



GWU2X Programming Guide_U2X_SPI

UG1004-1.0E, 6/29/2021

Copyright © 2021 Guangdong Gowin Semiconductor Corporation. All Rights Reserved.

GOWIN, Gowin, and GOWINSEMI are trademarks of Guangdong Gowin Semiconductor Corporation and are registered in China, the U.S. Patent and Trademark Office, and other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders. No part of this document may be reproduced or transmitted in any form or by any denotes, electronic, mechanical, photocopying, recording or otherwise, without the prior written consent of GOWINSEMI.

Disclaimer

GOWINSEMI assumes no liability and provides no warranty (either expressed or implied) and is not responsible for any damage incurred to your hardware, software, data, or property resulting from usage of the materials or intellectual property except as outlined in the GOWINSEMI Terms and Conditions of Sale. All information in this document should be treated as preliminary. GOWINSEMI may make changes to this document at any time without prior notice. Anyone relying on this documentation should contact GOWINSEMI for the current documentation and errata.

Revision History

| Date | Version | Description |
|-----------|---------|----------------------------|
| 6/29/2021 | 1.0E | Initial version published. |

Contents

| | |
|---|------------|
| Contents | i |
| List of Figures | ii |
| List of Tables | iii |
| 1 General Description | 1 |
| 2 Driver Installation and Uninstallation | 2 |
| 2.1 Use Zadig to Install Driver | 2 |
| 2.2 Uninstall Driver | 4 |
| 3 libusb_WinUSB Programming..... | 5 |
| 3.1 Libusb Initialization and Exit | 5 |
| 3.2 Open the Specified USB Device..... | 6 |
| 3.3 Interface Declaration | 8 |
| 4 Parameter Configuration | 9 |
| 5 U2X_SPI API Functions..... | 11 |
| 5.1 Parameter Configuration | 11 |
| 5.2 Send / Receive Byte Data | 11 |
| 5.3 Send / Receive Multiple Bits of Data | 13 |
| 5.4 Programming Example | 14 |
| 6 Error Code..... | 17 |
| Terminology and Abbreviations | 19 |
| Support and Feedback..... | 20 |

List of Figures

Figure 2-1 Check “List All Device” Option 2

Figure 2-2 Select the Device that Requires Driver Installation 3

Figure 2-3 Select the Driver Program to be Installed 3

Figure 2-4 Open Device Manager 4

Figure 2-5 Uninstall Device 4

List of Tables

| | |
|---|----|
| Table 6-1 Error Code List | 17 |
| Table A-1 Terminology and Abbreviations | 19 |

1 General Description

GWU2X is a USB to multiple protocol converter that enables SCLK function conversion and supports up to 12MHz SCLK clock frequency (current test result).

It supports standard SPI host mode; It supports full duplex data transceiver function, optional command fields, address fields, DummySCLK fields, and data fields.

2 Driver Installation and Uninstallation

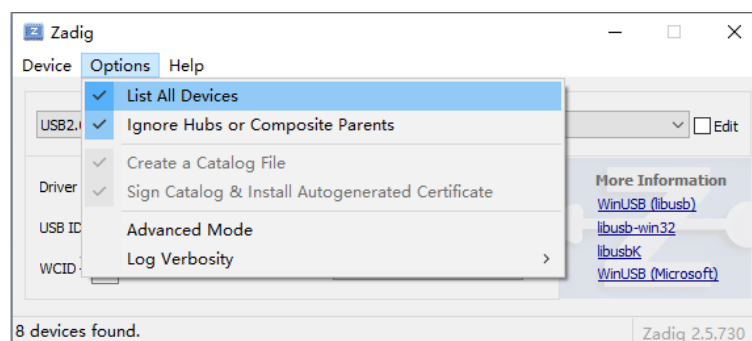
GWU2X_SPI can be programmed using libusb, the open source USB function library. To program with this function library, the “WinUSB.sys” USB driver program needs to be installed.

You can use Zadig(<https://zadig.akeo.ie/>), the open source driver installation tool, to install driver. The driver installation requires administrator privileges.

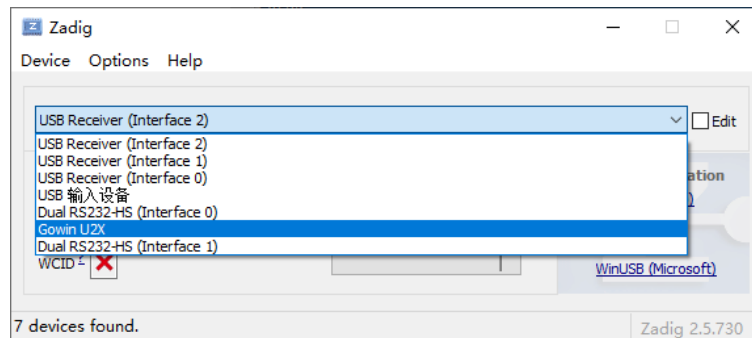
2.1 Use Zadig to Install Driver

Connect GWU2X device to the computer USB interface, double-click to open Zadig (administrator privileges required), click Options, and check the "List All Device" option. All USB devices connected to the computer will be listed.

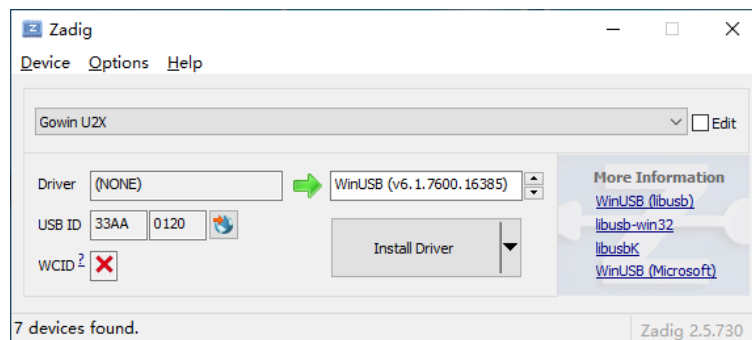
Figure 2-1 Check “List All Device” Option



Select GWU2X, the device that requires driver installation.

Figure 2-2 Select the Device that Requires Driver Installation

Select the driver to be installed, use libusb+WinUSB, and select WinUSB.

Figure 2-3 Select the Driver Program to be Installed

Click "Install Driver". The driver will be installed after a few moments.

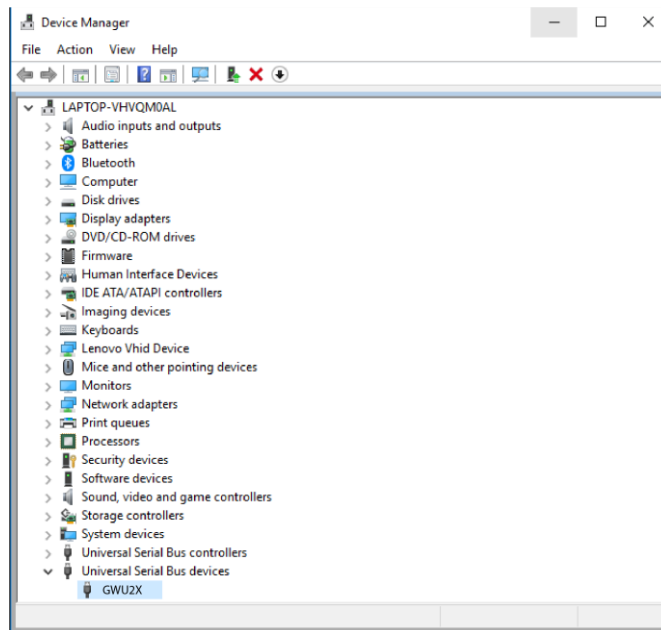
Note!

The button displays "Install Driver" if the driver is not currently installed, and "Replace Driver" if another driver is currently installed.

2.2 Uninstall Driver

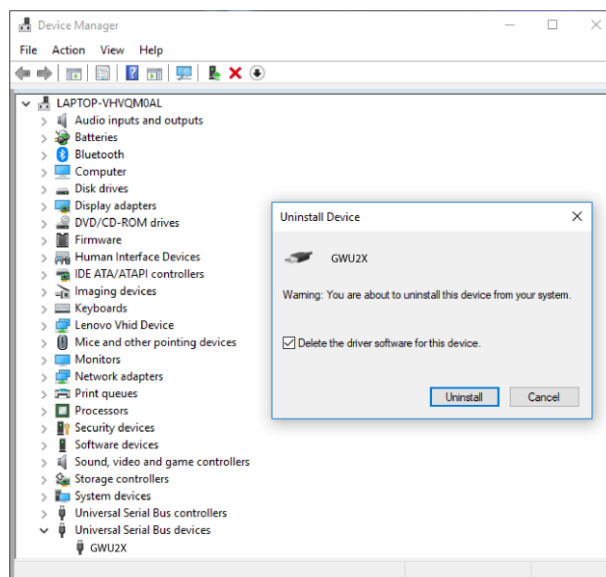
To uninstall the driver, connect GWU2X device to the computer, open the windows device manager, and find GWU2X device in the "Universal Serial Bus Devices" list. Right-click on the device name and select the "Uninstall Device" option in the pop-up menu.

Figure 2-4 Open Device Manager



In the pop-up dialog box, first check "Remove driver software for this device", and then click the "Uninstall" button to uninstall the driver.

Figure 2-5 Uninstall Device



3 libusb_WinUSB Programming

libusb is an open source USB function library.

The official website is: <https://libusb.info>

The source code is hosted on github at: <https://github.com/libusb/libusb>

You can download the pre-compiled version, the official GCC version and VS version, including dynamic and static libraries, through the official website. You can also download the source code through github and compile it on your own.

For libusb function descriptions, see the official reference at: <http://libusb.sourceforge.net/api-1.0>

3.1 Libusb Initialization and Exit

When programming with libusb, you need to call the function `libusb_init()` to initialize it first, and at the end of use, you should call the function `libusb_exit()` to exit it from the system.

Function declarations are as follows:

```
int libusb_init (libusb_context ** context)

void libusb_exit (libusb_context * ctx)
```

The parameter `libusb_context` is a libusb context struct that saves some configuration parameters for libusb. If `libusb_context` is not specified, a default context struct will be created, or if one already exists, it will be used directly and not reinitialized.

Programming examples are as follows:

```
int rc = libusb_init(NULL);

if (rc < 0)

    return rc;
```

```
libusb_exit(NULL);
```

3.2 Open the Specified USB Device

You can use the `libusb_open_device_with_vid_pid()` function to open the specified device based on VID/PID. You can also use the `libusb_get_device_list()` function to get all the USB devices, select the desired device from them, and use the `libusb_open()` function to get the handle of the device for subsequent operations. The function declaration is as follows:

Open the device with VID/PID:

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

The parameter `ctx` is the address of the context struct generated when initializing libusb. If the default context is used, use `NULL`. `vendor_id` and `product_id` are the VID and PID of the USB device, respectively. The VID of Gowin USB device is `0x33aa`, and the PID of the U2X_SPI device is `0x0020`.

The return value is the pointer to the operation handle of the first matching device found by libusb on this computer, otherwise it returns the null pointer `NULL`.

Examples of use are as follows:

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

Select the specified device after getting all USB devices.

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

The parameter `ctx` is the address of the context struct generated when initializing `libusb`. If the default context is used, use `NULL`. “list” is the pointer to storage device list.

At the end of use, the memory should be freed using the `libusb_free_device_list()` function.

If the function is executed correctly, the return value is the number of devices and the list saves the list of found devices. Otherwise, a `libusb_error` value less than zero is returned.

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

The parameter `dev` is the device in the device list, and `dev_handle` is the address that saves the pointer of the returned device handle.

If the device is opened successfully, the return value is zero, otherwise an `libusb_error` value less than zero is returned.

Examples of use are as follows:

```
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 Interface Declaration

USB devices usually contain one or more interfaces. libusb needs to declare the interface first when using the interface, and when the declaration is successful, it means that the interface is successfully opened and the endpoint contained in the interface can be received/transmitted.

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

The parameter dev_handle is the device handle; interface_number is the number of interface. In GWU2X device, the interface number is 0. If the interface is declared successfully, the return value is zero, otherwise an libusb_error value less than zero is returned.

Programming Examples:

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

4 Parameter Configuration

The struct of `u2x_spi_config` is used to configure and control the transmission of U2X_SPI parameters.

```
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`: Used to configure SCLK clock Frequency, in Khz;
- `DataShftDir`: Used to control the data shift mode. Set it to `MSB_FIRST` if high is first; set it to `LSB_FIRST` if low is first.
- `ClkPol`: Used to set the level of the SCLK clock during idle time. If it is low, set it to `CPOL_0`; if it is high, set it to `CPOL_1`.
- `ClkPha`: Set which edge of SCLK to send data and sample the received data. If send and sample on the first edge, set it to `CPHA_0`; if send and sample on the second edge, set it to `CPHA_1`.
- `SelPol`: Used to set the effective level of the chip select signal. Set it to `SEL_POL_LO` if the low level is valid; set it to `SEL_POL_HI` if the high level is valid.
- `ucAddrLen`: Used to control the length of the address segment to be sent. If it is not necessary to send the address segment, set this

parameter to 0. The maximum value of this parameter is 4, indicating that the length of the address segment is 4 bytes.

- **ucCmdEn:** Used to control whether to send a command segment. If set to 1, send a byte of the command segment; if set to 0, no command segment will be sent and the length of the command segment is fixed at 1 byte.

Please refer to the following enumeration variable definitions for values set above:

```
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 API Functions

5.1 Parameter Configuration

This function is used to configure U2X_SPI parameters. If you need to change the parameter configuration, use this function again to reconfigure it.

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

Parameters:

- devh: device handle of libusb;
- pSpiConfig: A pointer to the “u2x_spi_config” struct, use this struct to configure parameters for U2X_SPI device.
- uiTimeout: timeout parameter, in milliseconds;

Return Value:

Returns 0 if the function runs successful, otherwise an error code less than zero is returned.

5.2 Send / Receive Byte Data

This function can realize three data transmission modes of full duplex sending and receiving, sending only, and receiving only. It takes bytes as unit, and the maximum data length of a single transmission is 512 bytes.

```
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  
);
```

Parameters:

- devh: device handle of libusb;
- uiDataByteCnt: The count of bytes to control the data transmitting and receiving.
- pucWriteData: A pointer to that holds the data to be transmitted. If no data needs to be sent, set this parameter to NULL.
- pucReadData: A pointer to that holds the received data. If no data needs to be received, set this parameter to NULL.
- uiAddr: Address segment data. If the length of the address segment configuration is non-zero, the address data will be sent. The maximum length of the address segment is 4 bytes.
- ucCmd: Command segment data. If it's enabled, the byte data will be sent in the command segment.
- uiDummyCnt: DummySCLK, that is, the number of waiting clock cycles between the command segment, address segment, and data segment. If this parameter is non-zero, send the SCLK with the specified number of cycles after sending the command segment and address segment, and then send the data segment. The maximum value of this parameter is 65536. If DummySCLK segment is not needed, set this parameter to 0.
- pSpiConfig: A pointer to the "u2x_spi_config" struct, use this struct to control the reading and writing of U2X_SPI device.
- uiTimeout: timeout parameter, in milliseconds;

Return Value:

Returns 0 if the function runs successful, otherwise an error code less

than zero is returned.

5.3 Send / Receive Multiple Bits of Data

This function can realize three data transmission modes of full duplex sending and receiving, sending only, and receiving only. It takes bits as unit. The maximum data length of a single transmission is 2048 bits (512 bytes).

```
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  
);
```

Parameters:

- devh: device handle of libusb;
- uiDataBitCnt: The count of bits to control the data transmitting and receiving.
- pucWriteData: A pointer to that holds the data to be transmitted. If no data needs to be sent, set this parameter to NULL.
- pucReadData: A pointer to that holds the received data. If no data needs to be received, set this parameter to NULL.
- uiAddr: Address segment data. If the length of the address segment configuration is non-zero, the address data will be sent. The maximum length of the address segment is 4 bytes.
- ucCmd: Command segment data. If it's enabled, the byte data will be sent in the command segment.
- uiDummyCnt: DummySCLK, that is, the number of waiting clock cycles between the command segment, address segment, and data segment. If this parameter is non-zero, send the SCLK with the specified

number of cycles after sending the command segment and address segment, and then send the data segment. The maximum value of this parameter is 65536. If DummySCLK segment is not needed, set this parameter to 0.

- **pSpiConfig**: A pointer to the “u2x_spi_config” struct, use this struct to control the reading and writing of U2X_SPI device.
- **uiTimeout**: timeout parameter, in milliseconds;

Return Value:

Returns 0 if the function runs successful, otherwise an error code less than zero is returned.

5.4 Programming Example

```
//Global variables

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;

    // Open the GWU2X device

    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;

    }

    // Configure U2X_SPI parameters

    pSpiConfig->DataShftDir      = MSB_FIRST;

    pSpiConfig->ClkPol           = CPOL_0;

    pSpiConfig->ClkPha           = CPHA_0;

    pSpiConfig->SelPol           = SEL_POL_LO;

    pSpiConfig->uiSclkFreqKiloHz = 10000; // SCLK frequency is 10MHz

    pSpiConfig->ucAddrLen        = 3; // The length of adress segment is 3 bytes

    pSpiConfig->ucCmdEn          = 1; // Enable command segment

    u2x_spi_set_config(devh, pSpiConfig, 1000);


    //Full-duplex mode sends and receives 64 bytes of data simultaneously

    u2x_spi_read_write_bytes(devh,

        64, // The length of transmission data is 64 bytes.

        ucWrData, // Data to be sent

        ucRdData, // Store the received data

        0xaabbcc, // The address segment is 0xaabbcc

        0x32, // The Command segment is 0x32

        8, // Send eight cycles of Dummy SCLK after command segment and adress
        segment

        pSpiConfig, // Pointer to the struct of parameter configuration

        1000); // Set "timeout" parameter to 1000 milliseconds


    //Full-duplex mode sends and receives 56 bits data simultaneously

```

```
u2x_spi_read_write_bits(devh,  
  
    56, // The length of transmission data Bit 56 bits.  
  
    (unsigned char *)&ullWrData, // Data to be sent  
  
    (unsigned char *)&ullRdData, // Store the received data  
  
    0xaabbcc, // The address segment is 0xaabbcc  
  
    0x32, // The Command segment is 0x32  
  
    8, // Send eight cycles of Dummy SCLK after command segment and adress  
segment  
  
    pSpiConfig, // Pointer to the struct of parameter configuration  
  
    1000); // Set "timeout" parameter to 1000 milliseconds  
  
// Close the device and Exit  
out:  
  
if(devh)  
  
    libusb_close(devh);  
  
libusb_exit(NULL);  
  
return 0;  
  
}
```

6 Error Code

If the API function runs well, it returns 0; otherwise, it returns a negative number.

The meaning of the non-zero return values are shown in the table as below.

Table 6-1 Error Code List

| Value | Enumerator | |
|-------|-----------------------------|---------------------------------------|
| 0 | SUCCESS | Runs correctly |
| -1 | USB_ERROR_IO | USB input/output error |
| -2 | USB_ERROR_INVALID_PARAM | USB parameter error |
| -3 | USB_ERROR_ACCESS | No permission to access the device |
| -4 | USB_ERROR_NO_DEVICE | No USB device (device disconnected) |
| -5 | USB_ERROR_NOT_FOUND | No Entity |
| -6 | USB_ERROR_BUSY | USB device busy |
| -7 | USB_ERROR_TIMEOUT | Timeout |
| -8 | USB_ERROR_OVERFLOW | Memory overflow |
| -9 | USB_ERROR_PIPE | Pipe error |
| -10 | USB_ERROR_INTERRUPTED | The system function was interrupted |
| -11 | USB_ERROR_NO_MEM | Out of memory |
| -12 | USB_ERROR_NOT_SUPPORTED | Not supported on the current platform |
| -13 | U2X_SPI_ERROR_USBTRANS_ERR | USB data transmitting error |
| -14 | U2X_SPI_ERROR_INVALID_PARAM | Invalid parameter setting |
| -15 | U2X_SPI_ERROR_TIMEOUT | U2X_SPI transmitting timeout |

| Value | Enumerator | |
|-------|-----------------------|-----------------------|
| -16 | U2X_SPI_ERROR_CMD_ERR | U2X_SPI command error |
| -99 | ERROR_OTHER | Other errors |

Terminology and Abbreviations

The abbreviations and terminology used in this manual are as shown in Table A -1.

Table A -1 Terminology and Abbreviations

| Terminology and Abbreviations | Full Name |
|-------------------------------|-----------------------------|
| USB | Universal Serial Bus |
| SPI | Serial Peripheral Interface |

Support and Feedback

Gowin Semiconductor provides customers with comprehensive technical support. If you have any questions, comments, or suggestions, please feel free to contact us directly by the following ways.

Website: www.gowinsemi.com

E-mail: support@gowinsemi.com

