

# GWU2X プログラミング ガイド\_U2X\_SPI

UG1004-1.0J, 2021-06-29

## 著作権について（2021）

著作権に関する全ての権利は、**Guangdong Gowin Semiconductor Corporation** に留保されています。

**GOWIN高云**、Gowin、及びGOWINSEMIは、当社により、中国、米国特許商標庁、及びその他の国において登録されています。商標又はサービスマークとして特定されたその他全ての文字やロゴは、それぞれの権利者に帰属しています。何れの団体及び個人も、当社の書面による許可を得ず、本文書の内容の一部もしくは全部を、いかなる視聴覚的、電子的、機械的、複写、録音等の手段によりもしくは形式により、伝搬又は複製をしてはなりません。

## 免責事項

当社は、**GOWINSEMI Terms and Conditions of Sale**（GOWINSEMI取引条件）に規定されている内容を除き、（明示的か又は黙示的かに拘わらず）いかなる保証もせず、また、知的財産権や材料の使用によりあなたのハードウェア、ソフトウェア、データ、又は財産が被った損害についても責任を負いません。本文書における全ての情報は、予備的情報として取り扱われなければなりません。当社は、事前の通知なく、いつでも本文書の内容を変更することができます。本文書を参照する何れの団体及び個人も、最新の文書やエラッタ（不具合情報）については、当社に問い合わせる必要があります。

## バージョン履歴

日付	バージョン	説明
2021/06/29	1.0J	初版。

# 目次

目次.....	i
図一覧 .....	ii
表一覧 .....	iii
<b>1 機能の紹介 .....</b>	<b>1</b>
<b>2 ドライバーのインストールとアンインストール.....</b>	<b>2</b>
2.1 Zadig を使用してドライバーをインストール .....	2
2.2 ドライバーのアンインストール .....	4
<b>3 libusb_WinUSB プログラミングの説明.....</b>	<b>6</b>
3.1 libusb 関数ライブラリの初期化と終了 .....	6
3.2 指定された USB デバイスを開きます .....	7
3.3 インターフェース宣言 .....	9
<b>4 U2X_SPI パラメータの構成.....</b>	<b>10</b>
<b>5 U2X_SPI の API 関数.....</b>	<b>12</b>
5.1 パラメータの構成 .....	12
5.2 複数バイトのデータの送受信 .....	12
5.3 複数ビットのデータの送受信 .....	14
5.4 プログラミング例 .....	15
<b>6 エラーコード.....</b>	<b>18</b>
用語、略語 .....	20
テクニカル・サポートとフィードバック .....	21

## 図一覧

図 2-1 “List All Device”オプションを選択.....	2
図 2-2 デバイスを選択 .....	3
図 2-3 ドライバーを選択.....	3
図 2-4 デバイスマネージャーを開く .....	4
図 2-5 ドライバーのアンインストール.....	5

# 表一覧

表 6-1 エラーコード一覧.....	18
表 A-1 用語、略語 .....	20

# 1 機能の紹介

GWU2X は、USB2SPI 変換を実現できる USB からマルチプロトコルへのコンバータであり、最大 12MHz（現在のテスト結果）の SCLK クロック周波数をサポートします。

標準の SPI マスターモードをサポートします。全二重データ送受信機能をサポートし、オプションの命令セグメント、アドレスセグメント、DummySCLK セグメント、およびデータセグメントをサポートします。

# 2 ドライバーのインストールとアンインストール

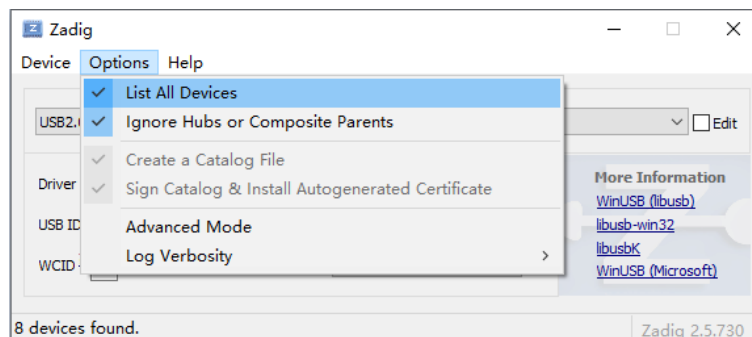
オープンソースの USB 関数ライブラリ libusb を使用して、GWU2X\_SPI をプログラミングすることができます。この関数ライブラリを使用してプログラミングする場合は、WINDOWS での USB ドライバーの WinUSB.sys をインストールする必要があります。

ドライバーをインストールするにはオープンソースのドライバー・インストー・ルツール Zadig (<https://zadig.akeo.ie/>) を使用できます。ドライバーをインストールするには、管理者権限が必要です。

## 2.1 Zadig を使用してドライバーをインストール

GWU2X デバイスをコンピューターの USB インターフェースに接続し、Zadig をダブルクリックして開き（管理者権限が必要）、Options > List All Device をチェックすると、コンピュータに接続されているすべての USB デバイスが一覧表示されます。

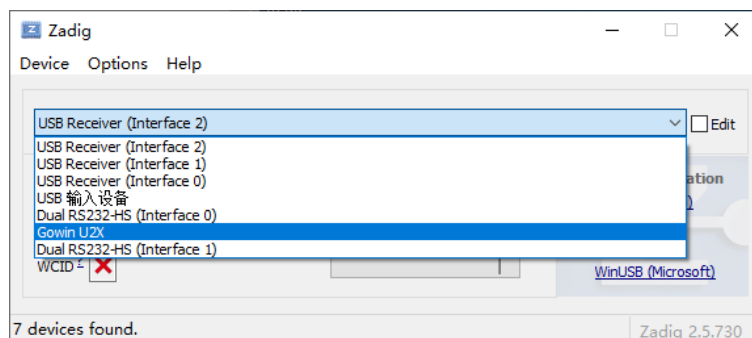
図 2-1 “List All Device” オプションを選択



デバイス GWU2X を選択して、ドライバーをインストールします。

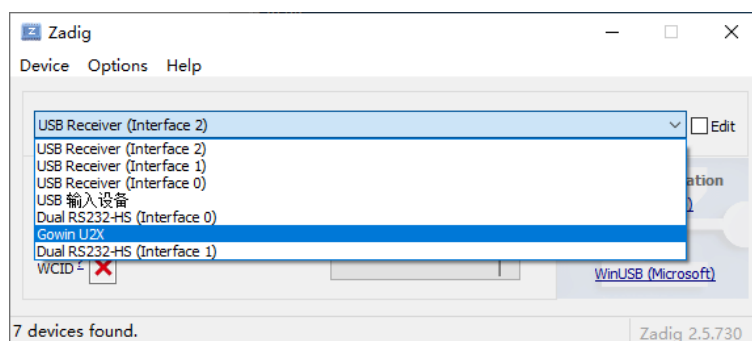


図 2-2 デバイスを選択



インストールするドライバーを選択します。libusb + WinUSB を使用する  
場合、WinUSB を選択してください。

図 2-3 ドライバーを選択



“Install Driver”<sup>[1]</sup> ボタンをクリックして、ドライバーをインストールし  
ます。

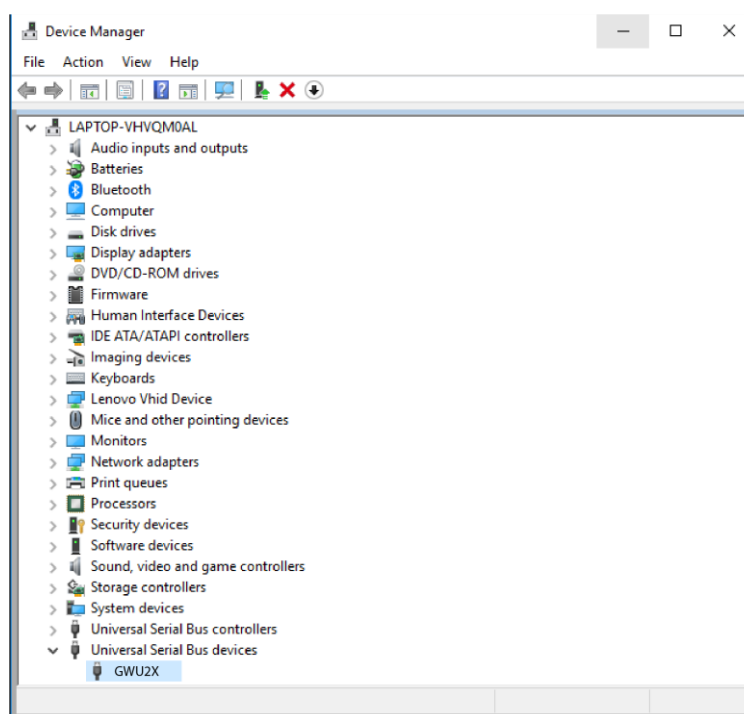
注記：

ドライバーが現在インストールされていない場合は“Install Driver”、他のドライバーがイ  
ンストールされている場合は“Replace Driver”と表示されます。

## 2.2 ドライバーのアンインストール

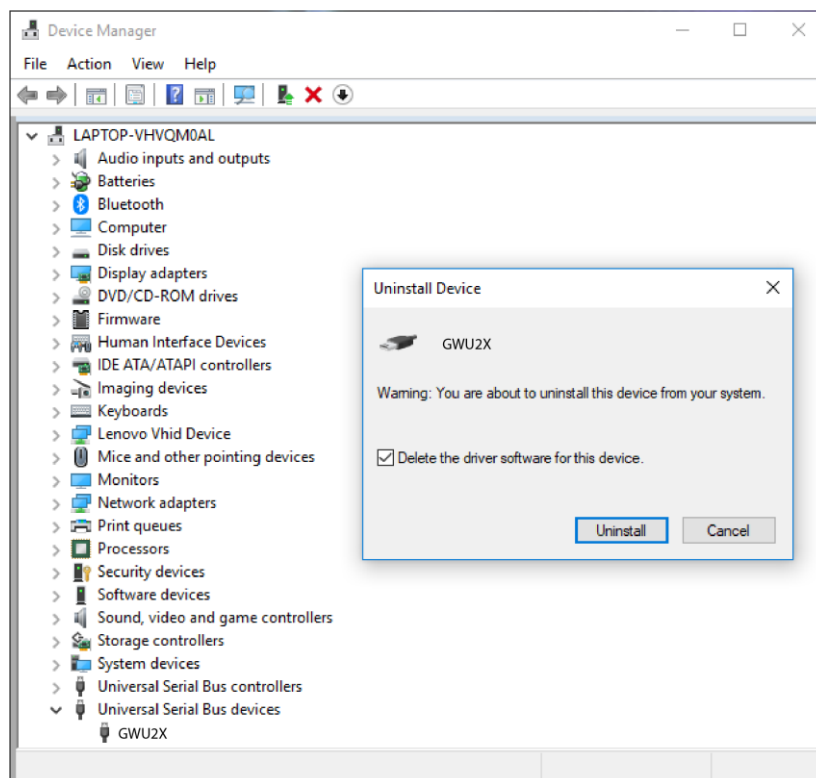
ドライバーをアンインストールするときは、GWU2X デバイスをコンピューターに接続し、Windows デバイスマネージャーを開いて、“ユニバーサル シリアル バス デバイス” リストで GWU2X デバイスを特定します。デバイス名を右クリックして、右クリックメニューの“削除”オプションを選択します。

図 2-4 デバイスマネージャーを開く



ポップアップダイアログボックスで、“このデバイスのドライバー ソフトウェアを削除する”にチェックを入れてアンインストールします。

図 2-5 ドライバーのアンインストール



# 3 libusb\_WinUSB プログラミングの説明

libusb はオープンソースの USB 関数ライブラリです。

その公式ウェブサイトは <https://libusb.info> です。

そのソースコードは <https://github.com/libusb/libusb> から入手できます。

動的ライブラリと静的ライブラリを含む、コンパイル済みバージョン、公式 GCC バージョン、および VS バージョンは、公式 Web サイトからダウンロードできます。github からソースコードをダウンロードして、自分でコンパイルすることもできます。

libusb 関数宣言については、公式リファレンス <http://libusb.sourceforge.net/api-1.0> を参照してください。

## 3.1 libusb 関数ライブラリの初期化と終了

libusb を使用してプログラミングする場合は、関数 `libusb_init()` を呼び出して初期化する必要があります。使用後は、関数 `libusb_exit()` を呼び出して終了させる必要があります。

関数宣言は次のとおりです。

```
int libusb_init (libusb_context ** context)

void libusb_exit (libusb_context * ctx)
```

パラメータ `libusb_context` は libusb コンテキスト構造体であり、libusb のいくつかの構成パラメータを保存するために使用されます。libusb\_context が指定されていない場合、デフォルトのコンテキスト構造が作成されます。コンテキスト構造がすでに存在する場合、その構造は再初期化せずに直接使用されます。

プログラミング例：

```
int rc = libusb_init(NULL);

if (rc < 0)
```

```
return rc;

libusb_exit(NULL);
```

## 3.2 指定された USB デバイスを開きます

`libusb_open_device_with_vid_pid()` 関数を使用して、VID/PID に従って指定されたデバイスを開くことができます。`libusb_get_device_list()` 関数を使用してすべての USB デバイスを取得し、それらから必要なデバイスを選択し、`libusb_open()` 関数を使用して後続の操作のためにデバイスのハンドルを取得することもできます。関数の説明は次のとおりです。

VID/PID を使用してデバイスを開きます。

```
libusb_device_handle* libusb_open_device_with_vid_pid(

    libusb_context * ctx,

    uint16_t vendor_id,

    uint16_t product_id

);
```

その中で、パラメータ `ctx` は、`libusb` が初期化されるときに生成されるコンテキスト構造のアドレスです。デフォルトのコンテキストが使用される場合、`NULL` が使用されます。`vendor_id` と `product_id` は、それぞれ USB デバイスの VID と PID です。Gowin USB デバイスの VID は `0x33aa` であり、U2X\_SPI デバイスの PID は `0x0020` です。

返り値は、このコンピュータで `libusb` によって最初に一致したデバイスの操作ハンドルポインターです。それ以外の場合は、`NULL` ポインタが返されます。

使用例は次のとおりです。

```
devh = libusb_open_device_with_vid_pid(NULL, 0x33aa, 0x0020);

if(NULL == devh) {

    printf("Open USB device failed\n");

    goto out;

}
```

すべての USB デバイスを取得した後、指定したデバイスを選択します。

```
ssize_t libusb_get_device_list (

    libusb_context *ctx,

    libusb_device *** list

)
```

その中で、パラメータ **ctx** は、**libusb** が初期化されるときに生成されるコンテキスト構造のアドレスです。デフォルトのコンテキストが使用される場合、**NULL** が使用されます。**list** はデバイスリストへのポインタです。

使用后、**libusb\_free\_device\_list ()** 関数を使用してそのメモリを解放する必要があります。

関数が正しく実行された場合、返り値はデバイスの数であり、見つかったデバイスのリストが **list** に保存されます。それ以外の場合は、ゼロ未満の **libusb\_error** 値が返されます。

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

その中で、パラメータ **dev** はデバイスリスト内のデバイスであり、**dev\_handle** は返されるデバイスハンドルへのポインタのアドレスです。

実行が成功した場合、**0** が返されます。成功しなかった場合、ゼロ未満の **libusb\_error** 値が返されます。

使用例は次のとおりです。

```
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 インターフェース宣言

USB デバイスには通常、1 つ以上のインターフェースが含まれています。`libusb` がインターフェースを使用する場合、最初にインターフェースを宣言（`claim interface`）する必要があります。宣言が成功すると、インターフェースが正常に開かれ、インターフェースに含まれるエンドポイントに対して受送信できることになります。

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

その中で、パラメータ `dev_handle` はデバイスハンドルであり、`interface_number` はインターフェース番号です。GWU2X デバイスでは、インターフェース番号は 0 です。インターフェースの宣言が成功した場合は 0 が返され、そうでない場合はゼロ未満の `libusb_error` 値が返されます。

プログラミング例

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

# 4 U2X\_SPI パラメータの構成

U2X\_SPI のパラメータ構成と転送制御用の構造体 `u2x_spi_config` が定義されています。

```
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** : SCLK クロック周波数 (KHz) の構成用に使用されます。
- **DataShftDir** : 転送中のデータシフト方法を制御するために使用されます。上位ビット最優先の場合は **MSB\_FIRST** に設定し、下位ビット最優先の場合は **LSB\_FIRST** に設定します。
- **ClkPol** : アイドル時の SCLK クロックのレベルを設定します。アイドル時の **Low** の場合は **CPOL\_0** に設定し、アイドル時の **High** の場合は **CPOL\_1** に設定します。
- **ClkPha** : どの SCLK のエッジでデータを送信し、受信データをサンプリングするかを設定します。最初のエッジで送信およびサンプリングする場合は **CPHA\_0** に設定し、2 番目のエッジで送信およびサンプリングする場合は **CPHA\_1** に設定します。
- **SelPol** : チップセレクト信号の有効レベルを設定します。アクティブ **Low** の場合は **SEL\_POL\_LO** に設定し、アクティブ **High** の場合は



SEL\_POL\_HI に設定します。

- **ucAddrLen** : 送信されるアドレスセグメントの長さを制御します。アドレスセグメントを送信する必要がない場合は、このパラメータを **0** に設定します。このパラメータの最大値は **4** です。これは、アドレスセグメントの長さが **4** バイトであることを意味します。
- **ucCmdEn** : 命令セグメントを送信するかどうかを制御します。**1** に設定する場合は **1** バイトの命令セグメントが送信され、**0** に設定する場合は命令セグメントは送信されません。命令セグメントの長さは **1** バイトに固定されます。

上記の値の設定については、次の列挙変数の定義を参照してください。

```
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 の API 関数

## 5.1 パラメータの構成

この関数は、U2X\_SPI のパラメータを構成するために使用されます。パラメータの構成を変更するには、この関数を使用して再構成する必要があります。

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

パラメータ：

- devh : libusb のデバイス操作ハンドル。
- pSpiConfig : u2x\_spi\_config 構造体へのポインタ。この構造体を使用して U2X\_SPI デバイスのパラメータを構成します。
- uiTimeout : タイムアウトパラメータ。単位はミリ秒です。

返り値：

実行が成功した場合、0 が返されます。成功しなかった場合、ゼロ未満のエラーコードが返されます。

## 5.2 複数バイトのデータの送受信

この関数は、全二重送受信、送信のみ、受信のみの 3 つのデータ転送モードを実現できます。単位はバイトで、1 回の転送の最大データ長は 512 バイトです。

```
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  
);
```

パラメータ :

- **devh** : libusb のデバイス操作ハンドル。
- **uiDataByteCnt** : 送受信データのバイト数を制御します。
- **pucWriteData** : 送信されるデータを指すポインタ。データを送信する必要がない場合は、このパラメータを **NULL** に設定します。
- **pucReadData** : 受信されるデータを指すポインタ。データを受信する必要がない場合は、このパラメータを **NULL** に設定します。
- **uiAddr** : アドレスセグメントデータ。パラメータ構成におけるアドレスセグメントの長さが **0** でない場合、このアドレスデータが送信されます。アドレスセグメントの最大長は **4** バイトです。
- **ucCmd** : 命令セグメントデータ。パラメータ構成において命令セグメント送信が有効になっている場合、このバイトデータは命令セグメントで送信されます。
- **uiDummyCnt** : **DummySCLK** セグメント。つまり命令セグメント、アドレスセグメントと、データセグメント間の待機クロックサイクル数です。このパラメータがゼロでない場合、命令セグメントとアドレスセグメントを送信した後、指定されたサイクル数の **SCLK** を送信してから、データセグメントを送信します。このパラメータの最大値は **65536** です。**DummySCLK** を送信する必要がない場合、このパラメータを **0** に設定します。
- **pSpiConfig** : **u2x\_spi\_config** 構造体を指すポインタ。この構造体を使用して、**U2X\_SPI** デバイスの読み出しと書き込みを制御します。
- **uiTimeout** : タイムアウトパラメータ。単位はミリ秒です。

返り値 :

実行が成功した場合、0 が返されます。成功しなかった場合、ゼロ未満のエラーコードが返されます。

## 5.3 複数ビットのデータの送受信

この関数は、全二重送受信、送信のみ、受信のみの 3 つのデータ転送モードを実現できます。単位はビットです。1 回の転送の最大データ長は 2048 ビット（256 バイト）です。

```
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  
  
);
```

パラメータ：

- **devh** : libusb のデバイス操作ハンドル。
- **uiDataBitCnt** : 送受信データのビット数を制御します。
- **pucWriteData** : 送信されるデータを指すポインタ。データを送信する必要がない場合は、このパラメータを **NULL** に設定します。
- **pucReadData** : 受信されるデータを指すポインタ。データを受信する必要がない場合は、このパラメータを **NULL** に設定します。
- **uiAddr** : アドレスセグメントデータ。パラメータ構成におけるアドレスセグメントの長さが 0 でない場合、このアドレスデータが送信されます。アドレスセグメントの最大長は 4 バイトです。
- **ucCmd** : 命令セグメントデータ。パラメータ構成において命令セグメント送信が有効になっている場合、このバイトデータは命令セグメントで送信されます。
- **uiDummyCnt** : DummySCLK セグメント。つまり命令セグメント、アドレスセグメントと、データセグメント間の待機クロックサイクル数です。このパラメータがゼロでない場合、命令セグメントとアドレスセグメン

トを送信した後、指定されたサイクル数の **SCLK** を送信してから、データセグメントを送信します。このパラメータの最大値は **65536** です。**DummySCLK** を送信する必要がある場合、このパラメータを **0** に設定します。

- **pSpiConfig** : **u2x\_spi\_config** 構造体を指すポインタ。この構造体を使用して、**U2X\_SPI** デバイスの読み出しと書き込みを制御します。
- **uiTimeout** : タイムアウトパラメータ。単位はミリ秒です。

返り値 :

実行が成功した場合、**0** が返されます。成功しなかった場合、ゼロ未満のエラーコードが返されます。

## 5.4 プログラミング例

```
// グローバル変数

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;

    // GWU2X デバイスを開きます

    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;

}

// U2X_SPI パラメータを構成します

pSpiConfig->DataShftDir      = MSB_FIRST;

pSpiConfig->ClkPol           = CPOL_0;

pSpiConfig->ClkPha           = CPHA_0;

pSpiConfig->SelPol            = SEL_POL_LO;

pSpiConfig->uiSclkFreqKiloHz = 10000; //SCLK 周波数は 10MHz です

pSpiConfig->ucAddrLen         = 3; //アドレスセグメントの長さは 3 バイトです

pSpiConfig->ucCmdEn           = 1; // 命令セグメントを有効にします

u2x_spi_set_config(devh, pSpiConfig, 1000);


// 全二重モードで 64 バイトのデータを同時に送受信します

u2x_spi_read_write_bytes(devh,

    64, //転送データの長さは 64 バイトです

    ucWrData, // 送信されるデータ

    ucRdData, //受信されたデータ

    0xaabbcc, //アドレスセグメントは 0xaabbcc です

    0x32,     // 命令セグメントは 0x32 です

    8,        // 命令セグメント、アドレスセグメントの後に 8 サイクルの Dummy SCLK を送信し
    ます

    pSpiConfig, // パラメータ構成構造体へのポインタ

    1000); // タイムアウトパラメータは 1000 ミリ秒に設定されます


// 全二重モードで 56 ビットのデータを同時に送受信します
```

```
u2x_spi_read_write_bits(devh,  
  
    56, // 転送データの長さは 56 ビットです  
  
    (unsigned char *)(&ullWrData), // 送信されるデータ  
  
    (unsigned char *)(&ullRdData), // 受信されたデータ  
  
    0xaabbcc, // アドレスセグメントは 0xaabbcc です  
  
    0x32, // アドレスセグメントは 0x32 です  
  
    8, // 命令セグメント、アドレスセグメントの後に 8 サイクルの Dummy SCLK を送信し  
    // ます  
  
    pSpiConfig, // パラメータ構成構造体へのポインタ  
  
    1000); // タイムアウトパラメータは 1000 ミリ秒に設定されます  
  
// デバイスを閉じて終了します  
  
out:  
  
if (devh)  
  
    libusb_close(devh);  
  
libusb_exit(NULL);  
  
return 0;  
  
}
```

# 6 エラーコード

API 関数が正常に実行されている場合、戻り値は **0** です。それ以外の場合は、負の数を返します。

0 以外の戻り値で表されるエラーの意味を次の表に示します。

表 6-1 エラーコード一覧

Value	Enumerator	
0	SUCCESS	エラーなし
-1	USB_ERROR_IO	USB 入出力エラー
-2	USB_ERROR_INVALID_PARAM	USB パラメータエラー
-3	USB_ERROR_ACCESS	デバイスにアクセスするための権限が不十分です
-4	USB_ERROR_NO_DEVICE	USB デバイスが見つかりません(デバイスが切断されています)
-5	USB_ERROR_NOT_FOUND	エンティティ (Entity) が見つかりません
-6	USB_ERROR_BUSY	USB デバイスがビジーです
-7	USB_ERROR_TIMEOUT	タイムアウト
-8	USB_ERROR_OVERFLOW	メモリオーバーフロー
-9	USB_ERROR_PIPE	パイプエラー
-10	USB_ERROR_INTERRUPTED	システム関数が中断されました
-11	USB_ERROR_NO_MEM	メモリ不足
-12	USB_ERROR_NOT_SUPPORTED	この操作は現在のプラットフォームではサポートされていません
-13	U2X_SPI_ERROR_USBTRANS_ERR	USB データ転送エラー
-14	U2X_SPI_ERROR_INVALID_PARAM	パラメータ無効



Value	Enumerator	
-15	U2X_SPI_ERROR_TIMEOUT	U2X_SPI 転送タイムアウト
-16	U2X_SPI_ERROR_CMD_ERR	U2X_SPI 命令エラー
-99	ERROR_OTHER	その他のエラー

## 用語、略語

表 A-1 に、本マニュアルで使用される用語、略語、及びその意味を示します。

表 A-1 用語、略語

用語、略語	正式名称	意味
USB	Universal Serial Bus	ユニバーサル・シリアル・バス
SPI	Serial Peripheral Interface	シリアル・ペリフェラル・インターフェース

## テクニカル・サポートとフィードバック

GOWIN セミコンダクターは、包括的な技術サポートをご提供しています。使用に関するご質問、ご意見については、直接弊社までお問い合わせください。

Web サイト : [www.gowinsemi.com](http://www.gowinsemi.com)

E-mail : [support@gowinsemi.com](mailto:support@gowinsemi.com)

