GWU2X 编程指南_U2X_SPI

UG1004-1.0, 2021-06-29

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，或以禁止发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	说明
2021/06/29	1.0	初始版本。

目录

图目录

表目录

1 功能简介

GWU2X 是一个 USB 转多种协议的转换器，可实现 USB2SPI 功能转换，支持最高 SCLK 时钟频率 12MHz（当前的测试结果）。

支持标准 SPI 的主机模式。支持全双工数据收发功能，支持可选的命令段、地址段、DummySCLK 段、数据段。

2 驱动的安装和卸载

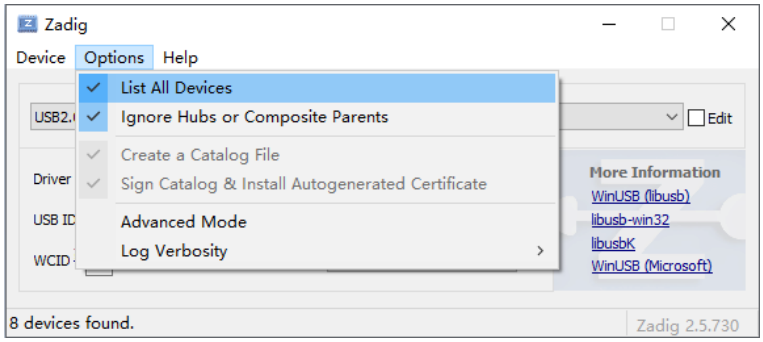
GWU2X_SPI 可使用开源 USB 函数库 libusb 进行编程操作。使用该函数库编程时，需安装 WINDOWS USB 驱动程序 WinUSB.sys。

安装驱动，可使用开源驱动安装工具 Zadig（https://zadig.akeo.ie/）。安装驱动需要使用管理员权限。

2.1 使用 Zadig 安装驱动程序

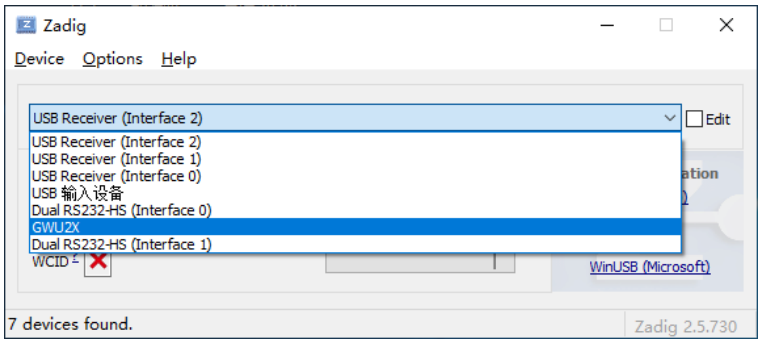
连接 GWU2X 设备到电脑 USB 接口，双击打开 Zadig（需要管理员权限），点击 Options，勾选“List All Device”选项，此时会列举出所有连接到电脑的 USB 设备。

图 2-1 选择“List All Device”选项



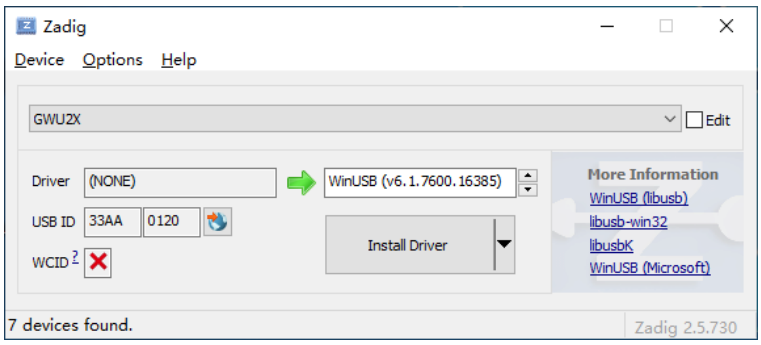
选择需要安装驱动的设备 GWU2X。

图 2-2 选择需要安装驱动的设备



选择要安装的驱动程序，使用 libusb+WinUSB 的形式，请选择 WinUSB。

图 2-3 选择要安装的驱动程序



点击“Install Driver”[1]按钮安装驱动。稍等片刻即可完成驱动安装。

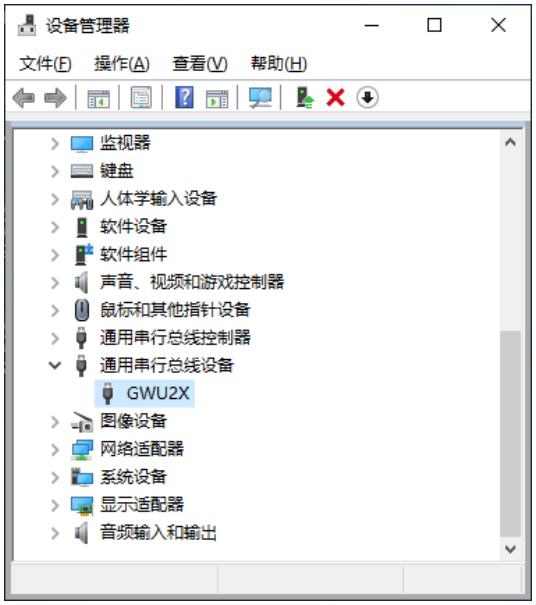
注！

若当前未安装驱动，该按钮显示为“Install Driver”；若当前安装了其他驱动，则显示为“Replace Driver”。

2.2 卸载驱动程序

卸载驱动程序时，连接 GWU2X 设备到电脑，打开 windows 设备管理器，在“通用串行总线设备”列表中，找到 GWU2X 设备。右单击该设备名称，在弹出菜单中选择“卸载设备”选项。

图 2-4 打开设备管理器



在弹出的对话框中，先勾选“删除此设备的驱动程序软件”，再点击卸载按钮，即可卸载驱动程序。

图 2-5 卸载设备

3 libusb_WinUSB 的编程指南

libusb 是一个开源的 USB 函数库。

其官方网址是 https://libusb.info

其源代码已托管至 github：https://github.com/libusb/libusb

可以通过其官方网址下载预编译版本，官方 GCC 版本和 VS 版本，包括动态库和静态库。亦可通过 github 下载源码并自行编译。

libusb 函数说明，请见官方参考：http://libusb.sourceforge.net/api-1.0

3.1 libusb 函数库的初始化和退出

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。

函数声明如下：

```
int libusb_init (libusb_context ** context)

void libusb_exit (libusb_context * ctx)
```

其中，参数 libusb_context 是一个 libusb 上下文结构体，用于保存 libusb 的一些配置参数。如果不指定 libusb_context，则会创建一个默认的上下文结构，若已存在一个上下文结构，则直接使用该结构且不重新初始化。

编程举例如下：

```
int rc = libusb_init(NULL);

if (rc < 0)

   return rc;

libusb_exit(NULL);
```

3.2 打开指定的 USB 设备

可使用 libusb_open_device_with_vid_pid()函数，根据 VID/PID 打开指定

设备。也可使用 libusb_get_device_list()函数，获取所有的 USB 设备，从中选出所需的设备，使用 libusub_open()函数获取该设备的句柄，用于进行后续的操作，函数说明如下：

使用 VID/PID 打开设备：

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

其中，参数 ctx 为初始化 libusb 时产生的上下文结构的地址，若使用默认上下文，则使用 NULL；vendor_id 和 product_id 分别为 USB 设备的 VID 和 PID，Gowin USB 设备的 VID 为 0x33aa，U2X_SPI 设备的 PID 为 0x0020。

返回值为 libusb 在此计算机上查找到的第一个符合的设备的操作句柄指针，否则返回空指针 NULL。

使用举例如下：

```
devh = libusb_open_device_with_vid_pid(NULL, 0x33aa, 0x0020);
if(NULL == devh) {
    printf("Open USB device failed\n");
    goto out;
}
```

获取所有 USB 设备后选取指定设备：

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

其中，参数 ctx 为初始化 libusb 时产生的上下文结构的地址，若使用默认上下文，则使用 NULL；list 为存储设备列表的指针。

在使用结束后，应当使用 libusb_free_device_list（）函数释放该段内存。

若函数执行正确，则返回值为设备数量，并且 list 中保存查找到的设备列表。否则，返回一个小于零的 libusb_error 值。

```
int libusb_open (
    libusb_device *dev,
    libusb_device_handle **dev_handle
```

```
)
```

其中，参数 dev 为设备列表中的设备，dev_handle 为保存返回设备句柄的指针的地址。

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。

使用举例如下：

```
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);
    }
}
```

3.3 接口声明

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）进行首发操作。

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

其中，参数 dev_handle 是设备句柄，interface_number 是接口编号。在 GWU2X 设备中，接口编号为 0。若声明接口成功，则返回 0，否则返还一个小于零的 libusb_error 值。

编程举例：

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

4 U2X_SPI 的参数配置

定义了一个结构体 u2x_spi_config，用于对 U2X_SPI 的参数配置和传输控制：

```
typedef struct _u2x_spi_config_{
    unsigned int          uiSclkFreqKiloHz;
    data_shift_direction  DataShftDir;
    sclk_polarity         ClkPol;
    sclk_phase            ClkPha;
    sel_polarity          SelPol;
    unsigned char         ucAddrLen;
    unsigned char         ucCmdEn;
} u2x_spi_config;
```

- uiSclkFreqKiloHz：用于配置 SCLK 时钟频率，以千赫兹位单位；
- DataShftDir：用于控制传输过程中数据的位移方式，若使用高位在前，则设置为 MSB_FIRST，若使用低位在前，则设置为 LSB_FIRST；
- ClkPol：设置 SCLK 时钟在空闲时的电平，若空闲时为低电平，则设置为 CPOL_0，若空闲时为高电平，则设置为 CPOL_1；
- ClkPha：设置在 SCLK 的哪个边沿发送数据和采样接收数据，若在第一个边沿发送和采样，则设置为 CPHA_0，若在第二个边沿发送和采样，则设置为 CPHA_1。
- SelPol：设置片选信号的有效电平，若使用低电平有效，则设置为 SEL_POL_LO，若使用高电平有效，则设置为 SEL_POL_HI。
- ucAddrLen：控制发送的地址段长度，若不需要发送地址段，则将该参数设置为 0，该参数的最大值为 4，表示地址段长度为 4 个字节；
- ucCmdEn：控制是否发送命令段，若设置为 1，则发送一个字节的命令段，

若设置为 0，则不发送命令段，命令段的长度固定为 1 个字节。

以上设置的值，请参考如下的枚举变量定义：

```
typedef enum _data_shift_direction_ {
      MSB_FIRST,
      LSB_FIRST
} data_shift_direction;

typedef enum _sclk_polarity_ {
      CPOL_0,
      CPOL_1
} sclk_polarity;

typedef enum _sclk_phase_ {
      CPHA_0,
      CPHA_1
} sclk_phase;

typedef enum _sel_polarity_ {
      SEL_POL_LO,
      SEL_POL_HI
} sel_polarity;
```

5 U2X_SPI 的 API 函数

5.1 参数配置

该函数用于配置 U2X_SPI 的参数，若需修改参数配置，则需使用该函数重新进行配置。

```
int u2x_spi_set_config(
    libusb_device_handle *devh,
    u2x_spi_config *pSpiConfig,
    unsigned int uiTimeout
);
```

函数参数:

- devh：libusb 的设备操作句柄；
- pSpiConfig：指向 u2x_spi_config 结构体的指针，使用该结构体对 U2X_SPI 设备进行参数配置；
- uiTimeout：超时参数，单位为毫秒；

返回值：

若运行成功，则返回值为 0，否则返回一个小于零的错误代码。

5.2 发送/接收多个字节数据

该函数可实现全双工的发送和接收、只发送、只接收等三种数据传输模式，以字节为单位，单次传输的最大数据长度为 512 字节。

```
int u2x_spi_read_write_bytes(
    libusb_device_handle *devh,
    unsigned int uiDataByteCnt,
```

```
    unsigned char *pucWriteData,
    unsigned char *pucReadData,
    unsigned int uiAddr,
    unsigned char ucCmd,
    unsigned int uiDummyCnt,
    u2x_spi_config *pSpiConfig,
    unsigned int uiTimeout
);
```

函数参数:

- devh：libusb 的设备操作句柄；
- uiDataByteCnt：控制发送和接收数据的字节数；
- pucWriteData：指向保存要发送数据的指针，若不需要发送数据，则将该参数设置为 NULL；
- pucReadData：指向用于保存接收数据的指针，若不需要接收数据，则将该参数设置为 NULL；
- uiAddr：地址段数据，若参数配置中地址段的长度为非 0，则发送该地址数据，地址段的最大长度为 4 字节；
- ucCmd：命令段数据，若参数配置中使能了命令段发送，则在命令段发送该字节数据；
- uiDummyCnt：DummySCLK 段，即命令段、地址段与数据段之间的等待时钟周期数。若该参数为非零值，则在发送完成命令段和地址段后，发送指定周期数的 SCLK，然后再发送数据段，该参数的最大值为 65536，若不需要发送 DummySCLK 段，则将该参数设置为 0；
- pSpiConfig：指向 u2x_spi_config 结构体的指针，使用该结构体对 U2X_SPI 设备的读写进行控制；
- uiTimeout：超时参数，单位为毫秒；

返回值：

若运行成功，则返回值为 0，否则返回一个小于零的错误代码。

5.3 发送/接收多个 BIT 数据

该函数可实现全双工的发送和接收、只发送、只接收等三种数据传输模式，以 BIT 为单位。单次传输的最大数据长度为 2048 bits（256 字节）。

```
int u2x_spi_read_write_bits(
    libusb_device_handle *devh,
```

```
    unsigned int uiDataBitCnt,
    unsigned char *pucWriteData,
    unsigned char *pucReadData,
    unsigned int uiAddr,
    unsigned char ucCmd,
    unsigned int uiDummyCnt,
    u2x_spi_config *pSpiConfig,
    unsigned int uiTimeout
);
```

函数参数:

- devh：libusb 的设备操作句柄；
- uiDataBitCnt：控制发送和接收数据的 BIT 数；
- pucWriteData：指向保存要发送数据的指针，若不需要发送数据，则将该参数设置为 NULL；
- pucReadData：指向用于保存接收数据的指针，若不需要接收数据，则将该参数设置为 NULL；
- uiAddr：地址段数据，若参数配置中地址段的长度为非 0，则发送该地址数据，地址段的最大长度为 4 字节；
- ucCmd：若参数配置中使能了命令段发送，则在命令段发送该字节数据；
- uiDummyCnt：DummySCLK 段，即命令段、地址段与数据段之间的等待时钟周期数。若该参数为非零值，则在发送完成命令段和地址段后，发送指定周期数的 SCLK，然后再发送数据段，该参数的最大值为 65536，若不需要发送 DummySCLK 段，则将该参数设置为 0；
- pSpiConfig：指向 u2x_spi_config 结构体的指针，使用该结构体对 U2X_SPI 设备的读写进行控制；
- uiTimeout：超时参数，单位为毫秒；

返回值：

若运行成功，则返回值为 0，否则返回一个小于零的错误代码。

5.4 编程示例

```
// 全局变量
static struct libusb_device_handle *devh = NULL;
static u2x_spi_config SpiConfig;
```

```
u2x_spi_config *pSpiConfig = &SpiConfig;
// Main Function
int main(int argc, char *argv[])
{
    unsigned char ucWrData[64];
    unsigned char ucRdData[64];
    unsigned long long ullWrData = 0x1234567812345678;
    unsigned long long ullRdData = 0x0;
int rc = 0;
rc = libusb_init(NULL);
if (rc < 0)
  return rc;
     // 打开 GWU2X 设备
devh = libusb_open_device_with_vid_pid(NULL, 0x33aa, 0x0120);
if(NULL == devh) {
printf("Open USB device failed\n");
goto out;
}
rc = libusb_claim_interface(devh, 0);
if (rc < 0) {
  goto out;
}
for(i = 0; i < DATA_BYTE; i++) {
  ucWrData[i] = i;
}
// 配置 U2X_SPI 参数
pSpiConfig->DataShftDir      = MSB_FIRST;
pSpiConfig->ClkPol           = CPOL_0;
pSpiConfig->ClkPha           = CPHA_0;
pSpiConfig->SelPol           = SEL_POL_LO;
pSpiConfig->uiSclkFreqKiloHz  = 10000; // SCLK 频率为 10MHz
```

```
pSpiConfig->ucAddrLen        = 3; // 地址段长度为 3 字节
pSpiConfig->ucCmdEn          = 1;  // 使能命令段
u2x_spi_set_config(devh, pSpiConfig, 1000);

// 全双工模式同时发送和接收 64 字节数据
u2x_spi_read_write_bytes(devh,
    64,  // 传输数据长度为 64 字节
    ucWrData, // 待发送数据
    ucRdData, // 保存接收数据
    0xaabbcc, // 地址段为 0xaabbcc
    0x32,      // 命令段为 0x32
    8,         // 命令段，地址段后发送 8 个周期的 Dummy SCLK
    pSpiConfig, // 指向参数配置结构体的指针
    1000); // 超时参数设置为 1000 毫秒

     // 全双工模式同时发送和接收 56BIT 数据
u2x_spi_read_write_bits(devh,
    56,    // 传输数据长度位 56 BIT
    (unsigned char *)(&ullWrData), // 待发送数据
    (unsigned char *)(&ullRdData), / 保存接收数据
    0xaabbcc, // 地址段为 0xaabbcc
    0x32, // 命令段为 0x32
    8,    // 命令段，地址段后发送 8 个周期的 Dummy SCLK
    pSpiConfig, // 指向参数配置结构体的指针
    1000); // 超时参数设置为 1000 毫秒

//关闭设备并退出
out:
if(devh)
  libusb_close(devh);
libusb_exit(NULL);
```

```
    return 0;
    }
```

6 错误代码

API 函数运行正常时，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下表所示。

表 6-1 错误代码列表

Value	Enumerator	
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
-13	U2X_SPI_ERROR_USBTRANS_ERR	USB 数据传输错误
-14	U2X_SPI_ERROR_INVALID_PARAM	该参数设定值不可用
-15	U2X_SPI_ERROR_TIMEOUT	U2X_SPI 传输超时
-16	U2X_SPI_ERROR_CMD_ERR	U2X_SPI 指令错误
-99	ERROR_OTHER	其他错误

术语、缩略语

表 A-1 中列出了本手册中出现的相关术语、缩略语及相关释义。

表 A-2 术语、缩略语

术语、缩略语	全称	含义
USB	Universal Serial Bus	通用串行总线
SPI	Serial Peripheral Interface	串行外设接口

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或建议，可直接与公司联系：

网址：www.gowinsemi.com.cn

E-mail：support@gowinsemi.com

Tel：+86 755 8262 0391


GOWIN高云
智慧逻辑 定制未来