PowerWriter用户手册

本节内容详细介绍PowerWriter 应用软件的详细使用方法，可以作为参考手册随时来查阅其中的功能，PowerWriter 应用软件的启动界面如下图所示。

标题栏

PowerWriter 标题栏包含:

- PowerWriter 版本号。
- PowerWriter 编译日期时间信息。
- PowerWriter 最小化，最大化，退出按钮。

PowerWriter 菜单

先了解快捷键

PowerWriter 客户端软件为所有的菜单均提供了快捷键，以便基于快捷键快速对功能进行操作，但是仍请注意：

> **⚠ 注意**
>
> 非全局快键键：
>
> PowerWriter 软件的快捷键为非全局的快捷键，这样可以尽可能的降低和其他软件冲突，在使用PowerWriter 快捷键时确保将PowerWriter 窗口为激活状态 (非强制置顶)，**如果快键键失效，可以使用鼠标拉动一下窗口来激活窗口，这样就可以使用快捷键了**

文件菜单

文件菜单包含保存项目、项目另存为、加载项目、退出四个常用功能，如下图所示：

保存项目

当完成项目设置之后，可将所有配置、数据打包成一个加密的项目文件，首次点击文件菜单将会弹出首次保存的路径和设置密码：

- **密码设置**：设置项目文件的密码，输入密码 (可留空)。
- **显示明文**：默认为不显示密码明文，如果需要查看明文密码，用鼠标点击右侧图标。
- **文件路径**：点击选择路径，在弹出的对话框中选择保存的路径，并设置保存文件名称，然后确认。

保存项目会在状态栏显示项目文件的路径,同时日志栏将显示保存的结果，如下(时间搓可能不同)：

02/19-11:00:14:858> 保存成功

> **提示**
>
> 1：创芯工坊加密文档格式同时具有高强度加密和良好的解压性能。
>
> 2：文档内部不保存任何密码的信息，采用自验证机制，如果用户忘记了密码，文件将无法解包，请您牢记密码。
>
> 3：密码最长为16位，但是是可选的，如果没有填写密码，则会用默认密码进行填充，实际的文件仍然是加密的。
>
> 4：第二次保存将不会再弹出保存文件的对话框，而是使用上一次选择密码和路径信息进行覆盖。

> **注意**
>
> 密码为可选，但是仍然推介使用自定义密码来保存项目。

项目另存为

项目另存为的定义和保存项目的定义**区别在于**：另存为每次都会弹出新的保存项目信息，而不是使用上次设置的路径和密码。

加载项目

通过加载项目功能，可以加载项目文件到PowerWriter 软件中，操作方式和保存项目一致：

- 填写项目的密码。
- 设置加载项目的路径。

退出

退出PowerWriter 软件

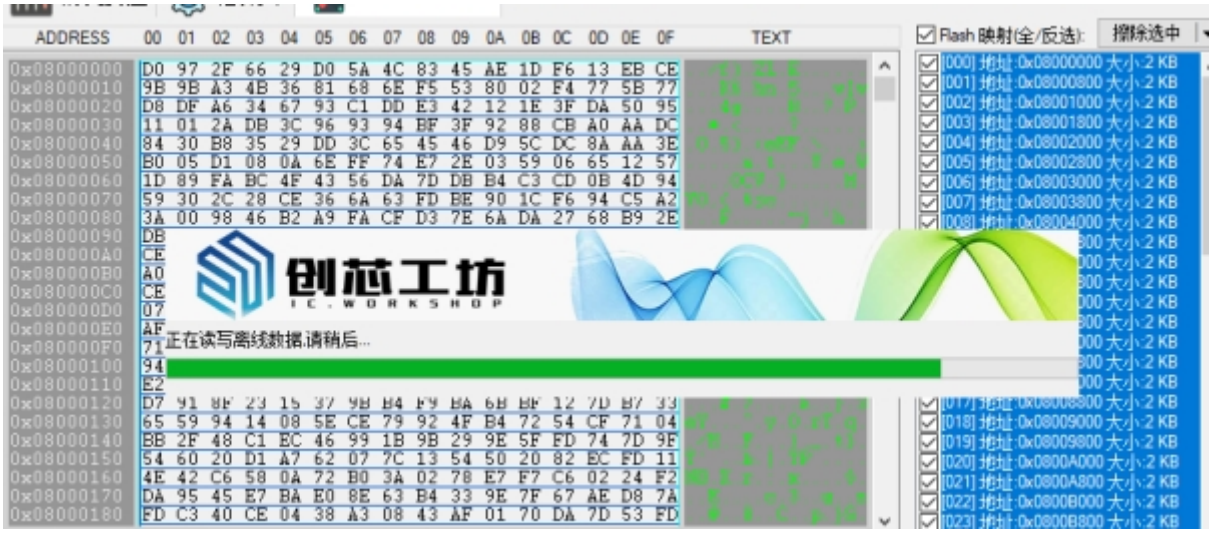
> **提示**
>
> 软件在执行耗时任务过程中无法退出。

执行菜单

执行菜单包含芯片在线操作和离线操作的常用功能

保存并离线加载

当项目的所有设置完成之后，可以将项目配置到PowerWriter 硬件，用于实际的产品生产，此功能会同时执行 **保存项目** 并加载项目到PowerWriter硬件,加载过程如图所示：




```
02/22-09:55:45:110> 保存成功
02/22-09:55:46:075> 加载离线数据成功
```

离线读取并保存

如果需要将PowerWriter中已存储的项目读回，可以执行离线读取并保存，此动作将会将项目从PowerWriter读回，并尝试加载为当前项目，需要填写项目密码（如果有）。

```
02/22-09:56:08:233> 加载离线数据成功
02/22-09:56:08:242> 芯片型号没有改变.不会更新设置...
02/22-09:56:08:334> 更新烧录器设置完成...
02/22-09:56:08:784> 更新芯片信息成功...
02/22-09:56:08:815> 加载成功
02/22-09:56:09:694> 目标芯片已连接...
02/22-09:56:09:715> 选项字节已经成功读取!
```

> **注意**
>
> 如果密码错误，将无法读取加载项目。

读取Program Memory

通过读取Program Memory 功能读取芯片的Program Memory 数据到PowerWriter 软件，设置图所示，读取完成后，数据将会在Program Memory 选项卡显示:

- 读取地址：可以自由设置读取地址，默认设置为芯片Program Memory的首地址。
- 读取大小：可以设置为1KB、2KB、4KB、8KB、16KB、32KB、64KB、128KB、256KB、512KB、1MB、2MB、4MB等。
- 整片读取，如果需要读取整片的Program Memory 数据，直接勾选整片读取，默认不勾选。

```
02/22-09:56:55:115> Program Memory 读取 成功!
```

注意

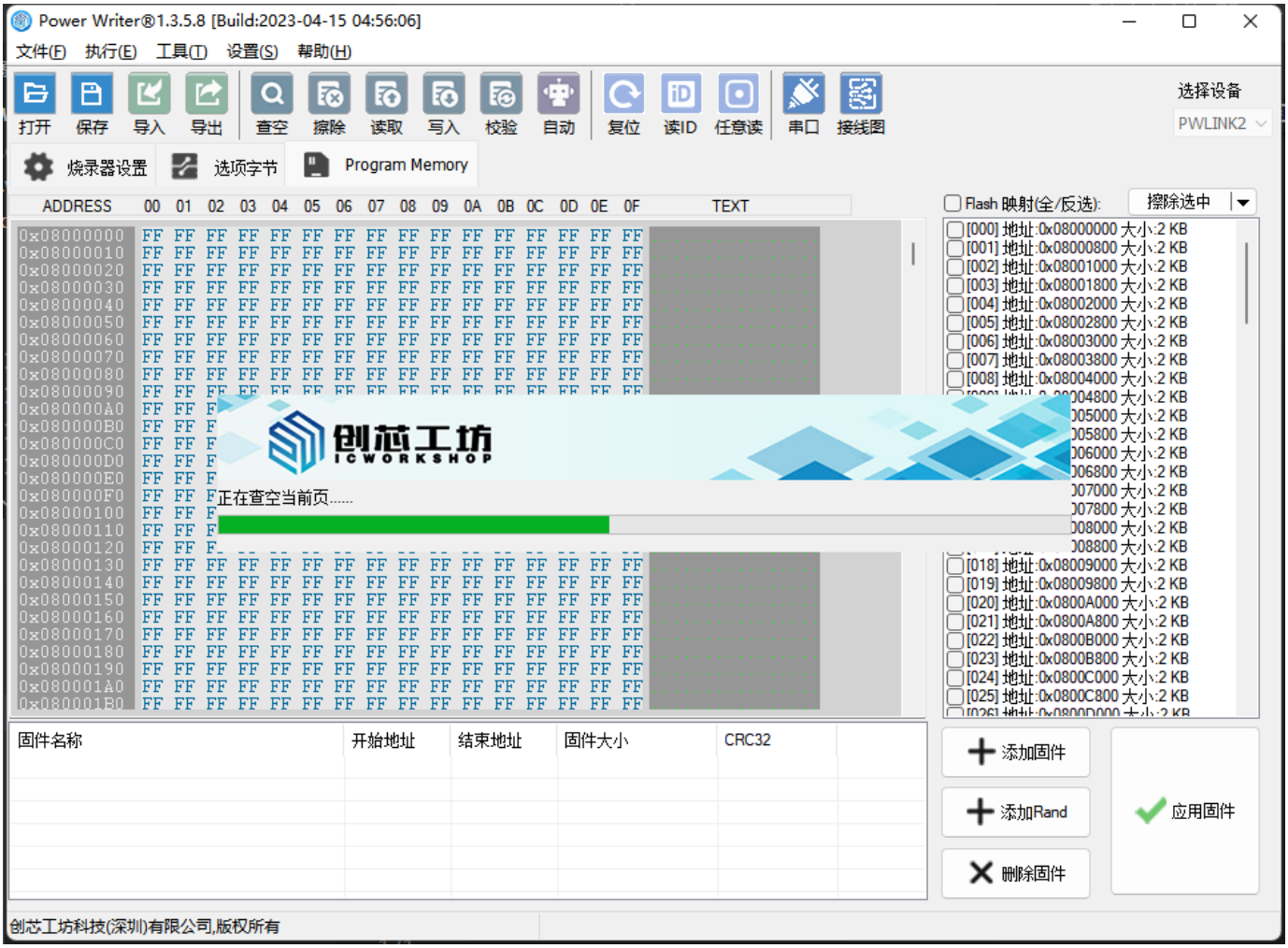
1：主流芯片有RDP，在未解锁的情况下，无法读取数据。

2：如果设置参数超过了芯片的容量，则会提示读取失败，后段数据无法读取，一般直接勾选整片读取即可。

3：极少芯片有读保护的情况下，也能读取数据，但是数据是不可预知的，需要根据芯片实际情况进行判断，大部分芯片PowerWriter都会主动判断有没有RDP。

查空Program Memory

通过此功能，可以快速检查芯片是否为空。

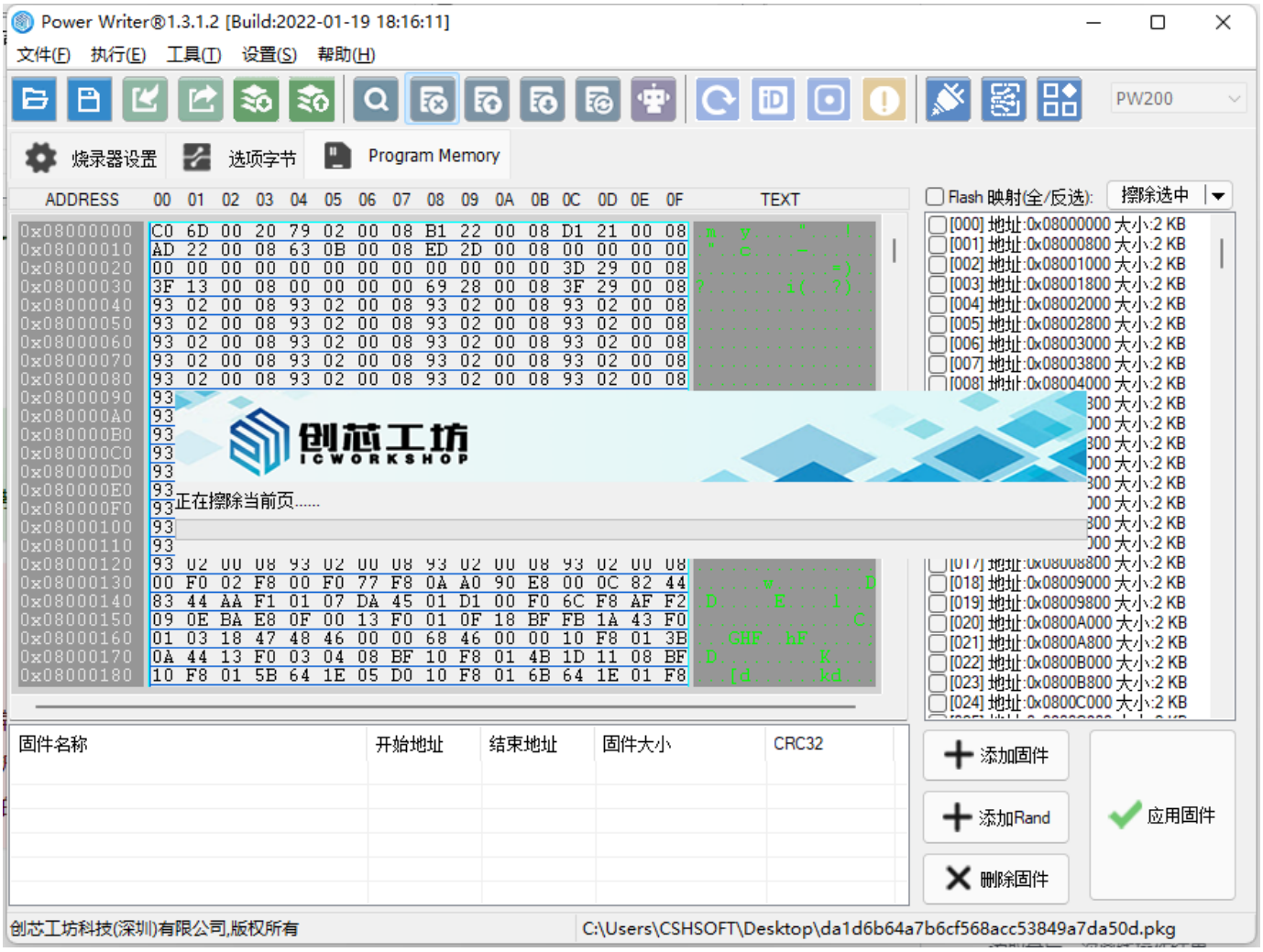



```
02/22-09:57:26:279> Program Memory 查空 成功!
```

擦除Program Memory

通过Program Memory 功能可以快速擦除芯片。

```
02/22-09:59:04:718> Program Memory 擦除 成功!
02/22-09:59:04:718> 如需要查看擦除后的芯片内存，请继续读取当前页。
```



提示

此功能执行的是片擦除（Chip Erase），如需扇区擦除，请参考对应扇区分页位置的扇区擦除功能。

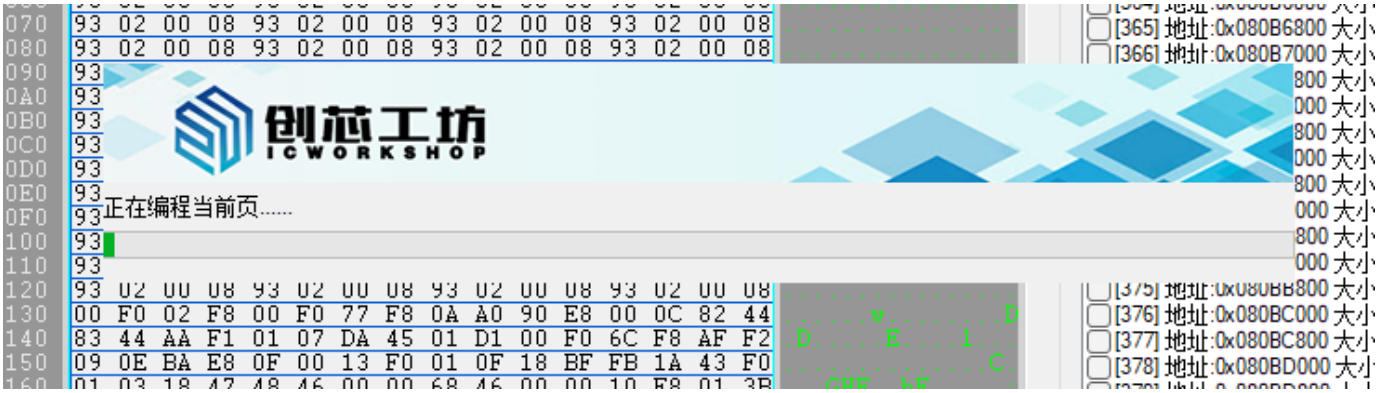
注意

如果擦除失败，**请按下面步骤进行排查**:

- 检查PowerWriter的状态指示灯状态是否常亮，如果常亮，说明芯片处于连接状态，否则请先检查连接线路。
- 执行读取选项字节，检查选项字节中的保护设置是否为0级，否则，则需修改成0级，然后写入以去除RDP
- 检查扇区写保护WRP，如果写保护是开启的，改为无保护，然后写入选项字节以去除WRP。

编程Program Memory

用于PowerWriter 在线对目标芯片进行烧写，烧写时，有以下两点特性需要留意：




```
02/22-10:02:08:565> Program Memory 编程 成功!
```

提示

当有添加固件时：执行编程Program Memory 操作将写入添加的所有 Program Memory 固件文件，支持多固件。

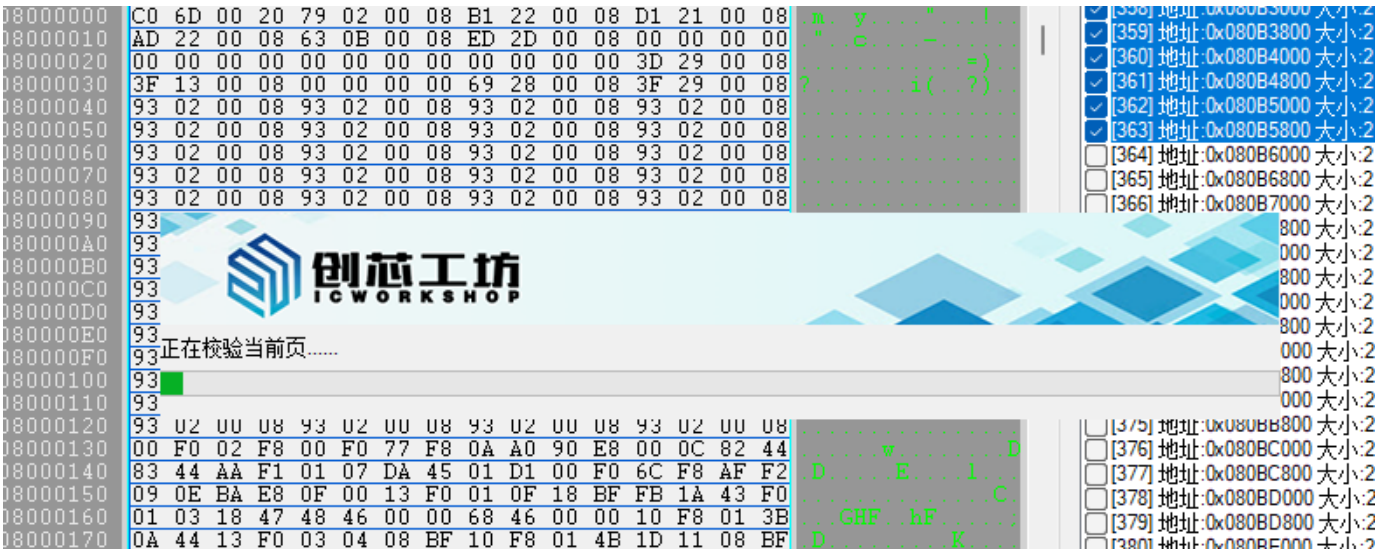
当无固件文件时：执行编程Program Memory 操作将写入的是Program Memory 缓冲区的所有数据。

写入失败原因汇总

- 常规问题：如芯片未连接，或者未正确连接线缆，请参考#查看芯片的接线图
- 芯片未擦除，处理方法为先擦除芯片，或者擦除指定的扇区，参考扇区擦除方法。
- 芯片RDP 开启，处理方法为将芯片保护级别去除，具体选项字节操作设。
- 芯片WRP开启，处理方法为将扇区写保护去除。具体选项字节操作。
- 其他原因请参考FAQ 中的问题汇总。

校验Program Memory

使用校验功能，对比芯片中的数据和 Program Memory 区域的数据是否一致。



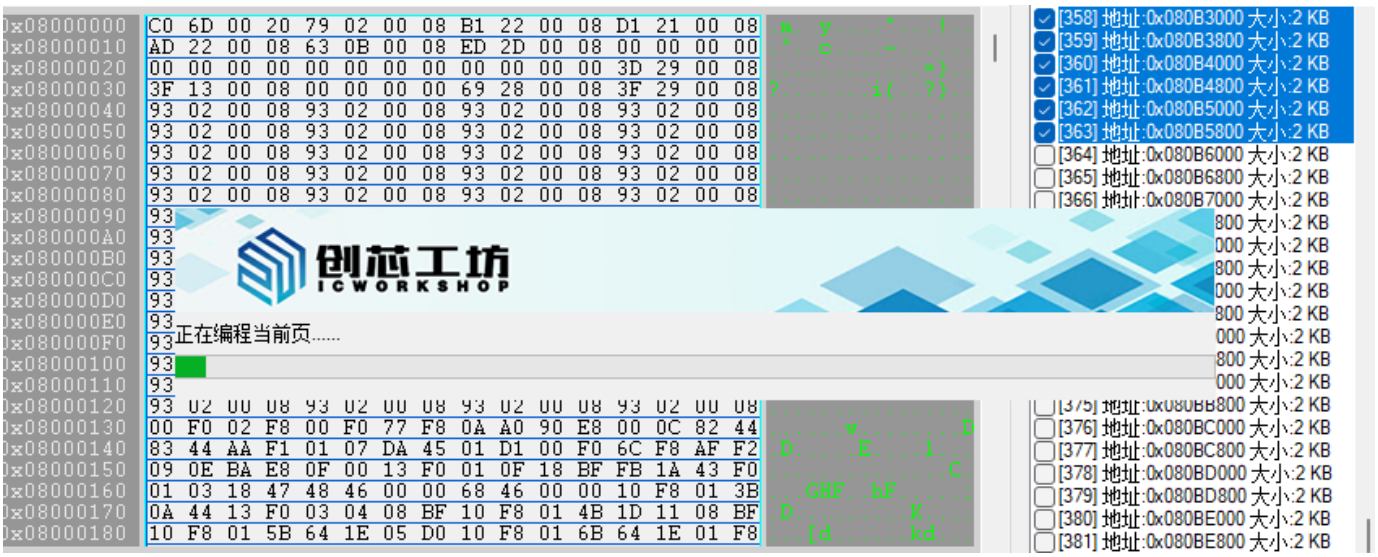

```
02/22-10:04:09:524> Program Memory 校验 成功!
```

> **提示**
>
> - 如果添加了分段固件，校验分段固件本身的数据。
> - 如果没有添加分段固件，校验的是整个Program Memory 缓冲数据区域。

Program Memory自动编程

Program Memory自动编程功能是 **擦除、写入、校验**的组合功能，执行此功能时，将会自动执行 Flash 擦除、写入、校验。

```
02/22-09:50:52:271> Program Memory 擦除 成功!
02/22-09:52:26:851> Program Memory 编程 成功!
02/22-09:52:44:917> Program Memory 校验 成功!
```



全功能自动编程

全功能自动编程的作用是将用户所有设定的数据，设置项，以在线的方式一次性写入，包括SN，和Matrix 签名数据(授权方式选择PowerWriter 内置的前提下)，并同步更新SN 的信息。

```
02/22-09:54:54:577> 保存成功
02/22-09:54:54:584> Power Writer®全功能在线编程...
02/22-09:54:54:589> 写入出厂默认OB...
02/22-09:54:54:838> 重新计算Program Memory 数据...
02/22-09:54:54:847> 智能在线擦除芯片...
02/22-09:54:55:017> 写入芯片数据
02/22-09:54:55:020> 写入用户自定义OB...
02/22-09:54:55:244> 全部完成!
```

> **注意**
>
> 在线全功能自动编程，包含ICWKEY签名的项目无法使用此方式写入。

其他数据区操作

PowerWriter 除支持**主存储**空间的在线操作之外，同时也支持EEPROM、OTP、DATA 等数据区的操作，操作支持以下几种功能

- 查空其他区域。
- 擦除其他区域(如果为可擦除的数据块)。
- 写入其他区域。
- 读取其他区域。
- 校验其他区域。

> **操作提示**
>
> 其他区域，比如EEPROM、OTP 操作方式和 Program Memory 相同

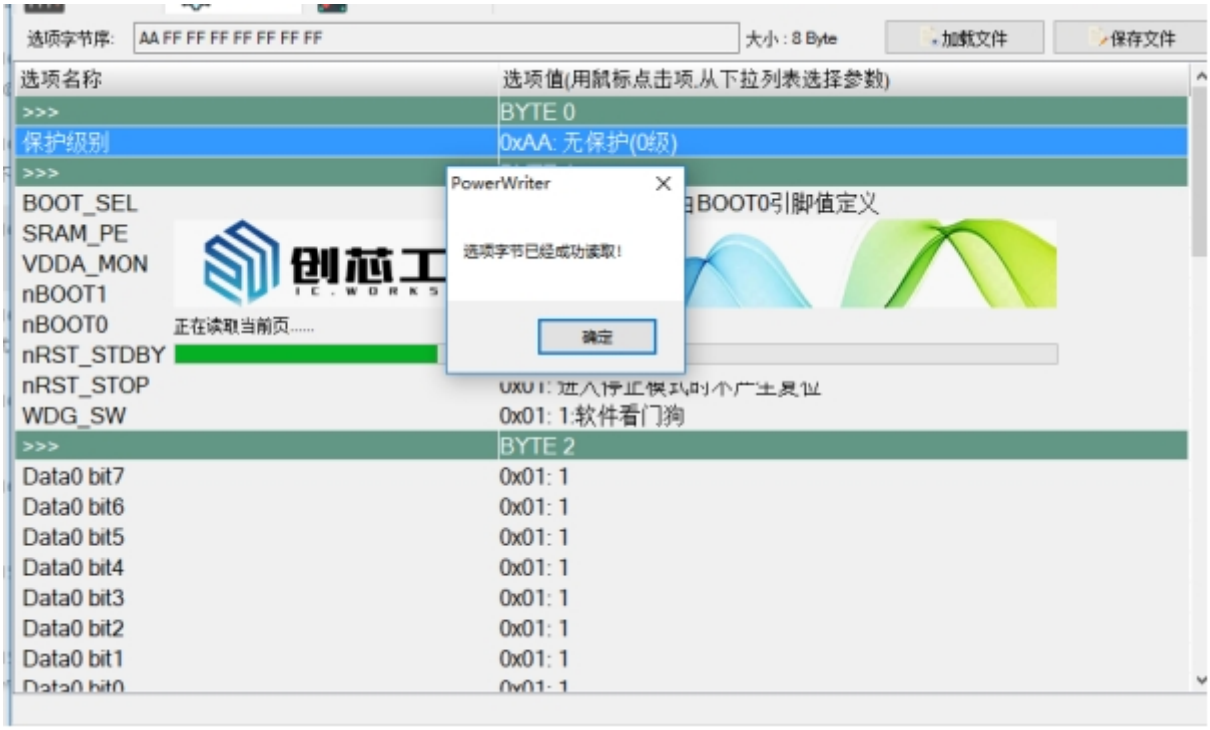
复位目标芯片

复位目标芯片功能，可以执行一次目标芯片的复位动作。此操作将同时触发：

- 执行一次软复位动作。
- RST 引脚输出一次外部复位信号。

读取选项字节

可以使用读取选项字节功能，读取目标芯片的选项字节，操作如下所示：

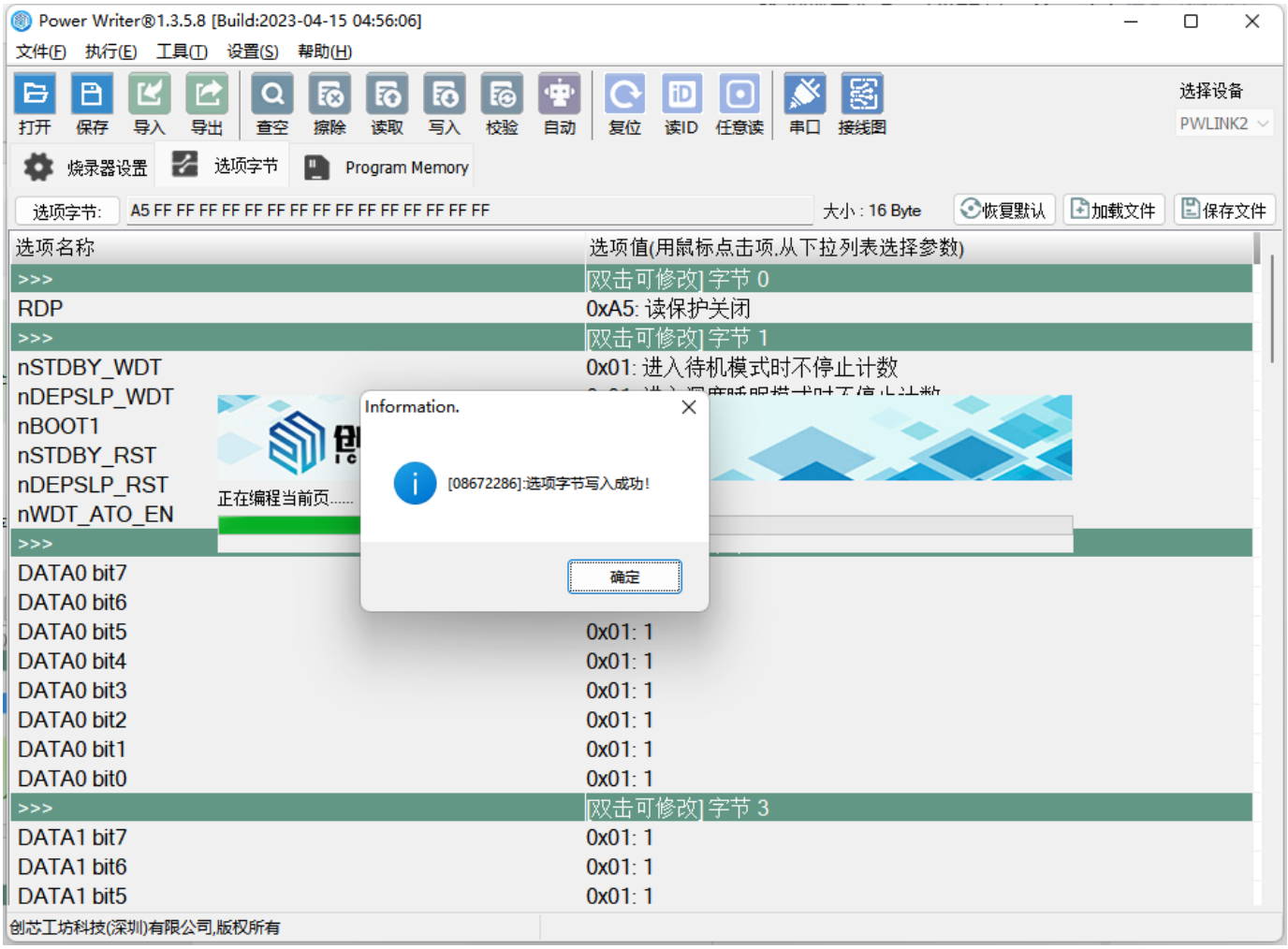



```
02/22-10:05:50:658> 选项字节已经成功读取！
```

> **提示**
>
> 如果用户选择的芯片和目标芯片不匹配，则无法读取选项字节并提示相关错误信息。

写入选项字节

当需要更新芯片的选项字节时，通过此功能，写入自定义的选项字节，操作成功如下图所示。




```
02/22-10:06:16:362> 选项字节写入成功!
```

> **提示**
>
> 写入选项字节无需手动断电或复位，此步骤在写入选项字节后自动完成。
>
> 部分芯片写入选项字节可能需要额外的通信引脚，请参考 #查看芯片的接线图

读取CID

读取芯片的ID，连接上芯片之后直接读取即可在日志栏查看芯片ID。

```
02/22-10:08:57:141>  mChipID[] =
{0x32,0x00,0xD9,0x05,0x33,0x42,0x33,0x35,0x49,0x39,0x22,0x43};
```

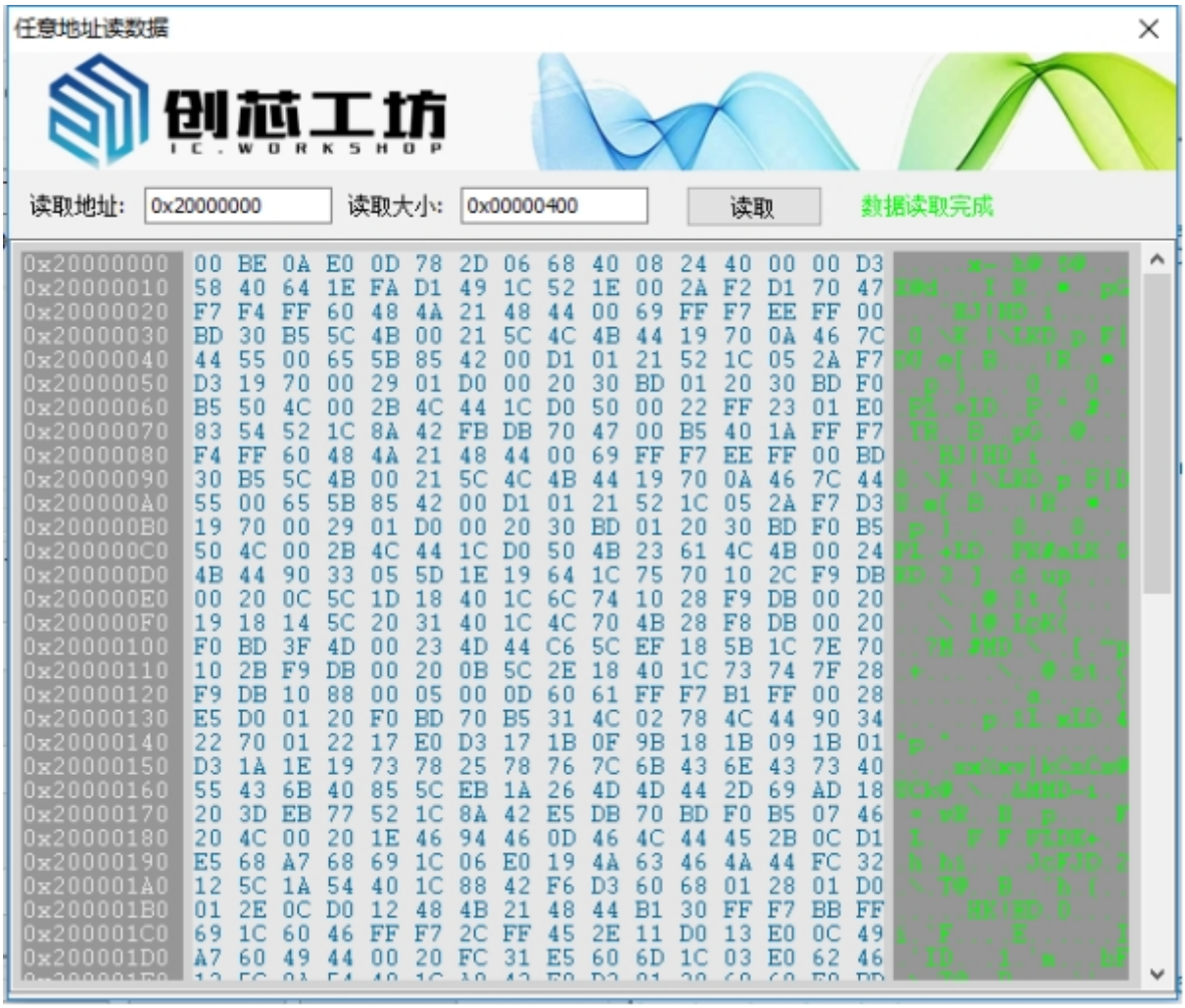
> **提示**
>
> 注：部分芯片ID 是非连续的，PowerWriter 显示的ID 是将不连续的ID 连续显示。

任意地址读数据

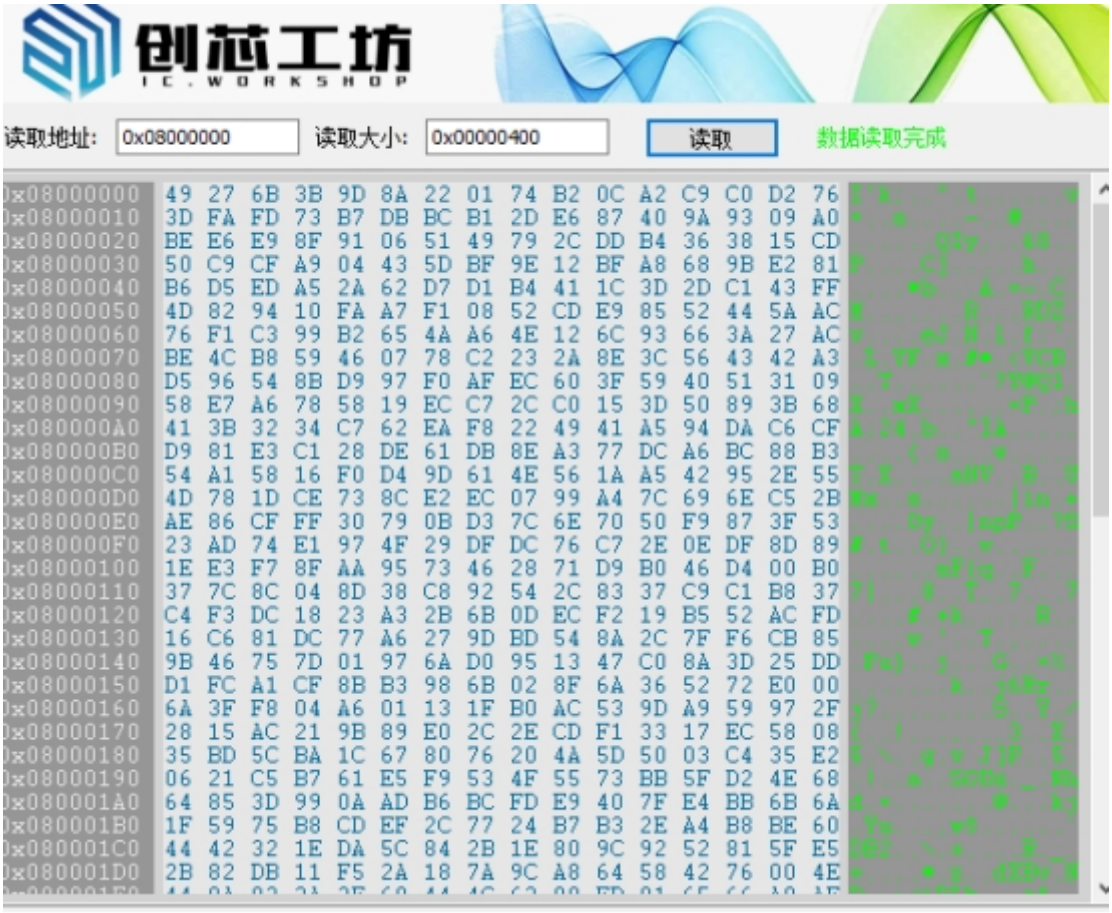
任意地址读数据是一个方便实用的功能，可以指定任意的芯片内部存储空间地址，包括但是不限于：

- 读取芯片RAM 数据并显示
- 读取芯片Flash 数据并显示
- 读取芯片所有寄存器数据并显示

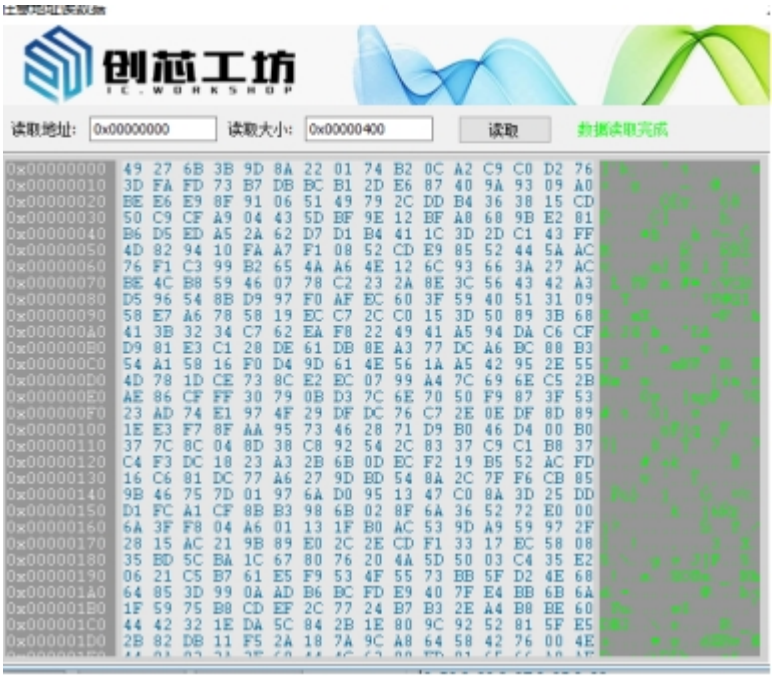
如读取芯片RAM 数据，从地址0x20000000 读取1K 的RAM 数据并显示如下：



读取任意的Flash 地址数据，如读取0x08000000 1K 大小的数据，显示如下：



读取芯片内部的其他地址数据，结果显示如下所示：



注意

- 任意地址读数据依赖于你对当前使用芯片的熟悉程度，读取芯片内部的数据时，需要设定好需要读取地址，以及大小。
- 读数据可能因为权限问题，或者是由于跨模块时读取数据会导致失败。

读取最后一次离线操作结果

在使用PowerWriter进行离线生产时，遇到错误，针对PowerWriter没有显示屏的产品，通过此功能，可以查看离线烧录错误的原因，

如下显示了烧录器未断开连接情况下进行离线烧录时的提示信息：

```
02/22-10:13:37:740> [0028] The writer needs to be disconnected from the online connection....
```

工具菜单

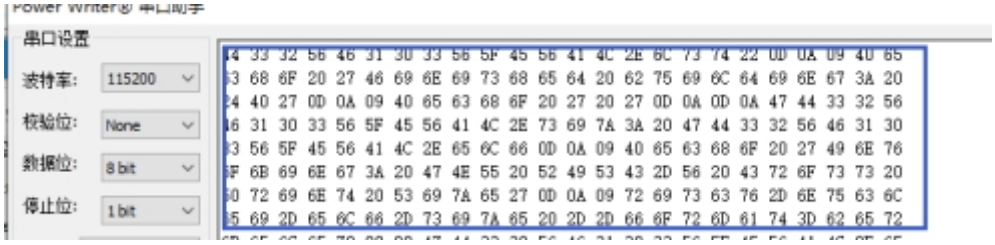
PowerWriter 除了作为开发工具常用的调试器、烧录器等功能外，为了降低使用难度，同时集成其他常用的工具，或者引导性的功能，包含：

- 串口调试助手
- PowerWriter 接口定义
- 芯片接线图参考
- 预留数据读写
- 离线项目高级设计
- 厂商插件
- UID 授权信息导出与加载

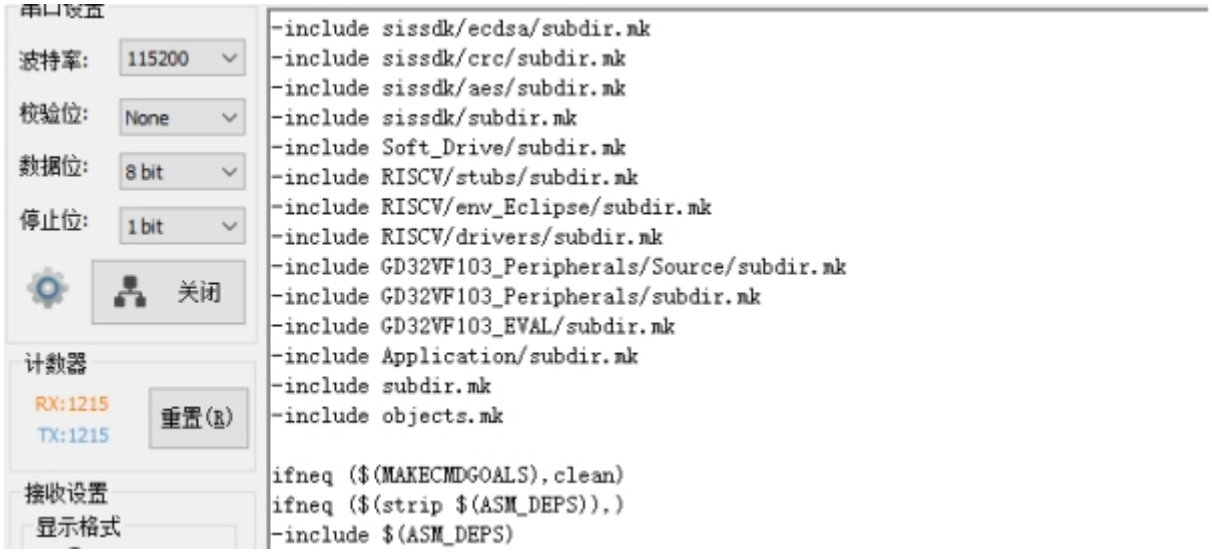
串口调试助手

内置串口调试助手，并预设串口代理转发参数，PowerWriter 串口调试助手界面如下所示：

- **波特率**：设置串口的波特率，默认为 115200。
- **校验位**：是否对串口的数据添加校验位，默认为无，可选奇校验、偶校验。
- **数据位**：设置串口通信的数据位数，默认为8bit，可选7bit、6bit、5bit等。
- **停止位**：设置停止位位数，默认为1bit,可选1.5bit、2bit。
- **打开或者关闭**：连接或者断开PowerWriter 的串口。
- **计数器指示器**：计数器用于显示当前串口助手发送和接收数据的总数。
- **计数器指数器重置**：重置计数器。
- **接收显示格式**：可以设置为16进制显示或者是字符串显示形式。
- **16进制模式**：将会按照16进制显示接收到数据，如下所示。

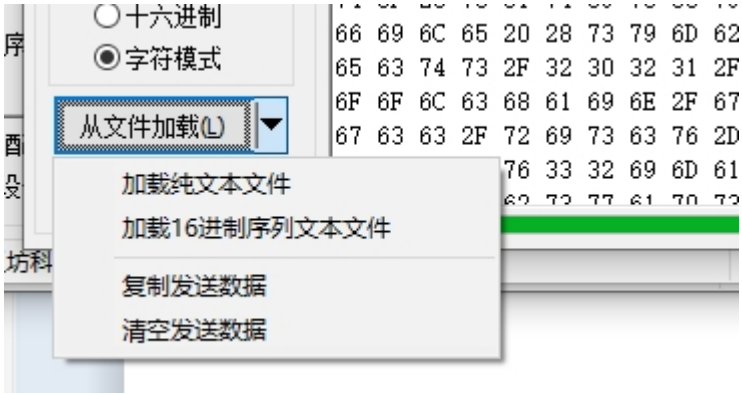


- **字符串模式(多字节)**：将会按照字符串的形式显示，可以使用换行符，\r\n 或者 \n 显示换行，如下图所示：



- **清空接收数据**：清空接收到的所有数据。
- **保存到文件**：将接收数据保存到磁盘文件。
- **发送格式**：发送格式用于设置PowerWriter 串口助手用何种格式来发送加载的文件。
- **字符模式**：默认把加载的文件按照字符(RAW 数据)发送，不会进行额外解析。
- **16进制模式**：如果选择16进制模式，PowerWriter 串口助手将会将缓冲区的数据解析为16进制字符串，地址从左到右，从上到下，如果高位缺失，则会丢弃此数值，如下图所示：

- **从文件加载**：从文件加载默认会加载为二进制文件，并将发送格式也切换为16进制模式，这样做的目的是，不对用户的文件进行任何解析，防止信息丢失，或者转换过程产生内容变化，无法达到用户的预期，如需加载其他的文件，则需要点开按钮后面的小箭头，打开扩展菜单，如下图所示：



- **加载纯文本文件**：主按钮默认按照二进制的模式加载文件，通过此菜单，则可以加载纯文本文件，并将发送格式改为字符串模式进行发送。

- **加载16进制序列文本文件**：可以通过此菜单加载16进制序列的文本文件，PowerWriter 将自动对数据序列化。

- **发送**：将缓冲区的数据发送到PowerWriter 的TX引脚，发送过程可查阅进度。

> **提示**
>
> 1：PowerWriter 的串口助手主要用于调试，支持ASCII码和 UNICODE 编码，暂无其他的字符编码支持编码支持，如有其他编码支持的需求，请给我们提意见，或者使用第三方串口工具。
>
> 2：PowerWriter 不同产品系列支持的波特率可能存在差异，如果使用第三方串口工具，请通过PowerWriter 软件查看当前产品支持的波特率列表。

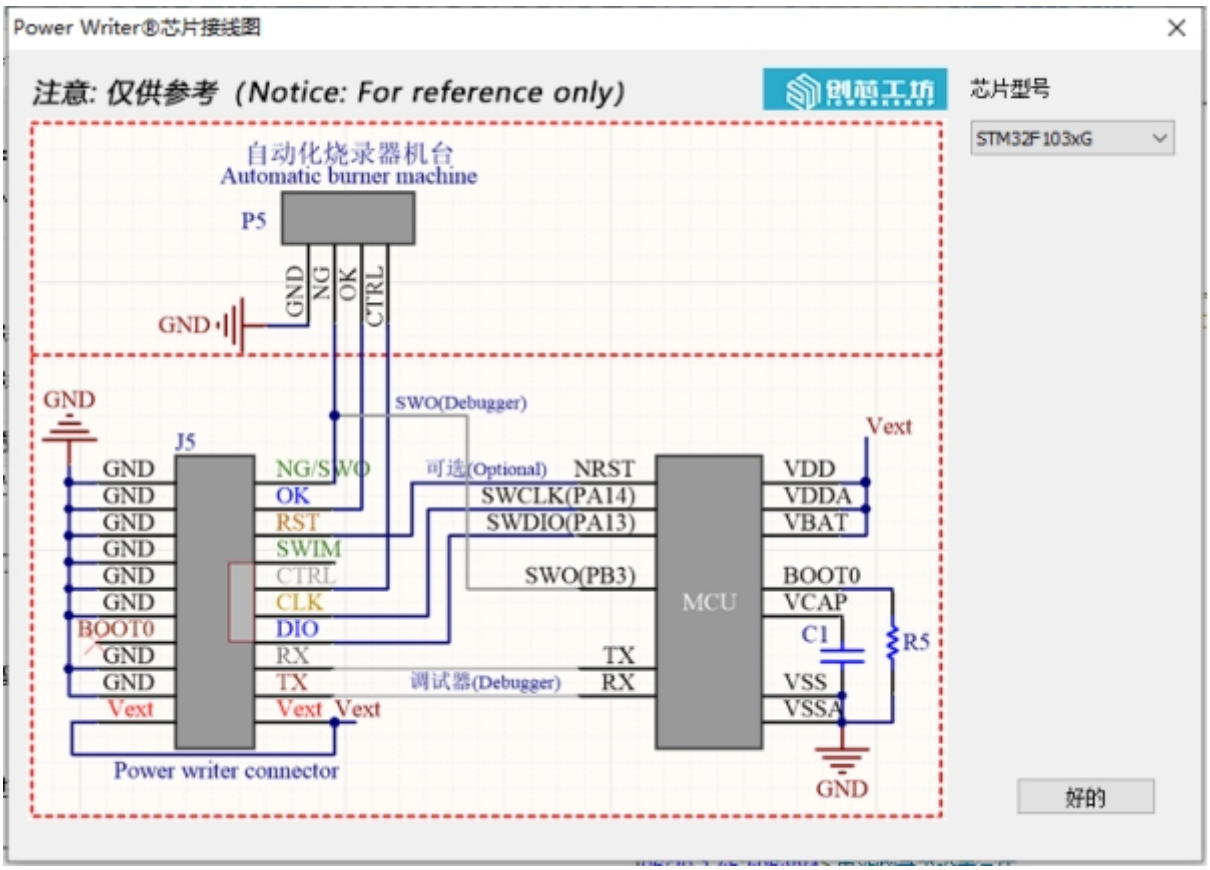
查看PowerWriter 接口定义

查询PowerWriter 的信号定义信息，点击菜单之后，将会弹出当前已连接的PowerWriter 产品类型的接口定义，以及不同硬件版本之间的差异，也可查询整个产品系列的接口定义定义信号，如下图所示:

> **提示**
>
> PowerWriter 接口定义可以查阅同类产品所有版本关于接口调整信息。

查看芯片的接线图

PowerWriter 对已经适配的芯片，提供接线图参考，在不确定如何连接线材，来进行读写的情况下，可以选择好芯片后，通过此功能快速查询当前芯片的接线图预览，如下图所示：

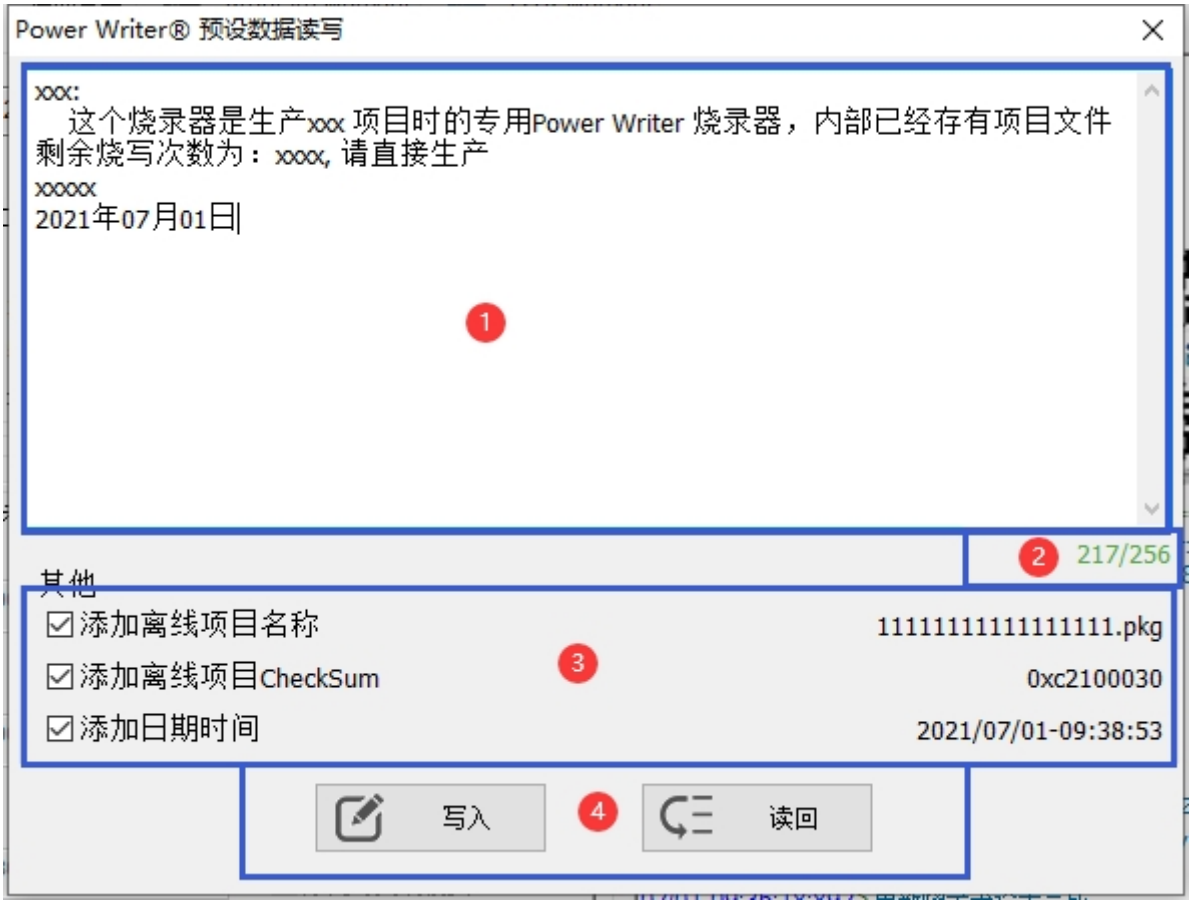


注意

- 提供的接线图为缩略图，只提供了最基础的接线、并没有完整提供芯片的整个最小系统。
- 提供的接线图根据芯片厂家资料核对整理，仅供参考，如有疑问，请及时与我们反馈。
- 芯片接线图默认会显示当前选择的芯片，也可以通过右侧的列表执行其他芯片查询。

预留数据读写

PowerWriter 预留数据读写功能，用于标记PowerWriter 硬件，最长为**256** 字节，其功能界面如下图所示：



PowerWriter 预留数据读写功能

- **用户自定义编辑区**：自定义编辑区可以由用户自行编辑备注内容，编辑时，中右侧将会有空间容量提示器，如上图所示。
- **用量提示**：实时提示当前数据区的使用量。
- **标准模板**：标准模板提供快速录入常规的信息，包含如下：
  - **添加离线项目名称**：快速添加离线项目名称, (需要先保存离线项目)。
  - **添加离线项目Check Sum**：快速添加离线项目checksum (需要先保存离线项目)。
  - **添加日期时间**：快速添加日期时间记录。
- **写入和读回**：写入到PowerWriter 或者 从PowerWriter 读回记录。

离线生产项目高级设置

除了硬件主机可以添加备注，也可以给项目文件添加备注，以及添加烧写权限设定，包含：离线文件内注释、离线文件限定PowerWriter 序列号、离线文件限定UID 烧录范围等功能，如下图所示：

PowerWriter 项目文件高级设置:

- **离线文件内注释**：通过离线文件内注释功能，可以给项目文件编写注释信息，和预留数据读写功能一样，区别在于：预留数据用于标记PowerWriter 主机，而离线文件内注释跟随项目文件，而不跟随PowerWriter 主机，同样，注释长度最长 256 字节，输入时支持提示器实时显示用量。

- **离线文件限定PowerWriter 序列号**:

  - **启用**：功能启用开关，启动此功能后，当前的pkg 文件只能用于列表中指定的 PowerWriter 烧录器烧录。

  - **序列号列表**：绑定项目文件的SN列表，列表中的烧录器可以烧录当前的项目文件，其他 PowerWriter 烧录器无法烧录 文件。

- **SN 序列号**：SN 序列号编辑框，用于再此输入对应烧录器的SN，然后即可添加到列表中。
- **添加SN**：将当前的SN 序列号添加到序列号列表，格式错误将会有提示。
- **添加当前设备**：快速添加当前连接的设备。
- **删除选中**：删除序列号列表中选中的SN，支持多选。

> **提示**
>
> PowerWriter 的SN 为32位 16进制字符，如果格式不对，将会无法添加，此外，输入的时候将会有输入长度提示。

- **离线文件限定UID 烧录范围：**
  - **启用**：功能开关，启用或者关闭，启用此功能时，pkg 文件将只能烧录 UID 指定范围内的芯片，超出范围，将会报错。提示Error: UID Limit。
  - **从0x~**：限定UID 范围的 低位UID 信息，默认为0x00。
  - **到0x~**：限定UID 范围的 高位 UID 信息，默认为0xff。

> **注意**
>
> a. 低位UID 表示任意数据用0x00 代替，因为任何一个数值都 >= 0x00。
>
> b. 高位UID 表示任意数据用 0xff 代替，因为任何一个数值都 <= 0xff。
>
> c. UID 限定功能只有在当前所选芯片有 UID 信息时才能开启，如果是无UID 的芯片，将会是灰色禁用状态。
>
> d. UID 限定会根据当前选选芯片，自动确定UID 的长度，大部分芯片的UID 均为12位，少数部分芯片的UID 为8位，或者是16位，请留意。

厂商特定功能

厂商特定功能属于PowerWriter 的插件接口，用于扩展部分厂家的特定功能，此部分功能不通用，属于厂家在PowerWriter 上的扩展功能，在特定的芯片情况下，才会使能，这部分内容请咨询芯片厂家关于PowerWriter 中定义的使用方法和参数，目前包含了一部分厂家的特定功能，如下所示：

- 扩展功能：Nuvoton 扩展设置

  请参考 Nuvoton FAQ 参考

  > **提示**
  >
  > 扩展功能在FAQ 中内容详细列出扩展功能的使用方法。

- 筛选功能：

  - SINOMICON NVR 筛选
  - CX32 NVR 筛选
  - HK32 NVR 筛选

> **特别提示**
>
> 厂家插件功能从菜单中移到了工具栏的最右侧厂家扩展插件按钮，见：**扩展插件**

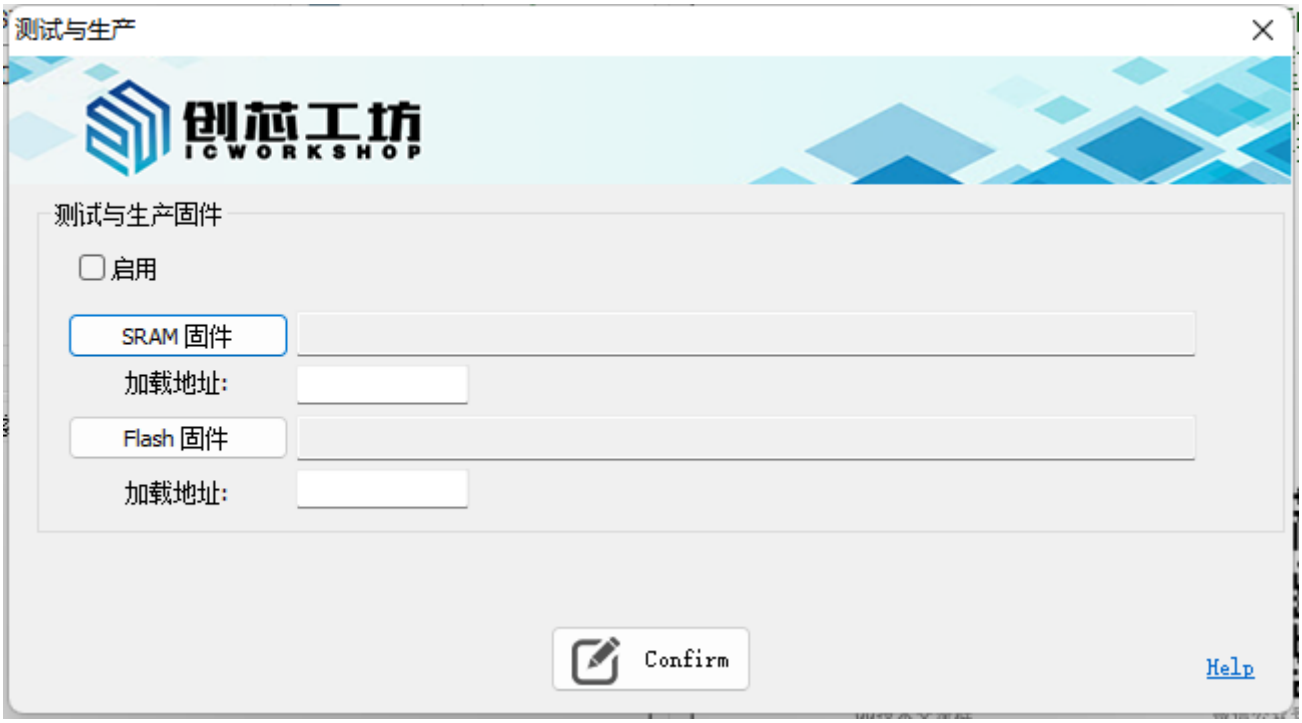
UID 授权配置导入和导出

UID 授权配置可以导出或者加载 UID 授权 配置节点，在授权控制方面，合作项目开发上更加便捷，比如库用户开发者要求项目在烧录时，必须指定一种合适的授权方法，但是用户固件可以由开发者自行添加，这种合作开发项目的方式，在PowerWriter 上可以得到支持，导入导出功能，在签名与授权页面一般选择模式为**锁定模式。**

> **提示**
>
> 导入和导出UID授权配置信息，可以设置密码对数据进行保护，对数据进行自定义密码加密，如果设置无密码模式，可以直接加载，配合授权与签名中加密模式为锁定模式时，加载 UID 授权配置也无法查看到配置信息。

测试与生产

PowerWriter 支持运行测试或者生产固件，默认提供两种类型生产测试固件的导入方式：SRAM 测试固件和Flash 测试固件，显示如下所示：



- **启用：** 开启测试固件导入功能。
- **SRAM固件**：导入生产测试的SRAM固件
  - **加载地址：**填写SRAM固件的加载地址
- **Flash 固件：**导入生产测试的Flash 固件
  - **加载地址：**填写Flash固件的加载地址

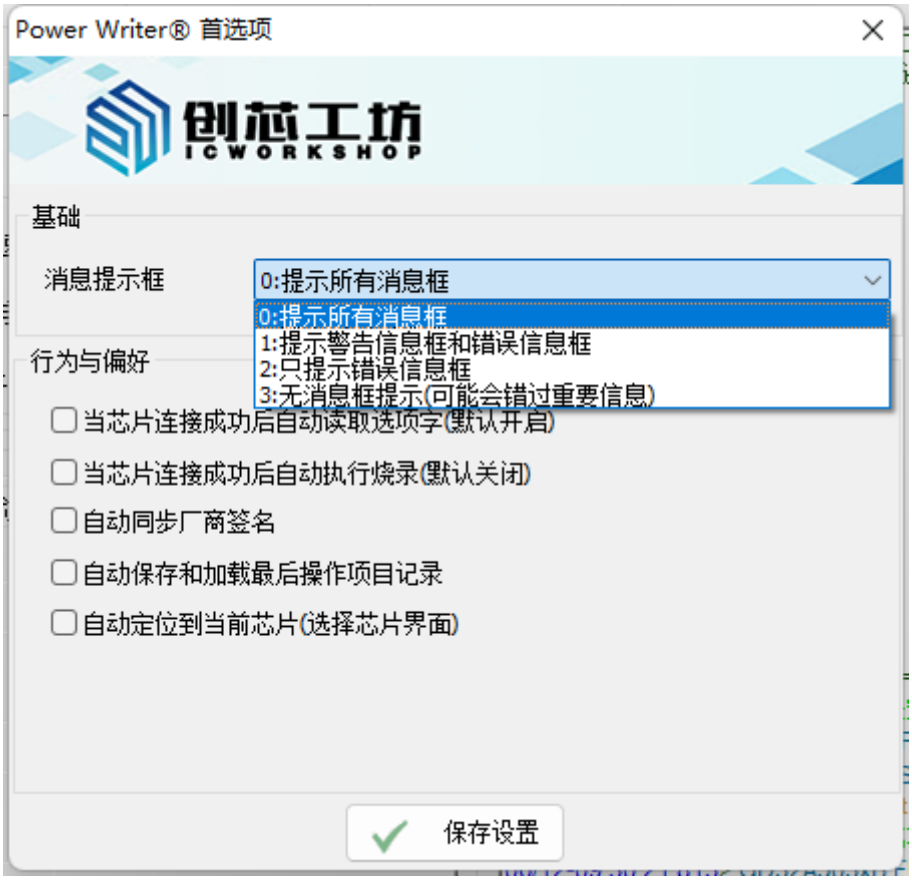
> **特别注意**
>
> 生产测试功能通过 AT 接口执行生产测试，AT 命令接入指南请参考: **PowerWriter AT 命令接口**

设置菜单

首选项

系统设置包含消息框提示设置和行为偏好设置。

消息提示框设置



PowerWriter 消息框提示设置：

- 0级：提示所有消息框。
- 1级：提示警告信息框和错误信息框。
- 2级：只提示错误消息框。
- 3级：无提示 (可能错过重要消息)。

> **提示**
>
> 注：消息框提示设置的是弹窗，所有的日志都可以在日志栏中查看，与当前设置并不关联。

行为与偏好

- 当芯片连接成功后自动读取选项字(默认开启)：此功能可以设置于在线模式，自动读取芯片的模式，开启时，芯片在连接时，自动读取选项字节到PowerWriter客户端软件，如果不需要此功能，可以选择关闭。
- 当芯片连接成功后自动执行烧录（默认关闭）：此功能可以设置于在线模式，用于批量烧录，此功能开启时，当客户端识别到芯片成功连接时将自动执行烧录操作，如果不需要此功能，可以选择关闭。
- 自动同步厂商签名：此功能用于设备pwlink2 lite,由于该设备在生产时默认签名几家芯片厂家，若需要其他已适配但pwlink2 lite未签名的芯片厂家，则需进行在线配置，此功能开启时，pwlink2 lite将自动同步网页的在线配置。

- 自动保存和加载最后操作项目记录：开启此功能后，Powerwriter客户端软件将自动保存最后操作项目记录，并在下次设备上电且连接上客户端时，自动加载上次最后操作项目。
- 自动定位到当前芯片（选择芯片界面）：此功能开启后，切换到选择芯片界面时，将自动定位到当前芯片所在位置。

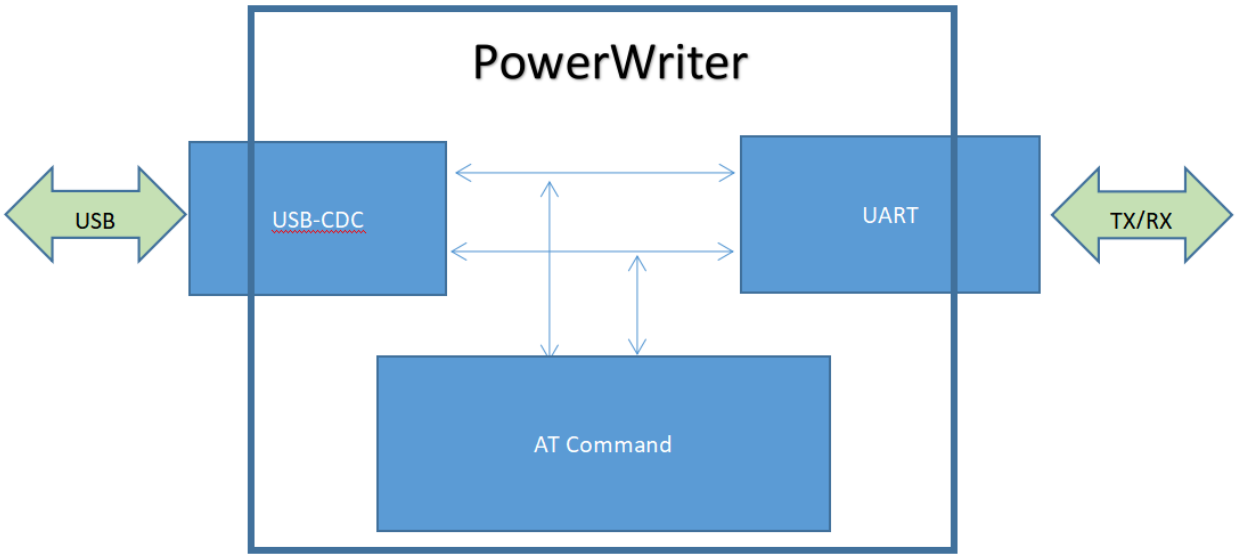
设备首选项

设备首选项中包含PowerWriter 硬件设备中的默认设置，界面显示如下所示：

- **保存全局电压：** PowerWriter 的默认输出电压为 3.3V，可以使用PowerWriter 软件设置输出电压为 1.8V 或者 5.5V 或者为外部参考电压，但是如果PowerWrietr 设备断电，则默认不保存输出电压设置，如果需要掉电保存PowerWriter 设置的输出电压，则需要开启此功能，并更新到设备。

> **特别注意**
>
> - 开启此功能的情况下，PowerWriter 上电后，会默认设置输出电压为上次设置的值，在给低压系统供电时，请特别留意PowerWriter 的输出电压，不要超过目标系统的耐压值，避免电压差异过大对目标芯片（系统）造成损坏。
> - 在离线烧写数据时，电压的取值会优先采用 Pkg 项目的预设值。

- **设备AT接口：**



开启PowerWriter AT 功能后，可使用PowerWriter 提供的 AT 命令接口，对PowerWriter 的行为进行控制，实现将PowerWriter 集成到各种各样的产品应用中，硬件原生AT 接口：可实现任意系统。或者硬件平台环境的深度集成，并提供最高的响应效率。

  - **开启USB 端口 AT 功能：**开启 PowerWriter USB CDC 端口为 AT 命令接口，此时，外部设备可通过PowerWriter USB 端串口实现对PowerWriter 的控制权，主要用于二次开发的Windows 、Linux、MacOS 等桌面系统软件实现对PowerWriter 设备的控制，也可以集成到其他可驱动USB CDC 的其他系统中。

  - **开启UART 端口 AT 功能：**开启PowerWriter 输出 UART 端的 AT 命令接口，此时，外部设备通过 PowerWiter 的输出UART（TX/RX）实现对PowerWriter 的控制，主要用于脱离桌面系统，需要集成PowerWriter 到硬件系统（嵌入式系统）中的使用情形。

  - **开启AT 数据加密：**

    AT 命令接口属于开放式接口，默认不加密收发数据，但是对于数据安全要求较高的生产环境中，需要对传输的数据进行保护，避免数据泄露而造成损失，如需加密文件数据，关键密文交换，则需要开启加密功能，AT 命令接口提供的加密算法为 **AES128bit -CBC** 加密，填充模式为 **0 填充**，需要**对齐到16 字节**，（AT 命令默认已对齐）。

    - **加密密码：**CBC 模式的加密密码, 16 个字节。
    - **初始向量：**CBC 模式的初始向量，16个字节。
    - **随机生成:** 随机生成 AT 命令的加密密码。
    - **复制到剪贴板:** 复制加密的密钥信息到系统剪贴板。

- **更新到设备**：将当前设置同步到PowerWriter 设备中

提示

- AT 命令功能只有在 PowerWriter 空闲状态下有效 (未连接到PowerWriter 软件)。
- USB-CDC、UART 端口中任意一个开启 AT 命令，都会导致PowerWriter 内置的透传功能关闭 （USBCDC <-> UART）
- AT 命令的串口波特率，如果开启了USB-CDC 的波特率，则UART 的波特率跟随 USB-CDC，确保两者速率一致，而不会导致缓冲区溢出风险，如果只单使用 UART 作为AT 命令接口，默认波特率为 **9600**，可通过 AT 指令更改默认的波特率（新波特率必须是PowerWriter支持的类型 ）。
- AT 命令的加密密码只可更新，不可读取，如忘记密码，只能重新设置。确保系统和 PowerWriter 一致，并保持加密模式一致（**CBC**）。
- 部分品牌无法使用 UART 端 AT 命令功能，(在烧录数字到目标芯片时，需要使用ISP 功能更新目标芯片数据)

关于加密算法的补充

问题1: 为何选择AES作为AT 加密算法?

AES 加密算法有着较高的安全性以及运算性能，且被广大开发者熟知，且部分MCU 带硬件AES，容易快速入手

问题2: 为何使用CBC?

CBC 模式相对于ECB 的字典模式，安全性较高，对于CFB，CTR等其他模式使用起来相对简单，且更常用

问题3：什么情况下需要开启加密

如有使用AT指令传输部分关键敏感数据，比如文件数据，或者是密码信息，则需要开启加密

AT 命令接入指南请参考: **PowerWriter AT 命令接口**

语言设置

PowerWriter 默认提供两种语言选择，中文和英文，软件在首次启动时，会自动根据系统的语言设定默认语言，在简体中文系统中会默认选择简体中文，其他非简体中文操作系统，默认将会选择英文界面，如果想更改软件的显示语言，可以在此处进行设置。

- 中文：设置为中文语言界面。
- 英文：设置为英语语言界面。

> **提示**
>
> - 切换语言时需要重启PowerWriter软件。
> - 软件在初次使用时自动检测系统语言，如系统为简体，显示中文，其他情况下，显示英文界面，可以通过菜单手动切换。
> - 暂未提供其他语言。

窗口置顶

通过窗口置顶设置，设置PowerWriter软件为桌面 Z 序的最顶端。

> **注意**
>
> 设置窗口置顶会将PowerWriter 我的主窗口设置在最顶端，在绝大部分情况下，不推介使用此方法，只有在少数情况下，如有此类需求，可以设置窗口为最顶端。

帮助菜单

帮助菜单显示常用帮助信息，比如软件手动升级，跳转创芯工坊官方网站，查看用户手册，以及查看发布信息。

安装驱动

如果软件找不到驱动程序，并且自动检测安装驱动也失败，PowerWriter软件无法连接硬件，可通过点击**菜单->帮助->安装驱动**来手动尝试驱动程序。

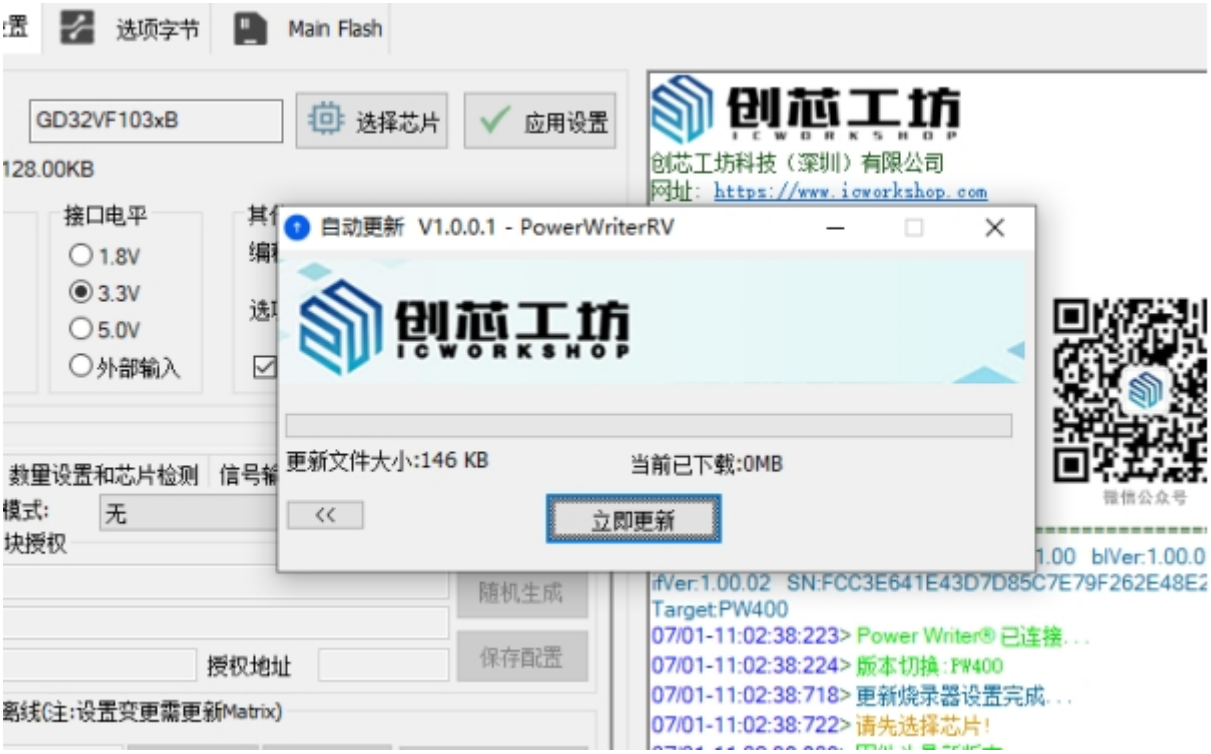
> **提示**
>
> 自助驱动问题解决方法，请参考 常见驱动问题解决方法参考

固件升级

固件升级在连接PowerWriter 烧录器时自动提示，如检测到新版本，则会提示升级。意外关闭取消后可手动点击固件升级菜单，如果发现固件有新版本，将会对重新弹出升级对话框，提示对固件进行升级，如图所示：

软件升级

在启动软件时自动检测更新，如果检测到新版本，则会提示升级。意外关闭之后，也可以通过手动点击菜单进行升级检测，如图所示：



官方网站

通过点击菜单可以快速达到创芯工坊官方站点：https://www.icworkshop.com。

用户手册

> **注意**
>
> 离线版本用户手册已经不在维护，请收藏本文档站点，查询在线文档内容。

关于

关于菜单可以查看PowerWriter 应用软件的发布信息，以及创芯工坊的用户协议。

工具栏

本章节将介绍工具栏的功能，工具栏作为常用功能的快捷入口设立，如图所示，从左到右分别包含如下功能:

- 从磁盘加载项目文件
- 保存项目到磁盘
- 从磁盘加载当前Tab页数据
- 保存当前Tab页数据到磁盘文件
- 保存项目到磁盘并加载到PowerWriter
- 读取PowerWriter 离线项目保存到磁盘并加载
- 查空当前Tab 页数据
- 擦除当前Tab 页数据
- 读取当前Tab 页数据
- 写入当前Tab 页数据
- 校验当前Tab 页数据
- 全功能在线写入所有数据
- 复位目标芯片，让芯片运行
- 读取芯片ID
- 芯片任意地址数据读取
- 获取最后一次离线操作结果
- 打开串口助手
- 查看芯片接线图
- 离线项目高级设置
- PowerWriter OEM 版本选择列表

工具栏

从磁盘加载项目文件

打开项目按键功能同菜单: **加载项目**

保存项目到磁盘

保存项目按键功能同菜单: **保存项目**

从磁盘加载当前Tab 页数据

加载文档到当前TAB页，属于动态功能，可以单独保存每一页Tab的配置数据和加载上次保存的数据，详细的支持特性如表所示：

TAB 名称	是否支持
烧录器设置TAB	支持保存到文件
选项字节TAB	支持保存选项字节到文件
Program Memory TAB	支持保存整个Program Memory到文件
OTP Memory TAB	支持
EEPROM TAB	支持
其他	支持

> **注意**
>
> - 烧录器设置从文件加载时，需要输入上次保存烧录器设置的文件密码，否则将无法加载，格式为pkg 格式。
> - 选项字节的加载支持PowerWriter 定义的Hex,bin,S19格式，非PowerWriter 导出的格式无法支持。
> - Program Memory 加载文件是加载到缓冲区，不显示分段固件信息。

保存当前Tab页数据到磁盘文件

保当当前TAB页到文档，是一个非常实用的功能，可以单独保存每一页Tab的配置数据和加载上次保存的数据，详细的支持特性:

TAB 名称	是否支持
烧录器设置TAB	支持保存到文件
选项字节TAB	支持保存选项字节到文件
Program Memory TAB	支持保存整个Program Memory到文件
OTP Memory TAB	支持
EEPROM TAB	支持
其他	支持

> **注意**
>
> - 烧录器设置保存到文件时，需要填写烧录器设置的文件密码 (如果有)，格式为pkg 格式。
> - 选项字节的保存支持PowerWriter 定义的Hex,bin,S19格式。
> - 保存 Program Memory 到文件是保存的是缓冲区，不显示分段信息，如果需要分段，请使用保存项目到文件，同样支持Hex,bin,s19格式。

保存项目到磁盘并加载到PowerWriter

此功能同菜单： **保存并离线加载**。

读取离线烧录档保存到磁盘并加载

此功能同菜单： **离线读取并保存**。

查空当前TAB页数据

当前TAB 页查空功能支持查空 当前TAB页面的数据，详细的支持特性如图表所示：

TAB 名称	是否支持
烧录器设置TAB	无需此功能

TAB 名称	是否支持
选项字节TAB	无需此功能
Program Memory TAB	支持，等同于查空program-memory
OTP Memory TAB	支持，菜单中#其他数据区操作
EEPROM TAB	支持，菜单中#其他数据区操作
其他动态页面	支持，与所选芯片有关

擦除当前TAB页数据

当前TAB 页擦除功能支持擦除当前TAB页面的数据，详细的支持特性如表所示：

TAB 名称	是否支持
烧录器设置TAB	无需此功能
选项字节TAB	无需此功能
Program Memory TAB	支持，等同于#擦除program-memory
OTP Memory TAB	支持，#其他数据区操作
EEPROM TAB	支持，#其他数据区操作
其他动态页面	支持,与所选芯片有关

> **注意**
>
> 部分芯片擦除后数据为0x00，绝大部分芯片擦除后数据为0xff。

读取当前TAB页数据

当前TAB 页写入功能支持写入当前TAB页面的数据，详细的支持特性如图表所示：

TAB 名称	是否支持
烧录器设置TAB	支持，**此功能可读取离线项目配置信息**
选项字节TAB	支持，等同于#读取选项字节
Program Memory TAB	支持，等同于#读取program-memory
OTP Memory TAB	支持，等同于#其他数据区操作
EEPROM TAB	支持，等同于#其他数据区操作
其他动态页面	支持,与所选芯片有关

注意

- 烧录器设置TAB 页面读取，将会读取烧录器的设置到PowerWriter 软件，但是部分安全相关信息将不会同步到PowerWriter。
- 可以通过此功能读取离线烧录项目文件离线剩余次数。

写入当前TAB页数据

当前TAB 页写入功能支持写入当前TAB页面的数据，详细的支持特性如图表所示：

TAB 名称	是否支持
烧录器设置TAB	支持，等同于"**应用设置**"按钮
选项字节TAB	支持，等同于#写入选项字节 菜单
Program Memory TAB	支持，等同于编程program-memory 菜单
OTP Memory TAB	支持，等同于菜单中#其他数据区操作
EEPROM TAB	支持，等同于菜单中#其他数据区操作
其他动态页面	支持,与所选芯片有关

校验当前TAB页数据

当前TAB 页校验功能支持校验当前TAB页面的数据，详细的支持特性如图表所示：

TAB 名称	是否支持
烧录器设置TAB	无需此功能
选项字节TAB	无需此功能
Program Memory TAB	支持，等同于#校验program-memory菜单
OTP Memory TAB	支持，等同于菜单中#其他数据区操作
EEPROM TAB	支持，等同于菜单中#其他数据区操作
其他	支持，等同于菜单中#其他数据区操作

全自动在线写入所有数据和设置

等同于菜单项中的全功能自动编程。

复位目标芯片

等同于菜单项中的复位目标芯片。

读取CID

等同于菜单项中的：读取CID。

读取芯片任意地址数据

等同于菜单项中的：任意地址读数据。

获取PowerWriter 最后一次离线操作结果

等同于菜单项中的：读取最后一次离线操作结果。

打开PowerWriter 串口助手

等同于菜单项中的：串口调试助手。

查看芯片接线图

等同于菜单项中的：查看芯片接线图。

扩展插件

此功能由菜单中 **厂商特定功能** 合并而来，如果当前所选芯片出现插件按钮，则存在插件功能，插件功能的使用方法，可以FAQ中找到，也可咨询对应芯片厂家获取使用方法(如未在FAQ中找到文档)。

PowerWriter OEM版本切换

PowerWriter客户端属于融合客户端，同一类型的产品，用同一个客户端，不同的PowerWriter产品会有OEM版本差异，不同的烧录器在部分功能上也有差异，如PWLINK 没有离线功能，则对应的菜单离线功能是禁用的，用户可以通过此列表找到对应的版本，执行切换，然后打包项目。

> **注意**
>
> - 如果PowerWriter 处于连接状态，将会自动匹配到对应的OEM 版本，并且为不可选状态。
> - 如果PowerWriter 处于断开状态，才是可以手动切换的状态。
> - 打开软件默认为PW200的OEM，如果需要切换到PW300 或者其他型号进行打包，请手动切换到PW300。
> - 不同OEM版本之间不能混用同一个项目文件!

PowerWriter Tab 标签页

PowerWriter 包含数个标签页，其中包含固定的三个标签页：烧录器设置和选项字节，以及Program Memory 标签，以及包含数个动态的标签页（根据所选择的芯片自动识别）：OTP Memory 和EEPROM 标签页等, 以下章节将详细介绍各个标签页的设置和使用方法。

烧录器设置Tab标签页

烧录器选项标签页作为PowerWriter 最重要的部分，包含了PowerWriter 数十项常用功能，本节内容较为重要，请仔细阅读其中的功能说明，烧录器选项Tab 如图所示：

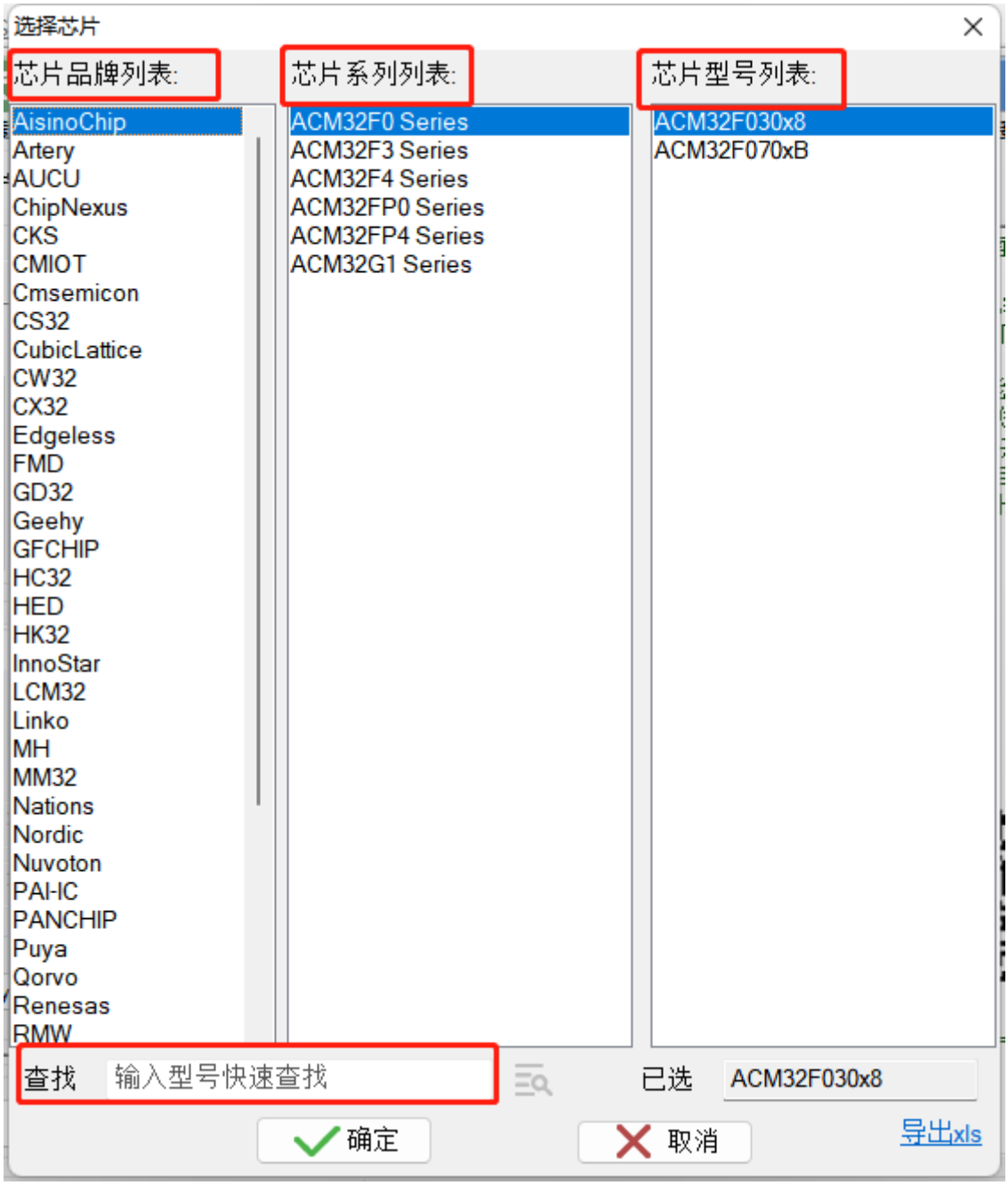
烧录器设置Tab 又分为：芯片设置、烧写功能设置、日志窗口三大部分，以下将对三大功能区域进行详细地介绍。

芯片设置

芯片设置部分包含芯片设置的常用基础功能，包含:

- **MCU 型号**：显示用户当前选择的MCU 型号。
- **选择芯片**：使用PowerWriter前需要选择对应的芯片，PowerWriter 会逐步增加常用芯片厂商芯片的适配，并对每个厂商芯片的更新进行追踪，确保适配厂家市面上可以买到的芯片在

PowerWriter中都能找到，点击选择芯片后将会弹出芯片选择对话框，左侧为芯片品牌分类，中间系列，右侧是芯片型号列表，也可通过左下方输入框进行芯片型号的快速查找，如图所示：



- **应用设置**：当修改了PowerWriter 的相关设置，需要手动点击一次将配置同步到 PowerWriter 硬件端。
- **擦除方式**：擦除方式用于设置用于离线烧录时 PowerWriter 的擦除方式，默认为全片擦除。

擦除方式	说明
不擦除	离线烧录时不对目标芯片进行擦除

擦除方式	说明
Chip 擦除	离线烧录时对芯片进行全片擦除
页面擦除	离线烧录时对芯片进行Sector 擦除(根据实际大小擦除)

- **接口电平**：PowerWriter 支持四种不同的电平匹配方式，见表：

接口电平	说明
1.8 V	设置接口电平为1.8V
3.3 V	设置接口电平为3.3V
5.0 V	设置接口电平为5.0V
外部输入	接口电平设置为根据外部参考电平

- **其他**：包含速度调节、选项字节更新方式、蜂鸣器选择方式：见下表3.1.4.1.1-6 所示：

功能	说明
编程速度调节	20MHZ (扩展)
10MHZ (默认)	
5MHZ	
2MHZ	
500KHZ	
200KHZ	
100KHZ	
50KHZ	
20KHZ	

功能	说明
10KHZ	
5KHZ	
选项字节更新方式	
烧录前无操作 -> 烧录后无操作	
烧录前无操作 -> 烧录后写入用户设置的Option Byte	
烧录前恢复出厂默认值->烧录后无操作	
烧录前恢复出厂->烧录写入用户设置的Option Byte	
蜂鸣器	是否开启蜂鸣器提示音

提示

- 较高的编程速度，对生产环境有更高的要求，在实际的测试中，PowerWriter 理论上任意速度调节，但是为了方便用户进行选择，我们进行了优化显示，提供常用的编程速度选择。
- 20 MHZ 的编程速度为扩展速度，如果实际生产环境支持这么快的时钟，也可以尝试切换到 20 MHZ 的时钟速度，来提交生产效率。更高的速度，将会在未来开发的 PowerWriter Pro/企业版上采用。
- 选项字节中的恢复出厂：可用于将芯片恢复出厂设置，而不写入数据，对于回收利用的芯片也同样友好。
- 芯片的分类格式以Flash 容量为准，例如 STM32F071xB，x表示可以代表任何的封装类型。
- PowerWriter 支持导出当前支持的所有芯片列表，在芯片选择对话框的右下角有一个导出xls 的按钮，见上图，点击此按钮后，将可以导出PowerWriter 支持的所有芯片列表。

注意

- 当选择芯片时，会自动同步设置到烧录器。
- 当烧录器自动连接时，会自动同步设置到烧录器。

- 应用设置之后，PowerWriter 将重新连接目标芯片，并同步当前芯片的选项字节到 PowerWriter 应用软件端。
- 为何PowerWriter 没有Bank擦除：
  - PowerWriter 对Bank自动化处理，无须用户选择Bank 1 还是Bank 2，无论是单Bank 还是双Bank，Memory Mapping 的地址为同一个地址，区别在于扇区表的大小不一样，而PowerWriter 支持完整的Sector类, 参考Program Memory 的 Flash 扇区映射表即可看到Bank 切换时，Sector 表的动态切换，PowerWriter 对于Bank 的擦除默认使用Sector擦除来替代。
- 选择外部输入时，需要外部接入VDD 到PowerWriter VEXT 引脚，以让PowerWriter 检测接口电平信号，来确定信号的逻辑电平。

烧写功能配置

烧写功能配置用于配置PowerWriter 附加功能，包括软件序列号的设置、离线烧录次数以及自动检测配置、信号输出控制、以及创芯工坊的签名功能。

序列号

PowerWriter 支持设置软件序列号，序列号长度为4字节，为了保持统一，序列号存储在Flash 中，序列号设置界面如图所示：

选项如下：

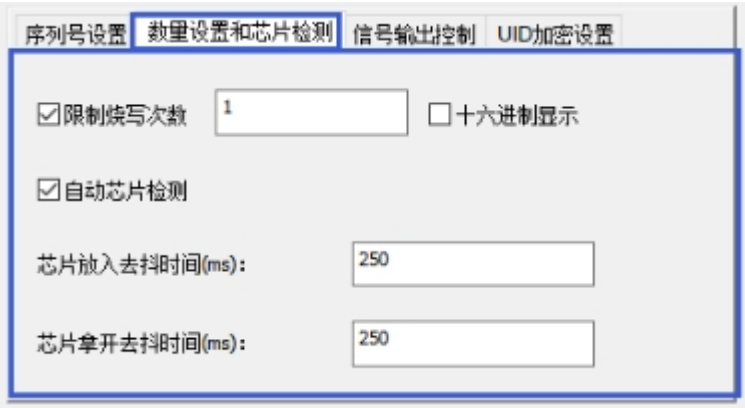
- 序列号初值：设置序列号的初始值，默认为0x00000000。
- 序列号增量：设置序列号的增量，默认为0x00000001。
- 序列号地址：设置序列号在Flash 中的地址，默认为当前芯片的最末尾地址 - 4 的位置，如图 128Kb 的芯片，地址在0x0801FFFC。
- 添加序列号功能：序列号功能的开关。
- 序列号10进制显示：切换序列号显示为10进制，默认为16进制。

- 序列号大端模式：序列号使用大端模式写入，默认为小端模式。

> **提示**
>
> 补充：大小端的资料请参考百度百科的描述，数据大小端百科。
>
> 参考视频：离线烧录时怎么设置序列号

数量与自检测

PowerWriter 支持离线量产，设置离线量产的烧录次数，默认长度为4字节长度，界面显示如图所示：



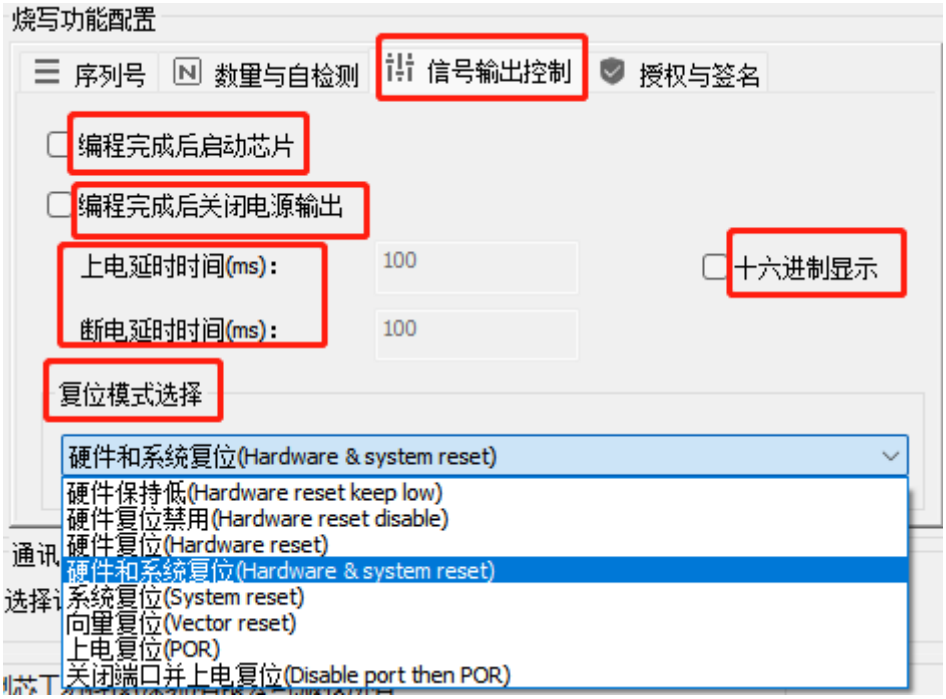
选项如下：

- 限制烧写次数：设置离线量产的次数，默认为0x00000001，同时可以切换是否需要开启或者是关闭离线次数限制。
- 自动检测芯片：当开始此功能后，PowerWriter 将自动检测芯片是否放入或者是芯片是否已拿走，无须再额外按烧录按键，或者是通过CTRL 给信号。
- 芯片放入去抖时间：当PowerWriter 检测到芯片时，等待一段时间在执行烧录的后续操作，防止因为抖动而导致数据烧写失败或者是错误，默认为250ms。
- 芯片拿开去抖时间：当PowerWriter 检测到芯片拿开时，等待一段时间再次判断芯片是否确定拿开，防止因为抖动而导致对同一个芯片执行重复烧录，默认为250ms。
- 16进制显示：切换显示模式，默认为16进制。

> **提示**
>
> 如果采用人工烧录遇到频繁的不良，或者重复烧录，可以尝试将去抖时间进行延长。
>
> 如果采用烧录机台记性烧录，或者是测试架，按压进行烧录，由于抖动的问题相对较少，可尝试将去抖时间进行改小。

信号输出控制

Power 提供烧录时灵活的编程完之后信号输出控制，如下图所示：



选项包含：

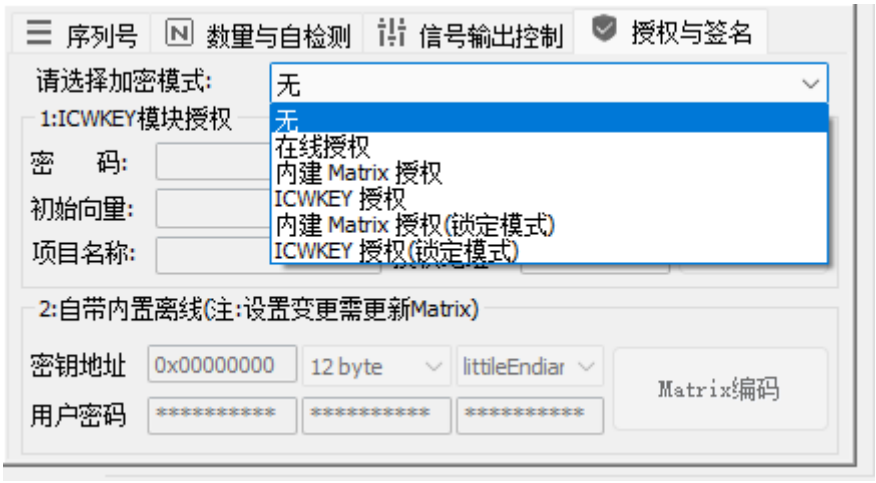
- **编程完成后启动芯片**：当编程完成之后，执行Reset and Run。
- **16进制显示**：16进制显示上电延时时间(ms)和断电延时时间(ms)。
- **编程完成后关闭电源输出**：当编程完成之后关闭电源输出，开启此选项后，每烧录完一颗芯片，都会断一下点，当下次启动烧录的时候，再根据上电延时时间和断电延时时间执行供电。
- **上电延时时间**：当给芯片上电之后多少ms 之后开始执行操作，默认为100ms。
- **断电延时时间**：当给芯片断电之后至少多少ms 之后开始执行下一次供电操作，默认为100ms。
- **复位模式选择**：当烧录完芯片后，将会从REST 引脚输出设定地输出信号类型，可以为如下几种模式：
  - 硬件保持低电平：RESET 引脚一直输出低电平
  - 硬件复位禁用：RESET 引脚切换为浮空输入模式,相当于禁用了此功能
  - 硬件复位：RESET引脚输出复位信号
  - 硬件复位和系统复位：RESET 引脚输出复位信号、同时执行内核复位(软复位)
  - 系统复位：执行内核复位(软复位)

- 向量复位：执行Vector Reset (复位到中断向量表ResetHandler)
- 上电复位：自行上电复位（Power on reset）
- 关闭端口并上电复位：执行上电复位，并关闭协议端口(高阻态)

> **注意**
>
> - (烧录完成后启动芯片、芯片自动检测) 和 烧录完关闭电源输出只能二选一，原因在于当关闭电源输出了，就无法启动芯片，也无法执行芯片自动检测。
> - 烧录过程中的复位操作由PowerWriter 自动控制，大部分芯片都可以执行under reset 复位功能，针对少部分芯片由于芯片本身的限制，需要硬件复位信号才可以复位，如果不清楚芯片是否可以执行软件复位，建议在烧录芯片时将 RESET 引脚也接上。

授权与签名

UID加密设置是PowerWriter 提供的签名功能，提供了随机的内置离线授权方法，一键生成动态的签名代码，同时提供在线授权服务器，支持第三方基于创芯工坊提供的授权服务器模板自建授权服务器，支持在线非对称授权方案，支持离线ICWKEY ECDSA 非对称授权算法，下面章节将详细介绍UID 授权算法的核心功能,授权设置界面如图所示：



> **提示**
>
> 参考视频：powerwriter授权前情提要

服务器在线授权方案

在线授权方案功能由创芯工坊官方提供，此时的烧录器内部不存储离线固件，而是将固件提交到创芯工坊的后台管理控制台以订单形式发布，客户再通过创芯工坊客户端实现远程量产烧录，烧录芯片时需要全程联网，从授权服务器获取授权数据。具体的在线授权使用方法，请参考License Server 章节/ 创芯工坊后台章节 / 以及在线授权方案的应用指导章节部分内容。本节做了一个概

览介绍，为了更加充分的理解在线授权方案，下面演示了PowerWriter的在线授权流程，如图所示：

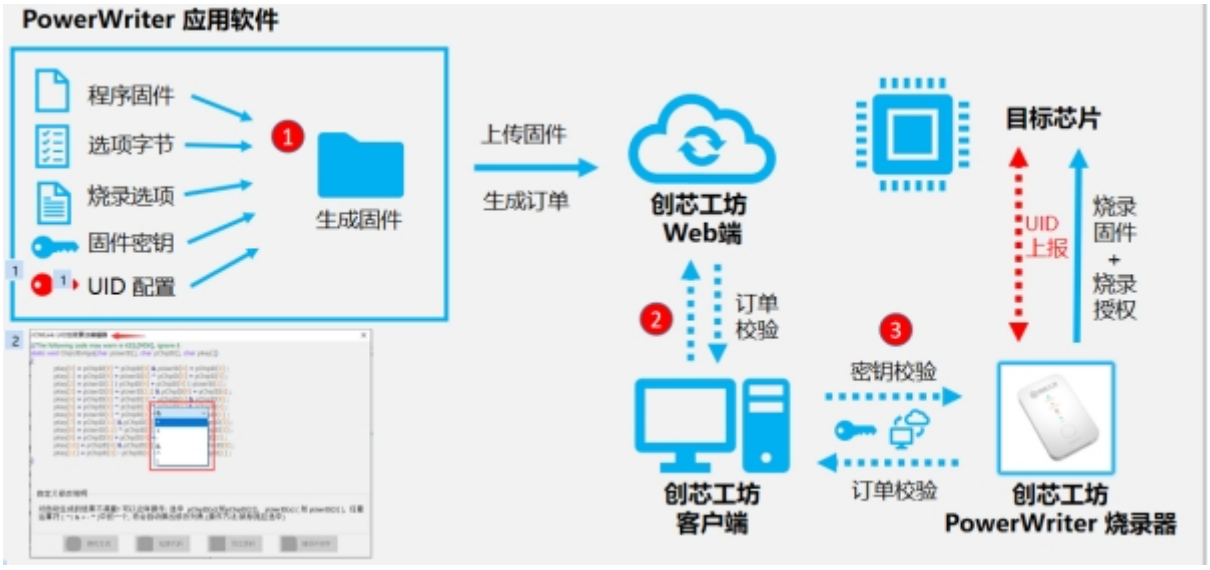
提示

托管授权方案：帮助中心 (icworkshop.com)

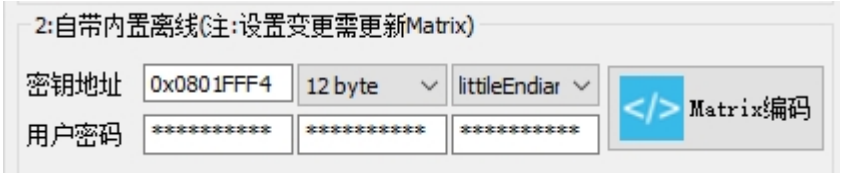
自建授权方案：帮助中心 (icworkshop.com)

参考视频：Powerwriter授权服务器如何使用

PowerWriter 内置离线授权



PowerWriter 内置了基于随机矩阵算法的 UID 离线授权方法，离线授权界面设置如图所示:



离线授权基础设置包含：

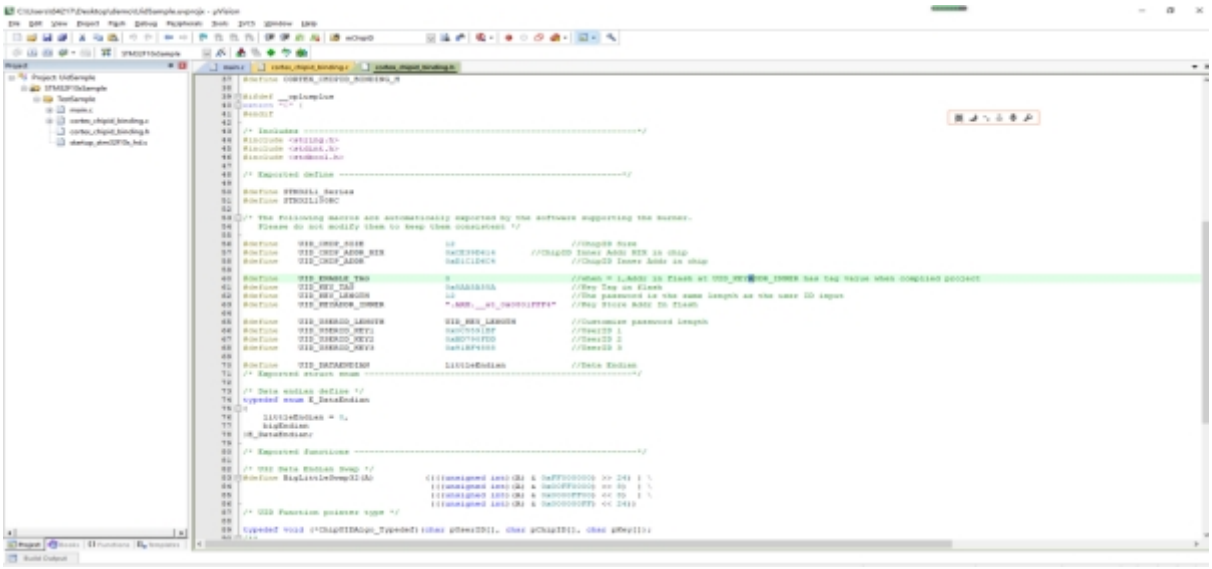
- **密钥地址**：密钥地址可以理解为存放授权信息的地址，他的默认地址设定为 芯片Flash 容量 - 12 的位置，上图是 STM32F071CB 的默认存储地址
- **用户密码长度**：这里可以填写用户设定密码长度，默认为12字节，可选4字节，8字节长度，未来这个功能将会升级为自动，匹配芯片UID长度。
- **数据存储模式**：数据存储模式分为小端模式和大端模式
- **用户密码(3组用户密码)**：根据设定可以设定最多三种用户密码
- **Matrix 编码**：Matrix 编码定义了用户可以编辑的离线授权的加密矩阵，如下图所示：

PowerWriter 离线授权随机矩阵主要包含三个区域，如下所示：

- UID 加密算法编辑器代码编辑预览：此区域提供了代码预览和代码快速编辑的功能，通过随机生成代码或者手动修改算法矩阵，都可以实时显示在此区域，如需手动编辑算法矩阵，请参考自定义UID 修改说明的操作流程。
- 随机生成：随机生成功能将会生成随机矩阵数组。
- 检查代码：验证当前矩阵数组的强度。
- 导出源码：设置好的信息导出为源码 (MDK 示例)。

名称	修改日期	类型	大小
cortex_chipid_binding.c	2020/3/30 9:32	C Source file	7 KB
cortex_chipid_binding.h	2020/3/30 9:32	C/C++ Header	5 KB
main.c	2020/3/30 9:32	C Source file	1 KB
startup_stm32f10x_hd.s	2020/3/30 9:32	Assembler Source	16 KB
UidSample.uvoptx	2020/3/30 9:32	UVOPTX 文件	9 KB
UidSample.uvprojx	2020/3/30 9:32	礦ision5 Project	15 KB

生成的项目文件见下图所示:



- **cortex_chipid_binding.c**：芯片UID 签名代码源文件，以下说明 cortex_chipid_binding 中的代码组成结构：

花指令：通过指令开启花指令，花指令的作用是在原始的矩阵中混入一些无关的代码，让反编译看到的结果和预想中有出入，增加分析难度，此方法对于芯片保护级别只有0和1 两级的芯片尤其有用，用过可以通过以下开关进行开启或者关闭花指令

```
#define ENABLE_JUNK_CODE    1   //Enable/Disable junk code
```

```
#define ENABLE_JUNK_CODE    1   //Enable/Disable junk code

#if ENABLE_JUNK_CODE
static char mJunkCodeVar[UID_KEY_LENGTH];   //junkCode for mix

//The following code may warn in KEIL(MDK), ignore it
static void ChipUIDAlgo(char pUserID[], char pChipID[], char pKey[])
{
    pKey[0] = pUserID[5] + pChipID[7] & pUserID[3] | pChipID[6] ;
    mJunkCodeVar[2] = pChipID[4] ^ pChipID[9] - pKey[11] ^ pKey[4] ;//junk
```

```
code,Your can remove or add your's junk code
    mJunkCodeVar[10] = mJunkCodeVar[11] + pUserID[0] * pUserID[1] & pChipID[8]
;//junk code,Your can remove or add your's junk code
    pKey[1] = pChipID[5] * pChipID[9] - pChipID[2] ^ pChipID[0] ;
    pKey[2] = pUserID[4] ^ pChipID[4] - pUserID[1] * pUserID[8] ;
    mJunkCodeVar[11] = pKey[3] | pChipID[10] | mJunkCodeVar[2] + pKey[2]
;//junk code,Your can remove or add your's junk code
    mJunkCodeVar[2] = pUserID[7] & pChipID[1] | mJunkCodeVar[9] + pKey[8]
;//junk code,Your can remove or add your's junk code
    mJunkCodeVar[10] = pUserID[9] & pUserID[7] * mJunkCodeVar[5] | pUserID[11]
;//junk code,Your can remove or add your's junk code
    pKey[3] = pUserID[10] & pUserID[7] + pUserID[2] | pChipID[1] ;
    mJunkCodeVar[3] = pChipID[1] | mJunkCodeVar[3] + pChipID[0] & pUserID[3]
;//junk code,Your can remove or add your's junk code
    mJunkCodeVar[2] = mJunkCodeVar[5] & pKey[8] | mJunkCodeVar[7] & pUserID[3]
;//junk code,Your can remove or add your's junk code
    mJunkCodeVar[8] = pKey[4] ^ pKey[8] * pChipID[8] * mJunkCodeVar[8] ;//junk
code,Your can remove or add your's junk code
    pKey[4] = pChipID[11] & pUserID[0] + pChipID[8] - pUserID[9] ;
    mJunkCodeVar[11] = pChipID[7] ^ mJunkCodeVar[6] + mJunkCodeVar[10] |
pUserID[6] ;//junk code,Your can remove or add your's junk code
    mJunkCodeVar[3] = pUserID[5] + mJunkCodeVar[4] - mJunkCodeVar[2] ^
pChipID[11] ;//junk code,Your can remove or add your's junk code
    mJunkCodeVar[2] = mJunkCodeVar[2] | pUserID[4] ^ mJunkCodeVar[9] + pKey[1]
;//junk code,Your can remove or add your's junk code
    pKey[5] = pUserID[6] * pChipID[10] ^ pChipID[3] | pUserID[11] ;
    mJunkCodeVar[5] = pKey[4] * pKey[4] + pChipID[3] ^ pUserID[1] ;//junk
code,Your can remove or add your's junk code
    pKey[6] = pUserID[0] - pChipID[1] + pUserID[10] | pUserID[2] ;
    mJunkCodeVar[10] = pChipID[11] * pKey[11] * pUserID[1] + mJunkCodeVar[6]
;//junk code,Your can remove or add your's junk code
    pKey[7] = pUserID[11] * pUserID[1] ^ pUserID[9] & pChipID[4] ;
    mJunkCodeVar[10] = pKey[4] - pKey[8] & pUserID[5] | pUserID[2] ;//junk
code,Your can remove or add your's junk code
    mJunkCodeVar[0] = pChipID[7] ^ pKey[3] - pChipID[4] ^ pChipID[0] ;//junk
code,Your can remove or add your's junk code
    mJunkCodeVar[1] = pChipID[5] ^ mJunkCodeVar[7] | pKey[7] - pKey[8] ;//junk
code,Your can remove or add your's junk code
    pKey[8] = pChipID[9] & pUserID[6] | pChipID[2] + pChipID[7] ;
    mJunkCodeVar[4] = mJunkCodeVar[5] | mJunkCodeVar[0] | pKey[4] + pChipID[7]
;//junk code,Your can remove or add your's junk code
    mJunkCodeVar[8] = pUserID[1] + pUserID[2] + pKey[10] - mJunkCodeVar[7]
;//junk code,Your can remove or add your's junk code
    pKey[9] = pUserID[7] - pUserID[4] * pChipID[6] ^ pChipID[11] ;
    pKey[10] = pChipID[3] ^ pUserID[3] * pChipID[8] - pUserID[8] ;
    mJunkCodeVar[0] = pUserID[6] | pUserID[4] - mJunkCodeVar[2] + pUserID[8]
;//junk code,Your can remove or add your's junk code
```

```
    mJunkCodeVar[11] = mJunkCodeVar[5] + pKey[0] ^ mJunkCodeVar[7] |
pChipID[11] ;//junk code,Your can remove or add your's junk code
    mJunkCodeVar[9] = pChipID[11] | pUserID[1] & pChipID[2] + pChipID[8]
;//junk code,Your can remove or add your's junk code
    pKey[11] = pUserID[5] & pChipID[10] | pChipID[5] + pChipID[0] ;
    mJunkCodeVar[0] = pKey[8] + pKey[0] ^ pUserID[9] + pKey[0] ;//junk
code,Your can remove or add your's junk code
    mJunkCodeVar[9] = mJunkCodeVar[0] & pUserID[11] + pChipID[8] + pKey[7]
;//junk code,Your can remove or add your's junk code
}

#else
//The following code may warn in KEIL(MDK), ignore it
static void ChipUIDAlgo(char pUserID[], char pChipID[], char pKey[])
{
    pKey[0] = pUserID[5] + pChipID[7] & pUserID[3] | pChipID[6] ;
    pKey[1] = pChipID[5] * pChipID[9] - pChipID[2] ^ pChipID[0] ;
    pKey[2] = pUserID[4] ^ pChipID[4] - pUserID[1] * pUserID[8] ;
    pKey[3] = pUserID[10] & pUserID[7] + pUserID[2] | pChipID[1] ;
    pKey[4] = pChipID[11] & pUserID[0] + pChipID[8] - pUserID[9] ;
    pKey[5] = pUserID[6] * pChipID[10] ^ pChipID[3] | pUserID[11] ;
    pKey[6] = pUserID[0] - pChipID[1] + pUserID[10] | pUserID[2] ;
    pKey[7] = pUserID[11] * pUserID[1] ^ pUserID[9] & pChipID[4] ;
    pKey[8] = pChipID[9] & pUserID[6] | pChipID[2] + pChipID[7] ;
    pKey[9] = pUserID[7] - pUserID[4] * pChipID[6] ^ pChipID[11] ;
    pKey[10] = pChipID[3] ^ pUserID[3] * pChipID[8] - pUserID[8] ;
    pKey[11] = pUserID[5] & pChipID[10] | pChipID[5] + pChipID[0] ;
}

#endif
```

U32DataEndiaSwap：32位数据大小端交换函数

```
/**
  * @brief  Data Endian Swap in uint32
  * @note   Please keep the burner and project consistent, do not modify it
  * @retval None
  */

void U32DataEndiaSwap(uint32_t * pBuffer, uint32_t size)
{
    int i;
    for(i = 0; i < size; i ++){
        pBuffer[i] = BigLittleSwap32(pBuffer[i]);
```

```
    }
}
```

导出的变量：包括签名信息等。

```
/* store full chipID */
static char mChipID[UID_CHIP_SIZE];

/* initial chip id */
static uint32_t mChipAddr;                  //global

/* calc key compare with  mKey */
static char mCalcKey[UID_KEY_LENGTH];


/* store full userID */
static char mUserID[] = {
#if (UID_USERID_LENGTH == 4)
    (UID_USERID_KEY1 & 0xff),((UID_USERID_KEY1>>8) & 0xff),
((UID_USERID_KEY1>>16) & 0xff),((UID_USERID_KEY1>>24) & 0xff),
#elif (UID_USERID_LENGTH == 8)
    (UID_USERID_KEY1 & 0xff),((UID_USERID_KEY1>>8) & 0xff),
((UID_USERID_KEY1>>16) & 0xff),((UID_USERID_KEY1>>24) & 0xff),
    (UID_USERID_KEY2 & 0xff),((UID_USERID_KEY2>>8) & 0xff),
((UID_USERID_KEY2>>16) & 0xff),((UID_USERID_KEY2>>24) & 0xff)
#else
    (UID_USERID_KEY1 & 0xff),((UID_USERID_KEY1>>8) & 0xff),
((UID_USERID_KEY1>>16) & 0xff),((UID_USERID_KEY1>>24) & 0xff),
    (UID_USERID_KEY2 & 0xff),((UID_USERID_KEY2>>8) & 0xff),
((UID_USERID_KEY2>>16) & 0xff),((UID_USERID_KEY2>>24) & 0xff),
    (UID_USERID_KEY3 & 0xff),((UID_USERID_KEY3>>8) & 0xff),
((UID_USERID_KEY3>>16) & 0xff),((UID_USERID_KEY3>>24) & 0xff)
#endif
};

/* Store the key calculated by the burner   */
#if UID_KEYADDR_PLACEHOLDER_EN
const static char mKey[UID_KEY_LENGTH]
__attribute__((section(UID_KEYADDR_INNER)))
#if UID_ENABLE_TAG
=
{

    #if (UID_KEY_LENGTH == 4)
        UID_KEY_TAG & 0xff,(UID_KEY_TAG >> 8) & 0xff,(UID_KEY_TAG >> 16) &
```

```
0xff,(UID_KEY_TAG >> 24) & 0xff
    #elif (UID_KEY_LENGTH == 8)
            UID_KEY_TAG & 0xff,(UID_KEY_TAG >> 8) & 0xff,(UID_KEY_TAG >> 16) &
0xff,(UID_KEY_TAG >> 24) & 0xff,
            UID_KEY_TAG & 0xff,(UID_KEY_TAG >> 8) & 0xff,(UID_KEY_TAG >> 16) &
0xff,(UID_KEY_TAG >> 24) & 0xff
    #else
            UID_KEY_TAG & 0xff,(UID_KEY_TAG >> 8) & 0xff,(UID_KEY_TAG >> 16) &
0xff,(UID_KEY_TAG >> 24) & 0xff,
            UID_KEY_TAG & 0xff,(UID_KEY_TAG >> 8) & 0xff,(UID_KEY_TAG >> 16) &
0xff,(UID_KEY_TAG >> 24) & 0xff,
            UID_KEY_TAG & 0xff,(UID_KEY_TAG >> 8) & 0xff,(UID_KEY_TAG >> 16) &
0xff,(UID_KEY_TAG >> 24) & 0xff
    #endif

}
#endif
;
#else
    //no license placeholder
#endif
```

mChipID： 用于存储芯片的UID，在启动的时候需要初始化一次，针对部分芯片UID 不连续的，在初始化的时候合并成连续的数据存储。

mChipAddr：用于存储混淆过的芯片UID 地址信息，混淆过后无法在二进制代码中找到芯片的ID地址信息。

mCalcKey：用于存储动态Matrix 编码的计算的结果，也就是存储正确密码。

mUserID：存储用户设定的密码信息。

mKey：存储PowerWriter 量产时写入的正确密码数据，在代码动态计算如果mCalcKey 和mKey不匹配，说明此代码被非法拷贝,从而实现拒绝运行的目的。

void ChipUIDInitial()：此函数用于初始化 UID 信息，为了隐藏芯片的真实UID ， PowerWriter在导出源代码时，会将UID_CHIP_ADDR 与 随机数 UID_CHIP_ADDR_MIX 进行异或运算，使其在编译好的Bin 文件 hex 中不会显示真实的ID 地址，然后在 ChipUIDInitial 中在动态初始化，将正确的UID 存储在内存中，此函数分为以下几个功能：

- 在代码运行时 计算UID 地址。
- 将UID 地址安全转移到另外一个变量。
- 检查UID 转移代码是否被非法篡改或者无效地址。

- 对于连续UID 地址和 非连续UID 地址拷贝到同一个UID数组中。

```
/**
  * @brief  Initialization chip ID. Called at initialization time
  * Note:   Please do not place ChipUIDAlgo_Calc in the same place as
ChipUIDAlgo_Check
  * @retval void
  */
void ChipUIDInitial()
{
    //restore real id Addr
    int i = 0;

        uint32_t chipIDAddr = 0;
    uint32_t mask = 0x0000000f;

    mChipAddr = UID_CHIP_ADDR;
    mChipAddr ^= UID_CHIP_ADDR_MIX;

    //Do not read directly from the address,you could use your's style

        for ( i = 0; i < sizeof(uint32_t) * 2; i++)
    {
        chipIDAddr |= mChipAddr & (mask << i * 4);
    }
    //If it has been modified by decompiling, exit
    if (chipIDAddr == 0 || chipIDAddr == 0xffffffff)
        return;
        //copy to array
#if ((defined STM32L1_Series) || (defined STM32L0_Series)) //offset
0x00,0x04,0x14
    for (i = 0; i < UID_CHIP_SIZE; i++)
    {
        mChipID[i] = *(uint8_t *)chipIDAddr;
        if (i != 7)
        {
            chipIDAddr++;
        }
        else
        {
            chipIDAddr += 0x0D;
        }
    }
#else
    for (i = 0; i < UID_CHIP_SIZE; i++)
```

```
    {
        mChipID[i] = *(uint8_t *)chipIDAddr;
        chipIDAddr++;
    }
#endif
}
```

ChipUIDAlgo_Calc: 用于计算UID Matrix 矩阵计算，这里并非直接调用 Matrix 矩阵算法：

```
/**
  * @brief  Calculate the key
  * Note:   Before calling. Must have called ChipUIDInitial
  * @retval pKey
  */

char * ChipUIDAlgo_Calc()
{
  ChipUIDAlgo_Typedef func = ChipUIDAlgo;
    (*func)(mUserID,mChipID,mCalcKey);
    if(UID_DATAENDIAN == bigEndian){
        U32DataEndiaSwap((uint32_t *)mCalcKey,sizeof(uint32_t));
    }
    return mCalcKey;
}
```

ChipUIDAlgo_Check：此函数用于验证动态计算的结果和PowerWriter 的结果是否完全一致。

```
/**
  * @brief  Check the key
  * Note:   Before calling. Must have called ChipUIDInitial
  * @retval if key is same then retun true,Otherwise, return false
  */

bool __inline ChipUIDAlgo_Check()
{
        const char *mLicAddr = 0;
    int mLicAddrI32 = 0;

        ChipUIDAlgo_Calc();

    mLicAddrI32 = UID_KEYADDR_INNER_MIXED;
    mLicAddrI32 ^= UID_KEYADDR_MIX;
    mLicAddr = (char *)(mLicAddrI32);
```

```
    return 0 == memcmp(mCalcKey, mLicAddr, UID_KEY_LENGTH);
}
```

cortex_chipid_binding.h

```
#define STM32F0_Series  //当前芯片的系列
#define STM32F071xB          //当前芯片的型号
#define UID_CHIP_SIZE        12  //当前芯片ID的长度，默认都是12字节
#define UID_CHIP_ADDR_MIX    0xE4E23C7A //用于隐藏芯片ID 地址在代码中的位置
#define UID_CHIP_ADDR       0xFB1DCBD6  //芯片ID在内存中的地址
#define UID_ENABLE_TAG      0  //当为1时,将会对有占位符的授权地址填充
**UID_KEY_TAG**
#define UID_KEY_TAG       0x5AA5A55A  //有占位符标记
#define UID_KEY_LENGTH    12         //用户输入的密码长度
#define UID_KEYADDR_INNER  ".ARM.__at_0x0801FFF4"  //授权数据再内存中的地址
#define UID_KEYADDR_PLACEHOLDER_EN  0    //是否在代码中开启授权数据占位符
#define UID_KEYADDR_MIX         0xF4A77775       //授权地址混淆随机数
#define UID_KEYADDR_INNER_MIXED  (0x0801FFF4 ^ UID_KEYADDR_MIX)  //授权数据存
储地址
#define UID_USERID_LENGTH        UID_KEY_LENGTH    //用户密码长度
#define UID_USERID_KEY1         0x097FA3EF        //用户密码 1
#define UID_USERID_KEY2         0x438A3070        //用户密码 2
#define UID_USERID_KEY3         0xC3F2AB5E        //用户密码 3
#define  UID_DATAENDIAN         littleEndian       //数端序定义
```

main.c：测试入口

```
#include "cortex_chipid_binding.h"

void SystemInit()
{
    //don't call ChipUIDInitial() at here
    //...
}

int main()
{
    //Initial Chip
    ChipUIDInitial();
```

```
    //user code
    //......

    //Check in your code
    bool ret = ChipUIDAlgo_Check();
    if(ret){
        //ok
    }else{
        //false
    }
}
```

- **编译并保存**：将设置好的PowerWriter 内置授权功能同步到PowerWriter 硬件。

> **注意**
>
> 如修改密钥地址、长度、端序、或者密码信息、需要**重新打开Matrix 编辑器**，重新点击编译并保存，避免出现信息不同步。
>
> 生成的Demo中实际需要使用的只有 cortex_chipid_binding.c 和 cortex_chipid_binding.h，其他文件为辅助文件。

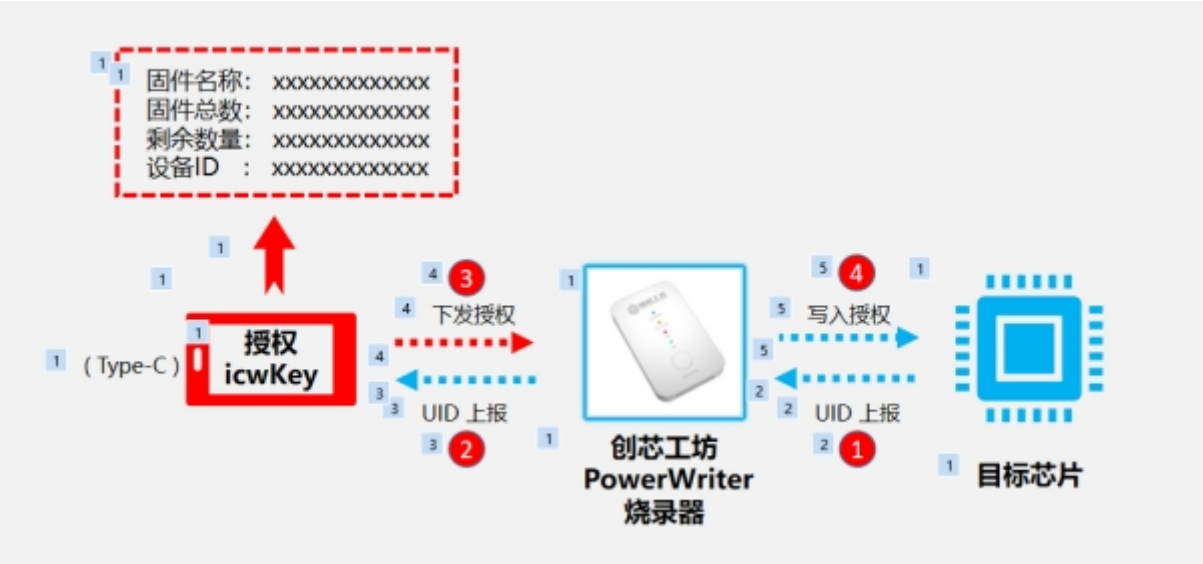
> **提示**
>
> 参考视频：powerwriter内置Matrix自定义授权
>
> PowerWriter内置Matrix集成到项目流程

ICWKEY 授权工具进行授权

PowerWriter 除了远程服务器在线授权、内置随机Matrix 随机授权算法授权之外，此外提供了更加专业的授权工具ICWKEY，通过ICWKEY 可以实现非对称ECDSA (ECC 非对称加密算法)授权,可以灵活将授权功能和PowerWriter 进行分离控制，并提供高强度的授权方案，针对对安全要求较高的产品，大批量产品授权，实现灵活的授权控制方案，ICWKEY 授权方法整个授权流程如图 所示。

PowerWriter & ICWKEY 授权方案需要用户基于ICWKEY 的SDK 和开发文档进行二次开发，ICWKEY 单独提供灵活的授权控制，以及完整的SDK 、文档和Sample Code。具体参考《ICWKEY 用户参考手册RM0002.pdf》。

本节将概要介绍ICWKEY 在PowerWriter 端配置，ICWKEY 配置工具和ICWKEY SDK 二次开发的基础流程。



ICWKEY 在PowerWriter 端配置：PowerWriter 的ICWKEY 配置功能如图 所示：

- **密码**：为了提供最高强度的通讯加密，PowerWriter 与 ICWKEY 的通讯采用AES128 CBC 模式加密，密码配置在PowerWriter 端随机生成，当密码框无焦点时认不显示密码，密码可以使用随机生成功能进行生成.不提供手动填入。
- **初始向量**：PowerWriter 和ICWKEY 的通讯除了 通信密码，同时提供一组初始向量，再通过创芯工坊的滚码算法，实现高强度的加密。
- **项目名称**：此名称将和 ICWKEY 的屏幕显示项目名称保持一致，默认格式为 :SafeLic_XXXXXXXX，如用户对默认的显示项目名称不满意。可以手动填写，默认最多为16个字节。
- **授权地址**：填写ICWKEY 在Flash 中的授权地址，PowerWriter 将根据用户填写的此地址，写入授权信息到 目标芯片的Flash 地址中。开发方法请参考《ICWKEY 用户参考手册 RM0002.PDF》，此地址的默认值为芯片Flash 的末尾 - 0x80 的位置。在基于ICWKEY 开发完成项目后基于MDK 导出的Mapping 信息找到授权的地址，在此处填写当前正确的授权地址信息。
- **随机生成**：点击此按钮PowerWriter 将随机生成 **密码、初始向量、项目代码。**
- **保存配置**：当用户完成设置后，点击保存，此时ICWKEY 配置信息将会保存到缓冲区。(注: 此时点保存并非同步到PowerWriter，同步到烧录器的方法统一通过离线加载并保存按钮菜

单进行操作)。

ICWKEY 端配置：ICWKEY 提供了灵活的权限控制与授权控制，应用软件界面如下图所示。

- 项目名称：同PowerWriter 端的项目名称,请保持一致。
- 密码：同PowerWriter 端的密码，请保持一致， 默认密码为30313233343536373839414243444546等同于字符串的“0123456789ABCDEF”。
- 向量：同PowerWriter 端的向量，请保持一致。
- 选择设备：连上ICWKEY之后刷新设备即可看到设备。
- 刷新设备：连上ICWKEY 之后刷新下设备。
- 连接设备：连接ICWKEY。

设备配置:

- 新项目名称：ICWKEY 默认的项目名称为ICWorkShop，需要改成和PowerWriter 端一致。
- 新密码：初次使用时，需要给ICWKEY配置新的通讯密码，否则进行授权时将会收到错误提示。
- 新向量：初次使用时，需要给ICWKEY 配置新的通讯向量，否则授权是将会收到错误提示。
- 设备序列号：设定ICWKEY 的序列号。
- UID 最小值：开启限制UID 烧录范围时有效，限定UID 的授权范围。
- UID 最大值：开启限制UID 收录范围时有效，限定UID 的授权范围。

- 授权数量：可以授权的芯片数量。
- 可配置次数：默认ICWKEY 可以供10个项目使用，在首次配置时设置，后面不能再更改。
- 实际剩余次数：ICWKEY 还剩多少次可以用于配置。
- 使能授权工具：可以开启或者关闭授权功能。
- 允许固件升级：可以开启或者关闭固件升级功能。
- 限制UID 授权范围：开启或者关闭UID 范围限制。
- 允许写入UID 算法：开启或者关闭UID算法写入。
- 开启日志记录：开启或者关闭ICWKEY 的日志记录功能。

UID算法：

- 向量矩阵加密：功能同PowerWriter 端内置的Matrix 授权功能。
- ECDSA 签名算法：ICWKEY 和PowerWriter 内置了,非对称加密算法ECC，采用高性能的mbedtls 技术框架，进行高度优化。产生签名的时间约为 300ms，验签的时间大约为700ms。
- 用户自定义：此模式下需要获得创芯工坊的授权，针对大客户，创芯工坊将开放ICWKEY配置软件、协议文档、ICWKEY 硬件端源码、以及相关SDK 资料。

日志信息：

ICWKEY 提供读取授权日志的功能，用于读取已授权的次数信息。

测试授权：

ICWKEY 提供测试授权的功能，用于测试授权的性能。

日志栏：

通过日志栏，可以实时看到ICWKEY 的日志信息，和PowerWriter 一样，提供分色显示的日志浏览，方便用户进行查阅。

关于锁定模式的补充

在授权与签名模式中，有锁定模式，锁定模式的特性如下：

- 在第一次启动项目时，所有的设置均为可编辑的模式。
- 在第二次加载项目时，所有的设置均为锁定模式，不可编辑，也不可查看。

> **提示**

> 锁定模式配合UID签名信息导入和导出功能可以实现签名功能和主项目权限分离，主要用于合作项目开发，库开发者管控权限，库使用者开发实际项目。

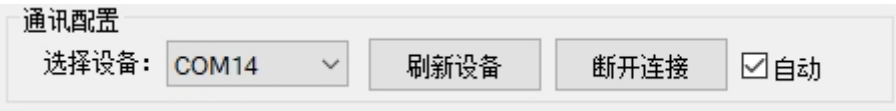
通用MCU 高级软件保护算法

> **提示**
>
> 内测中，暂未开放。

> **提示**
>
> 参考视频：powerwriter + ICWKEY授权工具介绍
>
> powerwriter + ICWKEY授权的设置
>
> ICWKEY授权工具配置软件讲解
>
> powerwriter + ICWKEY源码讲解
>
> ICWKEY授权效果演示

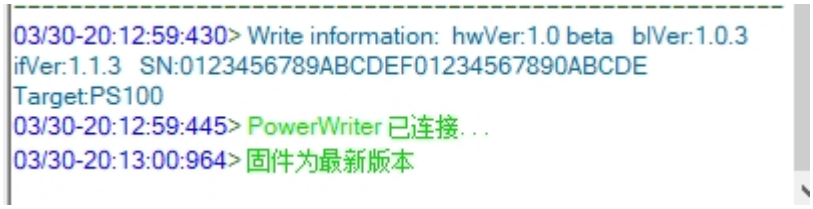
通信配置

PowerWriter 的通讯配置界面如下图所示：



- 选择设备列表：此列表显示当前扫描到的PowerWriter 硬件设备。
- 刷新设备：用户可以手动点击此按钮来刷新设备列表。
- 断开连接：用户可以手动点击断开连接，断开与PowerWriter 的通讯。
- 自动：默认勾选，将自动管理PowerWriter 设备的连接和断开，当插入PowerWriter 时，尝试自动连接到PowerWriter，当拔掉PowerWriter 时，将自动断开连接。

PowerWriter首次连接成功时，将会自动执行固件检查，获取并显示PowerWriter 信息，如图所示：



日志窗口

PowerWriter 提供详细的日志操作列表，并使用不同颜色显示，日志窗口界面如图所示：

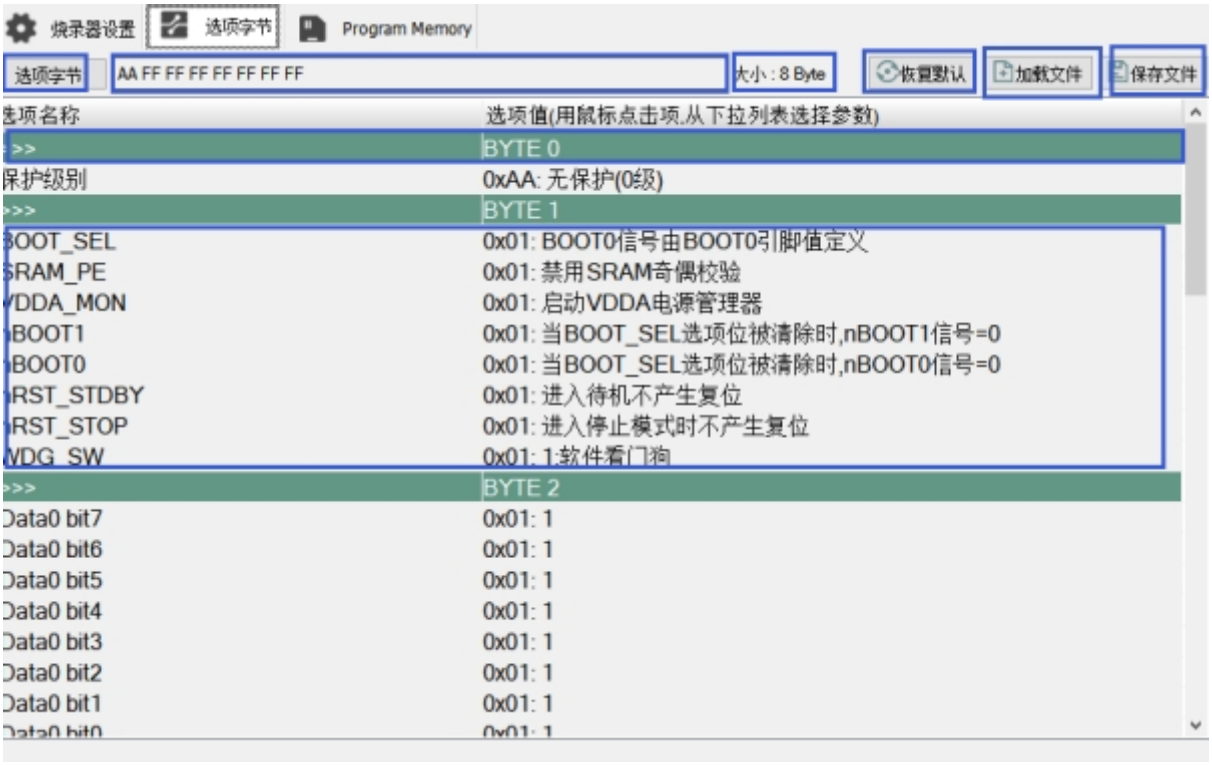
- 黑色：代表固定信息，如显示创芯工坊的官方联系方式。
- 浅蓝色：代表一般性信息，如显示PowerWriter 的硬件信息。
- 绿色：代表操作成功信息，比如烧录成功。
- 浅红色：代表警告信息，或者是需要注意的信息。
- 红色：代表错误信息，看到红色信息时，需要特别流程输出的内容，只有当遇到比较严重的错误才会出现此信息。

PowerWriter 的日志功能额外提供一个右键菜单，见图所示，提供清除日志，和拷贝日志功能：

- 清除日志：点击此菜单将清除PowerWriter 的所有操作日志。
- 复制日志到剪贴板：将日志以富文本格式拷贝到剪贴板，用户可以粘贴到记事本,或者是粘贴到Doc文档，当粘贴到Doc 文档时，将保留日志的原始排版信息。见图所示：

选项字节Tab 标签页

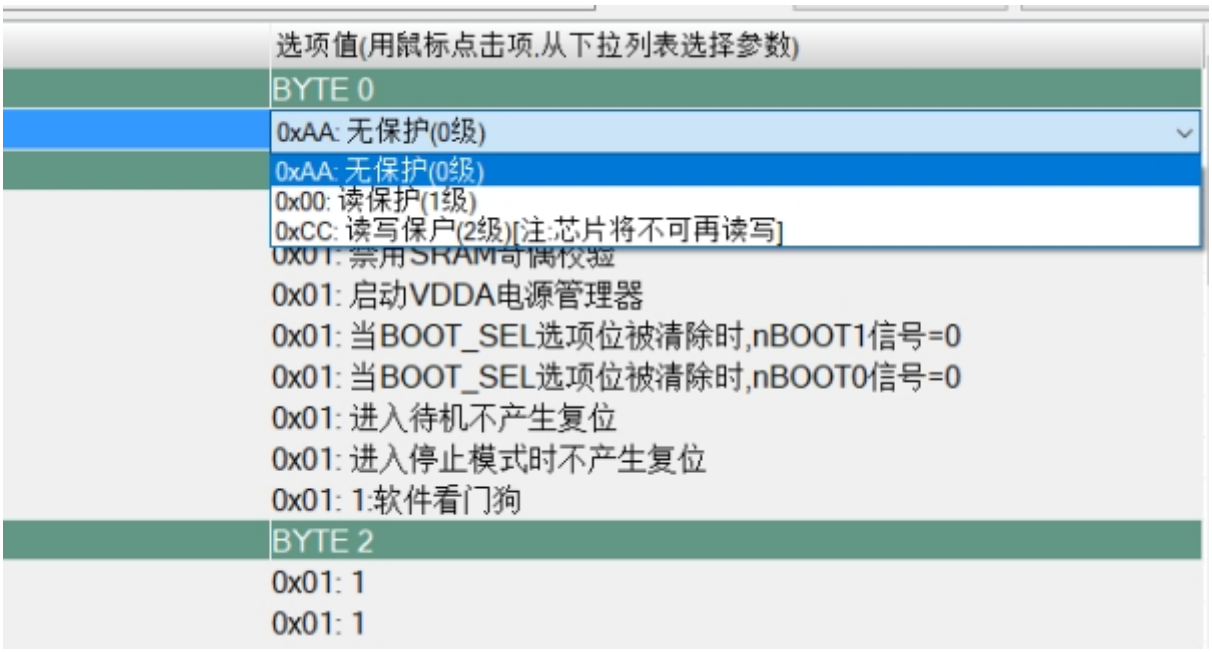
PowerWriter 提供完整的选项字节定义，所有数据均来自官方提供的用户手册，选项字节窗口的显示如图所示：



PowerWriter 提供每一个芯片的选项字节映射表，PowerWriter 的选项字节主要分为：

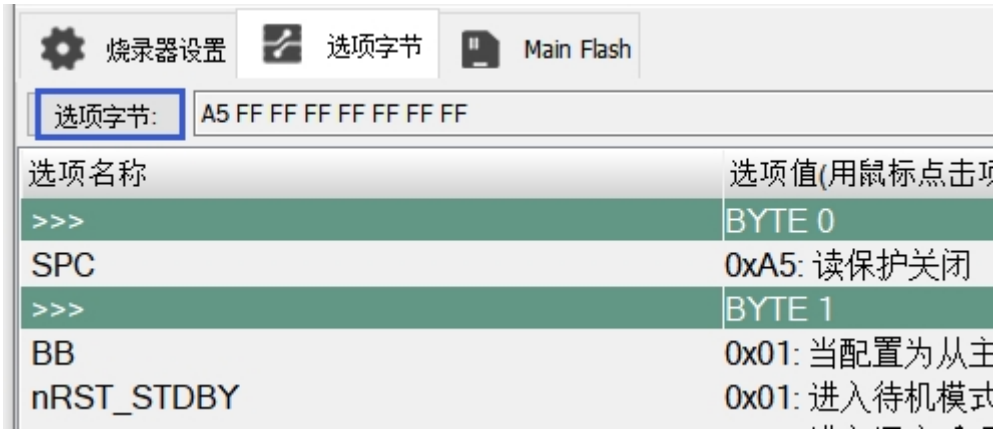
- **选项字节预览区：**选项字节预览区实时显示当前的选项字节设定，极大的方便开发者进行操作，为了确保用户的设定不会输入非法数据，PowerWriter 暂不提直接修改选项字节的功能。
- **选项长度显示：**选项长度显示了当前选项芯片选项字节的字节长度，如 大小：8 Byte 所示。
- **恢复默认:** 在调整选项字节过程中可以点击恢复默认将会恢复默认选项字节。
- **选项字节保存到文件：**选项可以保存到文件，支持hex，s19，bin 格式。
- **选项字节从文件加载：**选项字节也可以从已保存的文件中加载，支持hex，s19，bin格式。

- **选项字节Byte指示器：**Byte只是器如>>> BYTE 0所示，Byte指示器直观地将选项字节序按照指示器分开，并用底色显示。
- **选项字节设置项：**选项字节设置项位于选项字节窗口的左侧，显示名称和Data Sheet 上保持一致。
- **选项字节设置值：**选项字节设置的值，通过鼠标点击设置项的值，将会自动弹出选项字节设置列表，列表中显示当前选项支持设置的值，如图所示。



附加功能：

选项字节的编辑方式除了提供单项值的设定之外，还提供缓冲区直接编辑功能，点击选项字节按钮，如下所示，将会进入选项字节缓冲区编辑界面：



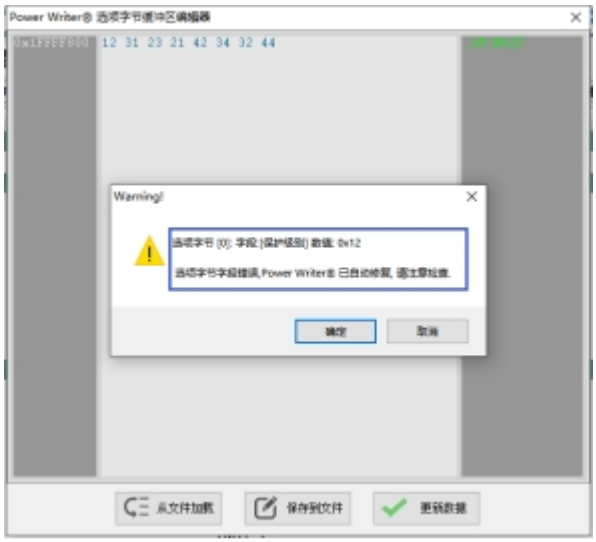
在选项字节缓冲区编辑界面可：

- 直接编辑缓冲区中的数值，编辑完成后，然后更新到选项字节缓冲区中。
- 从文件中加载选项字节，支持hex bin s19 格式文件。
- 将选项字节缓冲区保存到文件。

- 将修改后的选项字节应用到缓冲区。

注意

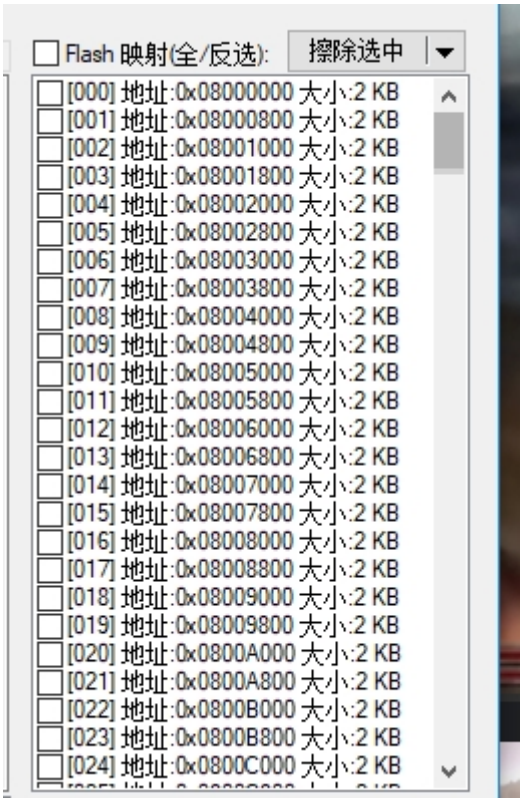
直接编辑选项字节缓冲区，提供了编辑选项字节的快速方法，但是容易导致编辑的值和芯片设定不一致，所以在更新数据时，可能会提示如下的提示框，PowerWriter 将自动对非法的选项字节进行修正：



附加补充：

- **选项字节的读取**：通过菜单->执行->读取选项字节，或者在选项字节窗口点击读取当前页工具栏按钮执行读取操作。
- **选项字节的写入**：通过菜单->执行->写入选项字节，或者在选项字节窗口点击写入当前页工具栏按钮执行写入操作。
- **选项字节的保存**：选项字节窗口点击**保存文件**，或者在选项字节窗口点击保存当前页工具栏按钮执行选项字节的保存操作。
- **选项字节恢复默认**：选项字节在多次编辑后可能存在混乱的情况，如果想回到默认的选项，可以通过点击恢复默认按钮，对选项字节进行重置。
- **选项字节的加载**：选项字节窗口点击**加载文件**，或者在选项字节窗口点击加载当前页工具栏按钮执行选项字节的加载操作。
- **选项字节的修改**：用鼠标点击设置项的选项值从下来列表进行操作。
- **选项字节的双Bank**：PowerWriter 支持自动识别双Bank，当前芯片为STM32G483xE,如图所示，可以看到双Bank 模式下的 Sector 分区表为每一个扇区为2KB，当切换为4K，再核对芯片手册RM0440.PDF,可以看出和芯片手册保持一致。

选项名称	选项值(用鼠标点击选项,从下拉列表选择参数)
>>>	BYTE 0
保护级别	0xAA: 无保护(0级)
>>>	BYTE 1
nRST_SHDW	0x01: 进入关机模式时不产生复位
nRST_STDBY	0x01: 进入待机不产生复位
nRST_STOP	0x01: 进入停止模式时不产生复位
BOR_LEV	0x00: BOR 0级. 复位电平阈值 1.7 V
>>>	BYTE 2
nBOOT1	0x01: nBOOT1设置为高
DBANK	0x01: 双bank模式与64位数据
BFB2	0x00: 双Bank引导禁用
WWDG_SW	0x01: 软件看门狗
IWDG_STBY	0x01: IWDG计数器在待机状态下工作


Flash 映射(全/反选):
擦除选中
[000] 地址:0x08000000 大小:2 KB
[001] 地址:0x08000800 大小:2 KB
[002] 地址:0x08001000 大小:2 KB
[003] 地址:0x08001800 大小:2 KB
[004] 地址:0x08002000 大小:2 KB
[005] 地址:0x08002800 大小:2 KB
[006] 地址:0x08003000 大小:2 KB
[007] 地址:0x08003800 大小:2 KB
[008] 地址:0x08004000 大小:2 KB
[009] 地址:0x08004800 大小:2 KB
[010] 地址:0x08005000 大小:2 KB
[011] 地址:0x08005800 大小:2 KB
[012] 地址:0x08006000 大小:2 KB
[013] 地址:0x08006800 大小:2 KB
[014] 地址:0x08007000 大小:2 KB
[015] 地址:0x08007800 大小:2 KB
[016] 地址:0x08008000 大小:2 KB
[017] 地址:0x08008800 大小:2 KB
[018] 地址:0x08009000 大小:2 KB
[019] 地址:0x08009800 大小:2 KB
[020] 地址:0x0800A000 大小:2 KB
[021] 地址:0x0800A800 大小:2 KB
[022] 地址:0x0800B000 大小:2 KB
[023] 地址:0x0800B800 大小:2 KB
[024] 地址:0x0800C000 大小:2 KB

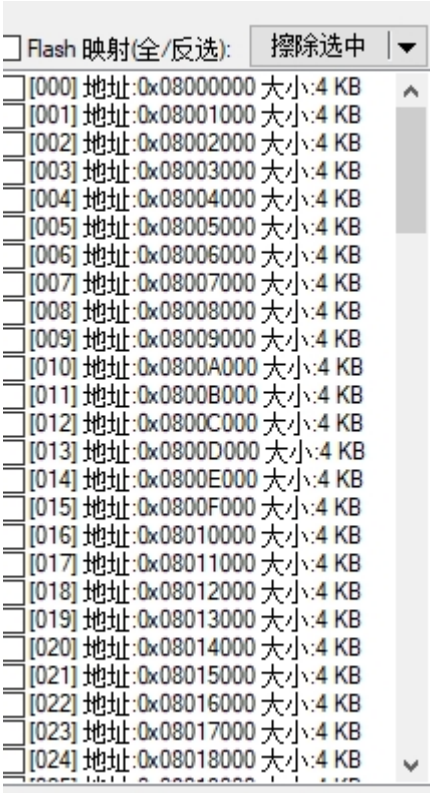

Flash 映射(全/反选):
擦除选中
[000] 地址:0x08000000 大小:4 KB
[001] 地址:0x08001000 大小:4 KB
[002] 地址:0x08002000 大小:4 KB
[003] 地址:0x08003000 大小:4 KB
[004] 地址:0x08004000 大小:4 KB
[005] 地址:0x08005000 大小:4 KB
[006] 地址:0x08006000 大小:4 KB
[007] 地址:0x08007000 大小:4 KB
[008] 地址:0x08008000 大小:4 KB
[009] 地址:0x08009000 大小:4 KB
[010] 地址:0x0800A000 大小:4 KB
[011] 地址:0x0800B000 大小:4 KB
[012] 地址:0x0800C000 大小:4 KB
[013] 地址:0x0800D000 大小:4 KB
[014] 地址:0x0800E000 大小:4 KB
[015] 地址:0x0800F000 大小:4 KB
[016] 地址:0x08010000 大小:4 KB
[017] 地址:0x08011000 大小:4 KB
[018] 地址:0x08012000 大小:4 KB
[019] 地址:0x08013000 大小:4 KB
[020] 地址:0x08014000 大小:4 KB
[021] 地址:0x08015000 大小:4 KB
[022] 地址:0x08016000 大小:4 KB
[023] 地址:0x08017000 大小:4 KB
[024] 地址:0x08018000 大小:4 KB

Table 7. Flash module - 512/256/128 KB dual bank organization (64 bits read width)

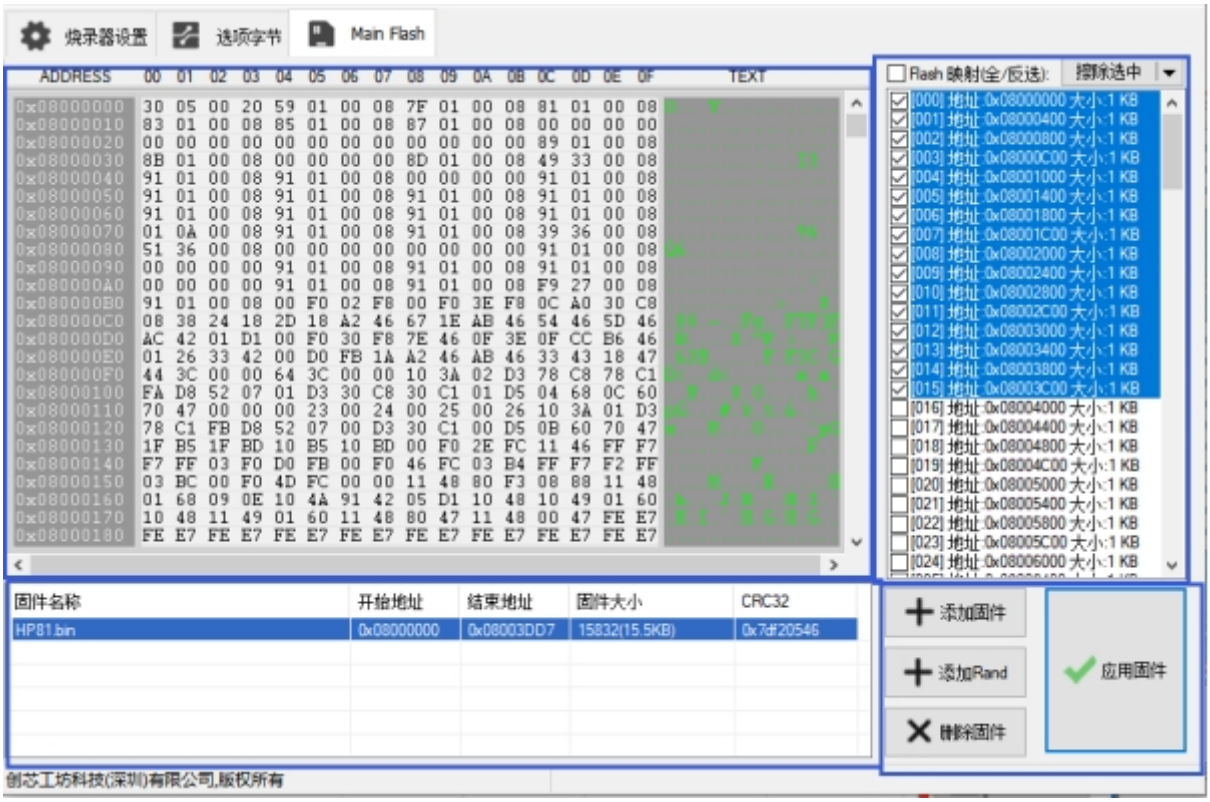
Flash area		Flash memory addresses	Size (bytes)	Name
Main memory (512/256/128 KB)	Bank 1 (256/128/64 KB)	0x0800 0000 - 0x0800 07FF	2 K	Page 0
		0x0800 0800 - 0x0800 0FFF	2 K	Page 1
		0x0800 1000 - 0x0800 17FF	2 K	Page 2
		0x0800 1800 - 0x0800 1FFF	2 K	Page 3
		- - - -	- - - -	- - - -
		0x0803 F000 - 0x0803 FFFF	2 K	Page 127
	Bank 2 (256/128/64 KB)	0x0804 0000 - 0x0804 07FF	2 K	Page 0
		0x0804 0800 - 0x0804 0FFF	2 K	Page 1
		0x0804 1000 - 0x0804 17FF	2 K	Page 2
		0x0804 1800 - 0x0804 1FFF	2 K	Page 3
		- - - -	- - - -	- - - -
		0x0807 F000 - 0x0807 FFFF	2 K	Page 127

Table 8. Flash module - 512/256/128 KB single bank organization (128 bits read width)

Flash area	Flash memory addresses	Size (bytes)	Name
Main memory (512/256/128 KB)	0x0800 0000 - 0x0800 0FFF	4 K	Page 0
	0x0800 1000 - 0x0800 1FFF	4 K	Page 1
	0x0800 2000 - 0x0800 2FFF	4 K	Page 2
	-	-	-
	-	-	-
	-	-	-
	-	-	-
	-	-	-
	-	-	-
	-	-	-
	0x0807 F000 0x0807 FFFF	4 K	Page 127

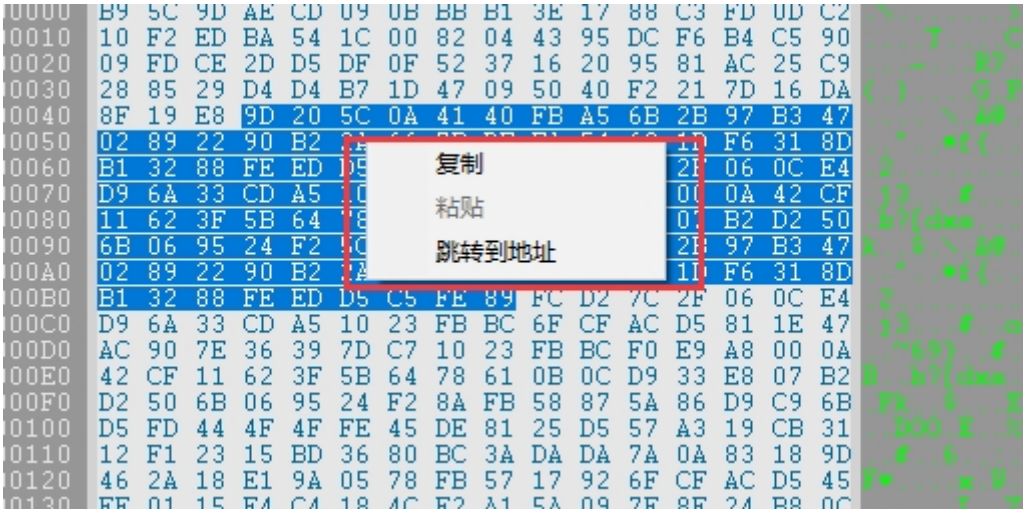
Program Memory Tab 标签页

PowerWriter 提供完整的Program Memory 页面，完整的显示当前芯片的整个Flash Memory 布局，如图所示：



PowerWriter Memory 标签页分为以下几大区域：

- **Hex 数据显示编辑区**：此区域显示将要写入到芯片的数据预览，或者是实时显示从芯片读取到数据，数据显示区左侧地址栏和芯片的实际地址信息对应。右侧实时显示数据的文本模式。此区域提供快速编辑功能，通过定位光标到缓冲区，输入键盘按键即可输入任意的数据到数据缓冲区。此外提供CTRL+C和CTRL+V 快速复制粘贴功能，以及提供菜单项，如图所示。



- **复制**：将选中的区段复制到剪贴板
- **粘贴**：将选中的区段粘贴到当前光标位置
- **跳转到地址**：PowerWriter 提供快速定位地址的功能，可以无须通过右侧滚动栏寻找数据地址，如图图3.1.4.3-3所示，设置好地址后，点击确定按钮，将自动跳转到地址显示。

- **分段固件信息显示区**：此区域实时显示PowerWriter 添加的分段固件，分段固件显示区域，支持多种格式的分段固件：文件、内存块、Mark标记块(如随机标记块)等多种混合格式。并且在分段固件的支持上，PowerWriter 没有任何的限制,此区域将显示：

固件名称：显示添加的固件名称

开始地址：显示当前分段固件的开始地址

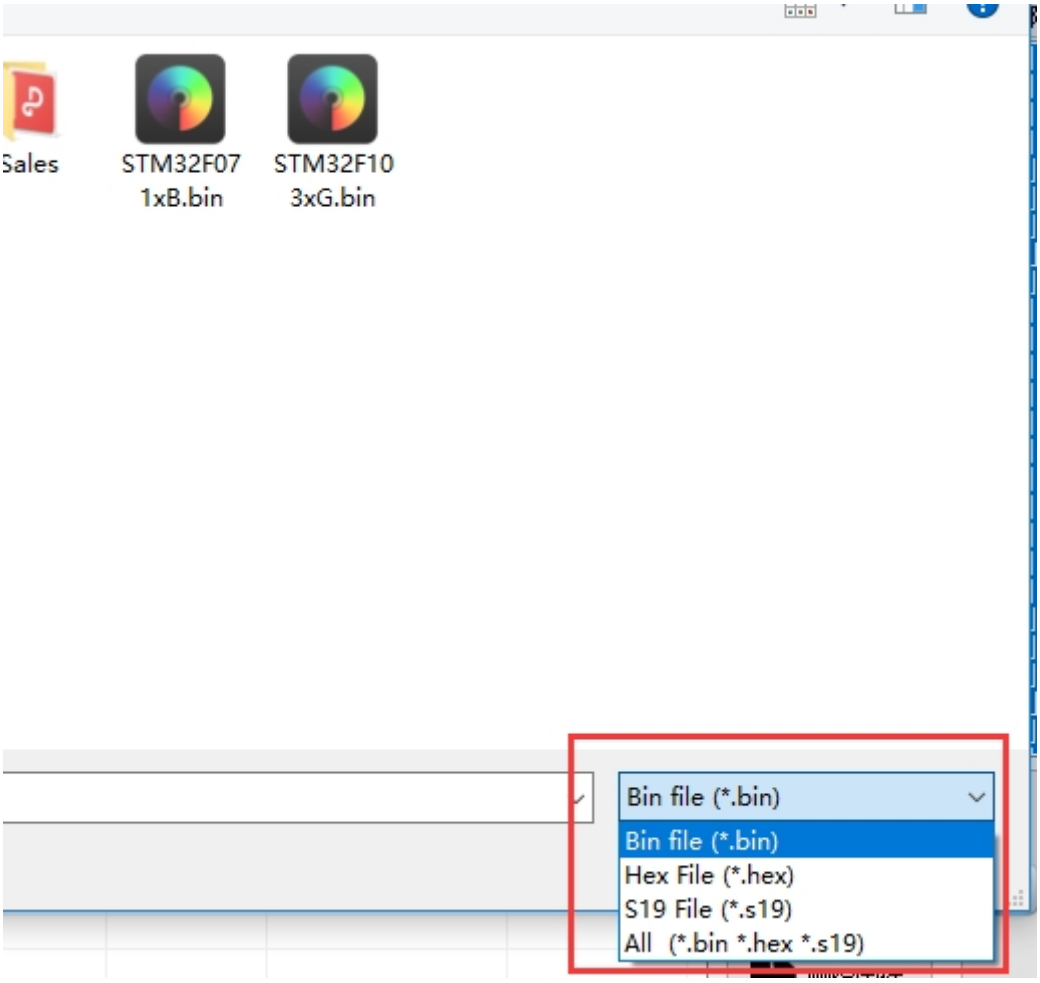
结束地址：显示当前分段固件的结束地址

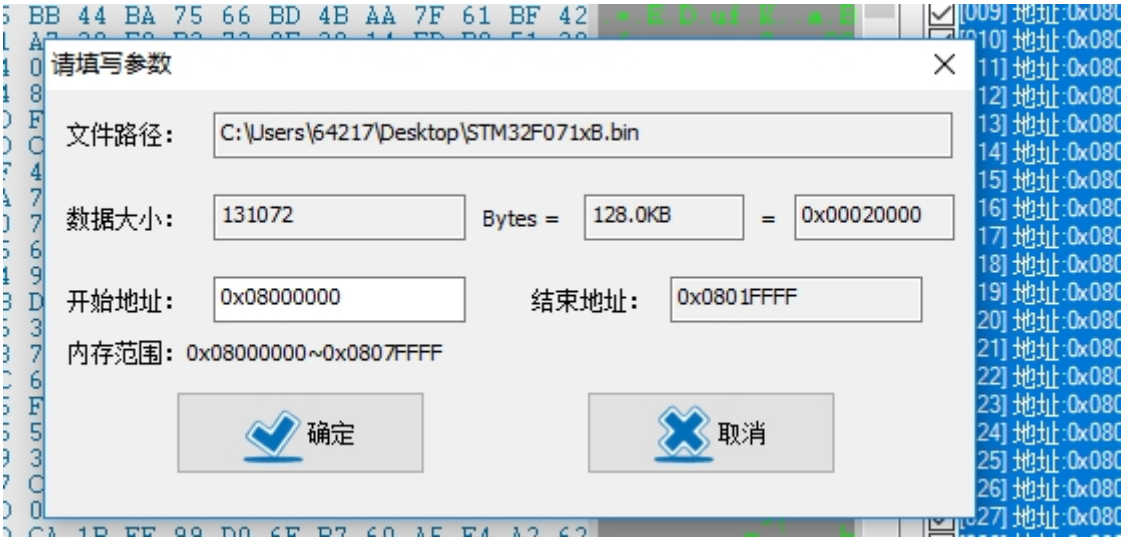
固件大小：显示当前分段固件的固件大小

CRC32: 显示当前分段固件的CRC32

分段固件操作区：分段固件的操作区域，包含以下几个功能：

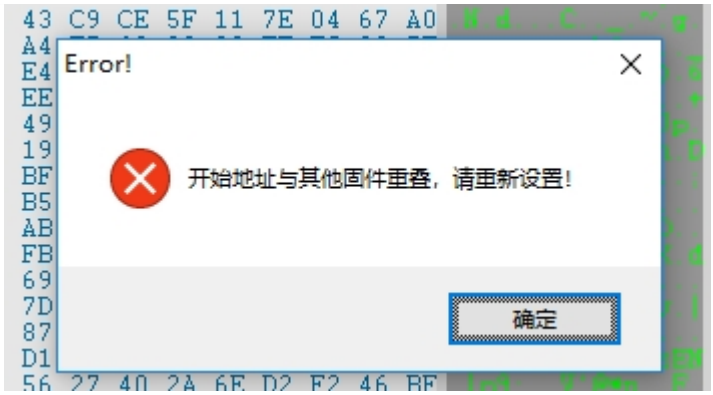
添加固件：PowerWriter 支持多种格式的固件，S19，Hex ，Bin格式，其他格式在后续的升级维护中，可能也会进行支持，比如elf，axf等格式，格式支持如图3.1.4.3-4所示，选择固件后点击**打开**按钮，将看到固件的设置对话框，如图所示，固件设置框，将显示当前固件的文件路径、数据大小、数据大小的KB数，以及16进制显示形式，同时显示固件的起始地址(Hex,S19 自动识别起始地址，bin格式默认显示Program Memory 的起始地址)，以及结束地址信息，核查起始地址无误后点击确定按钮，将在固件列表中看到添加的固件。



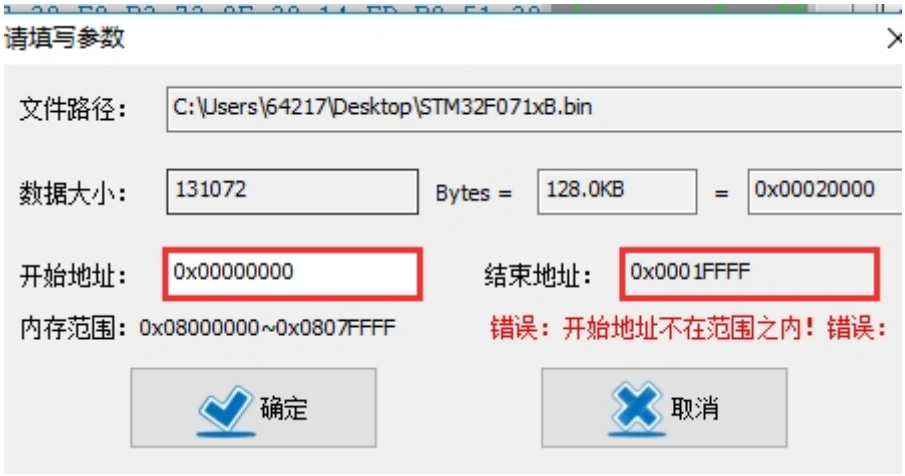


特别注意

1. 固件不能出现重叠，否则将会错误对话框，如图所示：



2. 固件不能超出Program Memory 空间，包括起始地址和结束地址都必须在Program Memory 所在的范围内。



3. 由于部分芯片写入时必须对齐到对应的字节数，PowerWriter 会尝试将用户固件回读以便补齐到需要对齐的基地址，此功能大部分时候都是可用的，但是针对部分芯片无法执行多次操作，此功能是一个实验性功能，用户在生成固件时，确保对齐固件到对应的首地址，比如芯片是4字节写，比如将首地址对齐到4字节的整数倍。如果是32字节写，必须对齐到32字节的整数倍，对齐操作依然是创芯工坊官方推介的操作，关于Flash 对齐操作的表格请参考附录 [PowerWriter Flash 起始地址对齐表](#_PowerWriter Flash 起始地址对其表)

添加Rand：用户除了可以无限制的添加应用固件之外，还可以在Flash 中添加随机数据块，点击添加Rand 按钮后将弹出如下的界面，如图所示：

在使用随机块功能时，PowerWriter 将自动搜索所有的内存数据，枚举出连续的空数据区块，供用户进行编辑调整。

调整随机快的首地址：点击起始地址执行编辑，对其到最小单位。

调整随机快的结束地址：点击结束地址执行编辑。

编辑完成后，即可选中需要添加的随机快，如图所示：

添加随机快之后，即可在固件列表中看到随机快数据区块，如图所示：

固件名称	开始地址	结束地址	固件大小	CRC32
Random memory	0x08001000	0x08001FFF	4096(4.00KB)	0xe7a915a0
Random memory	0x08002000	0x08002FFF	4096(4.00KB)	0x74e063f6
Random memory	0x08003000	0x08003FFF	4096(4.00KB)	0x58dcdbbf
Random memory	0x08006000	0x0800607F	128(0.125KB)	0x63dac59d
Random memory	0x080060A8	0x08006FFF	3928(3.84KB)	0x262eb5eb

⚠ 注意

随机数据块不能和其他区块进行重叠，否则将无法添加进入，对应区块将会以红色提示信息提示。

随机块必须基于对应的芯片地址对齐到最小操作单位,在窗口上有提示地址对齐

随机块的最小长度为 8 个字节（以PWRS 开头，以PWRE 结尾）

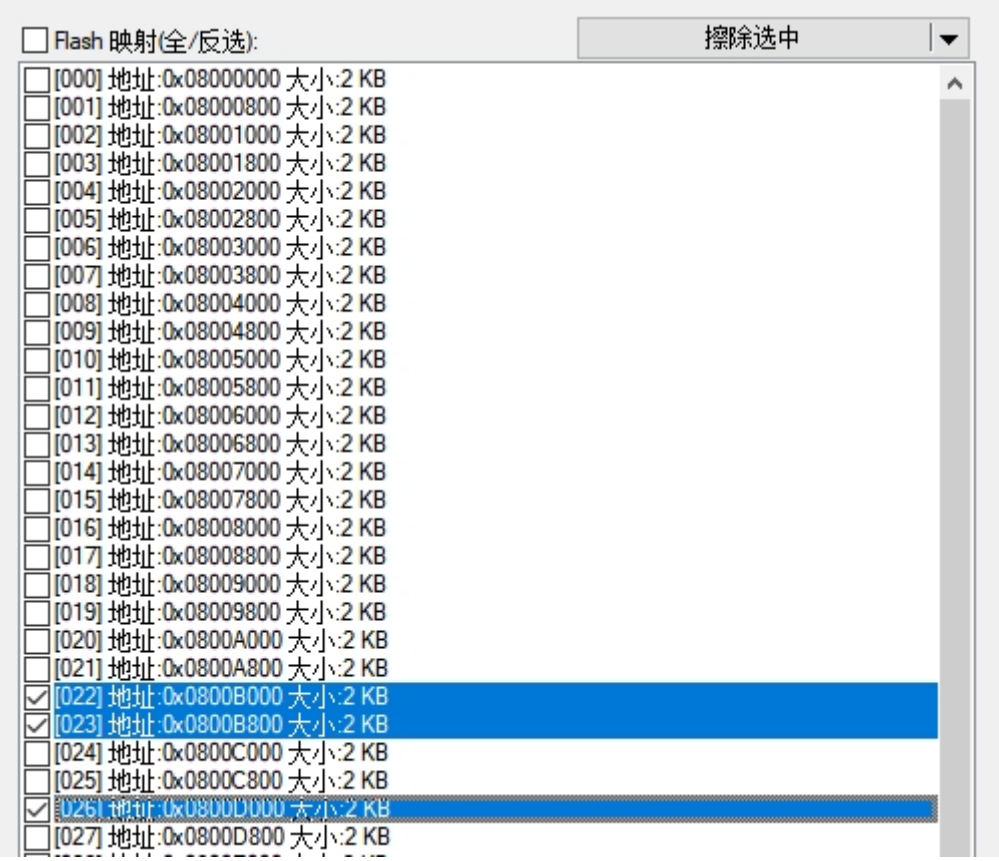
编辑时只可以将随机块改小，不可将随机块该大，因为列表中显示的就是最大的。

用户也可以添加外部随机块，参考上一条，只要固件以PWRS开头，PWRE结尾，PowerWriter 将会将其标记为随机快功能, 在烧录的过程中使用随机数据进行填充。

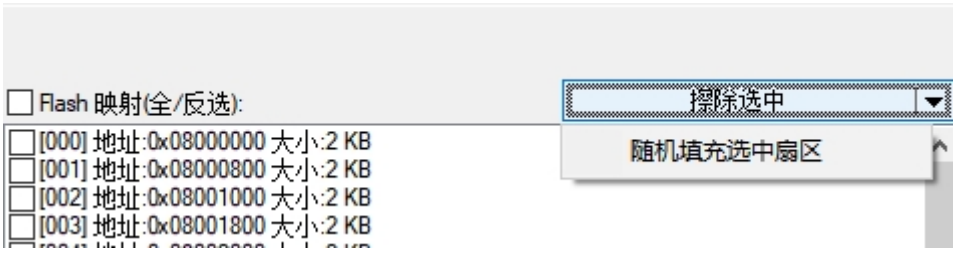
删除固件：删除固件按钮可以删除选择的固件。

应用固件：添加到固件列表的中固件，增删完成后，点击应用固件，PowerWriter 将固件刷新到缓冲区实时显示，同时会选中当前固件对应的扇区表。

Program Memory 扇区表：扇区表是PowerWriter 独有的功能，用户可以直观地从扇区表 看到芯片的Flash 结构，并直观地显示扇区表的索引、地址、扇区大小信息，如图所示：



PowerWriter 的扇区表功能除了直观地展示给用户之外，此外额外提供了快速定位， 随机填充选中扇区，擦除选中扇区功能。如下图所示：

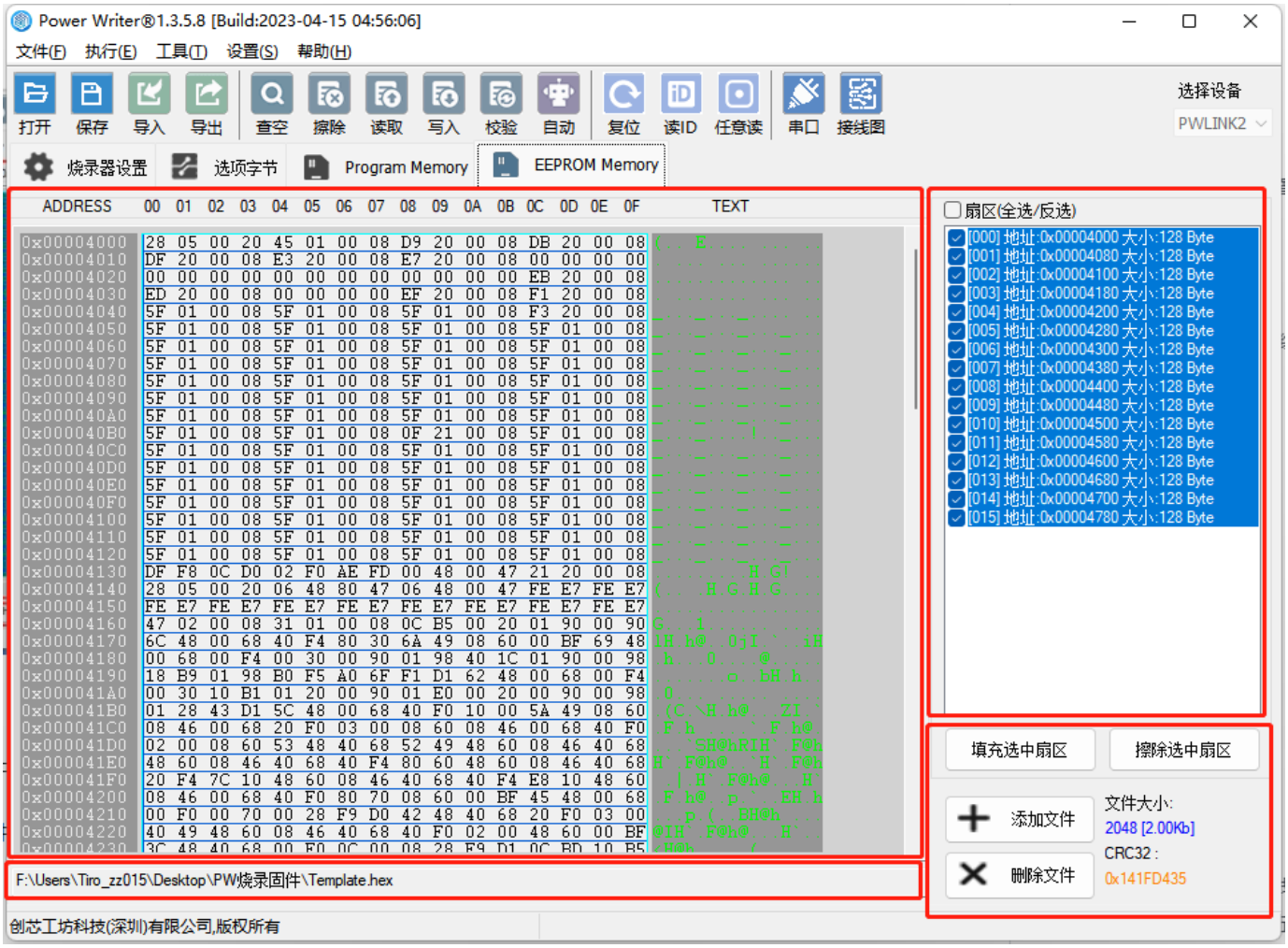


擦除选中：用户可以自由选择需要擦除的扇区，然后点击**擦除选中，**PowerWriter 将对用户选 中的扇区进行擦除。

随机填充：随机填充功能将随机数据填充到用户选中的扇区，常用于测试、填充Flash 的空白 数据，用于隐藏自己的真实数据等操作，用户可以自由发挥。

EEPROM Tab 标签页

部分芯片有EEPROM,如果选择有EEPROM 的芯片，会显示弹出EEPROM TAB 页，以供用户导入EEPROM 数据,EEPROM 表现页面显示为如下图所示：



EEPROM 页面功能包含：

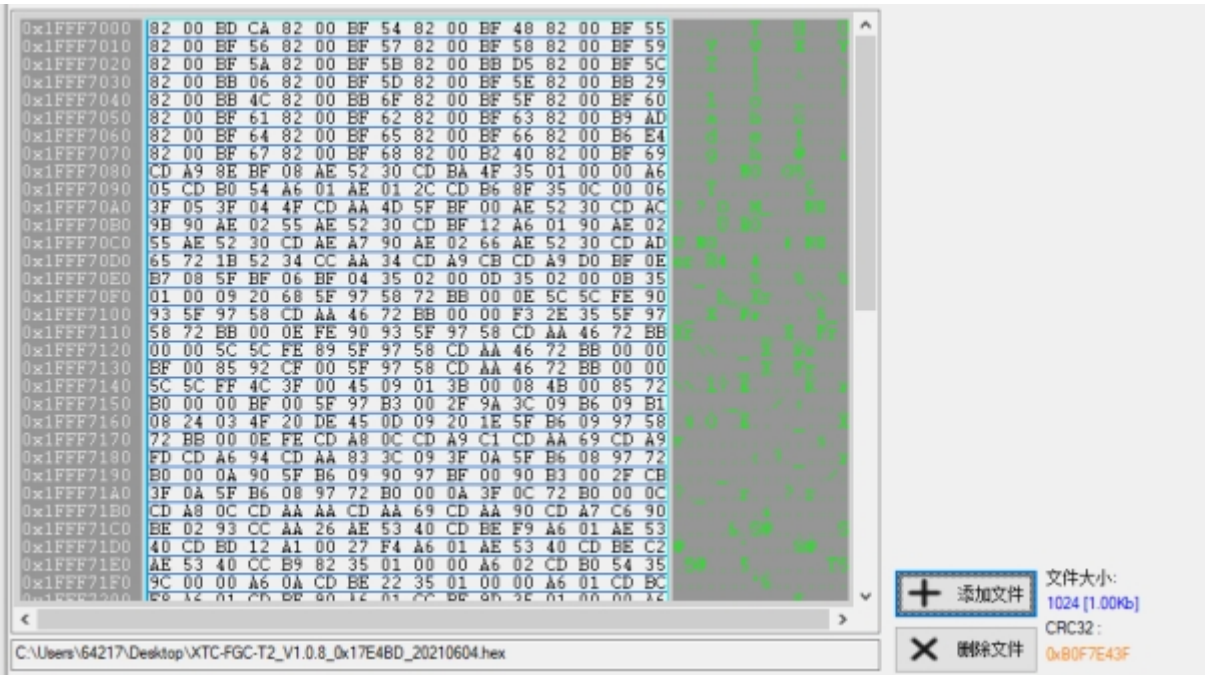
- **页面缓冲区**：同Main Flash(Program Memory)一样，此页面也包含缓冲区预览和实时编辑，如复制粘贴，跳转地址等。
- **加载文件路径导览**：加载文件后将看到加载文件的路径。
- **文件操作区:**
  - **添加文件**：通过此处添加文件到当前页面，支持Bin,Hex,S19 格式，和其他页面的添加文件功能一致。
  - **删除文件**：删除当前页面已添加的文件。
  - **文件大小**：实时查看添加文件的大小。
  - **CRC32**：实时查看添加文件的CRC校验值。
- **扇区操作区:**

- **扇区表**：同Main Flash(Program Memory)一样，此页面也包含扇区表，可以选中部分扇区进行填充，或者选中后擦除指定的扇区。
- **填充扇区**：对选中的扇区进行随机填充。
- **擦除当前扇区**：对选中的扇区进行页面擦除。

> **注意**
>
> - EEPROM Tab 与所选芯片有关，如果芯片存在EEPROM，才会有此Tab 页面显示。
> - EEPROM Tab 由于都是小容量，暂时没有做分段支持，只能添加一个。
> - EEPROM Tab 页面由于是数据空间，所以不会对地址信息和长度信息进行严格的审查，如果加载的文件地址不在缓冲区空间，将会修改为缓冲区的首地址。如果加载的文件大于缓冲区长度，将会进行截断处理。
> - EEPROM Tab的扇区表部分为 PowerWriter 模拟的，EEPROM 的数据实际上可以直接写，只有少部分芯片EEPROM 也需要擦除 ，为了给用户提供一致的使用体验，提供读取，写入， 擦除(重置功能)。

OTP Memory Tab 标签页

部分芯片有OTP,如果选择有OTP的芯片，会显示弹出OTP TAB 页，以供用户导入OTP数据,OTP表现页面显示为如下图所示：



OTP 页面功能包含：

- **页面缓冲区**：同Main Flash(Program Memory)一样，此页面也包含缓冲区预览和实时编辑，如复制粘贴，跳转地址等。

- **加载文件路径导览**：加载文件后将看到加载文件的路径。
- **文件操作区:**
  - **添加文件**：通过此处添加文件到当前页面，支持Bin,Hex,S19 格式，和其他页面的添加文件功能一致。
  - **删除文件**：删除当前页面已添加的文件。
  - **文件大小**：实时查看添加文件的大小。
  - **CRC32**：实时查看添加文件的CRC校验值。

> **注意**
>
> - OTP Tab 与所选芯片有关，如果芯片存在EEPROM，才会有这个Tab 页面显示...
> - OTP Tab 由于都是小容量，暂时没有做分段支持，只能添加一个。
> - OTP Tab 页面由于是数据空间，所以不会对地址信息和长度信息进行严格的审查，如果加载的文件地址不在缓冲区空间，将会修改为缓冲区的首地址。如果加载的文件大于缓冲区长度，将会进行截断处理。
> - OTP Tab 只能读取写入，不能擦除，所以OTP 页面没有扇区表!
> - OTP Tab 页面可以写入部分数据，比如写入指定的Byte,或者指定的bit。其他数据用默认值代替，如用0xff 代替(与芯片有关)，这样OTP 页面也可以实现类似多次写入的效果，在开发阶段做调试时将非常有效!

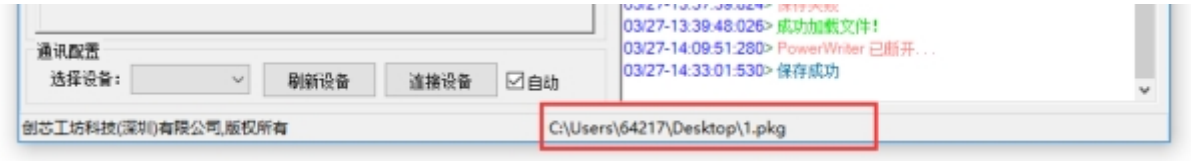
PowerWriter其他Tab 标签页

PowerWriter除了提供 Main Flash（Program Memory）页面、EEPROM页面、OTP 页面之外，在部分品牌的芯片上，也可能存在：

- Boot 页：系统引导加载页。
- Boot Loader 页：用户引导程序加载页。
- APP 页：用户代码加载页。
- User Data：用户普通数据。
- SPI Flash ： 分段代码，一部分用户代码在芯片内部，一部分代码在芯片外部。
- Configure ： 用户配置页(**和Option byte 不同，Option Byte为系统配置，此处为用户配置区**)。
- 其他未列举的页面。

PowerWriter 状态栏

状态栏显示当前加载的项目路径信息，如图所示



提示

下载本页PDF文件

编辑本页