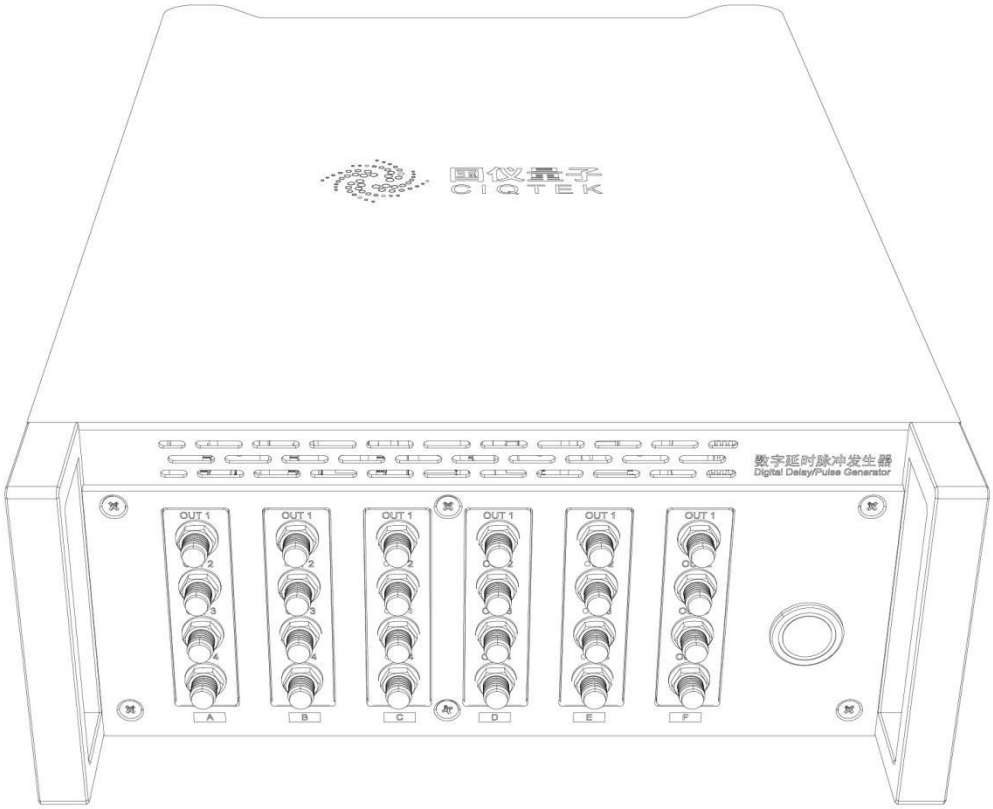
用户手册

数字延时脉冲发生器 ASG24100



国仪量子
CIQTEK

版权所有

商标信息

版本历史

本手册版本为 V1.0.0，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
V1.0.0	2021.08.15	该版本手册为 ASG24100 初版	
……			

保证和声明

软件版本

软件升级可能会修改产品功能，请联系国仪量子（合肥）技术有限公司升级软件，必要时我司会主动与您联系。

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。
- 未经我司事先书面许可，不得影印、复制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。

质保

公司货物免费保修一年，时间自最终验收合格并交付使用之日起计算。

联系我们

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

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 ,用户需了解以下安全措施，并按照规定使用本产品。

一般安全概要

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。

确保供电电源正确

为避免对操作人员造成伤害或损坏产品，请在使用产品前仔细阅读本手册，并确保产品供电电源正确。

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。

如果仪器的安装位置或环境，操作员无法接触到主电源开关或电源线，则必须在仪器附近 （操作员伸手可及的地方）提供单独的主输入电源断开开关。

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。

防静电保护

本产品包含可能因静电放电（ESD）而损坏的电子元件，若需要将仪器安装到系统中或放置于保护导电包装中，请先接触接地的裸金属表面或经批准的防静电垫，进行放电。

连接器注意事项

本产品上使用的连接器设计用于高信号质量和良好屏蔽。由于前面板上的空间有限，它们必须尽可能小以便在前面板上安装所需的信号连接；因此，如果使用不当，这些连连接器可能受到机械损坏，尤其是 SMCC 和 MMCX 连接器。

务必通过操作连接器本身而不是电缆来拆卸连接。始终从板连接器沿直线移动电缆连接器。断开连接电缆是，不要倾斜连接器接头。损坏的接头只能由原厂更换，且不在保修范围内。

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。

怀疑产品出故障时，请勿进行操作。

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。

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

为避免对产品面板上的按键、接口、指示灯等部件造成损坏，请注意搬运安全。

异常处理

异常故障	处理
风扇工作不正常或不工作	立即关闭仪器，防止电子元件过热而损坏。

电源线或电源插头损坏	立即关闭仪器，防止仪器过热、电击或起火等危险。 请使用本产品专用并经所在国家/地区认证的电源线。
仪器散发烟气、异常的噪音、气味或火花	立即关闭仪器，以免造成更大的损坏。
仪器损坏	立即关闭仪器，确保仪器不会发生意外操作。

安全术语和符号

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

以下术语和符号可能出现在产品上：

“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。

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。

此符号表示仪器对静电放电（ESD）敏感，静电放电可能会内部电路元件造成永久性损伤

避免人员接触以防灼伤

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。

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。

环保处置

本产品中包含的某些物质可能会对环境或人体健康有害，为了避免将有害物质释放到环境或危害人体健康，切勿将本仪器处理为未分类的废弃物，本仪器需做分类回收，以确保大部分材料可以正确地重复使用或回收，有关处理或回收信息，请联系当地相关部门。

目录

1、产品介绍

1.1 检查运输包装

ASG24100 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，若有任何损坏请立即向货运方或国仪量子（合肥）技术有限公司联系，因为此类损坏不在保修范围内。
- 打开仪器的包装，检查其是否有任何运输损坏，若有任何损坏请立即向货运方或国仪量子（合肥）技术有限公司联系，因为此类损坏不在保修范围内。
- 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。

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。

本产品提供以下的随机配件：

包装内容	**数量**	**单位**
ASG24100 数字延迟脉冲发生器	1	台

附件		
网线	1	根
电源线	1	根
ASG24100 用户手册	1	份
合格证	1	份
检验报告	1	份

1.3 检查整机

如仪器存在机械损坏，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。

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，保留干燥。
- 存储温度范围-30~60℃。
- 仪器若需要重新包装运输，需要注意以下要求：
  - 在重新包装运输时，使用足够强度和空间的纸箱放置仪器，并使用缓冲包装和缓冲材料包装防护。
- 在运输过程中，请注意避免剧烈震动影响。

1.5 产品技术规格参数

参数名称	参数值
时间分辨率	1ns

通道数	24
输出电平	3.3V@1MΩ/50Ω 5VΩ1MΩ
上升时间	≤800ps@3.3V/50Ω ≤1.4ns@3.3V/1MΩ ≤1.2ns@5V/1MΩ
脉宽范围	1~(2^{32}-1) ns
延迟范围	0~(2^{32}-1) ns
延迟分辨率	52ps
脉宽抖动（RMS）	≤35 ps

COUNTER 参数名称	参数值
通道数	4
输入电压阈值	1V@50Ω
最小序列宽度	1ns
脉冲输入最大频率	125MHz

触发信号名称	参数值
通道数	1
输入电压阈值	1V（50Ω）
外部触发到通道输出延迟	trig delay(168 ns)+timebase jitter(≤1 ns)

外时钟信号名称	参数值
时钟输入通道数	1
时钟输入电压范围	1~5V@50Ω
时钟输入时钟频率	10M/100MHz 时钟
参考时钟输出通道数	1
时钟输入电压范围	1~5V@50Ω
时钟输出时钟频率	40MHz~80MHz 可变时钟

2.使用前准备

为避免造成人身伤害或产品损坏，严禁不具备操作能力的人（未经过培训的人）使用本产品。

2.1 使用前检查

为了确保安全，在使用 ASG24100 之前请参阅本使用手册的前述部分的“安全注意事项”。

2.2 仪器的安装

2.2.1 一般注意事项

- 使用仪器时，须严格按照使用手册指导操作
- 打开电源前，确保供电电压与额定电压匹配
- 严禁仪器处于带故障操作，以免引起意外事故及控制失真
- 禁止乱拆仪器的机箱，以免仪器仪器使用异常

2.2.2 环境要求

特性	说明
温度	工作温度：-20℃～50℃ 存贮温度：-30℃～60℃
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
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 操作软件安装

用户可在官网上找到 ASG24100 产品详情页，在详情页中的资料下载模块可下载到软件安装程序，下载到本地后，双击 ASG24100_setup 可按照安装流程将软件安装至计算机上，如图 2.3.1 所示。

图 2.3.1 ASG 24100 安装程序

选择安装路径，勾选桌面快捷方式和用户许可协议后，点击快速安装即可，如图 2.3.2 所示

图 2.3.2 ASG 24100 安装流程

3.快速启动

本章节适用于初次拿到 ASG24100 设备的用户，我们通过演示设备在自由方波和连续方波模式下输出方波信号，来讲解如何快速启动和应用该仪器。同时我们将演示两台 ASG 设备同步使用的场景，该实验过程需要的具体设备清单如下：

- 2 台 ASG24100
- 若干根 SMA 连接同轴电缆
- 1 台示波器（SDS3104X）

3.1 准备工作

实验开始之前，需要完成以下准备工作：

- 保证上位机已安装软件及网口驱动
- 确认示波器采集通道可以正常工作
- 确认信号源输出通道可以正常工作

3.2 打开操作软件

1、ASG24100 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（如图 3.2.1 所示），打开软件，进入设备连接界面。

图 3.2.1 ASG24100 操作软件快捷方式

2、选中需要连接的设备,点击连接,连接成功会跳转设备控制页面。这里我通过路由器将两台 ASG24100 与主机连接，所以在设备列表会显示两条设备信息，如图 3.2.2 所示。

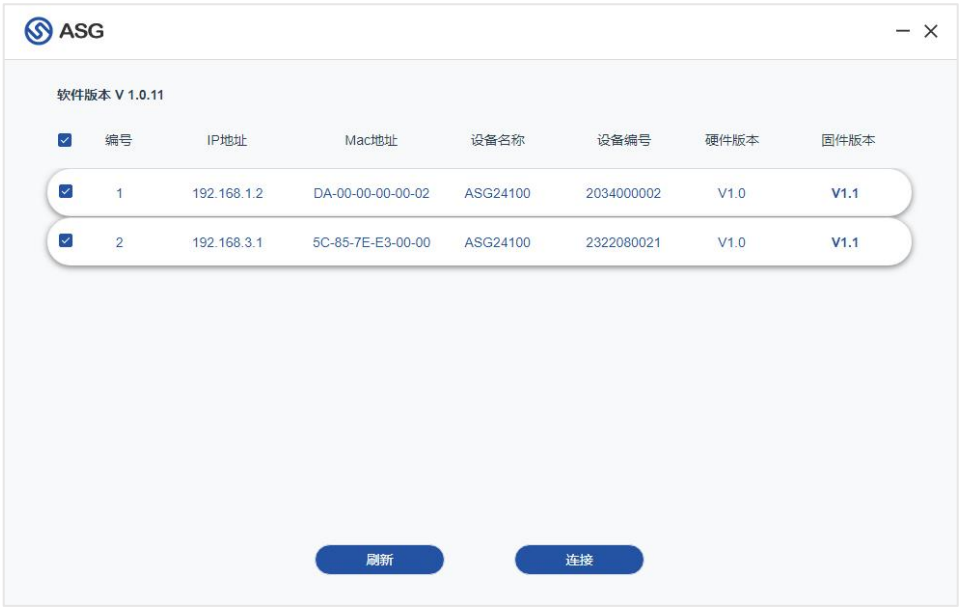


图 3.2.2 ASG24100 操作软件连接模块

3.3 输出设置及结果展示

本章将介绍 ASG24100 在连续方波及自由方波模式下进行方波输出，设备连接完成后直接跳转至操作界面，我们可直接进行参数设置。

3.3.1 连续方波输出设置

连续方波模块输出的波形为连续的周期性波形 ,涉及到的参数设置有周期、高电平、延迟 ,需要注意的是周期与高电平的设置需要是 4ns 的整数倍 ,延迟范围为 0~4000ns，延迟分辨率为 52ps，这三个参数在播放过程中是任意可调的。我们将 A、B 两张子卡的 OUT1 打开，实际上我们打开了通道 1 和通道 5，将周期设置为 40ns，高电平设置为 12ns，同时将 A 子卡的输出电平设置为 3.3V，匹配阻抗设置为 50Ω，将 B 子卡的的输出电平设置为 5V，匹配阻抗设置为 1MΩ，参数设置如图 3.3.1。

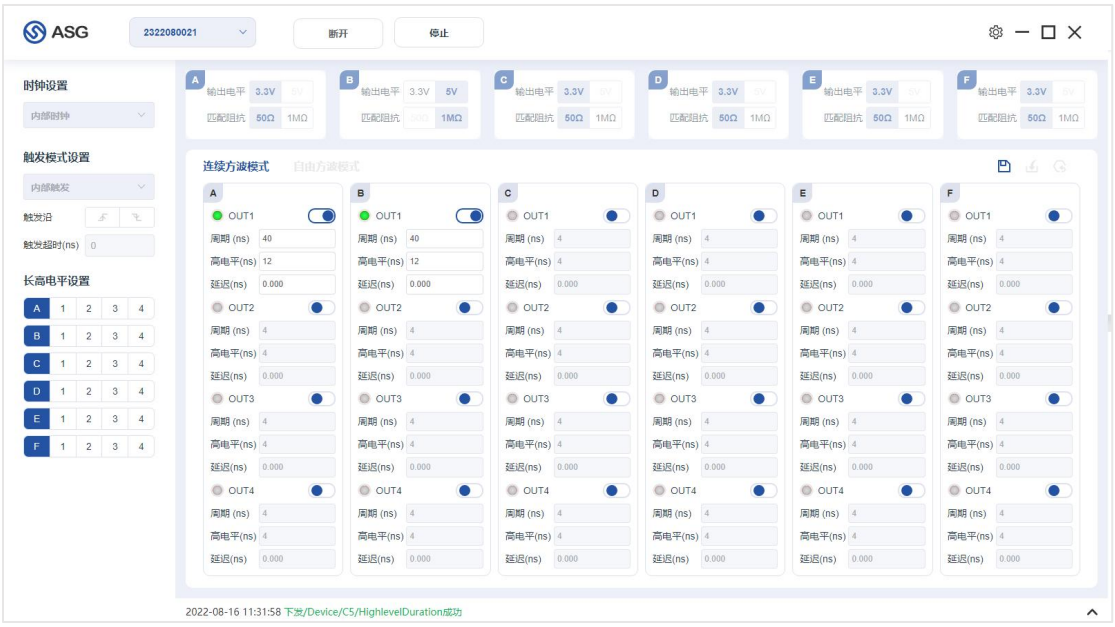


图 3.3.1 ASG24100 操作软件参数设置

参数设置完成后，点击播放按钮，在示波器上查看波形输出情况，如图 3.3.2。

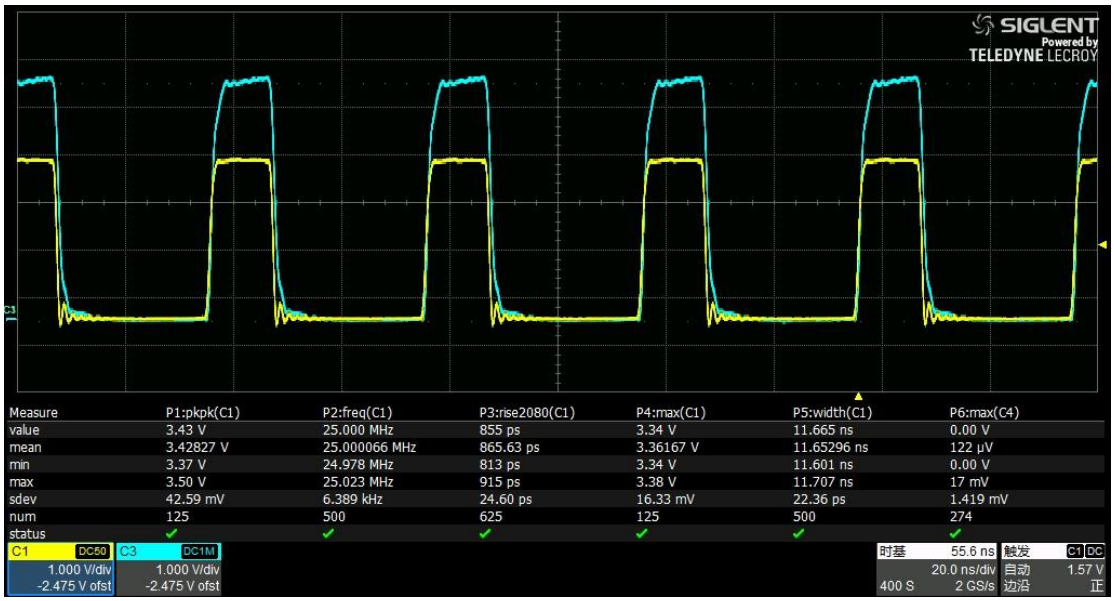


图 3.3.2 示波器输出结果

3.3.2 自由方波输出设置

在自由方波模式下可以编辑较为复杂的方波，并且提供了丰富的方波编辑函数（该内容将在 4.5 中详细介绍），同时在自由方波模式下可以使用 Counter 计数功能，这里我们将 A 板卡的 OUT3、OUT4 接入后面板的 Counter1、Counter2 接口，参数设置及采集结果如图 3.3.3 所示。

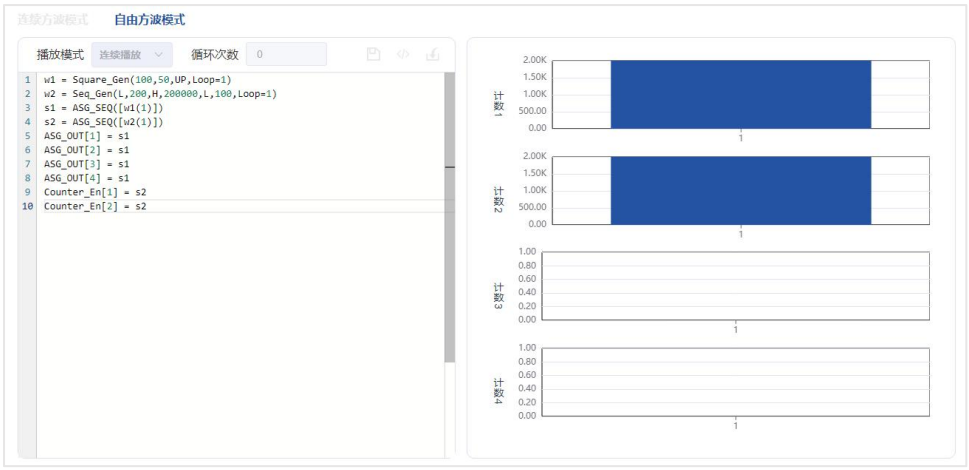


图 3.3.3 自由方波模式参数设置

在示波器上可以看到 OUT1、OUT2 接口输出波形，如图 3.3.4。

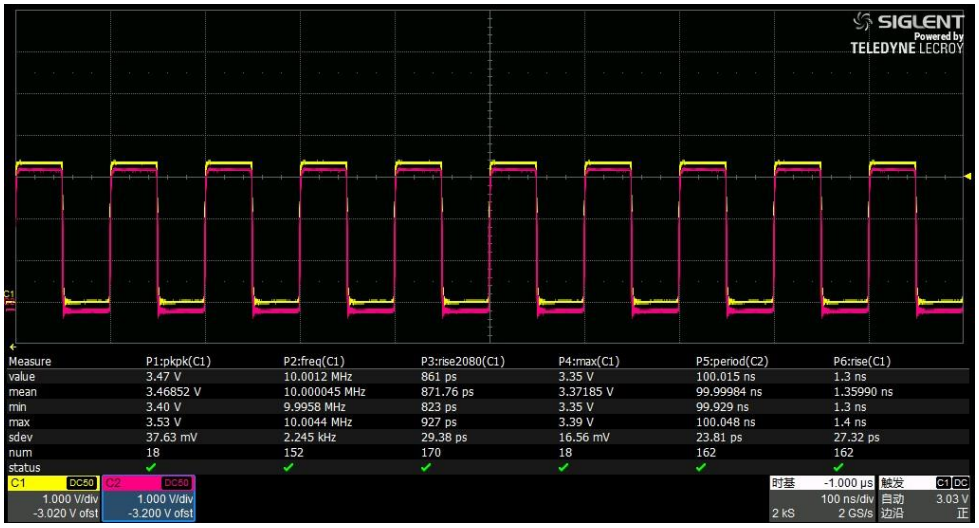


图 3.3.4 示波器输出结果

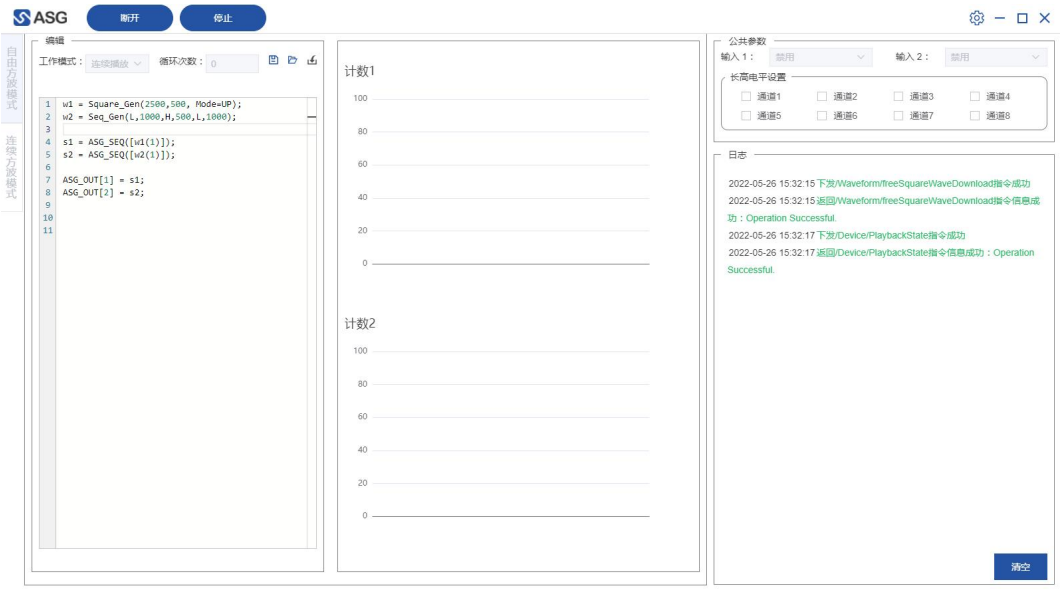


图 3.3.1 ASG24100 操作软件参数设置

参数设置完成后，点击编译按钮按钮，编译完成后点击播放，在示波器上查看波形输出情况，如图 3.3.2 所示。

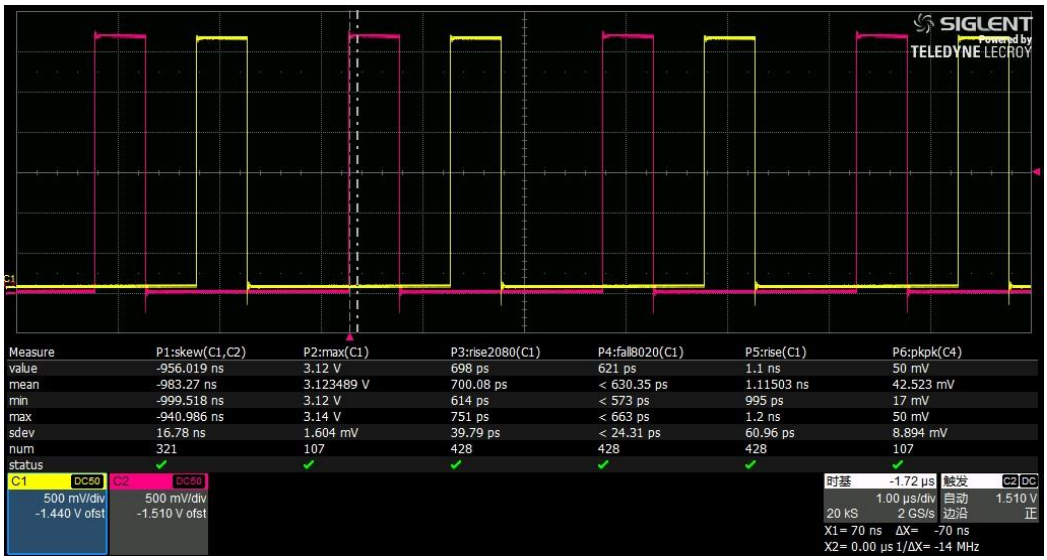


图 3.3.2 示波器输出结果

4.功能概述

4. 1 面板介绍

ASG24100 面板接口的布置参见如图 4.1.1 所示和表 4.1.1 说明。

图 4.1.1ASG24100 前后面板

表 4.1.1ASG24100 面板说明

位置	名称	描述
A	子卡 A	通道 1~4，通道输出 3.3V 或 5V 的波形，可匹配 50

		Ω、1MΩ
B	子卡 B	通道 5~8，通道输出 3.3V 或 5V 的波形，可匹配 50Ω、1MΩ
C	子卡 C	通道 9~12，通道输出 3.3V 或 5V 的波形，可匹配 50Ω、1MΩ
D	子卡 D	通道 13~16，通道输出 3.3V 或 5V 的波形，可匹配 50Ω、1MΩ
E	子卡 E	通道 17~20，通道输出 3.3V 或 5V 的波形，可匹配 50Ω、1MΩ
F	子卡 F	通道 21~24，通道输出 3.3V 或 5V 的波形，可匹配 50Ω、1MΩ
G	电源开关	电源开关按钮
H	时钟输出	输出 100MHz 时钟信号
I	参考时钟输入	可接入 20~80M 可变时钟信号
J	时钟输入	可接入 10/100M 时钟信号
K	触发输入	可接入外部触发信号，触发阈值 1V
L	Counter 计数 1	计数通道 1，输入电压范围[1,5V],最大输入频率 125MHz
M	Counter 计数 2	计数通道 2，输入电压范围[1,5V],最大输入频率 125MHz
N	Counter 计数 3	计数通道 3，输入电压范围[1,5V],最大输入频率 125MHz
O	Counter 计数 4	计数通道 4，输入电压范围[1,5V],最大输入频率 125MHz
P	LAN 口	千兆网口
Q	USB3.0	预留接口

R	供电接口	交流 220V，电源入口和电源开关

4.2 输出模式介绍

4.2.1 连续方波模式

在连续方波模式下，可以通过设置通道的周期、高电平时间以及延迟时间实现周期性方波的连续输出。在该种模式下，参数均可实时修改，修改完成后，序列将在下一个周期发生改变。需要注意的是，优于 ASG24100 内部使用 2 5 0 MHz 的时钟信号作为内部时钟，所以周期及高电平的设置需要为 4ns 的整数倍。延迟设置的分辨率为 52ps，若设置的延迟参数非 52ps 的整数倍，上位机将自动近似调整。

4.2.2 自由方波模式

在自由方波模式下，我们提供了详细的序列编辑接口，可编辑时序更为复杂的方波。需要注意的是，在自由方波模式下有连续播放和组合播放两种工作模式，如图 4.2.2 所示。下面我将详细介绍两种个播放模式的区别。

图 4.2.2 工作模式

我们以一段序列为例，首先我们编辑这样一段序列，如图 4.2.3 所示。

```
####使用函数库编辑基本序列集合
w1 = Seq_Gen(L,300,H,400);
w2 = Seq_Gen(L,300,H,600);

####使用ASG_SEQ定义基本序列的组合方式
s1 = ASG_SEQ([w1(2)]);
s2 = ASG_SEQ([w2(2)]);

####将ASG_SEQ定义的序列赋值给相应的通道
ASG_OUT[1] = s1;
ASG_OUT[2] = s2;
```

图 4.2.3 示例序列

改序列在不同的工作模式下将以不同的时序逻辑体现，首先在连续方波模式下，时序逻辑如图 4.2.4 所示。可以发现在连续方波模式下，通道间是相互独立的。

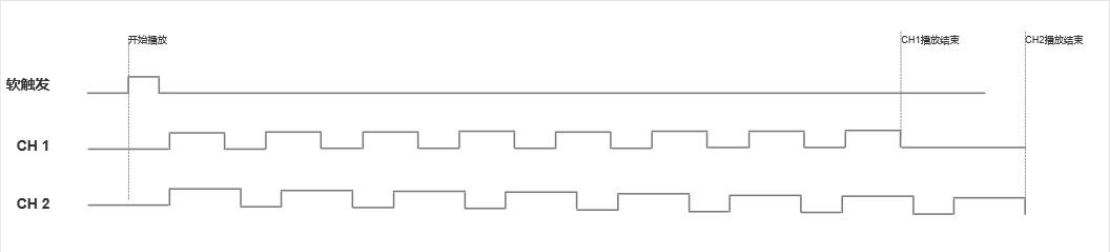


图 4.2.4 波形输出

在组合播放模式下，时序逻辑如图 4.2.5 所示。可以发现 CH1 多了两段等待时间，这是因为序列的输出单位是 SEQ，在组合播放模式下每个通道的 SEQ 是相互关联的，通道输出会等到最长的 SEQ 结束后再开始下一个周期的输出。

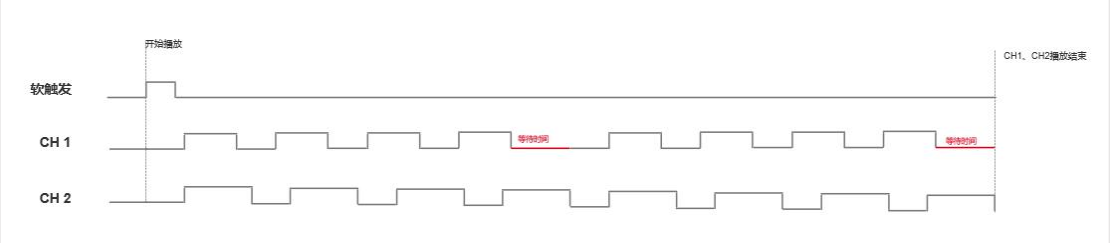


图 4.2.5 波形输出

4. 3 触发模式介绍

ASG24100 可以使用软触发下发开始工作指令 ,同时我们也提供了 Trigger in 接口外接触发信号，这个章节将详细介绍使用外触发，在自由方波和连续方波模式下的触发逻辑。

4.3.1 外触发下的连续方波

在连续方波模式下，通道输出的是固定周期和占空比的方波信号，所以在连续方波模式下，设备识别到外部触发信号后，通道就开始连续播放。播放逻辑如图 4.3.1。

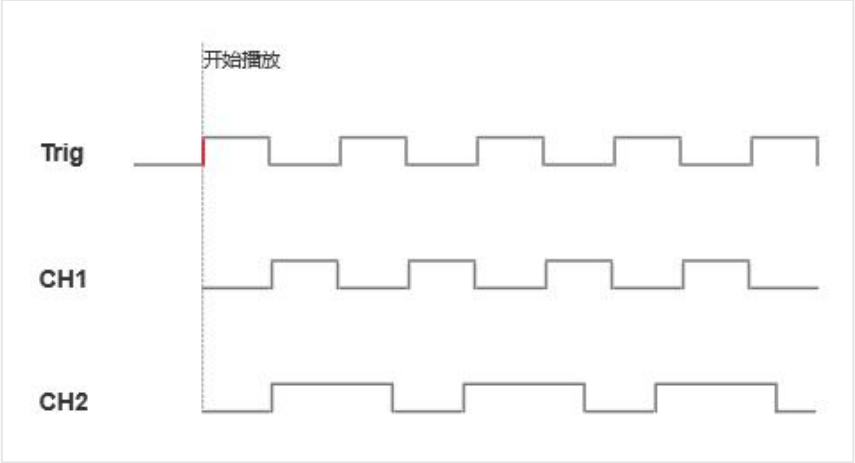


图 4.3.1 播放逻辑

4.3.2 外触发下的自由方波

在自由方波模式下需要区分连续播放和组合播放两种工作模式，在连续播放模式下，通道间是相互独立的，所以每段 SEQ 播放结束之后只需要等待下一个 Trigger 信号即可。在这里我们依然使用 4.2.2 章节中使用的示例代码作为通道 1、2 的输出信号，播放逻辑如图 4.3.2。

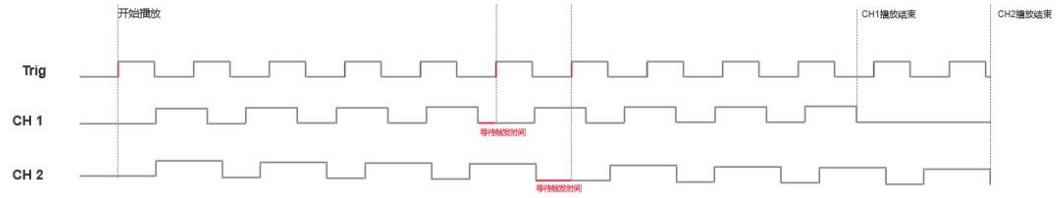


图 4.3.2 播放逻辑

在组合播放模式下，由于输出通道间的 SEQ 是相互关联，所以所有通道的第一段 SEQ 会全部播放结束后，等下一个 Trigger 信号到来，才会来时播放下一段 SEQ。

图 4.3.3 播放逻辑

4.4 Counter 计数功能

ASG24100 的 IN 1 、IN2 两个接口可以作为计数通道使用，下面这个章节将详细介绍 Counter 模块的计数逻辑。

首先我们将 IN 1 的工作模式设置为计数模式，我们在序列编辑框内编辑如图 4.4.1 示例序列。Counter 的计数逻辑为高电平采集，低电平不采集，高电平采集结果会做累加，每个高电平采集到的数据都会上传至上位机，并在计数区域将累加结果绘制出来，采集逻辑如图 4.4.2。

```
####使用函数库编辑基本序列集合
w1 = Seq_Gen(L,300,H,700);
w2 = Seq_Gen(L,100,H,10000,L,100,H,5000)

####使用ASG_SEQ定义基本序列的组合方式
s1 = ASG_SEQ([w1(1)]);
s2 = ASG_SEQ([w2(1)]);

####将ASG_SEQ定义的序列赋值给相应的通道
ASG_OUT[1] = s1;
Counter_En[1] = s2
```

图 4.4.1 示例序列

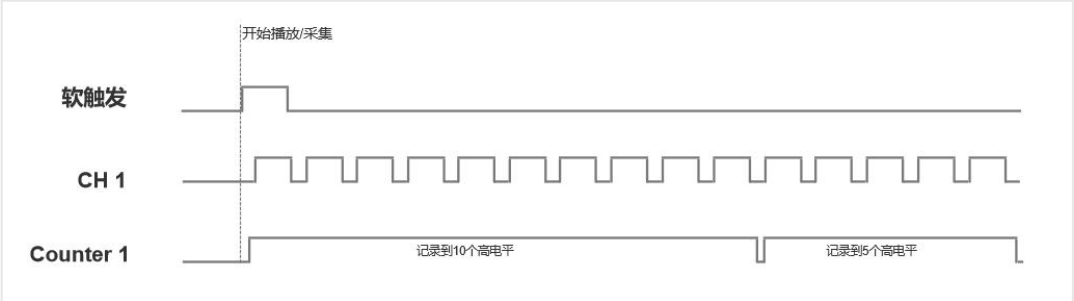


图 4.4.2 Counter 采集逻辑

需要注意的是在采集过程中 ,优于 Counter 序列中的每个高电平都会上传数据至上位机，上传的速率可至 n s 量级，然而上位机不可能以这种速率刷新，所以上位机图像绘制刷新的速率是每秒 5 次。如果需要 counter 采集到的所有数据，可以通过回调获取该数据。图 4.4.3 为 Counter 采集到的结果。

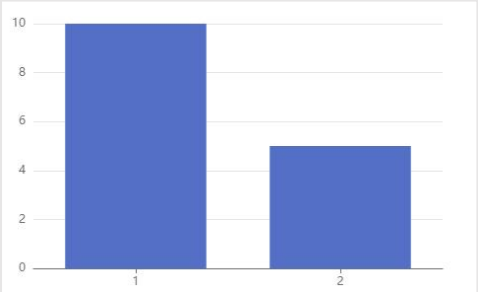


图 4.4.3 Counter 采集结果

4.5 序列编辑

4.5.1 序列编辑

4.5.1.1 函数类型

基本函数库包含了方波生成函数、自定义序列编辑函数和序列拼接函数。

1、方波生成函数

函数原型为 Square_Gen(Period, Width, Mode, Loop)，其相关参数解释如下：

Period: 必选，正整数，方波周期，单位 ns，精度 1ns，1~2^32-1

Width: 必选，正整数，方波脉冲宽度，单位 ns，精度 1ns，0~2^32-1

Mode: 可选，方波模式，UP 是先低后高，DOWN 是先高后低

Loop: 可选，正整数，方波循环次数，默认 1，0~2^32-1

示例：

```
w1 = Square_Gen(16,8, Mode=UP);
```

2、自定义序列编辑函数

函数原型为 Seq_Gen(Level1,Time1,Level2,Time2,…, Leveln,Timen,Loop)，其相关参数解释如下：

Level1，Level2，….，Leveln: 必选，H 表示高电平，L 表示低电平
Time1，Time2，….，Timen：必选，电平持续的时间，单位 ns，精度 1ns，0~2^32-1
Loop: 可选，正整数，方波循环次数，默认 1，1~2^32-1。

示例：

```
w1 = Seq_Gen(H,1600,L,300,H,100);
w2 = Seq_Gen(L,1600,H,300,H,100,Loop=10);
```

4.5.1.2 Seq_Join

函数原型为 Seq_Join (Seq_Gen1, Seq_Gen 2,..., Seq_Genn),其功能是可以将多个序列拼接在一起，组成一个新的序列。

其相关参数解释如下：

Seq_Gen1, Seq_Gen 2,..., Seq_Genn：自定义序列函数生成的序列

示例：

```
w4 = Seq_Join(W1,W2,W3); #将 W1,W2,W3 拼接组成一个新的序列
```

注意：由函数生成的基本序列的长度要满足 4*n（单位 ns），不足自动补零

4.5.1.3 ASG_SEQ 类型

一个 ASG_SEQ 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，子序列通过中括号来区分。
ASG_SEQ([w1(重复次数), …, wn(重复次数)]), 重复次数为 2^16-1
其中 w1,…,wn 是 n 个序列，n >= 1。重复次数为正整数。

示例：

```
# 写出所有波形
s1 = ASG_SEQ([w2(40),w3(2)]);                # 包含 2 个子序列
s2 = ASG_SEQ([w1(5000),w2(2),w3(1)])         # 包含 3 个子序列
s3 = ASG_SEQ([[w1(5),w2(2)],w3(1)])          # 包含 2 个子序列
s4 = ASG_SEQ([[w1(5),w2(2)],[w3(1), w4(100)]])# 包含 2 个子序列
```

4.5.1.4 ASG_ADVS 类型

通过调用一个 ASG_SEQ 类型可生成一个 ASG_ADVS，ASG_ADVS 支持“+”和“+=”运算。
多个 ASG_ADVS 可以拼接为一个 ASG_ADVS。

示例：

```
a1  = ASG_ADVS();
a1 += s1(3) + s2(2);
a2 = ASG_ADVS(s1(3) + s2(2)); # 与 a1 等效
```

```
a3 = s1(3) + s2(2);          # 自动处理类型，与 a1 等效
```

4.5.1.5 ASG_OUT[n]类型

n:范围为 1~8，分别代表相应的通道输出。

常高电平由软件界面控制。

4.5.1.6 Counter_En[n]类型

Counter_En[n]类型是用作 counter 计数用，仅内部在触发模式下使用，可以将 ASG_SEQ 类型的序列赋值给 Counter_En[n]。n 的值为 1 或者 2，代表计数通道 1 和 2。

5.软件功能介绍

5. 1 连接界面

ASG24100 数字延迟脉冲发生器的连接界面主要包括“设备列表模块”、“连接操作模块”，如图 5.1.1 所示。

图 5.1.1 设备连接界面

表 5.1.1 ASG24100 设备连接模块说明

序号	名称	描述
1	设备列表模块	在该模块用户可以看到通过 LAN 连接的所有 ASG 设备，列表中会显示设备编号、IP 地址、Mac 地址、设备名称、设备编号、硬件版本、固件版本，同时在左上角会显示目前软件版本状态
2	连接操作模块	1）“刷新”：点击该按钮可重新刷新获取设备列表信息； 2）“连接”：在设备列表中选中需要连接的 ASG24100 设备后，点击“连接”，连接成功后会跳转至设备操作页面。

5.2 操作界面

ASG24100 的操作界面主要包括“播放控制及窗口控制模块”、“公共参数设置”、“输出幅值及匹配阻抗设置”、“输出序列编辑”、“日志”这四个模块。

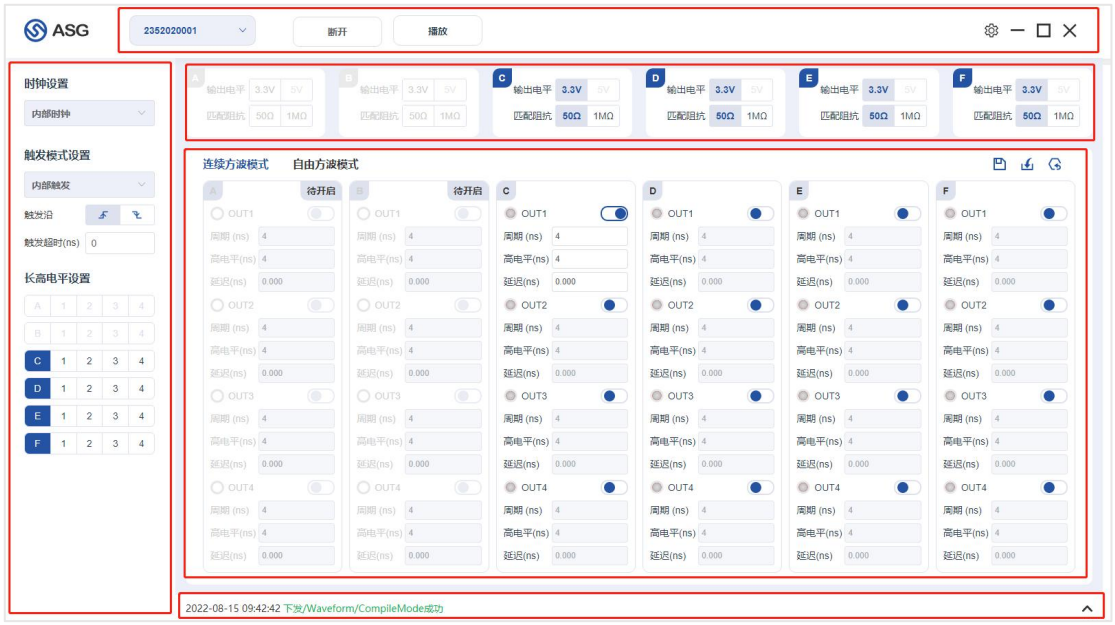


图 5.2.1 设备操作界面

5.2.1 播放控制及窗口控制模块

表 5.2.1 ASG24100 播放控制及窗口控制模块说明

序号	名称	描述
1	设备选择	如果在设备连接界面选中多个设备连接，那么点击“设备选择”会显示所有已连接的设备，用户可选择任意一台进行操作；
2	断开	触发“断开”按钮,设备会先保存此时参数,再跳转至连接页面；

3	播放	参数设置完成后，触发“播放”按钮，参数下发成功后，设备会开始正常输出；
4	设置	1、显示软件、固件版本信息，若存在最新的软件或者固件可在线进行软件和固件升级； 2、中英文切换
5	窗口操作	可进行最小化、最大化、关闭该窗口操作

5.2.2 公共参数设置模块

公共参数设置模块包含时钟设置、触发模式设置、长高电平设置，图 5.2.2 为公共参数设置模块。

图 5.2.2 公共参数设置

表 5.2.2ASG24100 公共参数设置说明

序号	名称	描述
1	时钟设置	１、内部时钟：设备默认使用内部时钟，内部时钟为２５０MHｚ； ２、外部１０M：若选择外部１０M，需要ｃｌｋ　in 接口接入１０M 的时钟信号，用于设备间的同步； ３、外部１００M：若选择外部１００M，需要ｃｌｋ in 接口接入１００M 的时钟信号，用于设备间的同步； ４、外部可变时钟：若选择外部可变时钟，ｃｌｋ　in 接口可接入２０～８０M 可变时钟，设备会将外接时钟信号作为 FPGA 的工作时钟进行工作。
2	触发模式设置	1、触发模式选择：设备默认是使用内部触发，即需要使用上位机的“播放”按钮控制设备的输出；若选择外部触发模式，需要 trig in 接口接入触发信号，设备会在接收到触发信号后开始工作； 2、触发沿选择：可选择“上升沿”或“下降沿” 3、触发超时：若用户选择外部触发，那么可以设置触发超时参数，若超过设置的超时时间还没接收到触发信号，那么该次播放结束，超时可以设置的范围 0~1s，分辨率为 1ns。
3	长高电平设置	若用户选择长高电平输出，通道可输出长高电平，此作做优先级最高，若通道处于长高电平输出状态，那么该

		通道不能做其他操作；

5.2.3 输出序列编辑模块

在输出序列设置模块，我们为用户提供了两种方式编辑序列，以应对不同的应用场景，分别是连续方波模式和自由方波模式。在连续方波模式下，设备可输出确定周期的连续方波，需要注意的是在任意可变时钟模式下，参数设置与其他时钟模式有一定区别，如图 5.2.3 所示。

图 5.2.3 连续方波编辑模块

表 5.2.3 ASG24100 连续方波编辑模块说明

序号	名称	描述
1	通道状态灯	通道状态灯展示通道输出状态，当通道正常输出时，状态灯显示绿色，当通道未输出或异常时，通道灯显示灰色；
2	通道开关	当通道开关处于打开状态时，参数设置才是有效的，点击“播放”按钮时，只有通道开关处于打开状态的通道

		才能正常输出；
3	周期	在非任意可变时钟状态下，周期设置为 4ns 的整数倍，范围[4,2^{32}ns]，若用户设置非 4ns 的整数倍时，上位机会近似为 4ns 的整数倍;在任意可变时钟状态下，周期设置为输入时钟信号周期的整数倍，范围[1,2^{32}]；
4	高电平	在非任意可变是时钟状态下，高电平设置为 4ns 的整数倍，范围[4,2^{32}ns]，需要注意的是，若高电平与周期相同时，通道输出长高电平;在任意可变时钟状态下，高电平设置为输入时钟信号周期的整数倍，范围[1,2^{32}]；
5	延迟	该延迟是指通道在接收到触发信号后(内部触发/外部触发)，通道会按照设置的延迟时间，等待一段时间才开始播放；在非任意可变时钟状态下，该参数的范围是[0,4000ns],分辨率是 52ps，在任意可变时钟状态下，该参数的范围是[0,12ns+可变时钟周期*n],分辨率是 52ps

在自由方波模式下，用户可编辑时序更加灵活的序列输出，同时在自由方波模式下，用户可使用 counter 接口开启计数功能，如图 5.2.4 所示。

5.2.4 日志模块

在日志模块可以清晰的看到每一步操作的反馈，会提示操作成功或者失败。这个功能对于序列编辑时是非常有效的，可以提示报错原因，以便于迅速定位到问题，如图 5.2.6 所示。

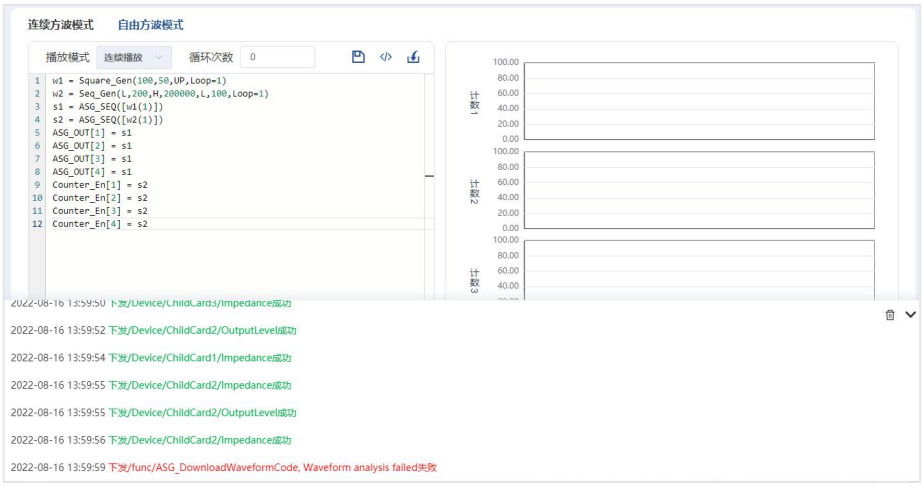


图 5.2.6 日志模块错误提示

6. 接口使用说明

6.1 C++接口说明

参数设置使用字符串指令集形式调用固定 API，在设置参数之前需要保证已打开服务端软件，并正确连接你想要连接的设备。在软件的日志区会实时打印出操作设备所下发的指令和值，务必按照日志的顺序和内容调用 SDK。同时，在某些参数设置中，需要同时设置 2 个或 2 个以上的参数，下面会单独列出各自 API。

以下按照软件页面和模块分类介绍各个参数设置。

注：每个 API 中列出的第一个参数是设备编号，第二个参数是该参数的指令，第二个参数的值是用于示例，并不代表设备内的参数就是该值。

1、SDK 初始化

函数	int ASG_Init(char *sdk_version);
功能	初始化 SDK 并传回 SDK 版本号
参数	[sdk_version] SDK 版本号，此内存需要调用者申请。
返回	正确返回 1，其他见错误码

2、释放资源

函数	int ASG_Release();
功能	释放资源、关闭 ARP 搜索等，在程序退出前调用此接口
参数	[sdk_version] SDK 版本号，此内存需要调用者申请。
返回	正确返回 1，其他见错误码

3、根据错误码获取错误信息

函数	const char *ASG_GetErrorInfo(int code);
功能	根据 code 错误码，获取对应错误信息
参数	[code] 错误码。

返回	错误信息

4、获取设备列表

函数	int ASG_GetDevicesList(ASGDevInfo *devices_info_list,int devNums)
功能	获取设备列表

参数	struct ASGDevInfo
	{
	char asgDev_name[10]{}; //设备名称
	char asgDev_id[15]{}; //设备编号
	char asgDev_hardV[8]{}; //硬件版本
	char asgDev_firmV[8]{}; //固件版本
	
	char asgDev_devIP[16]{}; //设备 IP
	char asgDev_devMAC[18]{}; //设备 Mac 地址
	char asgDev_localIP[16]{}; //本地网卡 IP
	char asgDev_localMAC[18]{}; //本地网卡 Mac 地址
	
	char asgDev_dsp[64]{}; //网卡描述
	int asgDev_connectType;//0:USB,1:TCP,2:UDP,3:SUDP
	};
	
	[devices_info_list] 参数回填指针,返回设备列表；
	[devNums] 设定一次可搜索最大设备数，默认为 5

返回	返回实际搜索到设备的数量；负数表示获取失败，见错误码
示例	ASGDevInfo *devInfos =

	(ASGDevInfo*)malloc(sizeof(ASGDevInfo)*devNums); int ret = ASG_GetDevicesList(devInfos, 5);

5、设备连接状态变化回调

函数	QMCResult ASG_DeviceListChangeCB(ASGDeviceListChangeCB callback);
功能	所连接的设备异常断电或断网回调，返回设备名称
参数	[callback] typedef bool(*ASGDeviceListChangeCB)(const char *)
返回	返回 QMCResult 类型，1 成功，其它失败

6、连接设备

函数	QMCResult ASG_ConnectDevice(const char *str_device_name，const char *lcoal_ip, const char *local_mac)
功能	连接单台设备
参数	[str_device_name] 设备名，"ASG24100"+设备编号； [local_ip] 本机 ip 地址； [local_mac] 本机 MAC 地址；
返回	正确返回 QMC_OK，其他见错误码

函数	QMCResult ASG_ConnectDevices(const char **str_device_names，const char *lcoal_ip, const char *local_mac,int devNums)
功能	同时（一次）连接多台设备
参数	[str_device_names] 设备名数组，"ASG24100"+设备编号...； [local_ip] 本机 ip 地址； [local_mac] 本机 MAC 地址； [devNums] 一次传入设备数量

返回	正确返回 QMC_OK，其他见错误码

7、关闭设备

函数	QMCResult ASG_DisConnectDevice(const char *str_device_name)
功能	断开连接（单个）
参数	[str_device_name] 设备的名，"ASG24100"+设备编号
返回	正确返回 QMC_OK，其他见错误码
示例	QMCResult result = ASG_DisConnectDevice(str_device_name);
函数	QMCResult ASG_DisConnectDevices(const char **str_device_names, int devNum)
功能	断开连接(同时断开多台设备接口)
参数	[str_device_name] 设备的名，"ASG24100"+设备编号
返回	正确返回 QMC_OK，其他见错误码
示例	QMCResult result = ASG_DisConnectDevice(str_device_name);

8、获取设备所有参数

函数	int ASG_GetAllParameters(const char *str_device_name, ASGParameter *asg_parameters);
功能	获取设备参数信息

参数	struct ASGParameter
	{
	char asgCmd[40]{};
	unsigned int value{};
	};
	[str_device_name] 设备的名，"ASG24100"+设备编号;
	[asg_parameters] 参数（结构体）回填指针;返回参数信息。

返回	实际参数总个数

示例	//pNums 为预设置参数个数，目前实际 160 个左右，可设置 200
	ASGParameter *asg_parameters = (ASGParameter*)malloc(sizeof(ASGParameter)*pNums);
	int ret = ASG_GetAllParameters(str_device_name, asg_parameters);

说明	此处细延迟值是实际值乘以 1000 后的结果，实际值需要除以 1000

9、加载参数

函数	QMCResult ASG_LoadAllParameters(const char *str_device_name, ASGParameter *asg_parameters ,int Nums)
功能	外部加载参数至设备（一）

参数	struct ASGParameter
	{
	char asgCmd[40]{};
	unsigned int value{};
	};
	
	[str_device_name] 设备的名，"ASG24100"+设备编号;
	[asg_parameters] 参数（结构体）指针。
	[Nums] 所加载参数的个数

返回	1 成功，其它失败
说明	细延迟需要乘以 1000 后取整

10、参数设置通用接口

- 设置参数（整数）

函数	QMCResult ASG_SetParamInt(const char *str_device_name, const char

	*str_cmd, const long long value);
功能	设置参数
参数	[str_device_name]设备名称 "ASG24100"+设备编号 [str_cmd] 指令 [value] 值
返回	正确返回 QMC_OK，其他见错误码

示例	播放停止功能下发(其他类似)：
	QMCResult result = ASG_SetParamInt("ASG2410022001","/Device/PlaybackState",1);//播放
	QMCResult result = ASG_SetParamInt("ASG2410022001","/Device/PlaybackState",0);//停止

- 设置参数（浮点数）

函数	QMCResult ASG_SetParamFloat(const char* str_device_name, const char* str_cmd, const double value)
功能	设置参数
参数	[str_device_name]设备名称 "ASG24100"+设备编号 [str_cmd] 指令 [value] 值
返回	正确返回 QMC_OK，其他见错误码

- 获取参数（整数）

函数	QMCResult ASG_GetParamInt(const char *str_device_name, const char *str_cmd, long long *value);
功能	获取参数
参数	[str_device_name]设备名称 "ASG24100"+设备编号

	[str_cmd] 指令 [value] 数据回填指针
返回	正确返回 QMC_OK，其他见错误码
示例	long long value; QMCResult result = ASG_GetParamInt("ASG2410022001", "/loop", &value);

- 获取参数（浮点数）

函数	QMCResult ASG_GetParamFloat(const char* str_device_name, const char* str_cmd, double *value)
功能	获取浮点型参数数值，目前只用于细延迟
参数	str_device_id:设备编号, str_cmd：指令，value：值
返回	正确返回 QMC_OK，其他见错误码

- 获取通道灯状态

函数	ASG_GetChannelLight(const char *str_device_name,int channel_lightMode[24]);
功能	主动获取 24 个通道对应状态灯数据
参数	[str_device_id] 设备编号。 [channel_lightMode[24]]整型数组，返回 24 个通道对应状态，0 暗，1 亮，(示例：[1,0,0,0,0,0,0,0])
返回	正确返回 1，其他见错误码

示例	int lights[24]{};
	QMCResult result = ASG_GetChannelLight("ASG2410022001", lights);
	for (int i = 0; i < 24; i++)
	printf("C:%d v:%d\n", i + 1, lights[i]);

函数	QMCResult ASG_ChannelLightCB(const char *str_device_name,channelLightStatesCB p_callback);
功能	通道状态灯回调(一)，单台设备

参数	typedef struct {
	char str_device_name[40];
	int channel_light_states[24];
	}ChannelLightStates;//通道状态信息结构体
	[str_device_id] 设备编号。
	[p_callback]typedef bool(*channelLightStatesCB)(ChannelLightStates *);

返回	正确返回 1，其他见错误码

函数	QMCResult ASG_ChannelLightCB_ALL(channelLightStatesCB p_callback);
功能	通道状态灯回调（二），多台设备连接时使用

参数	typedef struct {
	char str_device_name[40];
	int channel_light_states[24];
	}ChannelLightStates;//通道状态信息结构体
	[p_callback]typedef bool(*channelLightStatesCB)(ChannelLightStates *);

返回	正确返回 1，其他见错误码

- 获取子卡状态

函数	ASG_GetChannelLight(const char *str_device_name,int channel_lightMode[24]);
功能	主动获取 24 个通道对应状态灯数据
参数	[str_device_id] 设备编号。 [channel_lightMode[24]]整型数组，返回 24 个通道对应状态，0 暗，1 亮，(示

	例：[1,0,0,0,0,0,0,0])
返回	正确返回 1，其他见错误码

示例	int lights[24]{};
	QMCResult result = ASG_GetChannelLight("ASG2410022001", lights);
	for (int i = 0; i < 24; i++)
	printf("C:%d v:%d\n", i + 1, lights[i]);

函数	QMCResult ASG_ChannelLightCB(const char *str_device_name,channelLightStatesCB p_callback);
功能	通道状态灯回调(一)，单台设备

参数	typedef struct {
	char str_device_name[40];
	int channel_light_states[24];
	}ChannelLightStates;//通道状态信息结构体
	[str_device_id] 设备编号。
	[p_callback]typedef bool(*channelLightStatesCB)(ChannelLightStates *);

返回	正确返回 1，其他见错误码

函数	QMCResult ASG_ChannelLightCB_ALL(channelLightStatesCB p_callback);
功能	通道状态灯回调（二），多台设备连接时使用

参数	typedef struct {
	char str_device_name[40];
	int channel_light_states[24];
	}ChannelLightStates;//通道状态信息结构体
	[p_callback]typedef bool(*channelLightStatesCB)(ChannelLightStates *);

返回	正确返回 1，其他见错误码

- 获取子卡状态

函数	QMCResult ASG_GetChildCardStates(const char *str_device_name , int childCard_states[6]);
功能	获取子卡接入状态
参数	[str_device_name] 设备名称，"ASG24100"+设备编号。 [childCard_states] 子卡状态数组，6 个值分别对应 6 个子卡状态
返回	正确返回 1，其他见错误码

示例	int childCards[6]{};
	QMCResult result = ASG_GetChildCardStates(str_device_name, childCards);
	for (int i = 0; i < 6; i++)
	printf("C:%d v:%d\n", i + 1, childCards[i]);

函数	QMCResult ASG_ChildCardStatesCB(const char *str_device_name, childCardStatesCB p_callback);
功能	子卡接入状态回调（单设备）
参数	typedef struct { char str_device_name[40]; int child_card_states[6]; }SChildCardStates; [str_device_name] 设备名称，"ASG24100"+设备编号。 [callback]typedef bool(*childCardStatesCB)(SChildCardStates*);
返回	正确返回 1，其他见错误码

函数	QMCResult ASG_ChildCardStatesCB_ALL(childCardStatesCB p_callback);
功能	子卡接入状态回调（多设备，已连接设备批量注册）
参数	typedef struct { char str_device_name[40]; int child_card_states[6]; }SChildCardStates; [callback]typedef bool(*childCardStatesCB)(SChildCardStates*);
返回	正确返回 1，其他见错误码

- 获取 Counter 数据

函数	QMCResult ASG_GetCounterValue(const char *str_device_name ,const int counter_id ,unsigned int counter_value[4000],int &counter_size);
功能	主动获取 Counter 数据
参数	[str_device_name] 设备名称，"ASG24100"+设备编号。 [counter_id]Counter 编号（1：IN1,2:IN2）。 [counter_value]返回对应 counter_size 个数据。 [counter_size]返回的 counter 数据的个数
返回	正确返回 1，其他见错误码

示例	unsigned int counterV[4000]{};
	const int counterId = 1;//1、2、3、4
	int counterSize{};
	QMCResult result = ASG_GetCounterValue(devInfos[0].asgDev_id,counterId,counterV,counterSize);
	for(int i=0;i<counterSize;i++)
	{

	printf("counterSize:%d,counterNum:%d,counterValue:%d",counterSize,i,counterV[i]);
	}

函数	QMCResult ASG_CounterValueCB(const char *str_device_name,counterValueCB p_callback)
功能	Counter 数据回调
参数	typedef struct { char str_device_name[40]; int counter_id; unsigned int counter_value[4000]; int counter_size; }CounterValueStruct; [str_device_name] 设备名称，"ASG24100"+设备编号。 [p_callback] typedef bool(*counterValueCB)(CounterValueStruct*);
返回	正确返回 1，其他见错误码

函数	QMCResult ASG_CounterValueCB_ALL(counterValueCB p_callback)
功能	Counter 数据回调（批量回调注册，注册已连接所有设备）
参数	typedef struct { char str_device_name[40]; int counter_id; unsigned int counter_value[4000]; int counter_size;

	}CounterValueStruct; [p_callback] typedef bool(*counterValueCB)(CounterValueStruct*);
返回	正确返回 1，其他见错误码

- 固件升级

函数	QMCResult ASG_DealUpdateFirmware(const char *str_device_name,const char *filePath);
功能	进行固件升级
参数	[str_device_name] 设备名称，"ASG24100"+设备编号。 [filePath] 固件升级文件路径
返回	正确返回 1，其他见错误码

函数	QMCResult ASG_DealUpdateFirmwareCB(const char *str_device_name, updateFirmwareProcessCB p_callback);
功能	固件升级进度回调（单个）

参数	typedef struct {
	char str_device_name[40];//设备名
	int curr_progress;//当前进度
	}UpdateFirmwareProcessS;
	[str_device_name] 设备名称，"ASG24100"+设备编号。
	[p_callback] typedef bool(*updateFirmwareProcessCB)(UpdateFirmwareProcessS*);

返回	正确返回 1，其他见错误码

函数	QMCResult ASG_DealUpdateFirmwareCB_ALL(updateFirmwareProcessCB

	p_callback);
功能	固件升级进度回调（批量）

参数	typedef struct {
	char str_device_name[40];//设备名
	int curr_progress;//当前进度
	}UpdateFirmwareProcessS;
	[p_callback] typedef bool(*updateFirmwareProcessCB)(UpdateFirmwareProcessS*);

返回	正确返回 1，其他见错误码

- 下载波形代码

函数	QMCResult ASG_DownloadWaveformCode(const char *str_device_name,const char *waveform_code);
功能	下载自由方波下波形代码
参数	[str_device_name] 设备名称。 [waveform_code] 波形代码。
返回	正确返回 1，其他见错误码
示例	const char *code = "w1 = Seq_Gen(H,100,L,4,Loop = 10)\ns1 = ASG_SEQ([w1(10)])\nASG_OUT[1] = s1\n"; QMCResult result = ASG_DownloadWaveformCode(devInfos[0].asgDev_id,code); printf("result of download:%d",result);

11、参数设置

(1) 常规参数

- 设置编译模式

指令	ASG_SetParamInt(devlist[0]["asgDev_id"],"/Waveform/CompileMode",0)	
参数 说明	连续方波模式	0
	自由方波模式	1

- 设置板卡 1 输出电平

指令	ASG_SetParamInt("ASG2410022001","/Device/ChildCard1/OutputLevel", 0);
参数说明	0:3.3V 1:5V

- 设置匹配阻抗

指令	ASG_SetParamInt("ASG2410022001","/Device/ChildCard1/Impedance", 0);
参数说明	0:50Ω 1:1MΩ

- 时钟设置

指令	ASG_SetParamInt(devlist[0]["asgDev_id"],"/Device/IN1/ClockIn",0)
参数 说明	0：内部 1：外部 10M 2：外部 100M 3：外部可变时钟

- 触发模式

指令	ASG_SetParamInt(devlist[0]["asgDev_id"],"/Device/TriggerSwitch",0)

参数 说明	0：外部模式 1：内部模式

- 触发沿设置

指令	ASG_SetParamInt("ASG2410022001","/Device/TriggerEdge", 0);
参数说明	0:上升沿触发 1:下降沿触发

- 触发等待时间设置

指令	ASG_SetParamInt("ASG2410022001","/Device/TriggerWaitTime", 0);
参数说明	范围 0~1s，单位 ns；（0~1,000,000,000ns）

- 播放状态调整

指令	ASG_SetParamInt(devlist[0]["asgDev_id"],"/Device/PlaybackState",1)
参数 说明	0 停止，1 播放

- 播放次数（自由方波下）

指令	ASG_SetParamInt(devlist[0]["asgDev_id"],"/loop",20)
参数 说明	0 表示一直循环播放

- 播放模式（自由方波下）

指令	ASG_SetParamInt(devlist[0]["asgDev_id"],"/Waveform/PlayMode",0)

参数 说明	0 连续方波模式，1 自由方波模式

(2) 通道参数

- 通道使能开关

说明：24 个通道使用 C1,C2,C3...区分，此处以通道 1 为例

指令	ASG_SetParamInt(devlist[0]["asgDev_id"],"/Device/C1/Output",1)	
参数 说明	关	0
	开	1

- 通道周期设置

说明：24 个通道使用 C1,C2,C3...区分，此处以通道 1 为例

注意：通道周期应大于高电平宽度，否则被强制转换为相同

指令	ASG_SetParamInt(devlist[0]["asgDev_id"],"/Device/C1/Cycle",480)
参数 说明	通道 1 周期设置， “外部可变时钟”时，单位为倍数，其余情况为“ns”

- 通道高电平宽度设置

说明：24 个通道使用 C1,C2,C3...区分，此处以通道 1 为例

注意：通道周期应大于高电平宽度，否则被强制转换为相同

指令	ASG_SetParamInt(devlist[0]["asgDev_id"],"/Device/C1/HighlevelDuration",200)

参数 说明	通道 1 高电平设置， “外部可变时钟”时，单位为倍数，其余情况为“ns”.

- 通道粗延时设置

说明：24 个通道使用 C1,C2,C3...区分，此处以通道 1 为例

注意：通道延时下发必须先下发粗延时后下发细延时，且两个都要下发

指令	ASG_SetParamInt(devlist[0]["asgDev_id"],"/Device/C1/Delay",10)
参数 说明	“外部可变时钟”时，单位为倍数，其余情况为“ns”.

- 通道细延时设置

说明：24 个通道使用 C1,C2,C3...区分，此处以通道 1 为例

注意：通道延时下发必须先下发粗延时后下发细延时，且两个都要下发

指令	ASG_SetParamFloat(devlist[0]["asgDev_id"],"/Device/C1/SlightDelay",0.25)
参数 说明	延时设置-细延时设置，单位 ns

- 通道常高电平设置

说明：24 个通道使用 C1,C2,C3...区分，此处以通道 1 为例

指令	ASG_SetParamInt(devlist[0]["asgDev_id"],"/Device/C1/AlwaysHighlevel", 0)
说明	通道 1 常高电平设置，设置后通道 1 将恒定输出高电平,优先级最高

6.2 Python SDK 使用说明

ASG24100 Python版本SDK同样是基于C++ 版本SDK的封装。如“PythonSDK”目录所示。包含“asglib”文件夹、“asgparser”文件夹及“test.py”文件。

- “asglib”下是封装的接口文件及 C++版本动态库文件；
- “asgparser”下是波形解析模块（自由方波模式下使用）
- “test.py”为调用示例，包含“单步测试”、“自由方波模式测试”及“连续方波模式测试”。具体内容可参见内部代码。

调用时，需按照“PythonSDK”文件夹下文件位置配置，保持代码文件“XXX.py”与另两个文件夹处于相同目录下。

调用示例

```
# ************************************
#   Method:       ASG_Init 进 SDK 初始化
#   Returns(dict): result 执行结果，成功为 1，其他失败；version 返回 SDK 版本号
#   Mark:         执行 SDK 其它功能前必须先进行此操作
# ************************************
def ASG_Init()-> dict:
    ...
    return {"result": result, "version": version}

# ************************************
#   Method:      ASG_Release SDK 资源释放
#   Returns(int): result 执行结果，成功为 1，其他失败
#   Mark:         关闭软件前进行此操作
# ************************************
def ASG_Release()->int:
```

```
    ...
    return result


# ************************************
#   Method:      ASG_GetErrorInfo 根据错误代码获取具体错误信息
#   Parameter:   code 错误代码
#   Returns:     错误信息
#   Mark:        执行指令若出现错误返回，可根据此接口获取错误信息
# ************************************
def ASG_GetErrorInfo(code) -> str:
    ...
    return err_str


# ************************************
#   Method:      ASG_GetDevicesList 获取设备列表
#   Returns      :result 执行结果，成功为 1，其他失败；count 设备数量；value 设备列表
#   Parameter:   nums 限定一次可搜索到最大设备数量，默认 5
# ************************************
def ASG_GetDevicesList(nums = 5) -> dict:
    ...
    return {"result": 1, "count": count, "value": all_devices}


# ************************************
#   Method:      ASG_ConnectDevice 连接设备
#   Returns:     result 执行结果，成功为 1，其他失败
#   Parameter: str_device_name "ASG24100"+由 ASG_GetDevicesList 函数获取的设备编号
```

local_ip 上位机 IP 及 local_mac 上位机 MAC 地址均由 ASG_GetDevicesList 函数获取

```
#    Mark:          连接设备后，SDK 将主动同步设备数据
# **********************************
def ASG_ConnectDevice(str_device_name, local_ip, local_mac) -> int:
    ...
    return result

# **********************************
#    Method:       ASG_DisConnectDevice  断开连接
#    Returns:      result  执行结果，成功为 1，其他失败
#    Parameter: str_device_name "ASG24100"+由 ASG_GetDevicesList 函数获取的设备编号
# **********************************
def ASG_DisConnectDevice(str_device_name) -> int:
    ...
    return result

# **********************************
#    Method:       ASG_ResetDevice  重置设备
#    Returns:      result  执行结果，成功为 1，其他失败
#    Parameter: str_device_name "ASG24100"+由 ASG_GetDevicesList 函数获取的设备编号
# **********************************
def ASG_ResetDevice(str_device_name) -> int:
    ...
    return result

# **********************************
```

```
#    Method:       ASG_GetAllParameters 获取设备所有参数

#    Returns:     result 执行结果，成功为 1，其他失败；count 实际需同步寄存器数量；all_parameters 所有参数信息，指令+数值

#    Parameter: str_device_name "ASG24100"+由ASG_GetDevicesList函数获取的设备编号

# **********************************
def ASG_GetAllParameters(str_device_name) -> dict:
    ...
    return {"result": 1, "count": count, "value": all_parameters}


# **********************************

#    Method:       ASG_SetParamInt 设置参数（整型）

#    Returns:     result 执行结果，成功为 1，其他失败

#    Parameter: str_device_name "ASG24100"+由ASG_GetDevicesList函数获取的设备编号

#    Parameter: str_cmd 对应的功能指令

#    Parameter: value 待设置的数值

# **********************************
def ASG_SetParamInt(str_device_name, str_cmd, value) -> int:
    ...
    return result


# **********************************

#    Method:       ASG_SetParamFloat 设置参数（浮点型）

#    Returns:     result 执行结果，成功为 1，其他失败

#    Parameter: str_device_name "ASG24100"+由ASG_GetDevicesList函数获取的设备编号

#    Parameter: str_cmd 对应的功能指令
```

```
#   Parameter: value 待设置的数值
# **********************************
def ASG_SetParamFloat(str_device_name, str_cmd, value) -> int:
    ...
    return result


# **********************************
# Method:      ASG_GetParamInt 获取参数（整型）
# Returns:     result 执行结果，成功为 1，其他失败；value 读取到的数据
# Parameter: str_device_name "ASG24100"+由 ASG_GetDevicesList 函数获取的设备编号
# Parameter: str_cmd 对应的功能指令
# **********************************
def ASG_GetParamInt(str_device_name, str_cmd) -> dict:
    ...
    return {"result": result, "value": value.value}


# **********************************
# Method:      ASG_GetParamFloat 获取参数（浮点型）
# Returns:     result 执行结果，成功为 1，其他失败；value 读取到的数据
# Parameter: str_device_name "ASG24100"+由 ASG_GetDevicesList 函数获取的设备编号
# Parameter: str_cmd 对应的功能指令
# **********************************
def ASG_GetParamFloat(str_device_name, str_cmd) -> dict:
    ...
    return {"result": result, "value": value.value}
```

```
# ************************************
# Method:       ASG_GetChannelLight 获取通道状态灯数据
# Returns:     result 执行结果，成功为 1，其他失败；
#                  value 对应 24 个通道的灯状态 ,0 暗 ,1 亮 ,(示例 :[1,0,0,0,0,0,0,0,...])
# Parameter: str_device_name "ASG24100"+由 ASG_GetDevicesList 函数获取的设备编号
# ************************************
def ASG_GetChannelLight(str_device_name) -> dict:
    ...
    return {"result": result, "value": result_value}

# ************************************
# Method:       ASG_GetChildCardStates 获取通道状态灯数据
# Returns:     result 执行结果，成功为 1，其他失败；
#                  value 对应 6 个子卡连接状态 ,0 未连接 ,1 已连接 (示例 :[1,0,0,0,0,0])
# Parameter: str_device_name "ASG24100"+由 ASG_GetDevicesList 函数获取的设备编号
# ************************************
def ASG_GetChildCardStates(str_device_name) -> dict:
    ...
    return {"result": result, "value": result_value}

# ************************************
# Method:       ASG_GetCounterValue 获取外部输入对应的 Counter 数据
# Returns:     result 执行结果 ,成功为 1 ,其他失败 ;size 实际返回 counter 个数 ;value 对应 size 个数的 counter 值
# Parameter: str_device_name "ASG24100"+由 ASG_GetDevicesList 函数获取的设备编号
```

```
# Parameter: int_counter_id，1-4
# ************************************
def ASG_GetCounterValue(str_device_name, int_counter_id) -> dict:
    If...
        return {"result": result, "size": length.value, "value": result_value}
    else:
         return {"result": result}

# ************************************
# Method:       ASG_GetCounterValue_ALL 同时获取 4 个 Counter 数据
# Returns:     result 执行结果，成功为 1，其他失败；value 对应 4 个 Counter 输出通道数据 size 个数的 counter 值
# Parameter: str_device_name "ASG24100"+由 ASG_GetDevicesList 函数获取的设备编号
# ************************************
def ASG_GetCounterValue_ALL(str_device_name) -> dict:
    ...
    return {"result": 1, "value": all_counterV}

# ************************************
# Method:       ASG_DealUpdateFirmware 固件升级
# Returns:     result 执行结果（只表示该指令下发成功），成功为 1，其他失败；
# Parameter: str_device_name "ASG24100"+由 ASG_GetDevicesList 函数获取的设备编号
#                   str_file_path 固件绝对路径
# ************************************
def ASG_DealUpdateFirmware(str_device_name, str_file_path) -> int:
    ...
    return ret
```

```
# ************************************
# Method:       ASG_GetUpdateFirwareProgress 获取固件升级进度
# Returns:     result 执行结果(只表示该指令下发成功),成功为 1,其他失败； value 进度值
# Parameter: str_device_name "ASG24100"+由 ASG_GetDevicesList 函数获取的设备编号
# ************************************
def ASG_GetUpdateFirwareProgress(str_device_name) -> dict:
    return {"result": result, "value": value.value}

# ************************************
# Method:       ASG_DownloadWaveformCode 下载波形代码
# Returns:     result 执行结果(只表示该指令下发成功)，成功为 1，其他失败；
# Parameter: str_device_name "ASG24100"+由 ASG_GetDevicesList 函数获取的设备编号
#                  waveform_code 波形代码
# ************************************
def ASG_DownloadWaveformCode(str_device_id, waveform_code) -> int:
    ...
    return ret
```

6.3 LabVIEW 接口说明

ASG24100 LabVIEW SDK 实际上是对 C++ SDK 实现了进一步封装 ,将 C++各个接口封装成独立的.vi 文件，vi 文件名称及描图如下表所示。

接口名称	接口描述
ASG_ConnectDevice.vi	连接设备接口，需传入设备名称（由搜索设备接口获得）；
ASG_DealUpdateFirmware.vi	处理固件升级接口，需传入待升级的设备

	名称，固件路径；
ASG_DisConnectDevice.vi	断开连接，传入待断开连接设备名；
ASG_DownloadWaveformCode.vi	下载自由方波波形代码，传入设备名称、波形代码；
ASG_GetChannelLight.vi	获取通道灯状态（通道输出状态），传入设备名称；
ASG_GetChildCardStates.vi	获取子卡接入状态，传入设备名称；
ASG_GetCounterValue.vi	获取 Counter 数据，出入设备名称；
ASG_GetDeviceList.vi	获取设备列表，默认搜索最大设备数为 5；
ASG_GetErrorInfo.vi	获取错误信息，传入错误代码；
ASG_GetParamFloat.vi	获取浮点数参数值，传入设备名、参数指令（指令参照下文“指令参考”）；
ASG_GetParamInt.vi	获取整型参数值，传入设备名、参数指令（指令参照下文“指令参考”）；
ASG_GetUpdateFirwareProgress.vi	获取固件升级进度值，传入设备名；
ASG_Init.vi	SDK 初始化；
ASG_Release.vi	资源释放，用于软件关闭前；
ASG_ResetDevice.vi	重置设备参数，传入设备名称；
ASG_SetParamFloat.vi	设置浮点型参数，目前仅用于细延时下发，传入设备名及浮点数值；
ASG_SetParamInt.vi	设置整型参数，通常参数设置均从此接口下发，传入设备名及整型数值；
Example.vi	各功能模块调用示例；
global.vi	全局变量，包含 dll 绝对路径，本地 ip 及本地 mac（目前已做处理，ip 及 mac 会自动

	使用系统返回值，可不传入）
splitString.vi	分割字符串，传入字符串，返回分割后字符串数组。

各功能的具体使用请参照文件夹中“Example.vi”。

7. 维护保养

7.1 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；

7.2 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。

7.3 校准和标定

仪器需每年至少进行一次标定。使用高分辨率的示波器测试输出幅值、延迟精度、通道输出抖动等指标参数，检测设备产生和测量能力是否满足要求。如果仪器指标标定或使用中发现超差需进行返厂校准。