DP712 定时输出波形

用户手册

内部文件：[输入文件版本号]

颁布时间：[2023.7.19]

用户手册（DP712 指定波形输出）

1 引言

1.1 编写目的

本手册使用 visual studio 2010 进行开发编程，通过程序设置指定电压波形的参数，将预先设置的参数输出到程序控制电源 DP712 中，采用编程的方式，可以减少人为输入的时间，提高工作的效率。

1.2 背景

说明：

a. 开发环境 visual studio 2010，C#窗体应用程序
b. 本手册适用于对 DP700 系列电源有编程需求的开发人员

1.3 定义

DP712：普源生产的单通道可编程性直流电流

1.4 参考资料

列出有用的参考资料，如：

a. DP700_UserGuide_CN_tcm4-3991
b. DP700_ProgrammingGuide_CN_tcm4-3994
c. 启动稳定性测试电源波形 V_1.4

2 用途

2.1 功能

本手册用于输出《启动稳定性测试电源波形 V_1.4》中的六种波形，本文提供的方法可根据用户的需求，编写自己的波形程序，实现指定波形的输出。

2.2 性能

2.2.1 精度

本手册设计的参数，有电压（Voltage），电流（Current），持续时间，其中电压的最低值为 10mv，电流最小值为 0.01A，时间最小值 10ms。

2.2.2 时间特性

定时器的总输出数=输出组数*循环数；

输出组数最大值为 2048；

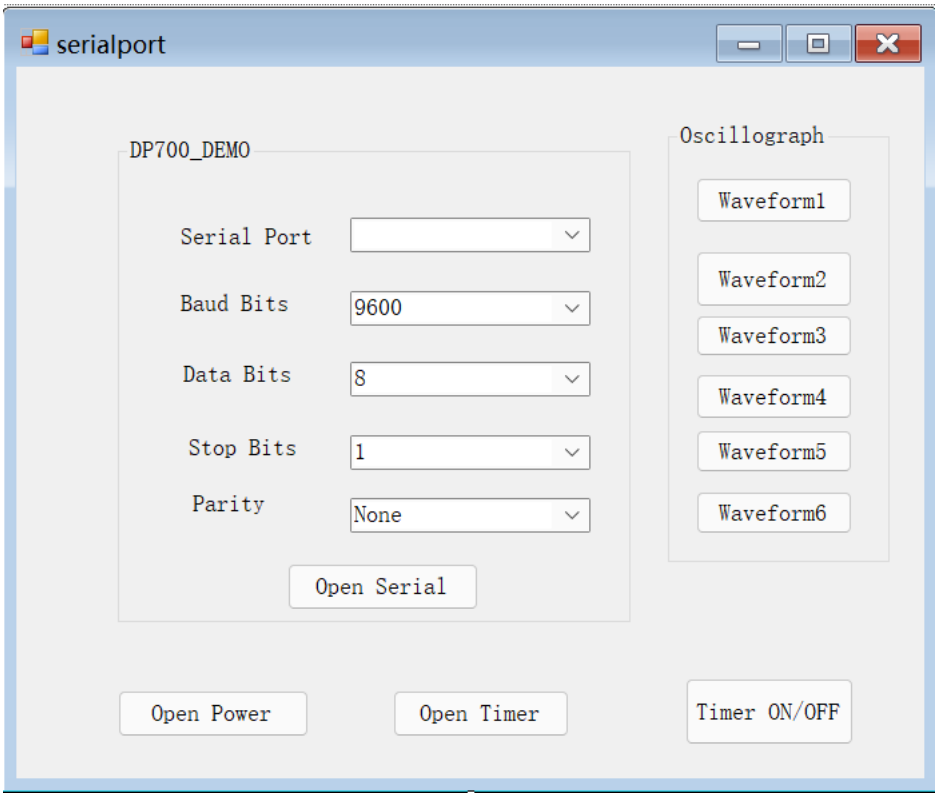
输出数太大时，在向 DP700 系列设定定时器的参数时，时间相对较久，此时只需要耐心等待

2.2.3 灵活性

说明本软件所具有的灵活性，即当用户需求（如对操作方式、运行环境、结果精度、时间特性等的要求）有某些变化时，本软件的适应能力。

3 窗体界面介绍

本示例所创建的窗体界面如下图所示，可以看出整个界面被分为了三个块，DP700_DEMO 块，Oscillograph 块，以及最下方的开关按钮。每个版块的详细信息，将在 3.1,3.2,3.3 中详细介绍。



3.1 DP700_DEMO

DP700_DEMO 块，主要包含五个属性值和一个串口开关。每个属性值由前面的标签和后面的下拉框组成，这一部分是用来设置 RS232 接口的，更多信息可以查看 *DP700_UserGuide_CN_tcm4-3991*

Serial Port：端口号。端口号的获取，采用最基本的串口扫描方式，如果找到可用的串口，则将其添加到下拉框控件的集合中。

Baud Bits:波特率。波特率可选为 7200bps、9600bps、14400bps、19200bps、38400bps、57600bps、115200bps。如果需要某种波特率，可以手动将其添加到控件的集合中。本例中使用的波特率为 9600bps。

Data Bits：数据位。默认为 8，不允许修改。

Stop Bits：停止位。默认为 1，不允许修改。

Parity：校验位。可选的校验位为奇校验，偶校验和无。本例中选择的是 None。

Open Serial：这是一个 button 控件，当我们手动选择扫描的串口，并且其他属性都配置好的情况下，点击该控件，串口打开，此时可用于通信。

3.2 Oscillograph

本模块，主要是通过设置定时器参数，从而输出六种特定的波形。六种波形的详细信息可参考《启动稳定性测试电源波形 V_1.4》，本手册中仅放出参考图，波形的参考图见 4.3，或者查询《启动稳定性测试电源波形 V_1.4》。

注：由于定时器内部参数分辨率的限制，电压最小为 10mv，电流最小 10mA，时间最小值 10ms，本手册输出的任何线性变化电压，均为估计计算，并不代表实际结果。

3.3Button 块

本块主要设计了一系列按键，用于模拟开关。
Open Power：用于打开和关闭机器的通道输出。
Timer ON/OFF：打开和关闭定时器输出。

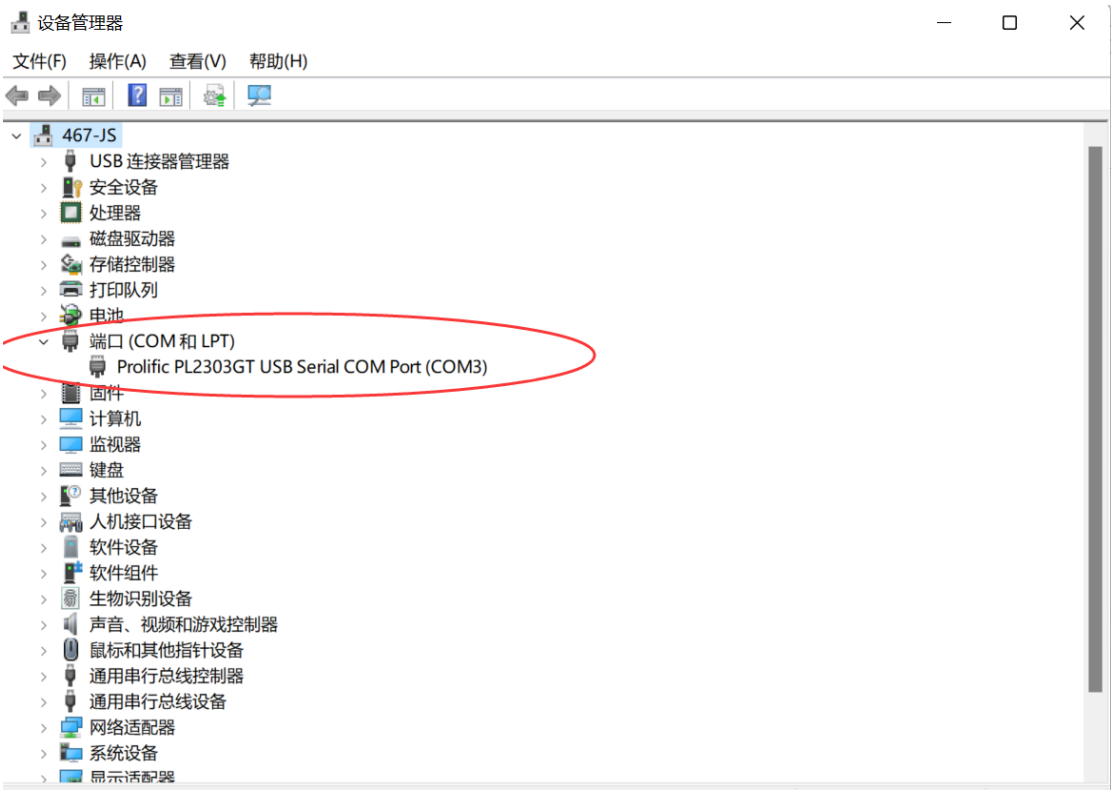
4 使用过程

在使用本实例时，务必保证已经阅读 DP700 系列的使用手册和编程手册。

4.1 安装与初始化

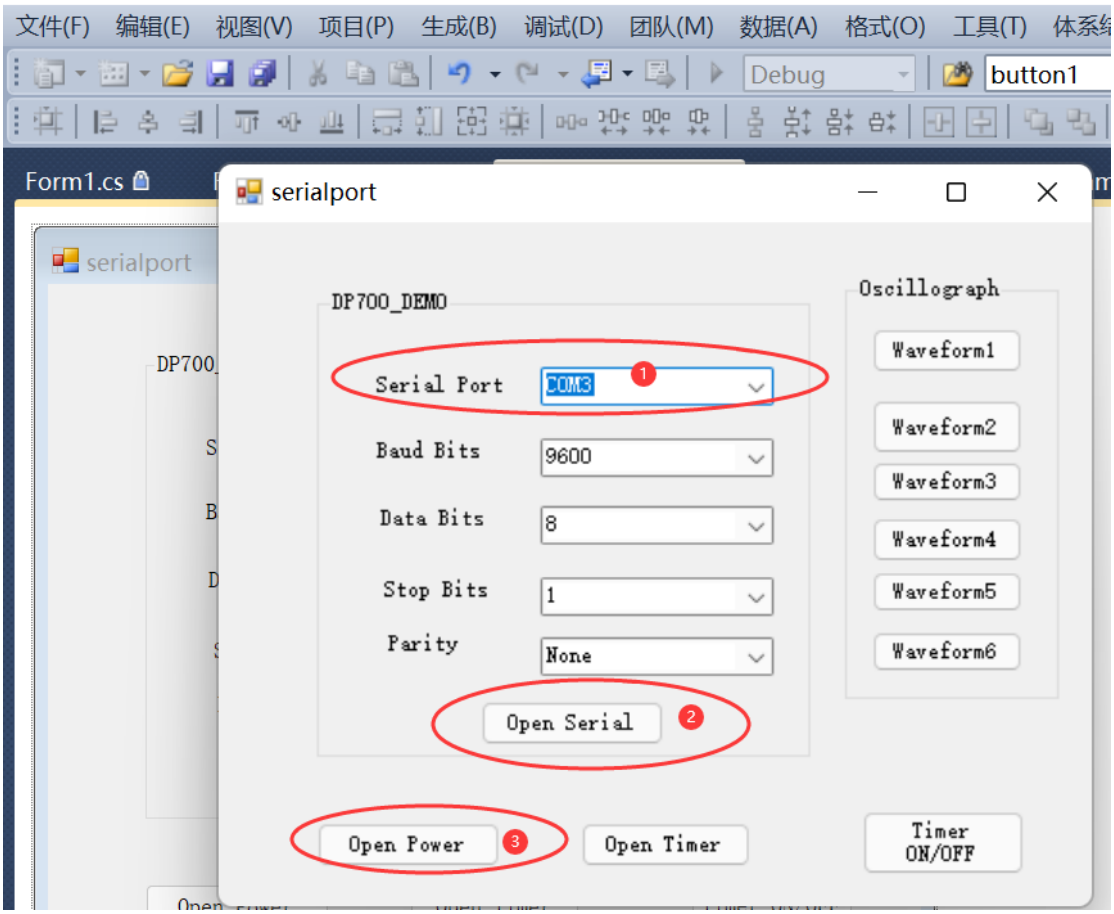
将程控电源通过 RS232 与上位机连接，打开控制面板-硬件和声音-设备管理器。

在打开的设备管理器页面中，查看端口的连接状况，如果如下图所示，则说明连接成功，如下图所示。如果连接不成功，例如出现 USB-serial controller 错误，则应该检查相应的驱动，随后在网上下载对应的驱动。



连接成功后，我们检查一下通信的情况，此处的通信检查可参考 DP700 系列使用手册第三章远程控制一节，此处不再赘述。

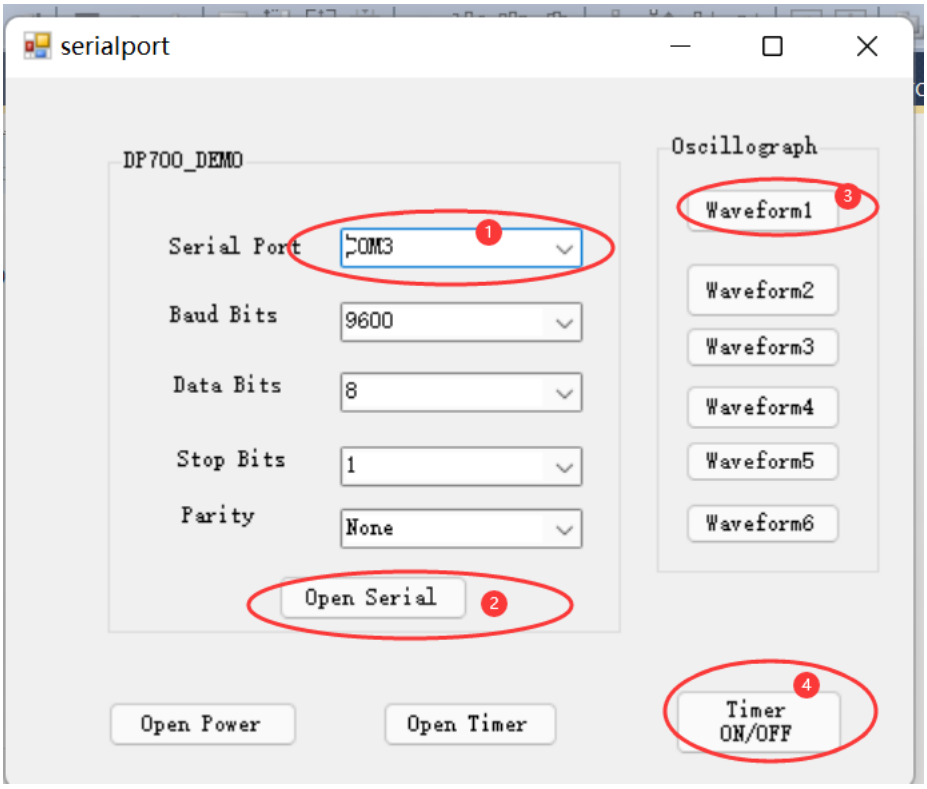
测试成功后，打开并运行程序，在窗口中选择串口，随后依次点击 Open Serial 和 Open Power，操作如下图所示。



操作完成后，我们看到了程控电源的通道输出被打开，同时 Open Power 变成了 Close Power，点击 Close Power，通道输出关闭。完成了本例的测试，说明已经可以进行下一步的定时器程序了。

4.2 输出定时器波形

右侧的 Oscillograph 模块中，有六种波形，我们先点击 Waveform1，随后点击 Timer ON/OFF，因为打开定时器输出相当于已经打开通道输出了，所以不需要点击 Open Power。打开波形一的流程如下图所示：

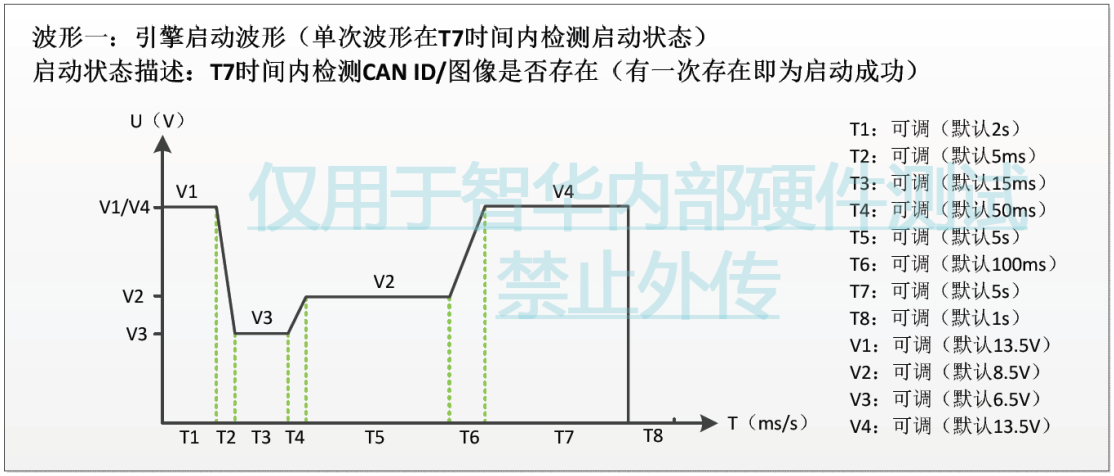


后面的几种波形的启动流程和上面的一致，所以不再赘述。

4.3 定时器输出说明

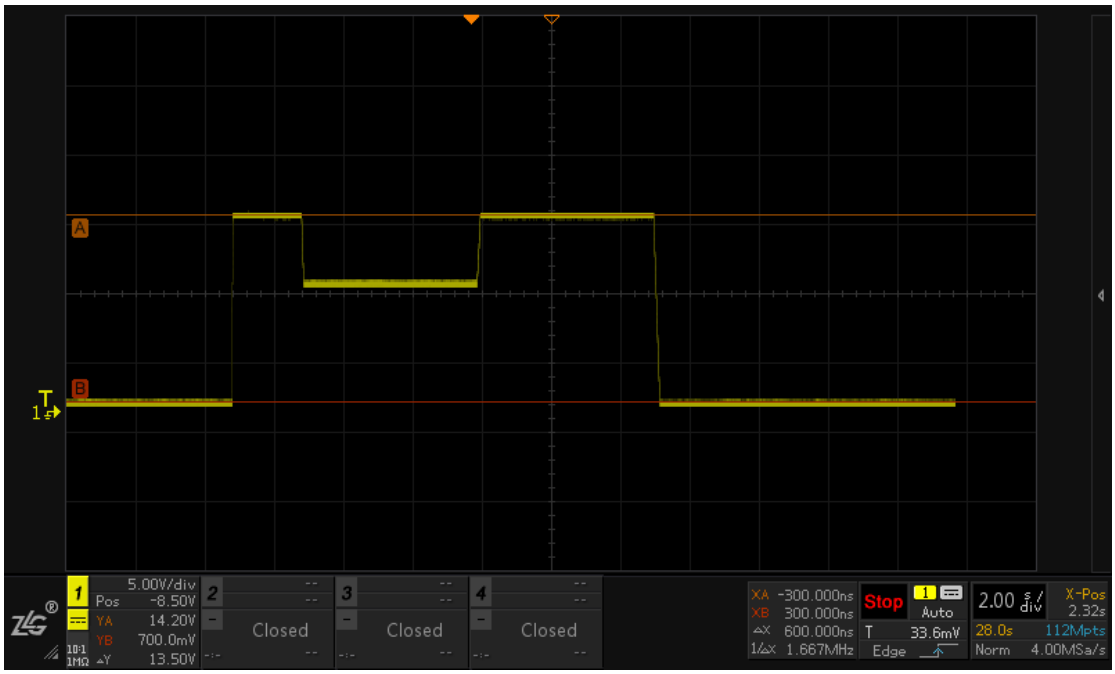
4.3.1 Waveform1

本输出说明是建立在阅读完 DP700 系列的编程使用手册的基础上，下面结合代码来说明实现的原理。对于波形一，如下图所示：



Waveform1 添加点击事件，命名为 Waveform1_Click。
结合波形一的曲线，我们编写了如下的函数，需要注意的是，曲线中没有提到电流设定值，因此电流值为一个经验值，要让仪器保持恒压输出，我们适当将电流值取得稍大，本例中电流值设定为 3。

使用示波器测得的电压实际波形如下图所示：



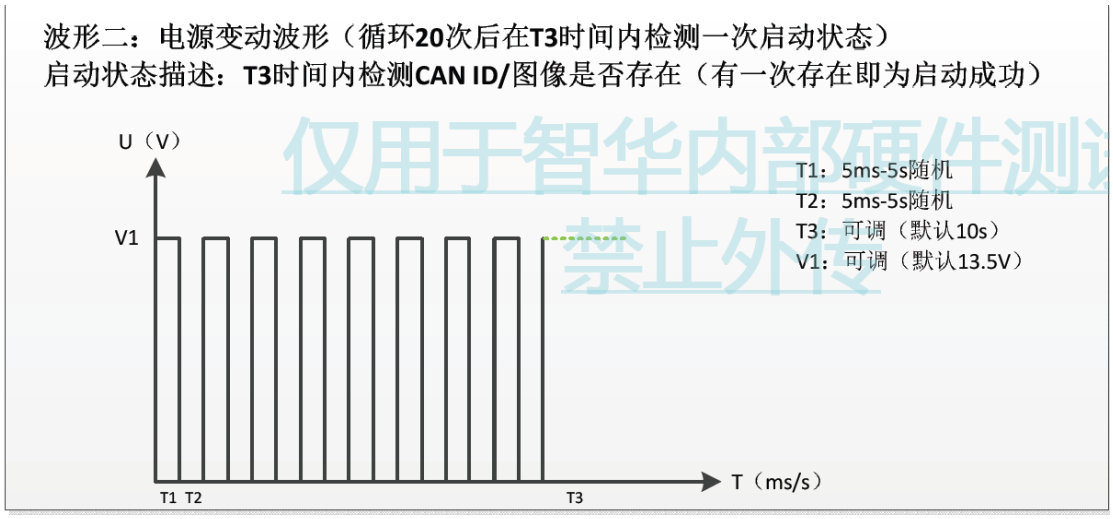
函数及解释如下：

```
private void Waveform1_Click(object sender, EventArgs e)
        {
            //定时器输出总组数=输出组数*循环数
            serialPort.Write(":TIMEr:STATe OFF\n");//关闭定时器
            serialPort.Write(":TIMEr:GROUPs 21\n");//设置定时器输出组数
            serialPort.Write(":TIMEr:CYCLEs N,1\n");//设置定时器循环数
            //设置定时器出发模式为自动（DEFault），意为按照定时器参数配置依次输出，直至完成总组数输出
            //另一种触发模式为 SINGle，意为定时器输出需要每次按 OK 键，不会按照输出组数依次往后输出
            serialPort.Write(":TIMEr:TRIGer DEFault\n");
            //设置定时器终止状态为最后一组（LAST），意为当输出完成后，仪器保持最后一组的输出状态
            //与之相对应的是关闭（OFF），意为当输出完成后，仪器关闭输出
            serialPort.Write(":TIMEr:ENDState LAST\n");

            serialPort.Write(":TIMEr:PARAmeter  1,13.5,3,2\n");// 设置第一组参数电压13.5V，电流 3A，持续事件 2s
            serialPort.Write(":TIMEr:PARAmeter  2,10,3,0.01\n");//线性变换的曲线无法输出，此处我们取值 v1 和 v3 的平均值，并让其输出 1s
            serialPort.Write(":TIMEr:PARAmeter 3,6.5,2,0.15\n");

/*在电压由 V3（6.5V）变到 V2（8.5V）的过程中，持续时间为 50ms，由于 DP700 系列自身的缺陷，设置定时器参数只能按照固有格式输出，不能使用变量，没有内置的线性波形方法，由于时间的最小值为 10ms，由于波形一的线性变化时间不久，所以我们按照 10ms 的最小时间变化，每过 10ms，电压上升 0.4V，五个 10ms 后，从 6.5V 变化到 8.5V，代码见下
```

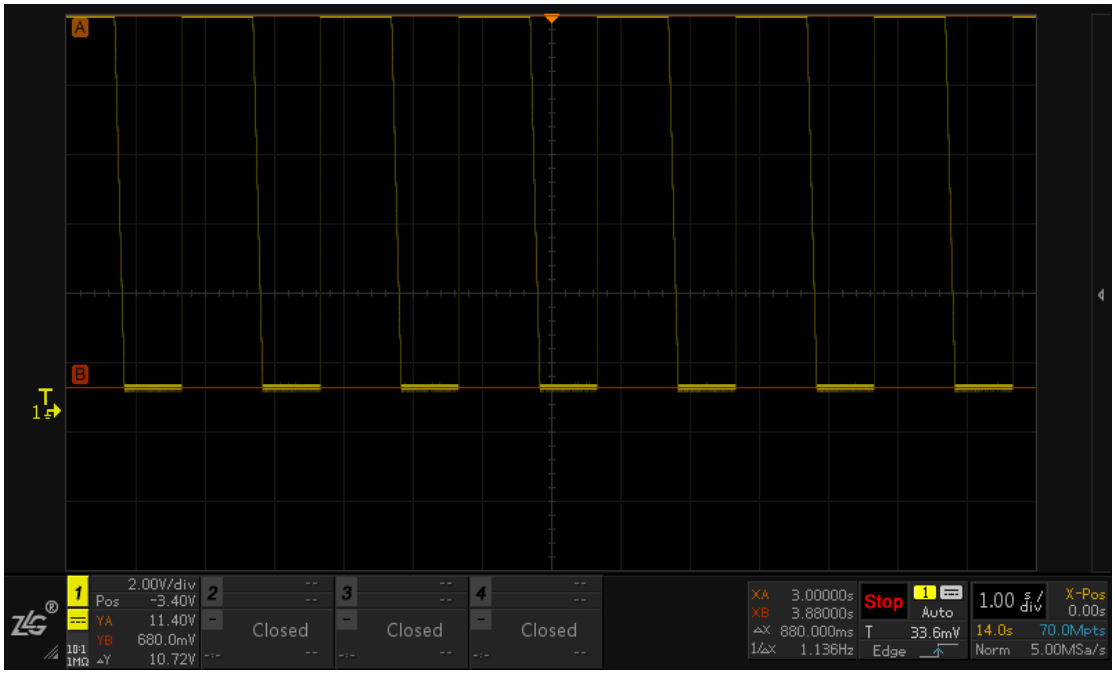
```
面的五行，定时器的第四组参数到第八组参数。*/
                serialPort.Write(":TIMEr:PARAmeter 4,6.9,2,0.01\n");
                serialPort.Write(":TIMEr:PARAmeter 5,7.3,2,0.01\n");
                serialPort.Write(":TIMEr:PARAmeter 6,7.7,2,0.01\n");
                serialPort.Write(":TIMEr:PARAmeter 7,8.1,2,0.01\n");
                serialPort.Write(":TIMEr:PARAmeter 8,8.5,2,0.01\n");

                serialPort.Write(":TIMEr:PARAmeter 9,8.5,2,5\n");

                serialPort.Write(":TIMEr:PARAmeter 10,9,2,0.01\n");
                serialPort.Write(":TIMEr:PARAmeter 11,9.5,2,0.01\n");
                serialPort.Write(":TIMEr:PARAmeter 12,10,2,0.01\n");
                serialPort.Write(":TIMEr:PARAmeter 13,10.5,2,0.01\n");
                serialPort.Write(":TIMEr:PARAmeter 14,11,0.01\n");
                serialPort.Write(":TIMEr:PARAmeter 15,11.5,2,0.01\n");
                serialPort.Write(":TIMEr:PARAmeter 16,12,2,0.01\n");
                serialPort.Write(":TIMEr:PARAmeter 17,12.5,2,0.01\n");
                serialPort.Write(":TIMEr:PARAmeter 18,13,2,0.01\n");
                serialPort.Write(":TIMEr:PARAmeter 19,13.5,2,0.01\n");

                serialPort.Write(":TIMEr:PARAmeter 20,13.5,2,5\n");
                serialPort.Write(":TIMEr:PARAmeter 21,0,2,1\n");
            }
```

4.3.2 Waveform2

波形二没有线性变化部分，只需要按照对应的电压电流时间传入参数即可。



使用示波器测得的电压实际波形如下图所示：

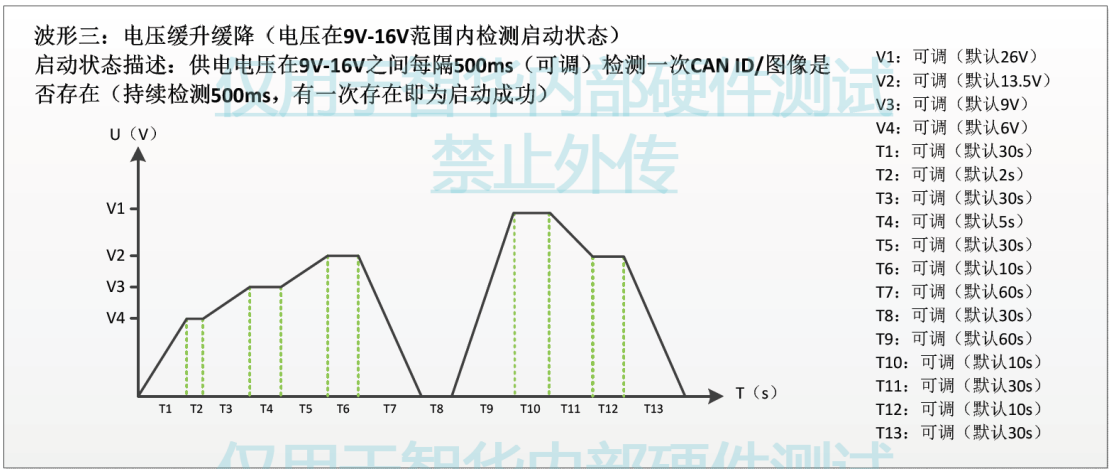



```
private void Waveform2_Click(object sender, EventArgs e)
        {
            serialPort.Write(":TIMEr:STATe OFF\n");
            serialPort.Write(":TIMEr:GROUPs 41\n");
            serialPort.Write(":TIMEr:CYCLEs N,2\n");
            serialPort.Write(":TIMEr:TRIGer DEFault\n");
            serialPort.Write(":TIMEr:ENDState LAST\n");
/* 根据波形二的波形图，循环输出 20 次，再将 13.5V 电压恒压输出 10s，此处不能靠循环组数来达到循环二十次 ，只能手动输入数值，就像下面的代码一样，从第一组输出到第 40 组，第 41 组再输出一个持续 10s 的 13.5V 的电压*/
            serialPort.Write(":TIMEr:PARAmeter 1,13.5,3,1\n");
            serialPort.Write(":TIMEr:PARAmeter 2,0,1,1\n");
            serialPort.Write(":TIMEr:PARAmeter 3,13.5,3,1\n");
            serialPort.Write(":TIMEr:PARAmeter 4,0,1,1\n");
            serialPort.Write(":TIMEr:PARAmeter 5,13.5,3,1\n");
            serialPort.Write(":TIMEr:PARAmeter 6,0,1,1\n");
            serialPort.Write(":TIMEr:PARAmeter 7,13.5,3,1\n");
            serialPort.Write(":TIMEr:PARAmeter 8,0,1,1\n");
            serialPort.Write(":TIMEr:PARAmeter 9,13.5,3,1\n");
            serialPort.Write(":TIMEr:PARAmeter 10,0,1,1\n");
            serialPort.Write(":TIMEr:PARAmeter 11,13.5,3,1\n");
            serialPort.Write(":TIMEr:PARAmeter 12,0,1,1\n");
            serialPort.Write(":TIMEr:PARAmeter 13,13.5,3,1\n");
            serialPort.Write(":TIMEr:PARAmeter 14,0,1,1\n");
            serialPort.Write(":TIMEr:PARAmeter 15,13.5,3,1\n");
            serialPort.Write(":TIMEr:PARAmeter 16,0,1,1\n");
            serialPort.Write(":TIMEr:PARAmeter 17,13.5,3,1\n");
            serialPort.Write(":TIMEr:PARAmeter 18,0,1,1\n");
```

```
                serialPort.Write(":TIMEr:PARAmeter 19,13.5,3,1\n");
                serialPort.Write(":TIMEr:PARAmeter 20,0,1,1\n");
                serialPort.Write(":TIMEr:PARAmeter 21,13.5,3,1\n");
                serialPort.Write(":TIMEr:PARAmeter 22,0,1,1\n");
                serialPort.Write(":TIMEr:PARAmeter 23,13.5,3,1\n");
                serialPort.Write(":TIMEr:PARAmeter 24,0,1,1\n");
                serialPort.Write(":TIMEr:PARAmeter 25,13.5,3,1\n");
                serialPort.Write(":TIMEr:PARAmeter 26,0,1,1\n");
                serialPort.Write(":TIMEr:PARAmeter 27,13.5,3,1\n");
                serialPort.Write(":TIMEr:PARAmeter 28,0,1,1\n");
                serialPort.Write(":TIMEr:PARAmeter 29,13.5,3,1\n");
                serialPort.Write(":TIMEr:PARAmeter 30,0,1,1\n");
                serialPort.Write(":TIMEr:PARAmeter 31,13.5,3,1\n");
                serialPort.Write(":TIMEr:PARAmeter 32,0,1,1\n");
                serialPort.Write(":TIMEr:PARAmeter 33,13.5,3,1\n");
                serialPort.Write(":TIMEr:PARAmeter 34,0,1,1\n");
                serialPort.Write(":TIMEr:PARAmeter 35,13.5,3,1\n");
                serialPort.Write(":TIMEr:PARAmeter 36,0,1,1\n");
                serialPort.Write(":TIMEr:PARAmeter 37,13.5,3,1\n");
                serialPort.Write(":TIMEr:PARAmeter 38,0,1,1\n");
                serialPort.Write(":TIMEr:PARAmeter 39,13.5,3,1\n");
                serialPort.Write(":TIMEr:PARAmeter 40,0,1,1\n");
                serialPort.Write(":TIMEr:PARAmeter 41,13.5,2,10\n");

            }
```

4.3.3 Waveform3

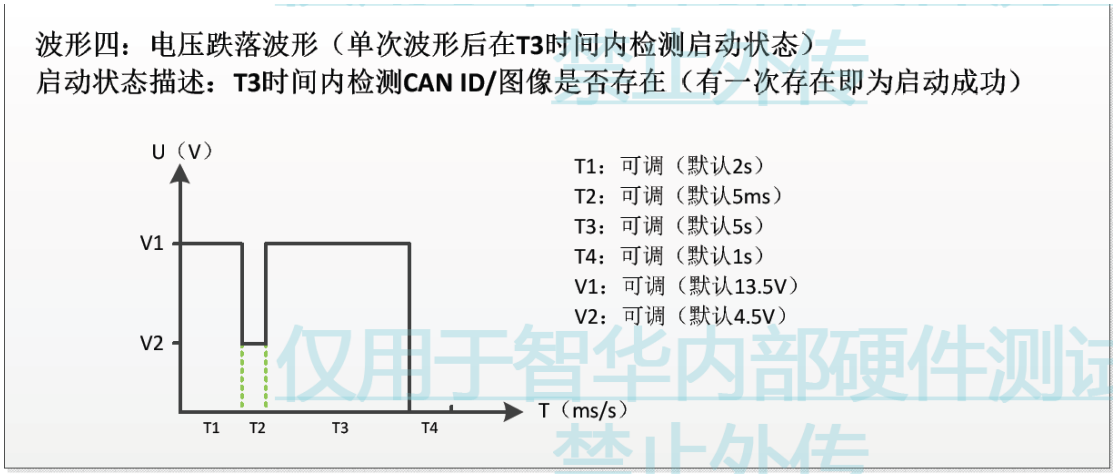
Waveform3 的波形是最复杂的，涉及了很多的线性上升和线性下降段。由前面的波形二的运行过程中，我们可以发现，按照顺序执行波形二的电压变化时，系统有一些迟缓，这是由于写入了 41 组的数据，这个延迟时间会随着写入数据的增多而变得尤其明显。按照第一个输出波形的方法，可以取最小的时间间隔 10ms，手动输入，以达到线性变化的效果。输出组数的最大限制为 2048，观察波形三的 T1，其持续时间为 30s，如果按照 10ms 的间隔，需要手动输入 3000 条代码，这已经超过了 2048，并且代码过长，传入的数据过大，也会造成系统卡顿。所以在本例中，我们取时间步长为 0.5s。由于前面已经有演示代码，此处不在贴出代码，可查看工程文件 DP700_DEMO_C。



使用示波器测得的电压实际波形如下图所示：

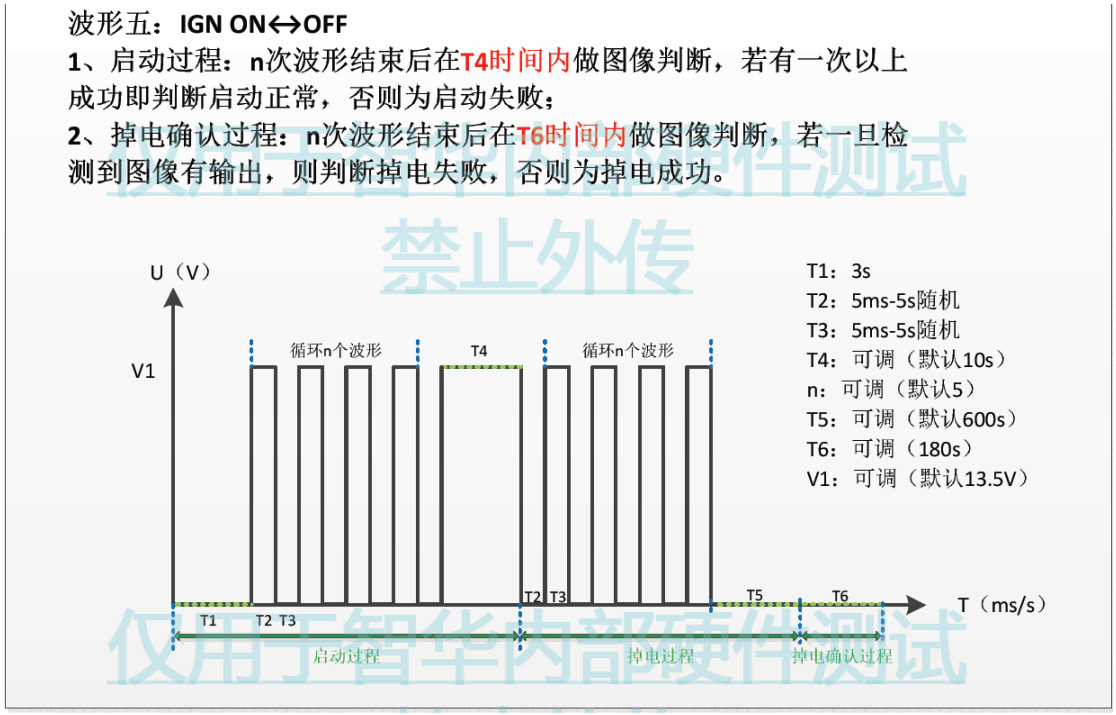
4.3.4 Waveform4

波形四的输出参考前面三个波形。

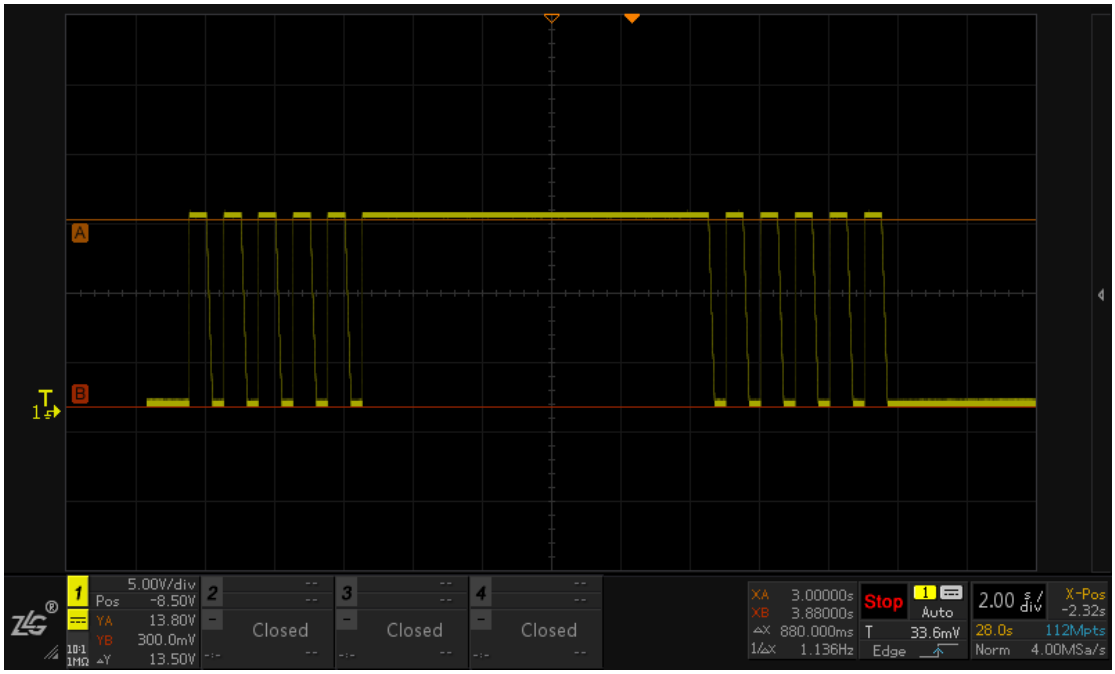


4.3.5 Waveform5

波形五也可参考前面三个波形。

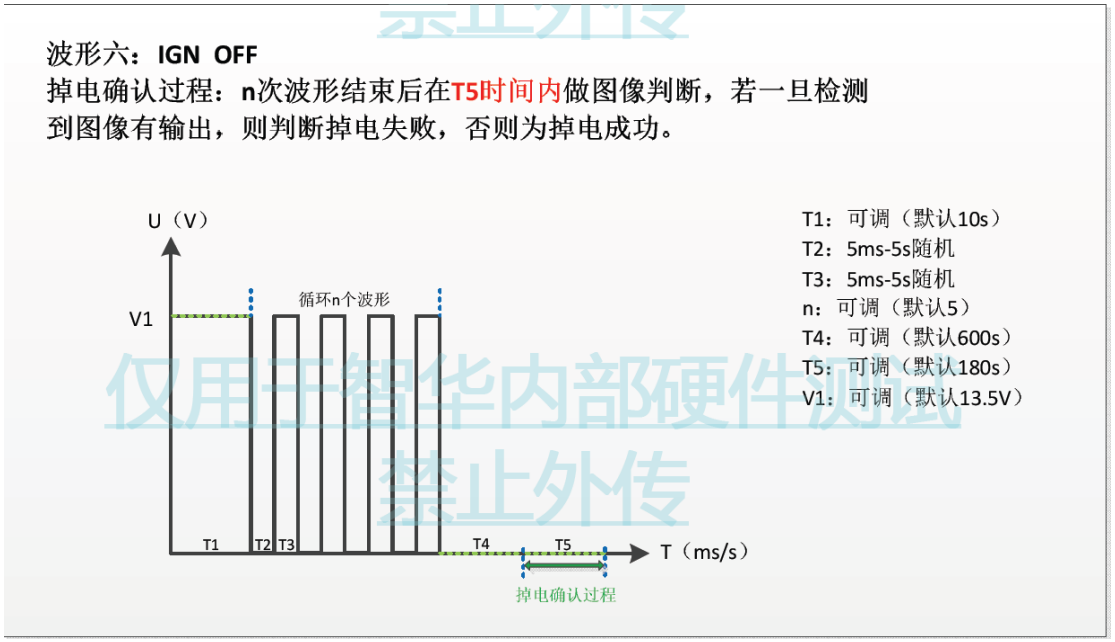


使用示波器测得的电压实际波形如下图所示：

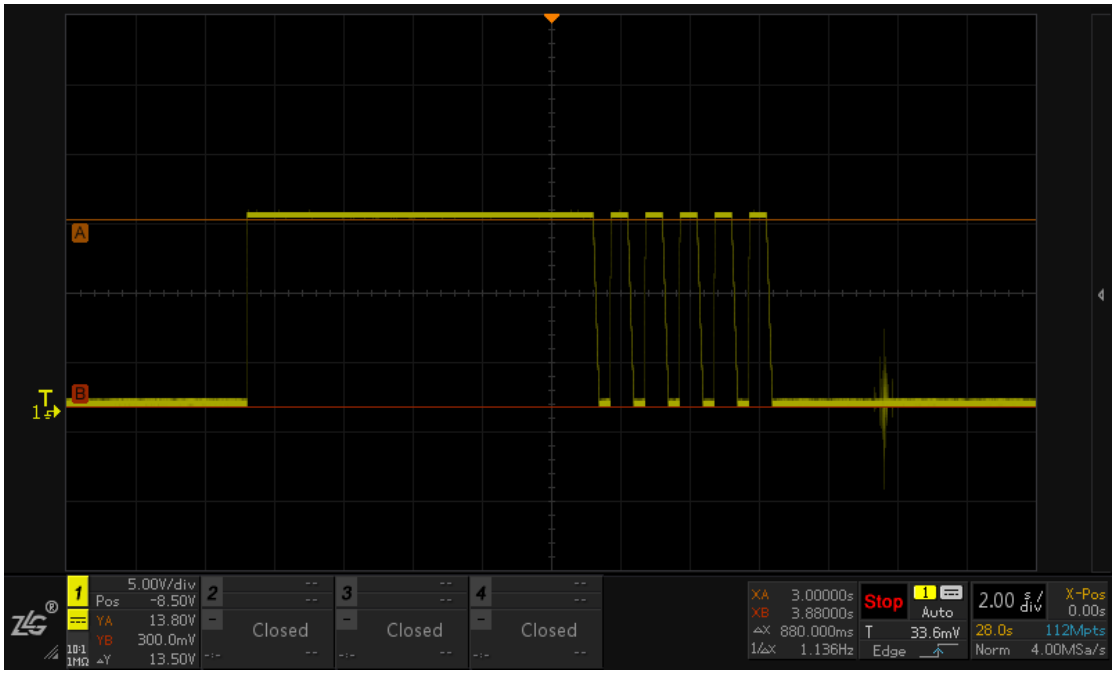


4.3.6 Waveform6

详细信息参考前面三个波形。



使用示波器测得的电压实际波形如下图所示：



4.4 出错处理

在向定时器输出参数时，仪器的主界面报错。如下图所示：

出现这种情况，大部分情况并不影响定时器的正常，只要正常点击 Timer ON/OFF。因为在调试模式下，单步执行并不会有上面的错误，所以这并不影响最后的结果。

出现上述情况的原因主要是由于仪器的写入的操作太快，仪器的算力跟不上，当查看本手册的工程文档时，在第四个波形中我们添加了延时命令，每个命令之间延迟 100ms，这时可完美解决问题。

在运行 Waveform3 的过程中，系统会出现卡顿，因为波形三的写入数据比较大，仪器的处理能力跟不上，所以会卡顿。当出现卡顿时，耐心等待，等我鼠标指示变为手型时再点击 Timer ON/OFF 键。