCycleNumber

功能： 设置AWG波形的通道循环次数

接口函数：

QMCResult AWG_SetSequenceCycleNumber(const char* str_device_name, const unsigned int value, int channel);

参数说明：

str_device_name: 设备名
value：无线循环 1-(2^32-1)表示循环次数
channel：通道号

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.31 AWG_GetSequenceCycleNumber

功能： 获取 AWG 波形的通道循环次数

接口函数:

QMCResult AWG_GetSequenceCycleNumber(const char* str_device_name, unsigned int* const pvalue, int channel);

参数说明：

str_device_name: 设备名
pvalue： 回填参数指针
channel：通道号

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.32 AWG_SetIdleOffset

功能： 设置AWG波形的通道循环次数

接口函数：

QMCResult AWG_SetIdleOffset(const char* str_device_name, const double value, int channel);

参数说明：

str_device_name: 设备名
value: -500.0 to 500.0
channel：通道号

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.33 AWG_GetIdleOffset

功能： 获取 AWG 波形的通道循环次数

接口函数:

QMCResult AWG_GetIdleOffset(const char* str_device_name, double* const pvalue, int channel);

参数说明：

str_device_name: 设备名

pvalue： 回填参数指针
channel：通道号

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.34 AWG_SetMarker

功能： 设置Maker开关状态

接口函数：

QMCResult AWG_SetMarker(const char* str_device_name, const unsigned int value, int channel);

参数说明：

str_device_name: 设备名
value: 0 or 1
channel：通道号

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.35 AWG_GetMarker

功能： 获取 Maker 开关状态

接口函数:

QMCResult AWG_GetMarker(const char* str_device_name, unsigned int* const pvalue, int channel);

参数说明：

str_device_name: 设备名
pvalue： 回填参数指针
channel：通道号

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.36 AWG_SetChannelTriggerMode

功能： 设置DIO模块通道的TriggerMode

接口函数：

QMCResult AWG_SetChannelTriggerMode(const char* str_device_name, const unsigned int value, int channel);

参数说明：

str_device_name: 设备名
value: 0:Rising 1:Falling 2:Both 3:Level
channel：通道号

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.37 AWG_GetChannelTriggerMode

功能： 获取 DIO 模块通道的 TriggerMode

接口函数:

QMCResult AWG_GetChannelTriggerMode(const char* str_device_name, unsigned int* const pvalue, int channel);

参数说明：

str_device_name: 设备名
pvalue： 回填参数指针
channel：通道号

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.38 AWG_SetSyncTriggerMode

功能： 设置DIO模块SYNC的TriggerMode

接口函数：

QMCResult AWG_SetSyncTriggerMode(const char* str_device_name, const unsigned int value);

参数说明：

str_device_name: 设备名
value: 0:Rising 1:Falling 2:Both

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.39 AWG_GetSyncTriggerMode

功能： 获取 DIO 模块 SYNC 的 TriggerMode

接口函数:

QMCResult AWG_GetSyncTriggerMode(const char* str_device_name, unsigned int* const pvalue);

参数说明：

str_device_name: 设备名
pvalue： 回填参数指针

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.40 AWG_SetThreshold

功能：设置DIO模块的Threshold

接口函数：

QMCResult AWG_SetThreshold(const char* str_device_name, const double value,int channel);

参数说明：

str_device_name: 设备名
value: 0.0 to 5.0
channel：通道号

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.41 AWG_GetThreshold

功能： 获取 DIO 模块的 Threshold

接口函数:

QMCResult AWG_GetThreshold(const char* str_device_name, double* const pvalue, int channel);

参数说明：

str_device_name: 设备名
pvalue： 回填参数指针
channel：通道号

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.42 AWG_SetChannelSweepStatus

功能： 设置Sweep模块的通道扫频开关状态

接口函数：

QMCResult AWG_SetChannelSweepStatus(const char* str_device_name, const unsigned int value, int channel);

参数说明：

str_device_name: 设备名
value: 0:关闭 1:打开
channel：通道号

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.43 AWG_GetChannelSweepStatus

功能： 获取 Sweep 模块的通道扫频开关状态

接口函数:

QMCResult AWG_GetChannelSweepStatus(const char* str_device_name, unsigned int* const pvalue, int channel);

参数说明：

str_device_name: 设备名
pvalue： 回填参数指针
channel：通道号

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.44 AWG_SetSweepCycleNumber

功能： 设置Sweep模块的通道循环次数

接口函数：

QMCResult AWG_SetSweepCycleNumber(const char* str_device_name, const unsigned int value, int channel);

参数说明：

str_device_name: 设备名
value: 0：无线循环 1-(2^32-1)表示循环次数
channel：通道号

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.45 AWG_GetSweepCycleNumber

功能： 获取 Sweep 模块的通道循环次数

接口函数:

QMCResult AWG_GetSweepCycleNumber(const char* str_device_name, unsigned int* const pvalue, int channel);

参数说明：

str_device_name: 设备名
pvalue： 回填参数指针
channel：通道号

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.46 AWG_SetSweepPlayStatus

功能： 设置Sweep模块的播放状态

接口函数：

QMCResult AWG_SetSweepPlayStatus(const char* str_device_name, const unsigned int value);

参数说明：

str_device_name: 设备名
value: 0:Stop 1:Play

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.47 AWG_GetSweepPlayStatus

功能： 获取 Sweep 模块的播放状态

接口函数:

QMCResult AWG_GetSweepPlayStatus(const char* str_device_name, unsigned int* const pvalue);

参数说明：

str_device_name: 设备名
pvalue： 回填参数指针

返回值：返回寄存器操作结果，参考返回值说明

8.2.1.48 AWG_GetCurrentTemperature

功能： 获取设备的温度和电流信息

接口函数:

QMCResult AWG_GetCurrentTemperature(const char* str_device_name,AWGCurrentTemperature* pvalue);

参数说明：

str_device_name: 设备名
pvalue： 回填参数(struct AWGCurrentTemperature 类型)指针

返回值：返回操作结果，参考返回值说明

8.2.1.49 AWG_GetChannelStatus

功能：获取设备的通道状态

接口函数:

QMCResult AWG_GetChannelStatus(const char* str_device_name,AWGChannelStatus* pvalue,int channel); * pvalue);

参数说明：

str_device_name: 设备名
pvalue： 回填参数(struct AWGChannelStatus 类型)指针

返回值：返回操作结果，参考返回值说明

8.2.1.50 AWG_SetBasicSin

功能：设置基础 SIN 波形

接口函数:

QMCResult AWG_SetBasicSin(const char* str_device_name, int channel, AWGBasicSin wave_paras);

参数说明：

str_device_name: 设备名
chennel：通道号
wave_paras：AWGBasicSin 类型变量

返回值：返回操作结果，参考返回值说明

8.2.1.51 AWG_SetBasicSquare

功能：设置基础 Square 波形

接口函数:

QMCResult AWG_SetBasicSquare(const char* str_device_name, int channel, AWGBasicSquare wave_paras);

参数说明：

str_device_name: 设备名
channel：通道号
wave_paras：AWGBasicSquare 类型变量

返回值：返回操作结果，参考返回值说明

8.2.1.52 AWG_SetBasicGauss

功能：设置基础 Guass 波形

接口函数:

QMCResult AWG_SetBasicGauss(const char* str_device_name, int channel, AWGBasicGauss wave_paras);

参数说明：

str_device_name: 设备名

channel：通道号
wave_paras：AWGBasicGauss 类型变量

返回值：返回操作结果，参考返回值说明

8.2.1.53 AWG_SetBasicTriangle

功能： 设置基础 Triangle 波形

接口函数:

QMCResult AWG_SetBasicTriangle(const char* str_device_name, int channel, AWGBasicTriangle wave_paras);

参数说明：

str_device_name: 设备名
channel：通道号
wave_paras：AWGBasicTriangle 类型变量

返回值：返回操作结果，参考返回值说明

8.2.1.54 AWG_SetBasicDC

功能： 设置基础 DC 波形

接口函数:

QMCResult AWG_SetBasicDC(const char* str_device_name, int channel, AWGBasicDC wave_paras);

参数说明：

str_device_name: 设备名
channel：通道号
wave_paras：AWGBasicDC 类型变量

返回值：返回操作结果，参考返回值说明

8.2.1.55 AWG_SetBasicPRBS

功能：设置基础 PRBS 波形

接口函数:

QMCResult AWG_SetBasicPRBS(const char* str_device_name, int channel, AWGBasicPRBS wave_paras);

参数说明：

str_device_name: 设备名
channel：通道号
wave_paras：AWGBasicPRBS 类型变量

返回值：返回操作结果，参考返回值说明

8.2.1.56 AWG_SetSweep

功能： 设置 Sweep 波形

接口函数:

QMCResult AWG_SetSweep(const char* str_device_name, int channel,AWGSweep wave_paras);

参数说明：

str_device_name: 设备名
channel：通道号
wave_paras：AWGSweep 类型变量

返回值：返回操作结果，参考返回值说明

8.2.1.57 AWG_SetSequence

功能： 设置 Sequence 代码

接口函数:

QMCResult AWG_SetSequence(const char* str_device_name, const char* code);

参数说明：

str_device_name: 设备名
code：python 代码 C 格式的字符串

返回值：返回操作结果，参考返回值说明

8.2.1.58 AWG_DownloadAlone

功能：独立通道下载

接口函数:

QMCResult AWG_DownloadAlone(const char* str_device_name, int channel, char* const tips = nullptr, int size = 0);

参数说明：

str_device_name: 设备名
channel：通道号
tips：下载提示信息字符数组
size：tips 数组的大小

返回值：返回下载结果，参考返回值说明

8.2.1.59 AWG_DownloadGroup

功能： 组合通道下载

接口函数:

QMCResult AWG_DownloadGroup(const char* str_device_name, int group, char* const tips = nullptr, int size = 0);

参数说明：

str_device_name: 设备名

group：组合模式的组合号，四通道设备（1、2、3），两通道设备（1）
tips：下载提示信息字符数组
size：tips 数组的大小

返回值：返回下载结果，参考返回值说明

8.2.1.60 AWG_DownloadSweep Sweep

功能：通道下载

接口函数:

QMCResult AWG_DownloadSweep(const char* str_device_name, char* const tips = nullptr, int size = 0);

参数说明：

str_device_name: 设备名
tips：下载提示信息字符数组
size：tips 数组的大小

返回值：返回下载结果，参考返回值说明

8.2.1.61 AWG_PlayAllDevice

功能： 在一台设备中播放所有设备

接口函数:

QMCResult AWG_PlayAllDevice(const char* str_device_name, unsigned int value);

参数说明：

str_device_name: 设备名
value：下载提示信息字符数组

返回值：返回操作结果，参考返回值说明

8.2.1.62 AWG_DownloadAllDevice

功能： 在一台设备中下载 sequence 中代码设置的所有设备

接口函数:

QMCResult AWG_DownloadAllDevice(const char* str_device_name, char* const tips = nullptr, int size = 0);

参数说明：

str_device_name: 设备名
tips：下载提示信息字符数组
size：tips 数组的大小

返回值：返回当前设备下载结果，参考返回值说明

8.2.2 C++ API 调用注意事项

1、C++调用版本要求：建议 VS2008 及以上版本。

2、32 位与 64 位动态库应与所建程序保持对应。

8.2.3 C++ API 接口调用示例

```
const char* code =
"\
w1 = DC(200, 240)\n\
w2 = Gauss(Var = 6, Len = 80)\n\
w3 = Square_p(Len = 80)\n\
w4 = WAVE([-500, -400, -300, -200, -100, 0, 100, 200, 300, 400, 500, 400, 300, 200, 100, 0, -100, -200, -300, -400, -500, -400, \
- 300, -200, -100, 0, 100, 200, 300, 400, 500, 400])\n\
w5 = WAVE([0, 6553.5, 9830.25, 11796.3, 13107, 17039.1, 24903.3, 27524.7, 32768, 39321, 45874.5, 52428, 58981.5, 65535], \"code\")\n\
w6 = Sin(5, 100)\n\
\n\
\n\
s1 = SQN([w1(1)])\n\
s2 = SQN([w2(1)])\n\
s3 = SQN([w3(1)])\n\
s4 = SQN([w4(1)])\n\
```

```
s5 = SQN([w5(1)])\n\
s6 = SQN([w6(1)])\n\
\n\
if GetDIO() <= 32:\n\
    OUT1 = s1\n\
elif GetDIO() > 32 and GetDIO() <= 58:\n\
    OUT1 = s2\n\
elif GetDIO() > 58 and GetDIO() <= 88:\n\
    OUT1 = s3\n\
elif GetDIO() > 88 and GetDIO() <= 120:\n\
    OUT1 = s4\n\
elif GetDIO() > 120 and GetDIO() <= 199:\n\
    OUT1 = s5\n\
else:\n\
    OUT1 = s6\n\
";

int TestSequence()
{
    /*设备搜索*/
    AWGDeviceInfo infos[10];
    int ret = AWG_FindDevice(infos, sizeof(infos));
    if (ret <= 0)
    {
        std::cout << "Find 0 Device,Exit!" << std::endl;
        return 0;
    }

    std::cout << "Find Device As Fallow:" << std::endl;
    for (int i = 0; i < ret; ++i)
    {
        std::cout << "["<< i + 1 << "].device_ip:" << infos[i].dev_ip
            << " interface_ip:" << infos[i].local_ip << std::endl;
    }

    std::cout << "Choose Device No:";
    int no;
    std::cin >> no;
    if (no > ret)
    {
```

```
        std::cout << "Choose Error,Exit!" << std::endl;
    }

    /*设备连接*/
    char device_name[128] = { 0 };
    if (strncmp(infos[no-1].firmware, "V3.", 3) == 0)
    {
        strcpy(device_name, infos[no-1].dev_name);
        strcat(device_name, infos[no-1].dev_id);

        QMCResult ret = AWG_ConnectDevice(infos[no-1]);
        if (ret != QMC_OK)
        {
            std::cout << "ConnectDevice() ERROR" << std::endl;
            return 0;
        }
        std::cout << "ConnectDevice " << device_name << " Success" << std::endl;
    }
    else
    {
        std::cout << "Old Firmware Version,Exit!" << std::endl;
        return 0;
    }

    /*时钟输入的 GET 与 SET*/
    QMCResult result(QMC_OK);
    result = AWG_SetClockIn(device_name, 1);
    if (result != QMC_OK)
    {
        std::cout << "SetClockIn() Error" << std::endl;
        return 0;
    }

    unsigned int clock_in;
    result = AWG_GetClockIn(device_name, &clock_in);
    if (result != QMC_OK)
    {
        std::cout << "GetClockIn() Error" << std::endl;
        return 0;
    }
```

```
std::cout << "GetClockIn() Result" << clock_in << std::endl;

/*时钟输出的 GET 与 SET*/
result = AWG_SetClockOut(device_name, 1);
if (result != QMC_OK)
{
    std::cout << "SetClockOut() Error" << std::endl;
    return 0;
}
unsigned int clock_out;
result = AWG_GetClockOut(device_name, &clock_out);
if (result != QMC_OK)
{
    std::cout << "GetClockOut() Error" << std::endl;
    return 0;
}
std::cout << "GetClockOut() Result" << clock_out << std::endl;


/*采样率的 GET 与 SET*/
result = AWG_SetSampleRate(device_name, 1.2);
if (result != QMC_OK)
{
    std::cout << "SetSampleRate() Error" << std::endl;
    return 0;
}
double sample_rate;
result = AWG_GetSampleRate(device_name, &sample_rate);
if (result != QMC_OK)
{
    std::cout << "GetSampleRate() Error" << std::endl;
    return 0;
}
std::cout << "GetSampleRate() Result" << sample_rate << std::endl;


/*波形形式  Waveform Mode  基础波形*/
result = AWG_SetWaveformMode(device_name, 2, 1);
if (result != QMC_OK)
{
```

```
        //std::cout << "SetWaveformMode() Error" << std::endl;
        return 0;
    }
    unsigned int wave_mode_out;
    result = AWG_GetWaveformMode(device_name, &wave_mode_out, 1);
    if (result != QMC_OK)
    {
        std::cout << "GetWaveformMode() Error" << std::endl;
        return 0;
    }
    std::cout << "GetWaveformMode() Result" << wave_mode_out << std::endl;


    /*播放模式（独立与组合）的 GET 与 SET*/
    result = AWG_SetPlayMode(device_name, 0);
    if (result != QMC_OK)
    {
        std::cout << "SetPlayMode() Error" << std::endl;
        return 0;
    }
    unsigned int play_mode_out;
    result = AWG_GetPlayMode(device_name, &play_mode_out);
    if (result != QMC_OK)
    {
        std::cout << "GetPlayMode() Error" << std::endl;
        return 0;
    }
    std::cout << "GetPlayMode() Result" << play_mode_out << std::endl;


    /*单通道触发源  Trigger Source*/
    result = AWG_SetChannelTriggerSource(device_name, 1, 1);
    if (result != QMC_OK)
    {
        std::cout << "SetChannelTriggerSource() Error" << std::endl;
        return 0;
    }
    unsigned int trigger_source_out;
    result = AWG_GetChannelTriggerSource(device_name, &trigger_source_out, 1);
    if (result != QMC_OK)
```

```
{
    std::cout << "GetChannelTriggerSource() Error" << std::endl;
    return 0;
}
std::cout << "GetChannelTriggerSource() Result" << trigger_source_out << std::endl;


/*Sequence 波形设置*/
result = AWG_SetSequence(device_name, code3);
if (result != QMC_OK)
{
    std::cout << "SetSequence() Error" << std::endl;
    return 0;
}


/*波形下载*/
char error[1024 * 512] = { 0 };
result = AWG_DownloadAlone(device_name, 1, error, 512 * 1024);
if (result != QMC_OK)
{
    std::cout << "DownloadAlone() Error" << std::endl;
    std::cout << "Error:" << error << std::endl;
    return 0;
}
std::cout << "Wave DownloadAlone() Success" << std::endl;


/*单通道播放寄存器  Play*/
result = AWG_SetChannelPlayStatus(device_name, 1, 1);
if (result != QMC_OK)
{
    std::cout << "SetChannelPlayStatus() Error" << std::endl;
    return 0;
}
unsigned int play_status_out;
result = AWG_GetChannelPlayStatus(device_name, &play_status_out, 1);
if (result != QMC_OK)
{
    std::cout << "GetChannelPlayStatus() Error" << std::endl;
```

```
        return 0;
    }
    std::cout << "GetChannelPlayStatus() Result" << play_status_out << std::endl;
    AWG_DisconnectDevice(device_name);

    return 0;
}

int main(int argc, char* agrv[])
{
TestSequence();
return 0;
}
```

8.3 Python 接口程序及调用注意事项

8.3.1 Python 接口程序

8.3.1.1 AWG_FindDevice

功能：搜索设备

接口函数:

def AWG_FindDevice() -> list

返回值：返回搜索到的设备列表

8.3.1.2 AWG_ConnectDevice

功能：连接设备

接口函数:

def AWG_ConnectDevice(device_info: dict) -> int

参数说明：

device_info: 连接的设备信息

返回值：返回连接结果，参考返回值说明

8.3.1.3 AWG_ConnectDeviceEx

功能：连接设备，在 PC 端有多网卡时会默认选择网卡

接口函数:

def AWG_ConnectDeviceEx(str_device_name: str) -> int

参数说明：

str_device_name:设备名

返回值：返回连接结果，参考返回值说明

8.3.1.4 AWG_DisconnectDevice

功能：断连设备

接口函数:

def AWG_DisconnectDevice(str_device_name: str) -> int:

参数说明：

str_device_name:设备名

返回值：返回断连结果，参考返回值说明

8.3.1.5 AWG_SetClockIn

功能：设置时钟输入参数

接口函数:

def AWG_SetClockIn(str_device_name: str, value: int) -> int

参数说明：

str_device_name: 设备名
value: Internal:0 Ext-10MHz:1 Ext-100MHz:2 Sync:3

返回值：返回寄存器操作结果，参考返回值说明

8.3.1.6 AWG_GetClockIn

功能：获取输入时钟参数

接口函数:

def AWG_GetClockIn(str_device_name: str) -> tuple

参数说明：

str_device_name: 设备名

返回值：返回寄存器操作结果和读取到的寄存器值，以元组形式返回

8.3.1.7 AWG_SetClockOut

功能： 设置输出时钟参数

接口函数:

def AWG_SetClockOut(str_device_name: str, value: int) -> int

参数说明：

str_device_name: 设备名
value：设备名 OFF:0 10MHz:1 100MHz:2

返回值：返回寄存器操作结果，参考返回值说明

8.3.1.8 AWG_GetClockOut

功能： 获取输出时钟参数

接口函数:

def AWG_GetClockOut(str_device_name: str) -> tuple

参数说明：

str_device_name: 设备名

返回值：返回寄存器操作结果和读取到的寄存器值，以元组形式返回

8.3.1.9 AWG_SetSampleRate

功能：设置采样率

接口函数:

def AWG_SetSampleRate(str_device_name: str, value: float) -> int

参数说明：

str_device_name: 设备名
value：采样率值 GSa/s

返回值：返回寄存器操作结果，参考返回值说明

8.3.1.10 AWG_GetSampleRate

功能： 获取采样率

接口函数:

def AWG_GetSampleRate(str_device_name: str) -> tuple

参数说明：

str_device_name: 设备名

返回值：返回寄存器操作结果和读取到的寄存器值，以元组形式返回

8.3.1.11AWG_SetWaveformMode

功能： 设置通道波形模式

接口函数：

def AWG_SetWaveformMode(str_device_name: str, value: int, channel: int) -> int

参数说明：

str_device_name: 设备名
value：None:0 Basic:1 Arbitrary:2
channel：通道号

返回值：返回寄存器操作结果，参考返回值说明

8.3.1.12 AWG_GetWaveformMode

功能： 获取通道波形模式

接口函数:

def AWG_GetWaveformMode(str_device_name: str, channel: int) -> tuple

参数说明：

str_device_name: 设备名
channel：通道号

返回值：返回寄存器操作结果和读取到的寄存器值，以元组形式返回

8.3.1.13 AWG_SetPlayMode

功能： 设置播放模式

接口函数：

def AWG_SetPlayMode(str_device_name: str, value: int) -> int

参数说明：

str_device_name: 设备名
value：Standalone:0 Combination:1

返回值：返回寄存器操作结果，参考返回值说明

8.3.1.14 AWG_GetPlayMode

功能： 获取播放模式

接口函数:

def AWG_GetPlayMode(str_device_name: str) -> tuple

参数说明：

str_device_name: 设备名

返回值：返回寄存器操作结果和读取到的寄存器值，以元组形式返回

8.3.1.15 AWG_SetChannelPlayStatus

功能： 设置独立模式下通道播放状态

接口函数：

def AWG_SetChannelPlayStatus(str_device_name: str, value: int, channel: int) -> int

参数说明：

str_device_name: 设备名
value：None: Stop:0 Play:1
channel：通道号

返回值：返回寄存器操作结果，参考返回值说明

8.3.1.16 AWG_GetChannelPlayStatus

功能： 获取独立模式下通道播放状态

接口函数:

def AWG_GetChannelPlayStatus(str_device_name: str, channel: int) -> tuple

参数说明：

str_device_name: 设备名
channel：通道号

返回值：返回寄存器操作结果和读取到的寄存器值，以元组形式返回

8.3.1.17 AWG_SetChannelTriggerSource

功能： 设置独立模式下通道触发源

接口函数：

def AWG_SetChannelTriggerSource(str_device_name: str, value: int, channel: int) -> int

参数说明：

str_device_name: 设备名
value：Continuous:0 Trigger Input1:1 Trigger Input2:2 Trigger Input3:3 Trigger Input4:4 Sync:5
channel：通道号

返回值：返回寄存器操作结果，参考返回值说明

8.3.1.18 AWG_GetChannelTriggerSource

功能：获取独立模式下通道触发源

接口函数:

def AWG_GetChannelTriggerSource(str_device_name: str, channel: int) -> tuple

参数说明：

str_device_name: 设备名
channel：通道号

返回值：返回寄存器操作结果和读取到的寄存器值，以元组形式返回

8.3.1.19 AWG_SetGroupPlayStatus

功能：设置组合模式下通道播放状态

接口函数：

def AWG_SetGroupPlayStatus(str_device_name: str, value: int, group: int) -> int

参数说明：

str_device_name: 设备名
value：None: Stop:0 Play:1
group：组合模式下组号

返回值：返回寄存器操作结果，参考返回值说明

8.3.1.20 AWG_GetGroupPlayStatus

功能： 获取组合模式下通道播放状态

接口函数:

def AWG_GetGroupPlayStatus(str_device_name: str, group: int) -> tuple

参数说明：

str_device_name: 设备名
group：组合模式下组号

返回值：返回寄存器操作结果和读取到的寄存器值，以元组形式返回

8.3.1.21 AWG_SetGroupTriggerSource

功能：设置组合模式下通道触发源

接口函数：

def AWG_SetGroupTriggerSource(str_device_name: str, value: int, group: int) -> int

参数说明：

str_device_name: 设备名
value：Continuous:0 Trigger Input1:1 Trigger Input2:2 Trigger Input3:3 Trigger Input4:4 Sync:5
group：组合模式下组号

返回值：返回寄存器操作结果，参考返回值说明

8.3.1.22 AWG_GetGroupTriggerSource

功能： 获取组合模式下通道触发源

接口函数:

def AWG_GetGroupTriggerSource(str_device_name: str, group: int) -> tuple

参数说明：

str_device_name: 设备名
group：组合模式下组号

返回值：返回寄存器操作结果和读取到的寄存器值，以元组形式返回

8.3.1.23 AWG_SetChannelInGroup

功能： 设置组合模式下Group的包含的通道

接口函数：

def AWG_SetChannelInGroup(str_device_name: str, value: int, group: int) -> int

参数说明：

str_device_name: 设备名
value：例如“1101”表示此组中包含了 1、2、4 三个通道
group：组合模式下组号

返回值：返回寄存器操作结果，参考返回值说明

8.3.1.24 AWG_GetChannelInGroup

功能： 获取组合模式下 Group 的包含的通道

接口函数:

def AWG_GetChannelInGroup(str_device_name: str, group: int) -> tuple

参数说明：

str_device_name: 设备名
group：组合模式下组号

返回值：返回寄存器操作结果和读取到的寄存器值，以元组形式返回

8.3.1.25 AWG_SetChannelOutput

功能： 设置通道输出模式

接口函数：

def AWG_SetChannelOutput(str_device_name: str, value: int, channel: int) -> int

参数说明：

str_device_name: 设备名
value：Direct:0 Amplifier:1
channel：通道号

返回值：返回寄存器操作结果，参考返回值说明

8.3.1.26 AWG_GetChannelOutput

功能： 获取通道输出模式

接口函数:

def AWG_GetChannelOutput(str_device_name: str, channel: int) -> tuple

参数说明：

str_device_name: 设备名
channel：通道号

返回值：返回寄存器操作结果和读取到的寄存器值，以元组形式返回

8.3.1.27 AWG_SetWaveformCycleNumber

功能： 设置基础波形的通道循环次数

接口函数：

def AWG_SetWaveformCycleNumber(str_device_name: str, value: int, channel: int) -> int

参数说明：

str_device_name: 设备名
value：无线循环 1-(2^32-1)表示循环次数
channel：通道号

返回值：返回寄存器操作结果，参考返回值说明

8.3.1.28 AWG_GetWaveformCycleNumber

功能： 获取基础波形的通道循环次数

接口函数:

def AWG_GetWaveformCycleNumber(str_device_name: str, channel: int) -> tuple

参数说明：

str_device_name: 设备名
channel：通道号

返回值：返回寄存器操作结果和读取到的寄存器值，以元组形式返回

8.3.1.29 AWG_SetSequenceCycleNumber

功能：设置AWG波形的通道循环次数

接口函数：

def AWG_SetSequenceCycleNumber(str_device_name: str, value: int, channel: int) -> int

参数说明：

str_device_name: 设备名
value：无线循环 1-(2^32-1)表示循环次数

channel：通道号

返回值：返回寄存器操作结果，参考返回值说明

8.3.1.30 AWG_GetSequenceCycleNumber

功能： 获取 AWG 波形的通道循环次数

接口函数:

def AWG_GetSequenceCycleNumber(str_device_name: str, channel: int) -> tuple

参数说明：

str_device_name: 设备名
channel：通道号

返回值：返回寄存器操作结果和读取到的寄存器值，以元组形式返回

8.3.1.31 AWG_SetIdleOffset

功能： 设置AWG波形的通道循环次数

接口函数：

def AWG_SetIdleOffset(str_device_name: str, value: float, channel: int) -> int

参数说明：

str_device_name: 设备名
value: -500.0 to 500.0
channel：通道号

返回值：返回寄存器操作结果，参考返回值说明

8.3.1.32 AWG_GetIdleOffset

功能： 获取 AWG 波形的通道循环次数

接口函数:

def AWG_GetIdleOffset(str_device_name: str, channel: int) -> tuple

参数说明：

str_device_name: 设备名
pvalue： 回填参数指针
channel：通道号

返回值：返回寄存器操作结果和读取到的寄存器值，以元组形式返回

8.3.1.33 AWG_SetMarker

功能： 设置Maker开关状态

接口函数：

def AWG_SetMarker(str_device_name: str, value: int, channel: int) -> int

参数说明：

str_device_name: 设备名
value: 0 or 1
channel：通道号

返回值：返回寄存器操作结果，参考返回值说明

8.3.1.34 AWG_GetMarker

功能： 获取 Maker 开关状态

接口函数:

def AWG_GetMarker(str_device_name: str, channel: int) -> tuple

参数说明：

str_device_name: 设备名
channel：通道号

返回值：返回寄存器操作结果和读取到的寄存器值，以元组形式返回

8.3.1.35 AWG_SetChannelTriggerMode

功能： 设置DIO模块通道的TriggerMode

接口函数：

def AWG_SetChannelTriggerMode(str_device_name: str, value: int, channel: int) -> int

参数说明：

str_device_name: 设备名
value: 0:Rising 1:Falling 2:Both 3:Level
channel：通道号

返回值：返回寄存器操作结果，参考返回值说明

8.3.1.36 AWG_GetChannelTriggerMode

功能： 获取 DIO 模块通道的 TriggerMode

接口函数:

def AWG_GetChannelTriggerMode(str_device_name: str, channel: int) -> tuple

参数说明：

str_device_name: 设备名
channel：通道号

返回值：返回寄存器操作结果和读取到的寄存器值，以元组形式返回

8.3.1.37 AWG_SetSyncTriggerMode

功能： 设置DIO模块SYNC的TriggerMode

接口函数：

def AWG_SetSyncTriggerMode(str_device_name: str, value: int) -> int

参数说明：

str_device_name: 设备名

value: 0:Rising 1:Falling 2:Both

返回值：返回寄存器操作结果，参考返回值说明

8.3.1.38 AWG_GetSyncTriggerMode

功能：获取 DIO 模块 SYNC 的 TriggerMode

接口函数:

def AWG_GetSyncTriggerMode(str_device_name: str) -> tuple

参数说明：

str_device_name: 设备名

返回值：返回寄存器操作结果和读取到的寄存器值，以元组形式返回

8.3.1.39 AWG_SetThreshold

功能： 设置DIO模块的Threshold

接口函数：

def AWG_SetThreshold(str_device_name: str, value: float, channel: int) -> int

参数说明：

str_device_name: 设备名
value: 0.0 to 5.0
channel：通道号

返回值：返回寄存器操作结果，参考返回值说明

8.3.1.40 AWG_GetThreshold

功能： 获取 DIO 模块的 Threshold

接口函数:

def AWG_GetThreshold(str_device_name: str, channel: int) -> tuple

参数说明：

str_device_name: 设备名
channel：通道号

返回值：返回寄存器操作结果和读取到的寄存器值，以元组形式返回

8.3.1.41 AWG_SetChannelSweepStatus

功能：设置Sweep模块的通道扫频开关状态

接口函数：

def AWG_SetChannelSweepStatus(str_device_name: str, value: int, channel: int) -> int:

参数说明：

str_device_name: 设备名
value: 0:关闭 1:打开
channel：通道号

返回值：返回寄存器操作结果，参考返回值说明

8.3.1.42 AWG_GetChannelSweepStatus

功能： 获取 Sweep 模块的通道扫频开关状态

接口函数:

def AWG_GetChannelSweepStatus(str_device_name: str, channel: int) -> tuple:

参数说明：

str_device_name: 设备名
channel：通道号

返回值：返回寄存器操作结果和读取到的寄存器值，以元组形式返回

8.3.1.43 AWG_SetSweepCycleNumber

功能：设置Sweep模块的通道循环次数

接口函数：

def AWG_SetSweepCycleNumber(str_device_name: str, value: int, channel: int) -> int

参数说明：

str_device_name: 设备名
value: 0：无线循环 1-(2^32-1)表示循环次数
channel：通道号

返回值：返回寄存器操作结果，参考返回值说明

8.3.1.44 AWG_GetSweepCycleNumber

功能： 获取 Sweep 模块的通道循环次数

接口函数:

def AWG_GetSweepCycleNumber(str_device_name: str, channel: int) -> tuple

参数说明：

str_device_name: 设备名
channel：通道号

返回值：返回寄存器操作结果和读取到的寄存器值，以元组形式返回

8.3.1.45 AWG_SetSweepPlayStatus

功能： 设置Sweep模块的播放状态

接口函数：

def AWG_SetSweepPlayStatus(str_device_name: str, value: int) -> int

参数说明：

str_device_name: 设备名

value: 0:Stop 1:Play

返回值：返回寄存器操作结果，参考返回值说明

8.3.1.46 AWG_GetSweepPlayStatus

功能： 获取 Sweep 模块的播放状态

接口函数:

def AWG_GetSweepPlayStatus(str_device_name: str) -> tuple

参数说明：

str_device_name: 设备名

返回值：返回寄存器操作结果和读取到的寄存器值，以元组形式返回

8.3.1.47 AWG_GetCurrentTemperature

功能： 获取设备的温度和电流信息

接口函数:

def AWG_GetCurrentTemperature(str_device_name: str) -> tuple

参数说明：

str_device_name: 设备名

返回值：返回操作结果和读取到的温度和电流信息，以元组形式返回

8.3.1.48 AWG_GetChannelStatus

功能： 获取设备的通道状态

接口函数:

def AWG_GetChannelStatus(str_device_name: str, channel: int) -> tuple

参数说明：

str_device_name: 设备名

channel：通道号

返回值：返回操作结果和读取到的通道状态信息，以元组形式返回

8.3.1.49 AWG_SetBasicSin

功能： 设置基础 SIN 波形

接口函数:

def AWG_SetBasicSin(str_device_name: str, channel: int, wave_paras: dict) -> int

参数说明：

str_device_name: 设备名
chennel：通道号
wave_paras：Sin 波形参数

返回值：返回操作结果，参考返回值说明

8.3.1.50 AWG_SetBasicSquare

功能： 设置基础 Square 波形

接口函数:

def AWG_SetBasicSquare(str_device_name: str, channel: int, wave_paras: dict) -> int

参数说明：

str_device_name: 设备名
channel：通道号
wave_paras：Square 波形参数

返回值：返回操作结果，参考返回值说明

8.3.1.51 AWG_SetBasicGauss

功能：设置基础 Guass 波形

接口函数:

def AWG_SetBasicGauss(str_device_name: str, channel: int, wave_paras: dict) -> int

参数说明：

str_device_name: 设备名
channel：通道号
wave_paras：Gauss 波形参数

返回值：返回操作结果，参考返回值说明

8.3.1.52 AWG_SetBasicTriangle

功能：设置基础 Triangle 波形

接口函数:

def AWG_SetBasicTriangle(str_device_name: str, channel: int, wave_paras: dict) -> int

参数说明：

str_device_name: 设备名
channel：通道号
wave_paras：Triangle 波形参数

返回值：返回操作结果，参考返回值说明

8.3.1.53 AWG_SetBasicDC

功能： 设置基础 DC 波形

接口函数:

def AWG_SetBasicDC(str_device_name: str, channel: int, wave_paras: dict) -> int

参数说明：

str_device_name: 设备名
channel：通道号
wave_paras：DC 波形参数

返回值：返回操作结果，参考返回值说明

8.3.1.54 AWG_SetBasicPRBS

功能： 设置基础 PRBS 波形

接口函数:

def AWG_SetBasicPRBS(str_device_name: str, channel: int, wave_paras: dict) -> int

参数说明：

str_device_name: 设备名
channel：通道号
wave_paras：PRBS 波形参数

返回值：返回操作结果，参考返回值说明

8.3.1.55 AWG_SetSweep

功能： 设置 Sweep 波形

接口函数:

def AWG_SetSweep(str_device_name: str, channel: int, wave_paras: dict) -> int

参数说明：

str_device_name: 设备名
channel：通道号

wave_paras：Sweep 波形参数

返回值：返回操作结果，参考返回值说明

8.3.1.56 AWG_SetSequence

功能： 设置 Sequence 代码

接口函数:

def AWG_SetSequence(str_device_name: str, code: str) -> int

参数说明：

str_device_name: 设备名
code：python 代码

返回值：返回操作结果，参考返回值说明

8.3.1.57 AWG_DownloadAlone

功能： 独立通道下载

接口函数:

def AWG_DownloadAlone(str_device_name: str, channel: int, size: int = 10240) -> tuple

参数说明：

str_device_name: 设备名
channel：通道号
size：tips 数组的大小

返回值：返回下载操作结果，参考返回值说明

8.3.1.57 AWG_DownloadGroup

功能：组合通道下载

接口函数:

def AWG_DownloadGroup(str_device_name: str, group: int, size: int = 10240) -> tuple

参数说明：

str_device_name: 设备名
group：组合模式的组合号，四通道设备（1、2、3），两通道设备（1）
size：tips 数组的大小

返回值：返回下载操作结果，参考返回值说明

8.3.1.58 AWG_DownloadSweep

功能： Sweep 通道下载

接口函数:

def AWG_DownloadSweep(str_device_name: str, size: int = 10240) -> tuple

参数说明：

str_device_name: 设备名
size：下载提示信息空间大小

返回值：返回下载操作结果，参考返回值说明

8.3.1.60 AWG_PlayAllDevice

功能： 在一台设备中播放所有设备

接口函数:

def AWG_PlayAllDevice(str_device_name: str, value:int) -> tuple

参数说明：

str_device_name: 设备名

value：0:Stop 1:Play

返回值：返回操作结果，参考返回值说明

8.3.1.61 AWG_DownloadAllDevice

功能： 在一台设备中下载 sequence 中代码设置的所有设备

接口函数:

def AWG_DownloadAllDevice(str_device_name: str, size: int = 10240) -> tuple

参数说明：

str_device_name: 设备名
size：下载提示信息空间大小

返回值：返回当前设备下载结果，参考返回值说明

8.3.2 Python 接口示例

```
def TestBasicSin():
    # 搜索接口
    result = AWG_FindDevice()

    if not len(result):
        print("Find 0 Device,Exit!")
        sys.exit(0)

    print("Find Device As Fallow:")
    for device in result:
        print("[{}].device_ip:{} interface_ip:{}".format(result.index(device) + 1,
device["dev_ip"], device["local_ip"]))
```

```
str_no = input("Choose Device No:")
no = int(str_no)

if no > len(result):
    print("Choose Error,Exit!")
    sys.exit(0)

# 连接接口
ret = AWG_ConnectDevice(result[no-1])
if ret != 1:
    print("ConnectDevice() Error")
    sys.exit(0)
print("ConnectDevice() Success")

str_device_name = result[no-1]["dev_name"] + result[no-1]["dev_id"]
# 时钟输入的 GET 与 SET
ret = AWG_SetClockIn(str_device_name, 2)
if ret != 1:
    print("SetClockIn() Error")
    sys.exit(0)

ret = AWG_GetClockIn(str_device_name)
if ret[0] != 1:
    print("GetClockIn() Error")
    sys.exit(0)
print("GetClockIn() result is {}".format(ret[1]))
```

```
# 时钟输出的 GET 与 SET
ret = AWG_SetClockOut(str_device_name, 1)
if ret != 1:
    print("SetClockOut() Error")
    sys.exit(0)

ret = AWG_GetClockOut(str_device_name)
if ret[0] != 1:
    print("GetClockOut() Error")
    sys.exit(0)
print("GetClockOut() result is {}".format(ret[1]))

# 采样率的 GET 与 SET
ret = AWG_SetSampleRate(str_device_name, 1.0)
if ret != 1:
    print("SetSampleRate() Error:{}".format(ret))
    sys.exit(0)

ret = AWG_GetSampleRate(str_device_name)
if ret[0] != 1:
    print("GetSampleRate() Error")
    sys.exit(0)
print("GetSampleRate() result is {}".format(ret[1]))

# 波形模式的 GET 与 SET 此出设置为基础波形
```

```
ret = AWG_SetWaveformMode(str_device_name, 1, 1)
if ret != 1:
    print("SetWaveformMode() Error")
    sys.exit(0)

ret = AWG_GetWaveformMode(str_device_name, 1)
if ret[0] != 1:
    print("GetWaveformMode() Error")
    sys.exit(0)
print("GetWaveformMode() result is {}".format(ret[1]))

# 播放模式（独立与组合）的 GET 与 SET
ret = AWG_SetPlayMode(str_device_name, 0)
if ret != 1:
    print("SetPlayMode() Error")
    sys.exit(0)

ret = AWG_GetPlayMode(str_device_name)
if ret[0] != 1:
    print("GetPlayMode() Error")
    sys.exit(0)
print("GetPlayMode() result is {}".format(ret[1]))

wave = {
    "freq": 100,
    "ampl": 205,
```

```
            "phase": 0,
            "offset": 0,
            "modulation": 0
        }
        ret = AWG_SetBasicSin(str_device_name, 1, wave)
        if ret != 1:
            print("SetBasicSin() Error")
            sys.exit(0)
        print("SetBasicSin() Success")

        ret = AWG_DownloadAlone(str_device_name, 1)
        if ret[0] != 1:
            print("DownloadAlone() Error")
            sys.exit(0)
        print("DownloadAlone() Success")

sys.exit(0)
```

8.3.3 Python 接口调用注意事项

8.3.3.1 使用说明

方法一：脚本运行环境为 Python3(建议版本 ＞3.6) 。使用时需要与模块‘awg4100’ 在同一目录下，如下所示，目录‘awg4100’中的内容缺一不可。**使用前请确保 AWG4100 能够正常运行**。

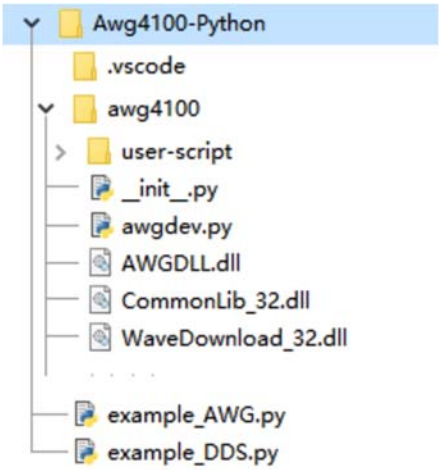

图 8.3.1 工程目录

方法二：将 awg4100 模块直接放到 Python 的安装目录下的 /Lib/site-packages 目录下面。

8.3.3.2 异常处理

若出现 OSError:[WinError126] ，可能是由于缺失 VC 运行时造成的，请安装 msvc2013 运行时环境（https://www.microsoft.com/zhcn/download/details.aspx?id=40784） 和 msvc2015 运行时环境 （https://www.microsoft.com/en-us/download/details.aspx?id=48145），建议 x86 版本与 x64 版本都要安装。

如果安装之后依旧报错。请将 msvcp120.dll、msvcr120.dll、msvcp140.dll vcruntime140.dll 放到 `awg4100` 目录下。请注意这几个 dll 是 32 位的还是

64 位的，如果使用 32 位 Python 则复制 32 位的，如果使用 64 位 Python 则复制 64 位的。

8.4 LabVIEW 接口调用及注意事项

8.4.1 LabVIEW 接口调用流程参考

打开【程序框图】窗口

在空白处鼠标右键，在出现的【函数选板】中选择【互连接口】-【库与可执行程序】-【调用库函数节点】

图 8.4.1 调用路径

双击函数，找到 dll 文件，选择函数。

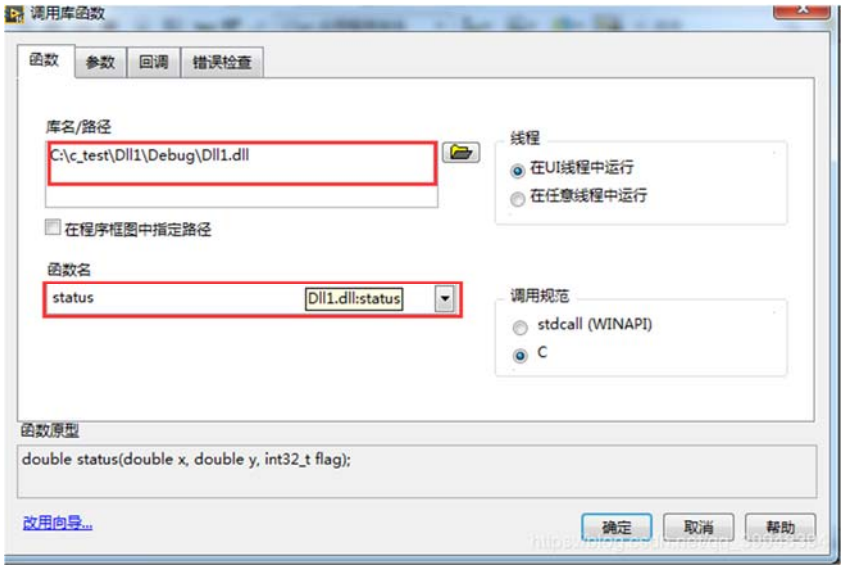

图 8.4.2 选择函数界面

参数里面添加返回值和参数，注意顺序和类型。

8.4.2 注意事项

dll 的目录中，需要包含如下图所示文件（其中 AWG4100-example.vi 为 LabVIEW 文件，其他文件为 AWGDLL 的依赖库，必须包含同一目录下）

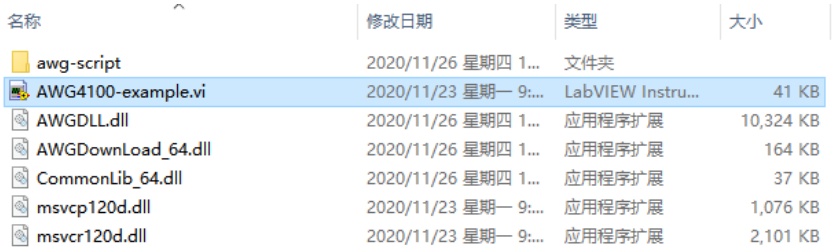

图 8.4.3 AWG4100-example.vi 的文件位置

相关的二次开发库，可以到官网到我司的官网 https://www.ciqtek.com 产品中心→量子测控系列产品→任意波形发生器（AWG4100）产品介绍界面下方下载。

9.技术规格指标

9.1 技术规格指标

1、 一般参数

参数	最小值	典型值	最大值
存储温度	-10 ℃		50 ℃
存储相对湿度			95%，无凝露
使用环境		室内安装、过电压等级 2 级，污染度 2 级	
工作海拔			2000 米
工作温度	0 ℃		40 ℃
工作相对湿度			90%，无凝露
指标温度		23±2℃；注：当环境温度变化达或以上时，产品性能会发生变化	
功耗			60 W
电源		220/240 V；50/60 Hz	
尺寸		218.3 mm（宽）×89 mm（高）×439.9 mm（长）	
重量		＜5 kg	

2、 主要参数

参数	最小值	典型值	最大值
通道数		4，可选择 2 通道版	
垂直分辨率		16bit	
波形存储深度		512 MSa/ch	
波形粒度		8 Sa	
最小波形长度		56 Sa	
序列编程器时钟频率		300 MHz	
Marker 输出		1 个标记/通道，SMA 连接器	
参考时钟输入/输出		各一个，BNC 连接器	
参考时钟输出幅值		3.3 Vpp	
参考时钟输出频率		10 MHz/100MHz	
模拟带宽		330M@1Vpp 150M@5Vpp	

3、 波形输出通道

参数	最小值	典型值	最大值
连接器类型		SMA	
输出阻抗		50 Ω	

输出耦合		DC	
输出模式		直接输出	
输出范围		± 0.5 V （into 50 Ω） ±2.5V（into 50 Ω）	
偏置调节范围		量程内偏置	
偏移电压			< 1 mV （into 50 Ω）
相位噪声			< -110 dBc/Hz （100 MHz@10 kHz）
电压噪声密度谱			< 20 nV/√Hz @100 kHz
触发到输出延迟			350ns

4、 输入

参数	最小值	典型值	最大值
触发输入		1 路/通道， SMA 连接器（前面板）	
触发输入阻抗		50 Ω	
触发输入幅值范围			5 V (into 50 Ω)
参考时钟输入		BNC 连接器 （后面板）	

参考时钟输入频率		10/100 MHz	

5、 时域频域特性

参数	最小值	典型值	最大值
采样率		1.2GSa/s	
上升时间（20% 到 80%）			< 1 ns （1Vpp）
通道间偏差			< 150 ps
SFDR (不含谐波)	74 dB (@10 MHz) 43 dB (@300 MHz) (1Vpp)		
谐波失真			HD2 -48 db@100 MHz HD3 -46 db@100 MHz （1Vpp）

6、 最大额定值

参数	最小值	典型值	最大值
波形输出（V）	-5.5		5.5
触发输入损坏阈值（V）	-0.5		5.5
Marker 损坏阈值（V）	-5.8		5.8
参考时钟输入损坏阈值（V）	0		3.3
参考时钟输出损坏阈值（V）	-5.8		5.8

7、 推荐 PC 配置

参数	描述
CPU	Intel Core i5-6400@2.7GHz 及以上
内存	4GB 以上
网口通讯	1GbE
操作系统	Windows 10

10.维护保养

10.1 使用和保养

1. 请勿将仪器放置在高温、湿气极重或受日光直射的地方；
2. 请勿将仪器暴露在灰尘、烟雾或蒸汽中；
3. 请勿将仪器放置在盐雾，酸碱及其它会产生腐蚀气体或物质环境中；
4. 请勿将液体或小颗粒掉入仪器中；
5. 请勿将仪器放置在倾斜、不平稳或易受振荡的地方；
6. 请勿投掷、掉落或踩踏仪器，或使仪器受到强烈的外力冲击；
7. 请勿在仪器上放置重物；
8. 请勿触摸或将异物插入仪器的端子部分。

10.2 清洁

请根据使用情况定期对仪器进行清洁，方法如下：

1. 请在开始清洁前，先自电源插座中拔出交流电源线；
2. 使用软布轻柔拭擦，请勿使用溶剂或其他化学药剂来清洁主机外壳；
3. 连接端子若不干净，请勿继续使用，使用干布或者棉质纱布擦拭灰尘，若在脏污时使用，可能损坏设备或者影响设备性能。

注意

请勿使任何腐蚀性的液体沾到仪器上，以免损坏仪器。

警告

重新通电之前，请确认仪器已经干透，避免因水分造成电气短路甚至人身伤害。

11. 常见故障或问题处理

11.1 计算机无法 ping 通设备

在支持千兆网卡的计算机上连接仪器，ping 命令没有响应，可进行下列尝试。

1、检查计算机是否是千兆网卡。

2、检查网线是否支持千兆网。

3、检查交换机是否为千兆交换机。

4、检查电脑 IP 地址是否与板卡是同网段(ping 协议需要同网段，但是连接不需要)。

5、重启设备。

11.2 计算机可以识别设备但软件中设备连接失败

计算机的设备管理器显示正确识别设备但是软件中设备连接失败，则重启软件进行尝试，如果还是显示无法连接，重启设备。

11.3 设置正确但无信号输出

1、检查 SMA 线缆是否接紧。

2、检查波形及其设置是否正确：模块开启的按钮是否正确；开启的模块，是否波形都已进行下载。

3、检查示波器等设置是否正确：示波器的触发模式是否设置正确（特别是对于 sequence 播放模式，如果设置的等待时间过长，使用示波器的自动(Auto)模式一般无法看到波形，需要切换到正常（Normal）模式或者单次触发（Trigger 模式）才可以看到波形。

4、断开连接，看是否能够重新连接设备，如果发现连接报错，检查网络的状态，看是否网络连接已经断开。

5、重启设备。

若以上操作完成后仍无法解决问题，请及时与我司联系。

11.4 连接正常，设置波形后有输出，但是输出异常

1、检查 SMA 线缆是否接紧。

2、检查波形及其设置是否正确。

3、检查示波器设置是否正确：示波器采集时，需要设置直流 50 欧姆的阻抗匹配（Coupling: DC 50Ω）。

4、断开连接，看是否能够重新连接设备，如果发现连接报错，检查网口的状态，看是否网络连接已经断开。

5、断开连接后重连成功，但是波形仍然不对，可尝试点击设置中的‘复位’按钮（在波形输出或者下载过程中，不可以切换时钟源），然后再进行测试。

6、重启设备。

若以上操作完成后仍无法解决问题，请及时与我司联系。

11.5 编写的波形函数输出的波形不完整，比如正弦波每个重复内最后的正弦波都不完整

这是因为波形长度必须是 8 的整数倍，如何保证输出的波形完整，详细计算方法如下：

计算公式：

Len*1/1.2G = 1000/f *N（f 在这里是 MHz）

Len=1000/f*N*1.2G

其中参数介绍如下：

Len：输出的波形长度

f：设置的波形频率(单位 MHz)

N：正整数

限制条件：

Len*Loop >=56

Len*Loop 必须位 8 的正整数倍。（Loop 为循环次数）

示例：

示例 1：M=10 MHz，Loop=1

则：Len=1000/10*N*1.2=120*N

Len*Loop=120*N，则可以同时满足两个限制条件，则可以设置 Len 为 120、240……，都是可以的。

示例 2：M=200 MHz，Loop=1

则：Len=1000/200*N*1.2=6*N

Len*Loop=6*N，则为了满足两个限制条件，则 N 可以是 16、20……，即 Len=96、120 等。

11.6 新安装软件后，搜索不到设备

1、请按照 4.1.1 章节对照查看操作步骤。

2、关闭软件后，在官网上下载驱动程序 WinPcap.exe，并安装。

3、如果是笔记本电脑，使用了 USB 转网口的转换器，则需要检查是否已经 USB 转网口的驱动。

4、如果是使用路由器连接，可以尝试使用直连的方式。

5、重新启动软件进行设备搜索。

11.7 使用 Python 调用 SDK 时，编译报错

1、SDK 分为 32 位、64 位两个版本，请确认使用了跟 Python 的环境匹配的版本。

2、确认是否是环境配置问题，请尝试简单的常规 Python 程序是否可以正常运行。

12. 索引

12.1 图形索引

12.2 表格索引

用 量 子 技 术 感 知 世 界

FEEL THE WORLD IN A QUANTUM WAY

国仪量子(合肥)技术有限公司

地址：合肥市高新区创新产业园二期E2楼

无锡量子感知技术有限公司

地址：无锡市惠山区惠山站区站前路2号

4000606976-602

www.ciqtek.com

service@ciqtek.com

国 仪 量 子 公 众 号

国仪量子售后服务小程序