



GWU2U Driver (Windows VCP)

User Guide

UG1007-1.1E, 12/23/2021

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
06/29/2021	1.0E	Initial version published.
12/23/2021	1.1E	Chapter 3 GWU2U Virtual Serial Port Programming Guide updated.

Contents

Contents	i
List of Figures	ii
List of Table	iii
1 GWU2U Virtual Serial Port Driver	1
2 Uninstall Driver	3
3 GWU2U Virtual Serial Port Programming Guide	5
3.1 Open Virtual Serial Port Device	5
3.2 Timeout Setting	8
3.3 Parameters Setting	9
3.4 } Clear Serial Port Buffer	10
3.5 Clear Serial Port Error	11
3.6 Transmit Data via Virtual Serial Device (Synchronization)	12
3.7 Receive Data via Virtual Serial Device (Synchronization)	12
3.8 Asynchronous Read and Write Serial Port	13
Terminology and Abbreviations	24
Support and Feedback	25

List of Figures

Figure 1-1 List All Devices 1

Figure 1-2 Select Device..... 2

Figure 1-3 Select Driver 2

Figure 2-1 Open Device Manager 3

Figure 2-2 Uninstall Device 4

List of Table

Table A-1 Terminology and Abbreviations	24
---	----

1 GWU2U Virtual Serial Port Driver

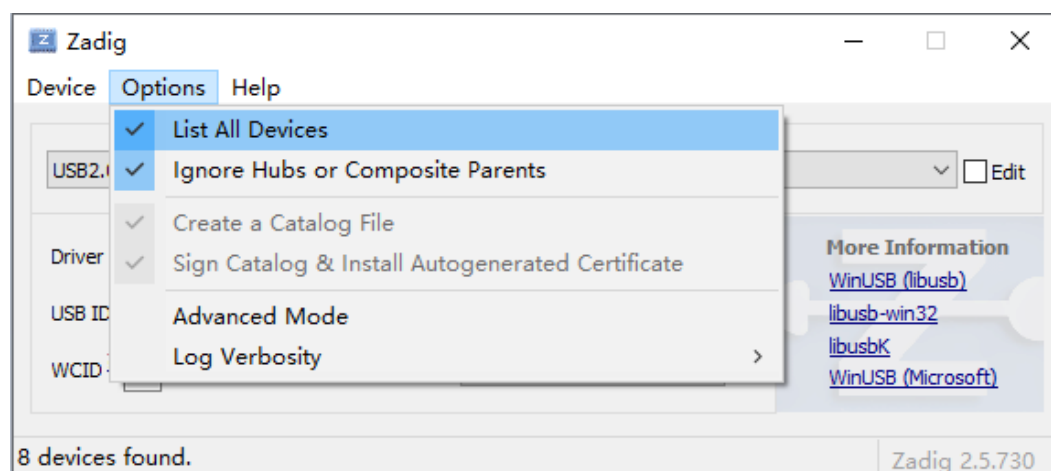
GWU2U can use usbser.sys, USB virtual serial port driver for programming operations to realize data receive/transmit and the setting of parameters such as baud rate, parity bit, stop bit and byte bit width.

Use Gowin driver installation tool or Zadig (<https://zadig.akeo.ie/>), open source driver installation tool, to install driver. The driver installation requires administrator privileges.

Use Zadig to Install Driver

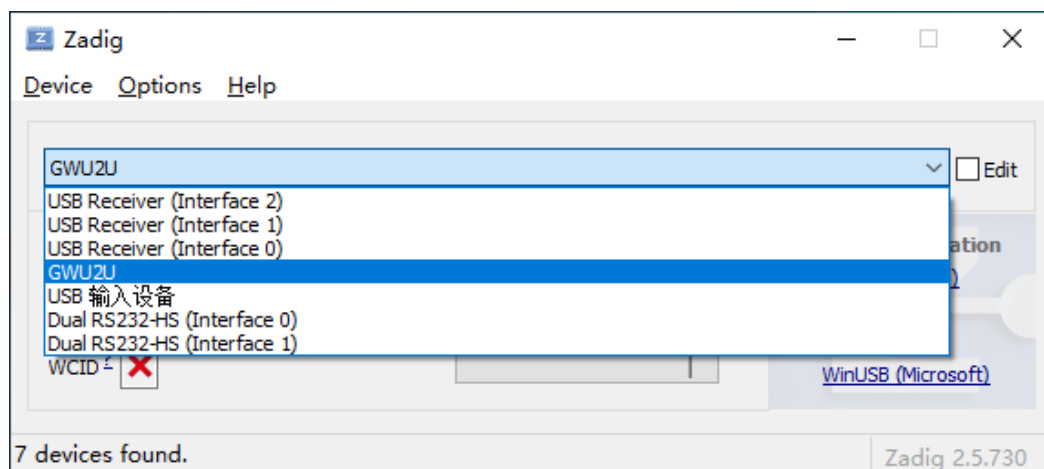
Connect GWU2U device to computer with USB interface, double-click to open Zadig (administrator privileges required), click Options, check the "List All Device" option, and all USB devices connected to the computer will be listed.

Figure 1-1 List All Devices



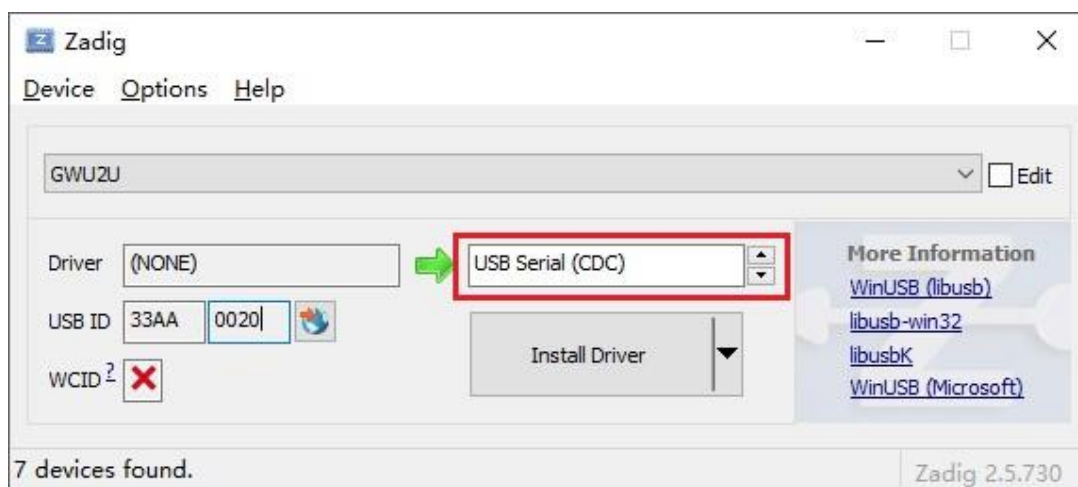
Select Gowin U2U, the device that requires driver installation.

Figure 1-2 Select Device



Select the driver to be installed, use virtual serial port driver (VCP), and select USB Serial (CDC).

Figure 1-3 Select Driver



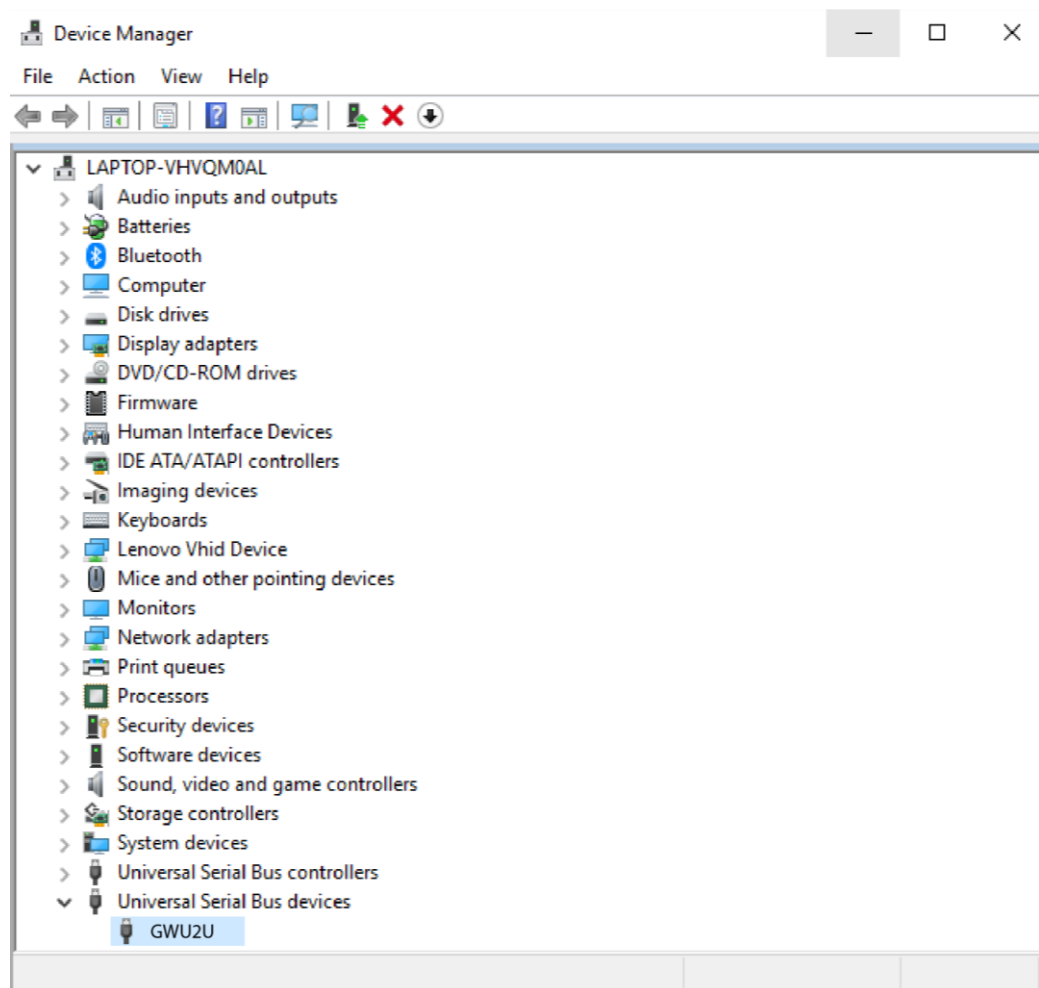
Click "Install Driver". The driver will be installed after a few moments.

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

2 Uninstall Driver

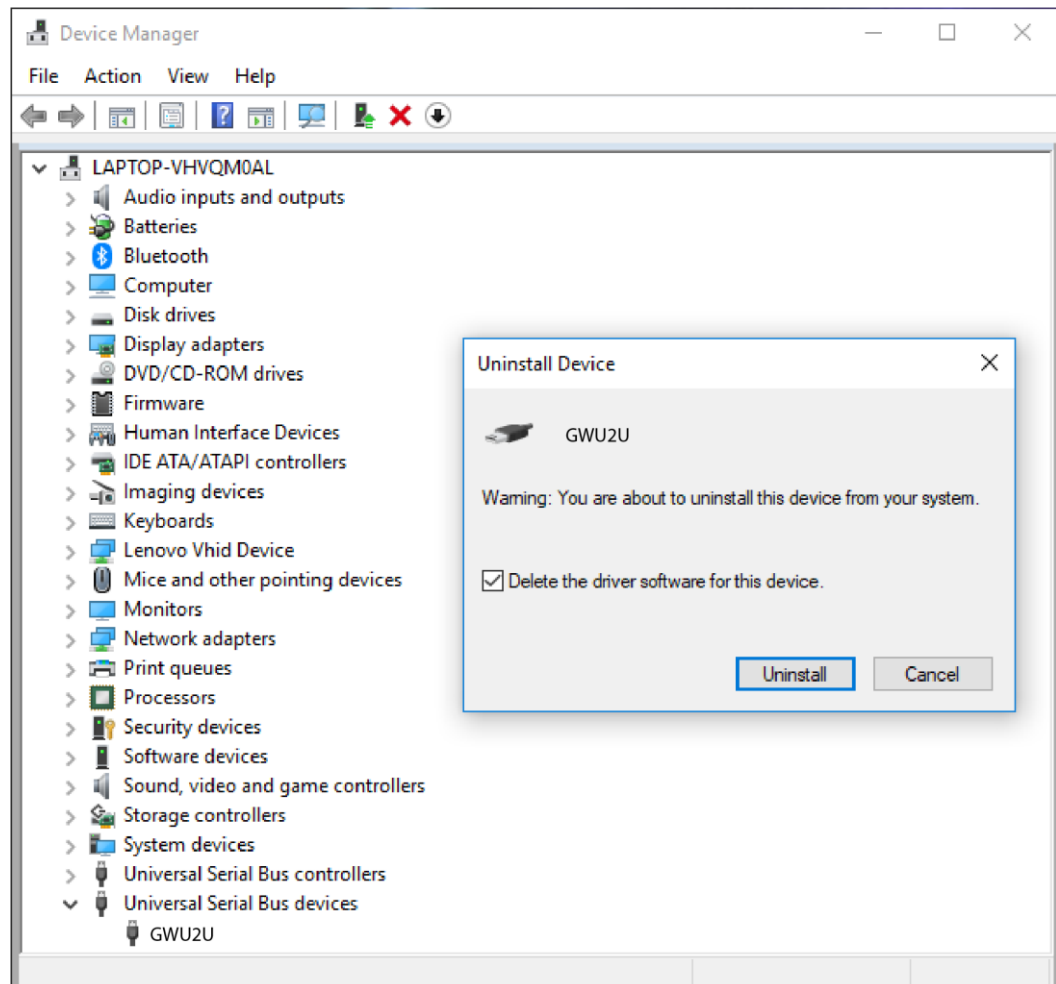
To uninstall the driver, connect GWU2U device to the computer, open the windows device manager, and find Gowin USB Serial device in the "Universal Serial Bus Devices" list. Right-click on the device name and select the "Uninstall Device" option in the pop-up menu.

Figure 2-1 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 2-2 Uninstall Device



3 GWU2U Virtual Serial Port Programming Guide

Use Windows library functions to operate the GWU2U virtual serial port devices. The header file <windows.h> needs to be included when programming.

3.1 Open Virtual Serial Port Device

Use the windows library function CreateFile () to open the virtual serial device and return a handle to the serial port. The function interface is defined as follows:

```
HANDLE WINAPI CreateFile(  
    _In_      LPCTSTR lpFileName, //Name of the file to open or  
create  
    _In_      DWORD dwDesiredAccess, //Type of Access  
    _In_      DWORD dwShareMode, //Share mode  
    _In_opt_  LPSECURITY_ATTRIBUTES lpSecurityAttributes,  
//Security attributes  
    _In_      DWORD dwCreationDisposition, //Action to specify  
that the file to be opened already exists or does not exist  
    _In_      DWORD dwFlagsAndAttributes, //File flags and  
attributes  
    _In_opt_  HANDLE hTemplateFile //A handle to template files  
);
```

Function parameters are defined as follows:

lpFileName: Name of the file to open or create.

dwDesiredAccess: Type of access.

- 0 is device query access method.
- GENERIC_READ is read access.
- GENERIC_WRITE is write access.

dwShareMode: Share mode.

- 0 indicates the file cannot be shared and other operations to open the file will fail.
- FILE_SHARE_READ indicates other read operations are allowed.
FILE_SHARE_WRITE indicates other write operations are allowed.
- FILE_SHARE_DELETE indicates other delete operations are allowed.

lpSecurityAttributes: Security attributes. A pointer to the SECURITY_ATTRIBUTES struct.

dwCreationDisposition: Action to take when creating or opening the file.

- OPEN_ALWAYS: Open the file, if it does not exist, then create it.
- TRUNCATE_EXISTING: Open the file and empty it (so GENERIC_WRITE permission is required), if the file does not exist, then it will fail.
- OPEN_EXISTING: Open the file, if the file does not exist, then it will fail.
- CREATE_ALWAYS: Create the file, if it already exists, then empty it;
CREATE_NEW: Create the file, if it exists, then it will fail.

dwFlagsAndAttributes: File flag attributes.

- FILE_ATTRIBUTE_NORMAL: General attribute.
- FILE_FLAG_OVERLAPPED: Asynchronous I/O flag, if this flag is not specified then the default is synchronous IO.
- FILE_ATTRIBUTE_READONLY: The file is read-only.
- FILE_ATTRIBUTE_HIDDEN: The file is hidden.
- FILE_FLAG_DELETE_ON_CLOSE: The files are deleted after all file handles are closed.
- For other flags and attributes, please refer to the official Microsoft documentation.

hTemplateFile: A handle to a file that must be opened with GENERIC_READ access. If this parameter is not NULL, the file will be created using the attributes and flags of the file associated with

hTemplateFile. If an existing file is opened, this parameter is ignored.

When using CreateFile() to open the serial port, it should be noted that:

- lpFileName file name, if the serial port is below COM10, then write the serial port number name directly, such as "COM1"; if the serial port is COM10 and above, then the format of serial port name should be: \\\\.\\COM10.
- dwShareMode shared mode should be 0, that is, the serial port should be exclusive mode.
- dwCreationDisposition open action should be OPEN_EXISTING, that is, the serial port must exist.

When closing the serial port, call CloseHandle() function to close the serial port, and the parameter of the function is the serial port handle.

Examples are as follows:

```
HANDLE hCom_1 =  
  
CreateFile (  
  
ComName,  
  
GENERIC_READ | GENERIC_WRITE,  
  
0,  
  
NULL,  
  
OPEN_EXISTING,  
  
FILE_FLAG_OVERLAPPED,  
  
// indicates that open the serial port in OVERLAPPED mode for  
asynchronous transmission, if synchronous transmission is used then  
use parameter 0  
  
NULL  
  
); // Open serial port device  
  
  
  
if (hCom_1 == INVALID_HANDLE_VALUE)  
{  
  
    int a = GetLastError();  
  
    printf("Error: %d\\n", a);  
  
    return -1;  
  
}
```

```
CloseHandle(hCom_1);    // Close serial port device
```

3.2 Timeout Setting

When calling `ReadFile()` and `WriteFile()` to read and write to the serial port, if no asynchronous operation is specified, both reading and writing will always wait for the specified size of data, and then we may want to set a timeout for reading and writing. Calling `SetCommTimeouts()` can set the serial port read/write timeout, `GetCommTimeouts()` can get the current timeout setting, generally use `GetCommTimeouts` to get the current timeout information to a `COMMTIMEOUTS` struct first, then customize this struct, then call `SetCommTimeouts()` to set it.

The member variables of `COMMTIMEOUTS` are described as follows:

ReadIntervalTimeout: The maximum delay between two characters. When reading serial data, once the time difference between two character transfers exceeds this time, the read function will return the existing data. A setting of 0 means that the parameter does not work. Specifies the maximum delay between the arrival of two characters on the communication line (in milliseconds). During the `ReadFile` operation, the time period is counted from the time the first character is received. If the interval between two characters received exceeds this value, the `ReadFile` operation is completed and all buffered data is returned. If `ReadIntervalTimeout` is 0, then the value does not work. If the value is `MAXDWORD`, and the `ReadTotalTimeoutConstant` and `ReadTotalTimeoutMultiplier` values are both 0, then the specified read operation returns immediately with the characters already received, even if none were received.

ReadTotalTimeoutMultiplier: Read the timeout between each character. Cumulative value that is specified in milliseconds. The total number of timeouts used to calculate read operations. For each read operation, the value is multiplied by the number of bytes to be read.

ReadTotalTimeoutConstant: The constant timeout for reading serial data at a time. So in a read serial operation, the timeout is `ReadTotalTimeoutMultiplier` multiplied by the number of bytes read plus `ReadTotalTimeoutConstant`. Set `ReadIntervalTimeout` to `MAXDWORD` and set `ReadTotalTimeoutMultiplier` and `ReadTotalTimeoutConstant` to 0 to indicate that the read operation will immediately return the characters stored in the input buffer.

WriteTotalTimeoutMultiplier: Write the timeout between each character.

WriteTotalTimeoutConstant: The constant timeout for writing serial data

at a time. So in a write serial operation, the timeout is WriteTotalTimeoutMultiplier multiplied by the number of bytes read plus WriteTotalTimeoutConstant.

Examples are as follows:

```
//Set read timeout
COMMTIMEOUTS timeouts;

GetCommTimeouts(hCom_1, &timeouts);

timeouts.ReadIntervalTimeout = 0;

timeouts.ReadTotalTimeoutMultiplier = 0;

timeouts.ReadTotalTimeoutConstant = 6000;

timeouts.WriteTotalTimeoutMultiplier = 0;

timeouts.WriteTotalTimeoutConstant = 0;

SetCommTimeouts(hCom_1, &timeouts);


//Set read/write buffer size
static const int nBufSize = 32768;

if(!SetupComm(hCom_1, nBufSize, nBufSize))
{
    printf("SetupComm() failed, comm port closed...\n");
    CloseHandle(hCom_1);
    return -1;
}
```

3.3 Parameters Setting

Windows provides a dedicated data struct DCB and a dedicated interface function GetCommState()/SetCommState() for setting serial port parameters.

DCB is a data structure used for setting serial port communication device. The variables for the basic settings of the serial port are as follows:

BaudRate: The maximum value that can be set is 256000.

Parity: NOPARITY, ODDPARITY, EVENPARITY, MARKPARITY, SPACEPARITY.

ByteSize: The number of bits to transfer a single byte.

StopBits: ONESTOPBIT, ONE5STOPBITS, TWOSTOPBITS.

Examples are as follows:

```
//Set comm port configurations
DCB dcb;

if (!GetCommState(hCom_1, &dcb))
{
    printf("GetCommState() failed, comm port closed...\n");
    CloseHandle(hCom_1);
    return -1;
}

//Configuration settings
int nBaud = 115200;

dcb.DCBlength = sizeof(DCB);
dcb.BaudRate = nBaud; //Baud rate
dcb.Parity = NOPARITY; //Parity mode
dcb.ByteSize = 8; //Byte Size
dcb.StopBits = TWOSTOPBITS; //Stop bit

if (!SetCommState(hCom_1, &dcb))
{
    printf("SetCommState() failed\n");
    CloseHandle(hCom_1);
    return -1;
}
```

3.4 }Clear Serial Port Buffer

Windows provides the PurgeComm() function for stopping read and write operations and clearing the read and write buffers. The read/write buffer should be cleared first before reading serial data or writing serial data for the first time, or if the serial port has not been used for a long time, or if there is an error on the serial port. Examples are as follows:

```
DWORD dwError;
```



```
COMSTAT cs;

if(!ClearCommError(hCom_1, &dwError, &cs))
{
    printf("ClearCommError() failed\n");
    CloseHandle(hCom_1);
    return -1;
}
```

3.5 Clear Serial Port Error

Windows provides the `ClearCommError()` function to clear errors in communication and to get the current status of communication. Call `ClearCommError()` function to clear the error and get the size of the data in the buffer before the read/write operation. Examples are as follows:

```
DWORD dwError;

COMSTAT cs;

if(!ClearCommError(hCom_1, &dwError, &cs))
{
    printf("ClearCommError() failed\n");
    CloseHandle(hCom_1);
    return -1;
}
```

3.6 Transmit Data via Virtual Serial Device (Synchronization)

Use WriteFile() function provided by Windows to send data. If it succeeds, then return TRUE; if it fails, then return FALSE. When transmitting data synchronously, the OVERLAPPED address data structure in the function parameter should be set to NULL. Examples are as follows:

```
BOOL bErrorFlag = FALSE;

char DataBuffer[] = "This is some test data to write to the virtual
com port.\r\n";

DWORD dwBytesToWrite = (DWORD)strlen(DataBuffer);

DWORD dwBytesWritten = 0;

bErrorFlag =
WriteFile (
    hCom_1,          // Serial port handle
    DataBuffer,      // Save the address of the data to be sent
    dwBytesToWrite,  // The number of data bytes to be sent
    &dwBytesWritten, // The variable address that saves the
actual number of bytes of data sent
    NULL            // OVERLAPPED data structure address, NULL
when sent synchronously
);
```

3.7 Receive Data via Virtual Serial Device (Synchronization)

Use ReadFile() function provided by Windows to receive data. If it succeeds, then return TRUE; if it fails, then return FALSE. When receiving data synchronously, the OVERLAPPED address data structure in the function parameter should be set to NULL. Examples are as follows:

```
char buf[101];

BOOL bErrorFlag = FALSE;

DWORD nLenOut = 0;

bErrorFlag =
```

```
ReadFile (
    hCom_1,          // Serial port handle
    buf,             // Saving address of received data
    100,             // The count of the data bytes to be received
    &nLenOut, // The variable address that saves the actual number
of bytes of data received
    NULL            // OVERLAPPED data structure address, NULL when
received synchronously
);

if(bErrorFlag)
{
    if(nLenOut)
    { // succeed
    }
    else
    { // timeout
    }
}
else
{ // fail
}
```

3.8 Asynchronous Read and Write Serial Port

Asynchronous read and write of serial port is similar to synchronous read and write except that the serial ports need to be opened in an overlapping mode. If the overlap operation does not complete immediately, `WaitCommEvent()` returns `FALSE` and `GetLastError()` returns `ERROR_IO_PENDING`, indicating that the operation is in progress in the background. The `hEvent` member of the parameter overlap structure is set to the no-signal state until `WaitCommEvent` returns. If an event or error occurs, it is set to signaled state, the application can call the wait functions (`WaitForSingleObject`, `WaitForSingleObjectEx`, etc.) to determine the state

of the time object, and the WaitCommEvent() parameter lpEvtMask saves the specific event that occurred.

There are two ways to wait or judge whether the overlapping operation is completed.

- The first method is to use WaitForSingleObject() to wait for the hEvent member of the parameter of OVERLAPPED type in the read/write function: when the ReadFile, WriteFile functions are called, the member will be automatically set to the no-signal state; when the overlapping operation is completed, the member variable will be automatically set to the signaled state when the overlap operation is completed.
- Another method is to call GetOverlappedResult() function to get the status of the overlapping operation to determine whether the overlapping operation is completed.

A complete example of virtual serial port asynchronous reading and writing is as follows:

```
//Header files

#include <stdio.h>

#include <windows.h>


DWORD WINAPI ThreadRxMsg(LPVOID lpParameter);

void getComName(char* ComNamePtr, unsigned int ComNum);

void printBuf(char *buf, unsigned int bufSize);


char ComName[13];

HANDLE hCom_1;


int main(int argc, char* argv[])
{
    DWORD dwError;

    unsigned int ComNum = 26;


    getComName (ComName, ComNum);
```

```
// open virtual serial port device in OVERLAPPED, the port name
is COM26

hCom_1 =
CreateFile (
    ComName,
    GENERIC_READ | GENERIC_WRITE,
    0,
    NULL,
    OPEN_EXISTING,
    FILE_FLAG_OVERLAPPED,
    0
);

if(hCom_1 == INVALID_HANDLE_VALUE)
{
    int a = GetLastError();
    printf("Error : %d\n", a);
    return -1;
}

// Timeout Setting
COMMTIMEOUTS timeouts;
GetCommTimeouts(hCom_1,&timeouts);
timeouts.ReadIntervalTimeout = 0;
timeouts.ReadTotalTimeoutMultiplier = 0;
timeouts.ReadTotalTimeoutConstant = 6000;
timeouts.WriteTotalTimeoutMultiplier = 0;
timeouts.WriteTotalTimeoutConstant = 0;
SetCommTimeouts(hCom_1,&timeouts);
// Set the size of read/write buffer
static const int nBufSize = 32768;
```

```
    if(!SetupComm(hCom_1,nBufSize,nBufSize))
    {
        printf("SetupComm() failed, comm port closed...\n");
        CloseHandle(hCom_1);
        return -1;
    }

// Set serial port parameters
DCB dcb;
if (!GetCommState(hCom_1,&dcb))
{
    printf("GetCommState() failed, comm port closed...\n");
    CloseHandle(hCom_1);
    return -1;
}

int nBaud = 115200;
dcb.DCBlength = sizeof(DCB);
dcb.BaudRate = nBaud; //Baud rate
dcb.Parity = NOPARITY; //Parity mode: No
dcb.ByteSize = 8; //Byte size: 8
dcb.StopBits = TWOSTOPBITS; //Stop bit: 1
if (!SetCommState(hCom_1,&dcb))
{
    printf("SetCommState() failed\n");
    CloseHandle(hCom_1);
    return -1;
}

// Create a new OVERLAPPED structure for the control of
asynchronous sending
```

```
OVERLAPPED wrOverlapped;

memset(&wrOverlapped, 0, sizeof(wrOverlapped));

if (wrOverlapped.hEvent != NULL)
{
    ResetEvent(wrOverlapped.hEvent);

    wrOverlapped.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
}

char DataBuffer[] = "This is some test data to write to the
virtual com port.\r\n";

DWORD dwBytesToWrite = (DWORD)strlen(DataBuffer);

DWORD dwBytesWritten = 0;

// Clear serial port errors and serial port transmit buffers
if (ClearCommError(hCom_1, &dwError, NULL)) {
    PurgeComm(hCom_1, PURGE_TXABORT | PURGE_TXCLEAR);
}

// Use WriteFile to transmit data and loop to determine if the
transmission is complete

if
(!WriteFile(hCom_1, DataBuffer, dwBytesToWrite, &dwBytesWritten, &wrOverlapped))
{
    if (GetLastError() == ERROR_IO_PENDING)
    {
        while
(!GetOverlappedResult(hCom_1, &wrOverlapped, &dwBytesWritten, FALSE))
        {
            if (GetLastError() == ERROR_IO_INCOMPLETE)
            {
                //Not finished, continue and get the result again
            }
        }
    }
}
```

```
        continue;
    }
    else
    {
        printf("Send Error Occured\n");
        ClearCommError(hCom_1, &dwError, NULL);
        break;
    }
}
printf("Data send finished, Bytes Written: %d\n",
dwBytesWritten);
}
}

// Clear read buffer
PurgeComm(hCom_1, PURGE_RXCLEAR | PURGE_RXABORT);
//clear error
COMSTAT cs;
if(!ClearCommError(hCom_1, &dwError, &cs))
{
    printf("ClearCommError() failed\n");
    CloseHandle(hCom_1);
    return -1;
}

SetCommMask(hCom_1, EV_RXCHAR); //Set event to receive data
//Create a new auxiliary thread to receive data
HANDLE hThread1 =
CreateThread(NULL, 0, ThreadRxMsg, NULL, 0, NULL);

while(1)
```



```
{
    // to do
}

CloseHandle(hCom_1);

return 0;
}

// Auxiliary thread for asynchronous receive serial port data
DWORD WINAPI ThreadRxMsg(LPVOID lpParameter)
{
    while(1)
    {
        // OVERLAPPED structure for controlling asynchronous receive
        OVERLAPPED osWait;
        memset(&osWait,0,sizeof(OVERLAPPED));
        osWait.hEvent = CreateEvent(NULL,TRUE,FALSE,NULL);
        DWORD dwEvtMask;
        if (WaitCommEvent(hCom_1,&dwEvtMask,&osWait))
        {
            //If receive buffer is not empty, handle the data in the buffer
            if(dwEvtMask & EV_RXCHAR)
            {
                DWORD dwError;
                COMSTAT cs;
                if(!ClearCommError(hCom_1,&dwError,&cs))
                {
                    printf("ClearCommError() failed\n");
                    CloseHandle(hCom_1);
                    return -1;
                }
            }
        }
    }
}
```

```
    }

    char buf[101] = {0};
    DWORD nLenOut = 0;
    DWORD dwTrans = 0;
    OVERLAPPED osRead;
    memset(&osRead, 0, sizeof(OVERLAPPED));
    osRead.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
    BOOL bReadStatus =
ReadFile(hCom_1, buf, cs.cbInQue, &nLenOut, &osRead);
    if(!bReadStatus)
    {
        if(GetLastError() == ERROR_IO_PENDING)
        {
            printf("GetLastError() == ERROR_IO_PENDING\n");
            // to do
        }
    }
    else
    {
        printBuf(buf, nLenOut);
    }
}
else
{
    if(GetLastError() == ERROR_IO_PENDING)
    {
        WaitForSingleObject(osWait.hEvent, INFINITE);
        if(dwEvtMask & EV_RXCHAR)
        {
```

```
    DWORD dwError;

    COMSTAT cs;

    if(!ClearCommError(hCom_1,&dwError,&cs))
    {
        printf("ClearCommError() failed\n");
        CloseHandle(hCom_1);
        return -1;
    }

    char buf[101] = {0};
    DWORD nLenOut = 0;
    DWORD dwTrans = 0;
    OVERLAPPED osRead;
    memset(&osRead,0,sizeof(OVERLAPPED));
    osRead.hEvent = CreateEvent(NULL,TRUE,FALSE,NULL);
    BOOL bReadStatus =
    ReadFile(hCom_1,buf,cs.cbInQue,&nLenOut,&osRead);
    if(!bReadStatus)
    {
        if(GetLastError() == ERROR_IO_PENDING)
        {
            printf("GetLastError() == ERROR_IO_PENDING\n");
            // to do
        }
    }
    else
    {
        printBuf(buf,nLenOut);
    }
}
```

```
    }  
}  
  
    return 1;  
}
```

// Generate the serial port name according to the serial port number, when the number is greater than or equal to 10, the serial port name should be in the format of "\\.\COMXX".

```
void getComName(char* ComNamePtr, unsigned int ComNum)  
{  
    if(ComNum > 9)  
    {  
        sprintf(ComNamePtr, "\\.\COM%d", ComNum);  
    }  
    else  
    {  
        sprintf(ComNamePtr, "COM%d", ComNum);  
    }  
  
    return;  
}
```

// Print Serial port data received

```
void printBuf(char *buf, unsigned int bufSize)  
{  
    for(int i = 0; i < bufSize; i++)  
    {  
        #ifdef PRINT_HEX  
            printf("%02x ", buf[i]);  
        #else  
            printf("%c", buf[i]);  
        #endif  
    }  
}
```

```
        printf("%c", buf[i]);  
    #endif  
}  
  
return;  
}
```

For details of the function interface to the Windows virtual serial port, see the official Microsoft documentation at [Serial Communications Functions](#).

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	Meaning
USB	Universal Serial Bus
UART	Universal Asynchronous Receiver/Transmitter

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

