Gowin USB to Multi Serial Protocol Bridge 软件 API

UG1251-1.2,2024-03-29

免责声明

本文档并未授予任何知识产权的许可，并未以明示或暗示，或以禁止反言或其它方式授予任何知识产权许可。除高云半导体在其产品的销售条款和条件中声明的责任之外，高云半导体概不承担任何法律或非法律责任。高云半导体对高云半导体产品的销售和／或使用不作任何明示或暗示的担保，包括对产品的特定用途适用性、适销性或对任何专利权、版权或其它知识产权的侵权责任等，均不作担保。高云半导体对文档中包含的文字、图片及其它内容的准确性和完整性不承担任何法律或非法律责任，高云半导体保留修改文档中任何内容的权利，恕不另行通知。高云半导体不承诺对这些文档进行适时的更新。

版本信息

日期	版本	描述
2023/12/29	1.0	初始版本。
2024/02/02	1.1	● 支持 Linux 系统和 MacOS 系统的软件 API； ● USB to UART 支持 libusb 编程方法。
2024/03/29	1.2	支持 GoBridgeDriver 驱动软件。

目录

图目录

表目录

1 关于本手册

1.1 手册内容

Gowin USB to Multi Serial Protocol Bridge 软件 API 用户指南主要包括 Windows 驱动安装与卸载方法、libusb 编程方法、UART 虚拟串口编程方法、USB to UART 软件 API 编程方法、USB to JTAG 软件 API 编程方法、USB to I2C 软件 API 编程方法、USB to SPI 软件 API 编程方法等，旨在帮助用户快速了解和编程 Gowin USB to Multi Serial Protocol Bridge 产品。

1.2 相关文档

通过登录高云半导体网站 www.gowinsemi.com.cn 可以下载、查看以下相关文档：IPUG1180，Gowin USB to Multi Serial Protocol Bridge IP 用户指南

1.3 术语、缩略语

本手册中出现的相关术语、缩略语及相关释义如表 1-1 所示。

表 1-1 术语、缩略语

术语、缩略语	全称	含义
API	Application Programming Interface	应用程序编程接口
I2C	Inter-Integrated Circuit	互连集成电路
IP	Intellectual Property	知识产权
JTAG	Joint Test Action Group	联合测试工作组
SDK	Software Development Kit	软件开发工具包
SPI	Serial Peripheral Interface	串行外设接口
UART	Universal Asynchronous Receiver/Transmitter	通用异步收发传输器
USB	Universal Serial Bus	通用串行总线

1.4 技术支持与反馈

高云半导体提供全方位技术支持，在使用过程中如有任何疑问，可直接与公司联系：

网址：www.gowinsemi.com.cn

E-mail：support@gowinsemi.com

Tel：+86 755 8262 0391

2开发平台

2.1 硬件目标

- DK_USB2.0_GW2AR-LV18QN88PC8I7_GW1NSR-LV4CMG64PC7I6_ V3.0
  - GW2AR-LV18QN88PC8/I7
  - GW1NSR-LV4CMG64PC7/I6

2.2 软件平台

- Windows 软件平台
  - CodeBlocks 20.03
- Linux 软件平台
  - CodeBlocks 20.03
- MacOS 软件平台
  - Xcode 12.5.1

3 软件开发工具包

- Windows SDK
  - Gowin_USB_To_Multi_Serial_Protocol_Bridge_SDK_win_Vx.x.zip
- Linux SDK
  - Gowin_USB_To_Multi_Serial_Protocol_Bridge_SDK_linux_Vx.x.zip
- MacOS SDK
  - Gowin_USB_To_Multi_Serial_Protocol_Bridge_SDK_mac_Vx.x.zip

注！

“Vx.x”中的“x”表示软件的版本号。

4 功能简介

Gowin USB to Multi Serial Protocol Bridge 可以实现 USB To JTAG、SPI、I2C、UART 四种不同的接口转换。

- USB To JTAG 实现 USB 转 JTAG 接口模式，可以用于读写 JTAG 接口设备，其中 TCK 时钟频率可以进行配置，最高可达 30 MHz。
- USB To SPI 实现 USB 转 SPI 接口模式，可以用于 SPI 主模式下的数据发送和接收。
- USB To I2C 实现 USB 转 I2C 接口模式，可以用于 I2C 主从模式下的数据接收和发送。
- USB To UART 实现 USB 转 UART 接口模式，可以实现数据的收发以及波特率、校验位等参数的设置。
  - Windows 系统和 Linux 系统支持 USB 虚拟串口编程方法和 libusb 编程方法；
  - MacOS 系统支持 libusb 编程方法。

5 驱动安装与卸载

Windows 系统中需要安装 USB 设备驱动程序。

注！

Linux 系统和 MacOS 系统可以自动识别 USB 设备，不需要安装 USB 设备驱动程序，请忽略此章节。

USB To JTAG、SPI、I2C、UART 四种接口模式，可以使用开源 USB 函数库“libusb”进行软件编程。使用该函数库编程时，须安装 Windows USB 驱动程序 WinUSB.sys。

USB To UART 接口模式也可以使用 USB 虚拟串口编程方法。使用该编程方法时，须安装“usbser.sys”驱动程序。

Windows 系统中，上述软件驱动可以使用高云自主研发的 USB 驱动软件 GoBridgeDriver 进行安装，安装该驱动时须管理员权限。

5.1 驱动安装

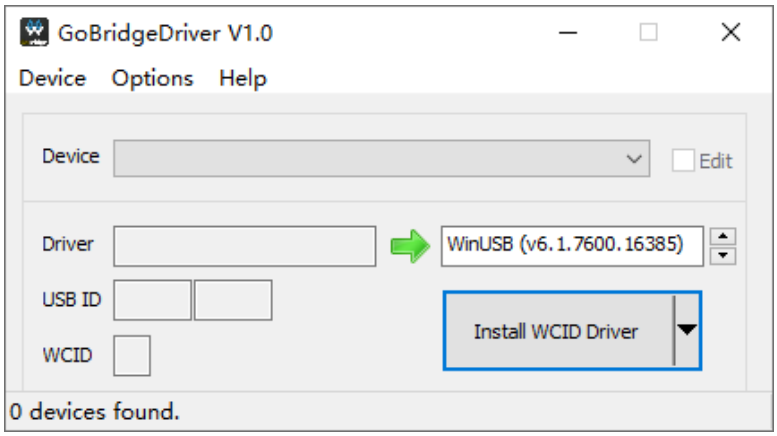
Windows 系统中，请参照以下步骤安装驱动程序。

步骤 1

连接 USB To Multi Serial Protocol Bridge 设备到本地 PC 主机的 USB 接口，打开 GoBridgeDriver（须管理员权限），如图 5-1 所示。

- Device：表示识别到的设备名称，默认无法在编辑框中输入设备名称，勾选复选框“Edit”后方可手动编辑设备名称。
- Driver：表示识别到的设备驱动类型。
- USB ID：表示设备的产品 ID、厂商 ID 和接口号。
- WCID：表示是否 Windows 兼容设备。

图 5-1 驱动软件界面



GoBridgeDriver 的各个选项功能描述如表 5-1 所示。

表 5-1 选项功能

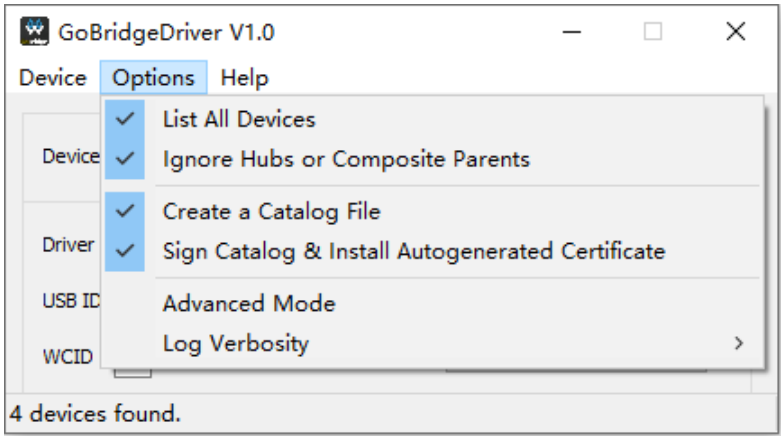
菜单	选项	功能描述
Device	Create New Device	创建一个需安装软件驱动的新设备
	Load Preset Device	载入预设的设备配置文件
Options	List All Devices	在 Device 下拉列表列出识别到的所有设备
	Ignore Hubs or Composite Parents	是否显示 Hub 设备和 Composite 设备
	Create a Catalog File	是否为安装的驱动产生目录文件（.cat）
	Sign Catalog & Install Autogenerated Certificate	是否为目录文件数字签名和安装自动生成的证书
	Advanced Mode	显示识别到的设备和驱动安装时的日志内容
	Log Verbosity	选择日志中显示的内容，可选择输出 Error、Warning、Info 或 Debug 中的一种
Help	Open Certificate Manager	打开证书管理窗口
	About	关于驱动软件的信息
Install Driver	Install Driver	为设备安装驱动
	Replace Driver	为设备替换驱动
	Reinstall Driver	为设备重新安装驱动
	Install WCID Driver	为设备安装 WCID 驱动
	Extract Files (Don't Install)	用户提取设备的驱动程序文件
	Install Filter Driver	为设备安装过滤器驱动

步骤 2

选择菜单栏“Options”，下拉列表中选择“List All Device”选项，此

时会列举出所有连接到 PC 主机的 USB 设备，如图 5-2 所示。

图 5-2 List All Devices 选项



步骤 3

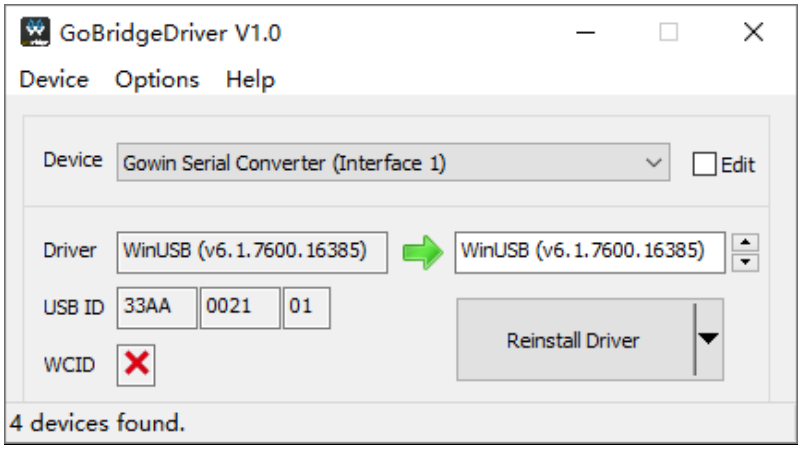
USB To JTAG、SPI、I2C 三种接口模式的驱动安装，如图 5-3 所示。

例如：

选择接口 Gowin Serial Converter (Interface 1)

选择驱动程序 WinUSB

图 5-3 USB to JTAG、SPI、I2C 驱动安装



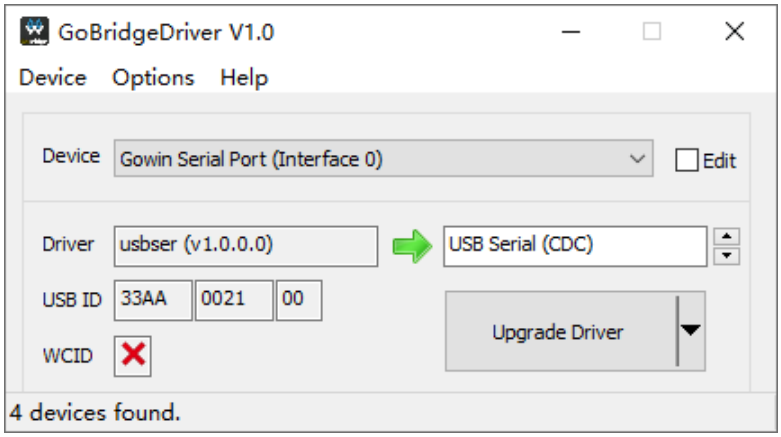
USB To UART 接口模式的驱动安装，如图 5-4 所示。

例如：

选择接口 Gowin Serial Port (Interface 0)

如果使用 USB 虚拟串口，选择驱动程序 USB Serial (CDC)

图 5-4 USB to UART 驱动安装



5.2 驱动卸载

Windows 系统中，请参照以下步骤卸载驱动程序。

步骤 1

连接 USB To Multi Serial Protocol Bridge 设备到本地 PC 主机。

步骤 2

打开 PC 主机的设备管理器，在“通用串行总线设备”下拉列表中，找到设备 Gowin Serial Converter；在“端口”下拉列表中，找到设备 Gowin Serial Port (Interface 0)。

步骤 3

右单击上述两个设备，在弹出的菜单中选择“卸载设备”选项，即可卸载驱动程序。

6 libusb 编程方法

6.1 主要特征

- libusb 是一个开源的 USB 函数库，官方网址是 https://libusb.info
- 源代码链接为 github：https://github.com/libusb/libusb
- 可以通过其官方网址下载预编译版本，包括 GCC 版本和 Visual Studio 版本，分为动态链接库和静态链接库；或者也可以通过 github 下载源代码进行编译
- libusb 函数说明，参照 http://libusb.sourceforge.net/api-1.0

6.2 函数接口

以下各节详细说明 libusb 的函数接口。

6.2.1 初始化和退出 libusb

使用 libusb 编程时，需先调用函数 libusb_init ()对其进行初始化，使用结束后，应当调用函数 libusb_exit ()令其从系统中退出。

原型：

*int libusb_init (libusb_context ** context);*

*void libusb_exit (libusb_context * ctx);*

参数：

context/ctx：libusb 的上下文结构体，用于存储 libusb 的配置参数。如果不指定该参数，系统会自动创建一个默认的上下文结构。如果已存在上下文结构，则直接使用该结构，且不重新初始化。

示例：

```
int rc = libusb_init (NULL);
if (rc < 0)
```

```
    return rc;
libusb_exit (NULL);
```

6.2.2 打开和关闭指定的 USB 设备

1. 可以调用函数 libusb_open_device_with_vid_pid ()，参照 VID 和 PID 打开指定设备。

原型：

```
libusb_device_handle* libusb_open_device_with_vid_pid (libusb_context * ctx, uint16_t vendor_id, uint16_t product_id);
```

参数：

- ctx：初始化 libusb 时产生的上下文结构的地址，如果使用默认的上下文结构，则指定为 NULL
- vendor_id：USB 设备的 VID，例如 0x33aa
- product_id：USB 设备的 PID，例如 0x0021

返回值：

libusb 在本地 PC 主机上查找到的第一个符合设备的操作句柄指针，否则返回空指针 NULL。

示例：

```
devh = libusb_open_device_with_vid_pid (NULL, 0x33aa, 0x0021);
if (NULL == devh)
{
    printf ("Open USB device failed\n");
    goto out;
}
```

2. 也可以调用函数 libusb_get_device_list ()，获取所有的 USB 设备，从中选出所需的设备。

原型：

```
ssize_t libusb_get_device_list (libusb_context *ctx, libusb_device *** list);
```

参数：

- ctx：初始化 libusb 时产生的上下文结构的地址，如果使用默认的上下文结构，则指定为 NULL
- list：指向存储设备列表的指针

返回值：

如果函数执行正确，则返回值为设备数量，并且 list 中存储查找到的设备列表，否则返回一个小于零的 libusb_error 值。

说明：

在调用结束后，应当调用函数 libusb_free_device_list ()释放设备列表。

3. 然后调用函数 libusub_open ()获取该设备的句柄，用于进行后续的操作。

原型：

*int libusb_open (libusb_device *dev, libusb_device_handle **dev_handle);*

参数：

- dev：设备列表中的设备
- dev_handle：存储返回设备句柄的指针的地址

返回值：

若设备打开成功，则返回值为 0，否则返回一个小于零的 libusb_error 值。

说明：

调用完成后需要关闭 USB 设备，可以调用函数 libusb_close ()，该函数的参数为设备的操作句柄指针。

示例：

```
cnt = libusb_get_device_list (NULL, &devs);
if (cnt < 0)
{
    // get device list failed
    return -1;
}
for (int i = 0; i < cnt; i++)
{
    libusb_open (dev[i], dev_handle);
    if (/*the wanted device is opened*/)
    {
        break;
```

```
        }
        else
        {
            //the current device is not wanted, close it and check the next
        one.
            libusb_close (dev_handle);
        }
    }
```

6.2.3 声明接口

USB 设备通常包含一个或多个接口（interface），libusb 在使用接口时，需先声明接口（claim interface），声明成功后，则表示该接口被成功打开，可以对该接口包含的端点（endpoint）进行收发操作。

原型：

*int libusb_claim_interface (libusb_device_handle * dev_handle, int interface_number);*

参数：

- dev_handle：设备句柄
- interface_number：接口编号

返回值：

Gowin USB to Multi Serial Protocol_Bridge 设备中，UART 接口模式的接口编号为 0，JTAG、I2C、SPI 接口模式的接口编号为 1。如果声明接口成功，则返回 0，否则返还一个小于零的 libusb_error 值。

示例：

```
rc = libusb_claim_interface (devh, 0);
if (rc < 0)
{
    printf ("Error claiming interface: %s\n", libusb_error_name(rc));
    goto out;
}
```

7 UART 虚拟串口编程方法

Windows 系统和 Linux 系统支持 Gowin USB to Multi Serial Protocol_Bridge 进行 UART 虚拟串口编程方法。

Windows 系统中，调用 Windows 库函数可以对 Gowin USB to Multi Serial Protocol_Bridge 进行虚拟串口操作。软件编程时，须包含头文件<windows.h>。操作串口时，须参照设备管理器识别到的端口号，修改 gw_usb2uart.h 头文件中定义的端口号。

Linux 系统中，调用 Linux 库函数可以对 Gowin USB to Multi Serial Protocol_Bridge 进行虚拟串口操作。软件编程时，须包含用于文件状态的头文件<stat.h>、<fcntl.h>等。操作串口时，须参照设备识别到的端口号，修改 gwusb_demo.c 文件中定义的端口号。

7.1 Windows 系统虚拟串口编程方法

7.1.1 打开和关闭虚拟串口设备

1. 调用 Windows 库函数 CreateFile ()可以打开虚拟串口设备，并返回一个该串口的句柄。

原型：

```
HANDLE WINAPI CreateFile (
    _In_        LPCTSTR lpFileName, //要打开或创建的文件名
    _In_        DWORD dwDesiredAccess,     //访问类型
    _In_        DWORD dwShareMode,         //共享方式
    _In_opt_    LPSECURITY_ATTRIBUTES lpSecurityAttributes, //安全属性
    _In_        DWORD dwCreationDisposition,    //指定要打开的文件已存在或不存在的动作
    _In_        DWORD dwFlagsAndAttributes,     //文件属性和标志
    _In_opt_    HANDLE hTemplateFile            //一个指向模板文件的
```

句柄

);

参数：

- lpFileName：要打开或创建的文件名
- dwDesiredAccess：访问方式：
  - 0：设备查询访问方式
  - GENERIC_READ：读访问
  - GENERIC_WRITE：写访问
- dwShareMode：共享方式：
  - 0：表示文件不能被共享，其它打开文件的操作都会失败
  - FILE_SHARE_READ：表示允许其它读操作
  - FILE_SHARE_WRITE：表示允许其它写操作
  - FILE_SHARE_DELETE：表示允许其它删除操作
- lpSecurityAttributes：安全属性，一个指向 SECURITY_ATTRIBUTES 结构的指针
- dwCreationDisposition：创建或打开文件时的动作：
  - OPEN_ALWAYS：打开文件，如果文件不存在则创建它
  - TRUNCATE_EXISTING：打开文件，且将文件清空，需要 GENERIC_WRITE 权限，如果文件不存在则会失败
  - OPEN_EXISTING：打开文件，文件若不存在则会失败
  - CREATE_ALWAYS：创建文件，如果文件已存在则清空
  - CREATE_NEW：创建文件，如文件存在则会失败
- dwFlagsAndAttributes：文件标志属性：
  - FILE_ATTRIBUTE_NORMAL：常规属性
  - FILE_FLAG_OVERLAPPED：异步 I/O 标志，如果不指定此标志则默认为同步 IO
  - FILE_ATTRIBUTE_READONLY：文件为只读
  - FILE_ATTRIBUTE_HIDDEN：文件为隐藏
  - FILE_FLAG_DELETE_ON_CLOSE：所有文件句柄关闭后文件被删除
  - 其它标志和属性请参照微软官方文档
- hTemplateFile：一个文件的句柄，且该文件必须是以 GENERIC_READ 访问方式打开的。如果该参数不是 NULL，则会使用 hTemplateFile 关联的文件的属性和标志来创建文件。如果是打开一个

现有的文件，则该参数被忽略。

说明

使用 CreateFile ()打开串口时需要注意：

- lpFileName 文件名，如果是 COM10 以下的串口，则直接写串口号名，如“COM1”；如果是 COM10 及以上的串口，则串口名格式应为 \\\\.\\COM10
- dwShareMode 共享方式应为 0，即串口应为独占方式
- dwCreationDisposition 打开时的动作应为 OPEN_EXISTING，即串口必须存在

2. 关闭串口时，调用函数 CloseHandle ()关闭串口，该函数的参数为串口句柄。

7.1.2 超时设置

调用函数 ReadFile ()和函数 WriteFile ()读写串口的时候，如果没有指定异步操作，则读写都会一直等待指定大小的数据，这时我们需要设置一个读写的超时时间。

调用函数 SetCommTimeouts ()可以设置串口读写超时时间，调用函数 GetCommTimeouts ()可以获得当前的超时设置。一般应用流程是先调用函数 GetCommTimeouts ()获得当前超时信息，存储在一个 COMMTIMEOUTS 结构里，然后对这个结构自定义，再调用函数 SetCommTimeouts ()进行设置。

7.1.3 串口参数设置

Windows 系统提供专用数据结构 DCB 和专用接口函数 GetCommState ()和函数 SetCommState ()，用于设置串口参数。

DCB 是一个用于串口通信设备设置的数据结构，其中串口基本设置的变量如下：

- BaudRate：波特率设置，可以设置的最大值为 256000
- Parity：校验位设置，可以设置为无校验（NOPARITY）、奇校验（ODDPARITY）、偶校验（EVENPARITY）、1 校验（MARKPARITY）、0 校验（SPACEPARITY）
- ByteSize：传输单个字节的比特数
- StopBits：停止位设置，可以设置为 1 位停止位（ONESTOPBIT）、1.5 停止位（ONE5STOPBITS）、2 停止位（TWOSTOPSBITS）

7.1.4 清除串口缓冲

Windows 系统提供函数 PurgeComm ()，用于停止读写操作和清除读写缓冲区。

第一次读取串口数据、写串口数据之前，或串口长时间未使用、串口出现错误等情况下，应当先清空读写缓冲区。

7.1.5 清除串口错误

Windows 系统提供函数 ClearCommError ()来清除通信中的错误，以及获得当前通信的状态。

在读写操作之前，可以调用函数 ClearCommError ()来清除错误，获得缓冲区内数据大小。

7.1.6 异步读写串口

异步读写串口的方式需要以重叠方式打开串口。如果重叠操作不能立即完成，则调用函数 WaitCommEvent ()时会返回 FALSE，调用函数 GetLastError ()时会返回 ERROR_IO_PENDING，表示操作正在后台进行。在函数 WaitCommEvent ()返回之前，参数重叠结构中的 hEvent 成员会被设置为无信号状态，当事件发生或错误发生时，其被设置为有信号状态，应用程序可以调用等待函数 WaitForSingleObject ()、WaitForSingleObjectEx ()等来判断时间对象的状态，而函数 WaitCommEvent ()的参数 lpEvtMask 会存储具体发生的事件。

有两种方法可以等待或判断重叠操作是否完成：

- 调用函数 WaitForSingleObject ()来等待读写函数中 OVERLAPPED 类型的参数的 hEvent 成员。当调用函数 ReadFile ()、函数 WriteFile ()时，该成员会自动被置为无信号状态。当重叠操作完成后，该成员变量会自动被置为有信号状态
- 调用函数 GetOverlappedResult ()获得重叠操作的状态，来判断重叠操作是否完成

7.2 Linux 系统虚拟串口编程方法

7.2.1 打开虚拟串口设备

执行该函数时，可以打开系统中的串口并返回串口设备文件描述符。

原型：

int UART0_Open (char port);*

参数：

Port：串口号

返回值：

如果执行成功，返回一个文件描述符值，否则返回值为-1。

7.2.2 关闭虚拟串口

执行该函数时，可以关闭系统中的串口。

原型：

void UART0_Close (int fd);

参数：

fd：文件描述符

返回值：

无

7.2.3 初始化串口

执行该函数时，可以对打开的串口进行初始化。

原型：

int UART0_Init (int fd, int speed, int flow_control, int databits, int stopbits, int parity);

参数：

- fd：文件描述符
- speed：串口波特率
- flow_control：数据流控制
- databits：数据位
- stopbits：停止位
- parity：校验类型

返回值：

如果执行成功，则返回值为 0，否则返回-1。

7.2.4 设置串口配置参数

执行该函数时，可以对打开的串口进行参数配置。

原型：

int UART0_Set (int fd, int speed, int flow_control, int databits, int stopbits, int parity);

参数：

- fd：文件描述符
- speed：串口波特率
- flow_control：数据流控制
- databits：数据位
- stopbits：停止位
- parity：校验类型

返回值：

如果执行成功，则返回值为 0，否则返回-1。

7.2.5 串口发送数据

执行该函数时，通过串口发送数据。

原型：

*int UART0_Send (int fd, char *send_buf, int data_len);*

参数：

- fd：文件描述符
- buf：存放要发送的数据
- data_len：要发送数据的个数

返回值：

如果执行成功，则返回值为 0，否则返回-1。

7.2.6 串口接收数据

执行该函数时，通过串口接收数据。

原型：

*int UART0_Recv (int fd, char *rcv_buf, int data_len);*

参数：

- fd：文件描述符
- rev_buf：存放要接收的数据
- data_len：要接收数据的个数

返回值：

如果执行成功，则返回值为 0，否则返回-1。

8 软件 API 编程方法

8.1 USB To JTAG 接口模式编程

USB To JTAG 接口模式软件 API 编程，可以实现以下功能：

- 设置 TCK 频率
- 读取器件 IDCODE 和寄存器状态
- SRAM 擦除和写入数据
- 内置 Flash 擦除、读取和写入数据
- 外置 Flash 读取 Flash ID、擦除、读取和写入数据

8.1.1 设置 TCK 频率

该函数通过 JTAG 接口发送数据，来选择不同的 TCK 频率。

原型：

*int usb2jtag_tck_select (libusb_device_handle *devh, unsigned int select_tck, unsigned int uiTimeout);*

参数：

- devh：libusb 设备操作句柄
- select_tck：TCK 频率选择
  - 频率的设置遵循 30MHZ/(select_tck *2)，上电默认是 15MHZ
  - select_tck 为 0x0，频率为 30MHZ
  - select_tck 为 0x1，频率为 15MHZ
  - select_tck 为 0x2，频率为 7.5MHZ
  - select_tck 为 0x3，频率为 5MHZ
  - ……
- uiTimeOut：超时参数，单位为毫秒

返回值：

如果执行成功，则返回值为 0，否则返回一个小于零的错误代码。

8.1.2 JTAG TAP 状态转换

1. JTAG 状态复位

执行该函数时，JTAG 发送连续 TMS 高电平信号，使得 JTAG 设备状态跳转到 TEST LOGIC RESET。

原型：

*int usb2jtag_tap_reset (libusb_device_handle *devh, unsigned int uiTimeout);*

参数：

- devh：libusb 设备操作句柄
- uiTimeOut：超时参数，单位为毫秒

返回值：

如果执行成功，则返回值为 0，否则返回一个小于零的错误代码。

2. 从复位状态到 IDLE 状态

执行该函数时，JTAG 发送一个 TMS 低电平，使得 JTAG 设备从 TEST LOGIC RESET 状态跳转到 Run-Test-Idle 状态。

原型：

*int usb2jtag_from_rst_to_idle (libusb_device_handle *devh, unsigned int uiTimeout);*

参数：

- devh：libusb 设备操作句柄
- uiTimeOut：超时参数，单位为毫秒

返回值：

如果执行成功，则返回值为 0，否则返回一个小于零的错误代码。

3. Shift-IR 移位指令寄存器状态

执行该函数时，JTAG 发送连续 TMS 高低电平信号，使得 JTAG 设备状态跳转到 Shift IR 状态。

原型：

*int usb2jtag_shift_ir (libusb_device_handle *devh, unsigned int uiTimeout);*

参数：

- devh：libusb 设备操作句柄

- uiTimeOut：超时参数，单位为毫秒

返回值：

如果执行成功，则返回值为 0，否则返回一个小于零的错误代码。

4. Shift-DR 移位数据寄存器状态

执行该函数时，JTAG 发送连续 TMS 高低电平信号，使得 JTAG 设备状态跳转到 Shift DR 状态。

原型：

*int usb2jtag_shift_dr (libusb_device_handle *devh, unsigned int uiTimeout);*

参数：

- devh：libusb 设备操作句柄
- uiTimeOut：超时参数，单位为毫秒

返回值：

如果执行成功，则返回值为 0，否则返回一个小于零的错误代码。

8.1.3 发送命令数据

执行该函数时，JTAG 首先跳转至 Shift IR 状态，发送命令数据后返回至 Run-Test-Idle 状态。

原型：

*int usb2jtag_ConfigData_Send (libusb_device_handle *devh, unsigned char *Usb2JtagCmd, unsigned int uiTimeout);*

参数：

- devh：libusb 设备操作句柄
- Usb2JtagCmd：要发送的命令
- uiTimeOut：超时参数，单位为毫秒

返回值：

如果执行成功，则返回值为 0，否则返回一个小于零的错误代码。

8.1.4 读取器件 IDCODE

执行该函数时，JTAG 发送读取 IDCODE 指令至器件，再通过 JTAG 返回 IDCODE，并打印输出。

原型：

*int usb2jtag_Read_IDCODE (libusb_device_handle *devh, unsigned int uiTimeout);*

参数：

- devh：libusb 设备操作句柄
- uiTimeOut：超时参数，单位为毫秒

返回值：

如果执行成功，则返回值为 0，否则返回一个小于零的错误代码。

8.1.5 读取器件状态寄存器

执行该函数时，JTAG 发送读取状态寄存器数据的命令至器件，再通过 JTAG 返回状态寄存器数值，并打印输出。

原型：

*int usb2jtag_Read_Status_Register (libusb_device_handle *devh, unsigned int uiTimeout);*

参数：

- devh：libusb 设备操作句柄
- uiTimeOut：超时参数，单位为毫秒

返回值：

如果执行成功，则返回值为 0，否则返回一个小于零的错误代码。

8.1.6 擦除 SRAM

执行该函数时，可以实现擦除器件 SRAM。

原型：

*int usb2jtag_Erase_Sram (libusb_device_handle *devh, unsigned int uiTimeout);*

参数：

- devh：libusb 设备操作句柄
- uiTimeOut：超时参数，单位为毫秒

返回值：

如果执行成功，则返回值为 0，否则返回一个小于零的错误代码。

8.1.7 编程 SRAM

执行该函数时，可以实现下载二进制文件至器件 SRAM。

原型：

*int usb2jtag_Config_Sram (libusb_device_handle *devh, const char *filename, unsigned int uiTimeout);*

参数：

- devh：libusb 设备操作句柄
- filename：需要下载的二进制文件
- uiTimeOut：超时参数，单位为毫秒

返回值：

如果执行成功，则返回值为 0，否则返回一个小于零的错误代码。

8.1.8 内置 Flash 编程延时设置

执行该函数时，参照不同的 TCK 频率，可以设置不同的延时时间。

原型：

*void usb2jtag_TckSet_Delay (libusb_device_handle *devh, unsigned int select_tck, int uiTimeout);*

参数：

- devh：libusb 设备操作句柄
- select_tck：TCK 频率，参照 8.1.1 设置 TCK 频率
- uiTimeOut：超时参数，单位为毫秒

返回值：

无

8.1.9 T 型工艺内置 Flash 擦除

执行该函数时，可以擦除 T 型工艺系列器件内置 Flash。

原型：

*int usb2jtag_Erase_T_Flash (libusb_device_handle *devh, unsigned int uiTimeout);*

参数：

- devh：libusb 设备操作句柄
- uiTimeOut：超时参数，单位为毫秒

返回值：

如果执行成功，则返回值为 0，否则返回一个小于零的错误代码。

8.1.10 T 型工艺内置 Flash 读取

执行该函数时，读取 T 型工艺系列器件内置 Flash 中的数据，与原始的下载数据对比，并打印输出错误数据。

原型：

*int usb2jtag_read_emb_Flash (libusb_device_handle *devh, const char *filename, unsigned int uiTimeout);*

参数：

- devh：libusb 设备操作句柄
- filename：需要对比的二进制文件
- uiTimeOut：超时参数，单位为毫秒

返回值：

如果执行成功，则返回值为 0，否则返回一个小于零的错误代码。

8.1.11 编程 T 型工艺内置 Flash

执行该函数时，可以写入二进制数据到 T 型工艺系列器件内置 Flash 中。

原型：

*int usb2jtag_embed_Program_Flash (libusb_device_handle *devh, const char *filename, int verify, unsigned int uiTimeout);*

参数：

- devh：libusb 设备操作句柄
- filename：需要下载的二进制文件
- verify：设置为 0 是自启动，设置为 1 是可回读
- uiTimeOut：超时参数，单位为毫秒

返回值：

如果执行成功，则返回值为 0，否则返回一个小于零的错误代码。

8.1.12 读取外置 Flash ID

执行该函数时，可以读取外置 Flash 型号，并打印输出型号 ID。

原型：

*int usb2jtag_read_SPI_Flash_id (libusb_device_handle *devh, unsigned int uiTimeout);*

参数：

- devh：libusb 设备操作句柄
- uiTimeOut：超时参数，单位为毫秒

返回值：

如果执行成功，则返回值为 0，否则返回一个小于零的错误代码。

8.1.13 擦除外置 Flash

执行该函数时，可以擦除外置 Flash。

原型：

*int usb2jtag_Erase_SPI_Flash (libusb_device_handle *devh, unsigned int uiTimeout);*

参数：

- devh：libusb 设备操作句柄
- uiTimeOut：超时参数，单位为毫秒

返回值：

如果执行成功，则返回值为 0，否则返回一个小于零的错误代码。

8.1.14 读取外置 Flash 数据

执行该函数时，可以读取外置 Flash 数据，与原始二进制数据对比，并打印输出错误数据。

原型：

*int usb2jtag_read_SPI_Flash (libusb_device_handle *devh, const char *filename, unsigned int uiTimeout);*

参数：

- devh：libusb 设备操作句柄
- filename：需要对比的二进制文件
- uiTimeOut：超时参数，单位为毫秒

返回值：

如果执行成功，则返回值为 0，否则返回一个小于零的错误代码。

8.1.15 编程外置 Flash

执行该函数时，可以下载二进制数据至外置 Flash。

原型：

*int usb2jtag_Extern_Program_Flash (libusb_device_handle *devh, const char *filename, unsigned int uiTimeout);*

参数：

- devh：libusb 设备操作句柄
- filename：需要下载的二进制文件
- uiTimeOut：超时参数，单位为毫秒

返回值：

如果执行成功，则返回值为 0，否则返回一个小于零的错误代码。

8.2 USB To I2C 接口模式编程

USB To I2C 接口模式软件 API 编程，可以实现以下功能：

- I2C 主从模式下的数据接收和发送
- 支持不同的频率选择

8.2.1 参数配置

执行该函数时，调用 I2C 参数配置结构体，对 USB To I2C 设备进行参数配置。

原型：

*int iic_set_config (libusb_device_handle *devh, USB2IIC_Config *pusb2iic_config, unsigned int uiTimeout);*

参数：

- devh：libusb 设备操作句柄
- pusb2iic_config：指向存储 I2C 参数配置结构体的指针
- uiTimeOut：超时参数，单位为毫秒

返回值：

如果执行成功，则返回值为 0，否则返回一个小于零的错误代码。

8.2.2 获取配置参数

执行该函数时，可以获取当前 USB To I2C 设备的配置参数。

原型：

*int iic_get_config (libusb_device_handle *devh, USB2IIC_Config *pusb2iic_config, unsigned int uiTimeout);*

参数：

- devh：libusb 设备操作句柄
- pusb2iic_config：指向存储 I2C 参数配置结构体的指针
- uiTimeOut：超时参数，单位为毫秒

返回值：

如果执行成功，则返回值为 0，否则返回一个小于零的错误代码。

8.2.3 打印配置参数

执行该函数时，可以打印输出 I2C 参数配置结构体的内容。

原型：

*int print_iic_config (USB2IIC_Config *pusb2iic_config);*

参数：

- pusb2iic_config：指向存储 I2C 参数配置结构体的指针

返回值：

如果执行成功，则返回值为 0，否则返回一个小于零的错误代码。

8.2.4 设备复位

执行该函数时，可以复位 USB To I2C 设备。

原型：

*int usb2iic_reset (libusb_device_handle *devh, unsigned int timeout);*

参数：

- devh：libusb 设备操作句柄
- uiTimeOut：超时参数，单位为毫秒

返回值：

如果执行成功，则返回值为 0，否则返回一个小于零的错误代码。

8.2.5 主机模式写数据

执行该函数时，实现主机模式下的写数据操作。

原型：

*int usb2iic_master_write (libusb_device_handle *devh, int DataCnt, unsigned char *TransData, unsigned int uiTimeout);*

参数：

- devh：libusb 设备操作句柄
- DataCnt：要发送的数据的字节数
- TransData：指向要发送的数据存储地址的指针
- uiTimeOut：超时参数，单位为毫秒

返回值：

如果执行成功，则返回值为 0，否则返回一个小于零的错误代码。

8.2.6 主机模式读数据

执行该函数时，实现主机模式下的读数据操作。

原型：

*int usb2iic_master_read (libusb_device_handle *devh, int DataCnt, unsigned char *RcvData, unsigned int uiTimeout);*

参数：

- devh：libusb 设备操作句柄
- DataCnt：要接收的数据的字节数
- TransData：指向要接收的数据存储地址的指针
- uiTimeOut：超时参数，单位为毫秒

返回值：

如果执行成功，则返回值为 0，否则返回一个小于零的错误代码。

8.2.7 从机模式写数据

执行该函数时，实现从机模式下的写数据操作。

原型：

*int usb2iic_slave_write (libusb_device_handle *devh, int DataCnt, unsigned char *TransData, int *DataTransed, unsigned int uiTimeout);*

参数：

- devh：libusb 设备操作句柄
- DataCnt：要发送的数据的字节数
- TransData：指向要发送的数据存储地址的指针
- DataTransed：指向存储实际发送完成的字节数的变量的指针
- uiTimeOut：超时参数，单位为毫秒

返回值：

如果执行成功，则返回值为 0，否则返回一个小于零的错误代码。

8.2.8 从机模式读数据

执行该函数时，实现从机模式下的读数据操作。

原型：

*int usb2iic_slave_read (libusb_device_handle *devh, unsigned char *RcvData, int *ptr_RcvDataBytes, unsigned int uiTimeout);*

参数：

- devh：libusb 的设备操作句柄
- RcvData：指向存储接收数据的地址的指针
- RcvDataBytes：指向存储实际接收完成的字节数的变量的指针
- uiTimeOut：超时参数，单位为毫秒

返回值：

如果执行成功，则返回值为 0，否则返回一个小于零的错误代码。

8.3 USB to SPI 接口模式编程

USB to SPI 接口模式软件 API 编程，可以实现以下功能：

- 支持 SPI Master 模式数据接收和发送功能

8.3.1 参数配置

执行该函数时，调用 SPI 参数配置结构体，对 USB To SPI 设备进行参数配置。

原型：

*int usb2spi_set_config (libusb_device_handle *devh, USB2SPI_Config *pusb2spi_config, unsigned int uiTimeout);*

参数：

- devh：libusb 设备操作句柄
- pusb2spi_config：指向存储 SPI 参数配置结构体的指针
- uiTimeOut：超时参数，单位为毫秒

返回值：

如果执行成功，则返回值为 0，否则返回一个小于零的错误代码。

8.3.2 获取配置参数

执行该函数时，获取当前 USB To SPI 设备的配置参数。

原型：

*int usb2spi_get_config (libusb_device_handle *devh, USB2SPI_Config *pusb2spi_config, unsigned int uiTimeout);*

参数：

- devh：libusb 设备操作句柄
- pusb2iic_config：指向存储 SPI 参数配置结构体的指针
- uiTimeOut：超时参数，单位为毫秒

返回值：

如果执行成功，则返回值为 0，否则返回一个小于零的错误代码。

8.3.3 设备复位

执行该函数时，复位 USB To SPI 设备。

原型：

*int usb2spi_reset (libusb_device_handle *devh, bool bTxFifoRst, bool bRxFifoRst, bool bSpiRst, unsigned int uiTimeout);*

参数：

- devh：libusb 设备操作句柄
- bTxFifoRst：是否对 TX FIFO 复位
- bRxFifoRst：是否对 RX FIFO 复位
- bSpiRst：是否对 SPI 复位
- uiTimeOut：超时参数，单位为毫秒

返回值：

如果执行成功，则返回值为 0，否则返回一个小于零的错误代码。

8.3.4 打印配置参数

执行该函数时，打印输出 SPI 参数配置结构体的内容。

原型：

*int usb2spi_print_config (USB2SPI_Config *pusb2spi_config);*

参数：

- pusb2iic_config：指向存储 SPI 参数配置结构体的指针

返回值：

如果执行成功，则返回值为 0，否则返回一个小于零的错误代码。

8.3.5 Merge 模式发送字节

执行该函数时，主模式下以 Merge 模式发送数据。

原型：

*int usb2spi_master_write_bytes_merge (libusb_device_handle *devh, USB2SPI_Config *pusb2spi_config, unsigned int uiCmd, unsigned int uiAddr, unsigned char *pucTxData, int iTxCnt, unsigned int uiTimeout);*

参数：

- devh：libusb 的设备操作句柄

- pusb2spi_config：指向存储 SPI 参数配置结构体的指针
- uiCmd：发送的指令段，指令段为 1 个字节。需要注意的是，如果不需要发送指令，则可填充任意值
- uiAddr：发送的地址段，地址段最多为 4 个字节，地址段的字节数在设置参数时进行配置
- puiTxData：指向要发送的数据单元的指针，单个数据单元的最大位宽为 32
- iTxCnt：要发送的数据单元个数，该数值应当与参数配置中的 WrTranCnt 相同
- uiTimeOut：超时参数，单位为毫秒

返回值：

如果执行成功，则返回值为 0，否则返回一个小于零的错误代码。

8.3.6 Merge 模式接收字节

执行该函数时，主模式下以 Merge 模式接收数据。

原型：

*int usb2spi_master_read_bytes_merge (libusb_device_handle *devh, USB2SPI_Config *pusb2spi_config, unsigned int uiCmd, unsigned int uiAddr, unsigned char *pucRxData, int iRxCnt, unsigned int uiTimeout);*

参数：

- devh：libusb 设备操作句柄
- pusb2spi_config：指向存储 SPI 参数配置结构体的指针
- uiCmd：发送的指令段，指令段为 1 个字节。需要注意的是，如果不需要发送指令，则可填充任意值
- uiAddr：发送的地址段，地址段最多为 4 个字节，地址段的字节数在设置参数时进行配置
- pucRxData：指向要接收的数据单元的指针
- iRxCnt：要接收的数据单元个数，该数值应当与参数配置中的 RdTranCnt 相同
- uiTimeOut：超时参数，单位为毫秒

返回值：

如果执行成功，则返回值为 0，否则返回一个小于零的错误代码。

8.3.7 Merge 模式读写字节

执行该函数时，主模式下以 Merge 模式读写数据。

原型：

*int usb2spi_master_write_read_bytes_merge (libusb_device_handle *devh, USB2SPI_Config *pusb2spi_config, unsigned int uiCmd, unsigned int uiAddr, unsigned char *pucTxData, unsigned char *pucRxData, int iTxCnt, int iRxCnt, unsigned int uiTimeout);*

参数：

- devh：libusb 设备操作句柄
- pusb2spi_config：指向存储 SPI 参数配置结构体的指针
- uiCmd：发送的指令段，指令段为 1 个字节。需要注意的是，如果不需要发送指令，则可填充任意值
- uiAddr：发送的地址段，地址段最多为 4 个字节，地址段的字节数在设置参数时进行配置
- puiRxData：指向要接收的数据单元的指针
- pucTxData：指向要发送的数据单元的指针
- iTxCnt：要发送的数据单元个数，该数值应当与参数配置中的 WrTranCnt 相同
- iRxCnt：要接收的数据单元个数，该数值应当与参数配置中的 RdTranCnt 相同
- uiTimeOut：超时参数，单位为毫秒

返回值：

如果执行成功，则返回值为 0，否则返回一个小于零的错误代码。

8.4 USB To UART 接口模式编程

USB to UART 接口模式软件 API 编程，可以实现数据的收发以及波特率、校验位等参数的设置。

8.4.1 参数配置

执行该函数时，对 USB To UART 设备进行参数配置。

原型：

*int usb2uart_set_config (libusb_device_handle *devh, usb2uart_config_setting* pusb2uart_config, unsigned char interface, unsigned int uiTimeout);*

参数：

- devh：libusb 设备操作句柄
- pusb2uart_config：指向存储 UART 参数配置结构体的指针
- interface：UART 接口号
- uiTimeOut：超时参数，单位为毫秒

返回值：

如果执行成功，则返回值为 0，否则返回一个小于零的错误代码。

8.4.2 打印配置参数

执行该函数时，可以打印输出 UART 参数配置结构体的内容。

原型：

int print_usb2uart_config (usb2uart_config_setting pusb2uart_config);*

参数：

- pusb2uart_config：指向存储 UART 参数配置结构体的指针

返回值：

如果执行成功，则返回值为 0，否则返回一个小于零的错误代码。

8.4.3 发送数据

执行该函数时，UART 发送数据。

原型：

*int usb2uart_send_data (libusb_device_handle *devh, unsigned char endpoint_out, int DataCnt, unsigned char *TransData, unsigned int uiTimeout);*

参数：

- devh：libusb 设备操作句柄
- endpoint_out：UART 发送数据的端点号
- DataCnt：要发送的数据单元个数
- TransData：指向要发送的数据单元的指针
- uiTimeOut：超时参数，单位为毫秒

返回值：

如果执行成功，则返回值为 0，否则返回一个小于零的错误代码。

8.4.4 接收数据

执行该函数时，UART 接收数据。

原型：

*int usbuart_receive_data (libusb_device_handle *devh, unsigned char endpoint_in, int *DataCnt, unsigned char* TransData, unsigned int uiTimeout);*

参数：

- devh：libusb 设备操作句柄
- endpoint_in：UART 接收数据的端点号
- DataCnt：要接收的数据单元个数
- TransData：指向要接收的数据单元的指针
- UiTimeOut：超时参数，单位为毫秒

返回值：

如果执行成功，则返回值为 0，否则返回一个小于零的错误代码。

8.5 错误代码

软件 API 编程执行正常时，返回值为 0，否则返回一个负数。

非 0 返回值代表的错误含义如表 8-1 所示。

表 8-1 错误代码列表

值	代码	描述
0	SUCCESS	运行正确无错误
-1	USB_ERROR_IO	USB 输入/输出错误
-2	USB_ERROR_INVALID_PARAM	USB 参数错误
-3	USB_ERROR_ACCESS	权限不足，不能访问设备
-4	USB_ERROR_NO_DEVICE	没有找到 USB 设备（设备已断开）
-5	USB_ERROR_NOT_FOUND	未找到实体（Entity）
-6	USB_ERROR_BUSY	USB 设备忙
-7	USB_ERROR_TIMEOUT	超时
-8	USB_ERROR_OVERFLOW	内存溢出
-9	USB_ERROR_PIPE	管道错误
-10	USB_ERROR_INTERRUPTED	系统函数被中断
-11	USB_ERROR_NO_MEM	内存不足
-12	USB_ERROR_NOT_SUPPORTED	当前平台不支持该操作

值	代码	描述
-13	USB2IIC_ERR_SENDCMD_FAIL	USB2I2C 指令发送错误
-14	USB2IIC_ERR_GETREG_FAIL	USB2I2C 获取寄存器值错误
-15	USB2IIC_ERR_SETREG_FAIL	USB2I2C 设置寄存器值错误
-16	USB2IIC_ERR_INVALID_PARAM	USB2I2C 参数设置不可用
-17	USB2IIC_ERR_TXDATA_FAIL	USB2I2C 发送数据错误
-18	USB2IIC_ERR_RXDATA_FAIL	USB2I2C 接收数据错误
-19	USB2IIC_ERR_CMPL_TIMEOUT	USB2I2C 等待完成超时
-20	USB2SPI_ERR_SENDCMD_FAIL	USB2SPI 指令发送错误
-21	USB2SPI_ERR_GETREG_FAIL	USB2SPI 获取寄存器值错误
-22	USB2SPI_ERR_SETREG_FAIL	USB2SPI 设置寄存器值错误
-23	USB2SPI_ERR_INVALID_PARAM	USB2SPI 参数设置不可用
-24	USB2SPI_ERR_TXDATA_FAIL	USB2SPI 发送数据错误
-25	USB2SPI_ERR_RXDATA_FAIL	USB2SPI 接收数据错误
-26	USB2SPI_ERR_CMPL_TIMEOUT	USB2SPI 等待完成超时
-27	U2J_ERROR_USBTRANS_ERR	USB 数据传输错误
-28	U2J_ERROR_INVALID_PARAM	Usb2JTAG 参数设定值不可用
-99	ERROR_OTHER	其他错误


GOWIN高云
智慧逻辑 定制未来