Cypress USB-Serial VCP I^{2}C/SPI API Guide

Doc. No. 001-99398 Rev. **

Cypress Semiconductor

198 Champion Court

San Jose, CA 95134-1709

Phone (USA): 800.858.1810

Phone (Intnl): 408.943.2600

http://www.cypress.com

Copyrights

Trademarks

Source Code

Disclaimer

Contents

1. Introduction

Cypress USB-Serial virtual COM port (VCP) I^{2}C/SPI APIs provide an easy start to using USB-Serial Communications Device Class (CDC) devices in the Windows operating system.

USB-Serial devices connect USB with I^{2}C/SPI/UART. In USB CDC mode, I^{2}C, SPI, or UART devices are accessible to Windows applications using the VCP. The COM port is designed for UART devices, so users are expected to use the Win32 Serial communication API to communicate with USB-to-UART bridge devices.

However, the Win32 native VCP APIs are not designed to be used for I^{2}C or SPI. For this special functionality, the Cypress VCP library comes in handy. The Cypress library adds a custom protocol wrapper on top of the Native Serial Communication layer and this protocol wrapper targets I^{2}C and SPI device access using the Cy7C6521xA part. Library APIs are exposed through Windows dynamically linked library (DLL) "C" functions. These APIs simplify accessing the I2C/SPI functionality.

To configure a device as a USB CDC I^{2}C/SPI device, refer to the USB-Serial Configuration Utility User Guide.

1.1 Software Requirements

Table 1-1 lists the software prerequisites for using the DLL.

Table 1-1. Software Requirements

	Software	Version
1.	Operating system	Microsoft Windows XP or later
2.	Run-time libraries	Microsoft VC++ 2008 SP1 run time redistributable

Note: The Microsoft VC++ 2008 run-time redistributable is packaged with the USB-Serial SDK for Windows. The redistributable is located in the *sdk_install_path*>\prerequisite directory. This package can also be downloaded and installed from the Microsoft website at http://www.microsoft.com/en-us/download/details.aspx?id=11895.

1.2 Supported USB-Serial Devices

Table 1-2 lists the USB-Serial API supported devices.

Table 1-2. Supported USB-Serial Devices

No.	Part Number	Description
1	CY7C65211A-24LTXI	USB-Serial (Single Channel)
2	CY7C65215A-32LTXI	USB-Serial (Dual Channel)

2. USB-Serial VCP Device Use

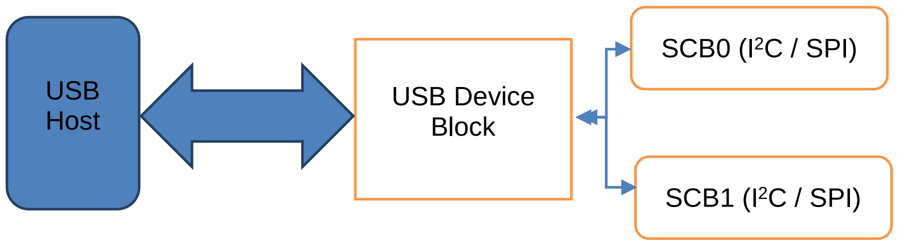
Users are expected to use the **USB-Serial Configuration Utility** to configure the device according to their requirements. To use the VCP API library, the device needs to be configured as described in this chapter.

This library does not provide any support for the USB device protocol configured as Vendor or PHDC USB class. This library is intended for devices configured for the USB – CDC protocol that bridges I^{2}C and SPI devices. Refer to the vendor API user guide and vendor library functionality for the USB Vendor or PHDC USB class configuration.

2.1 Identifying VCP Number for CY7C65215A

The CY7C65215A Serial-Bridge Controller has two Serial Communication Blocks (SCBs), as shown in Figure 2-1.

Figure 2-1. SCBs

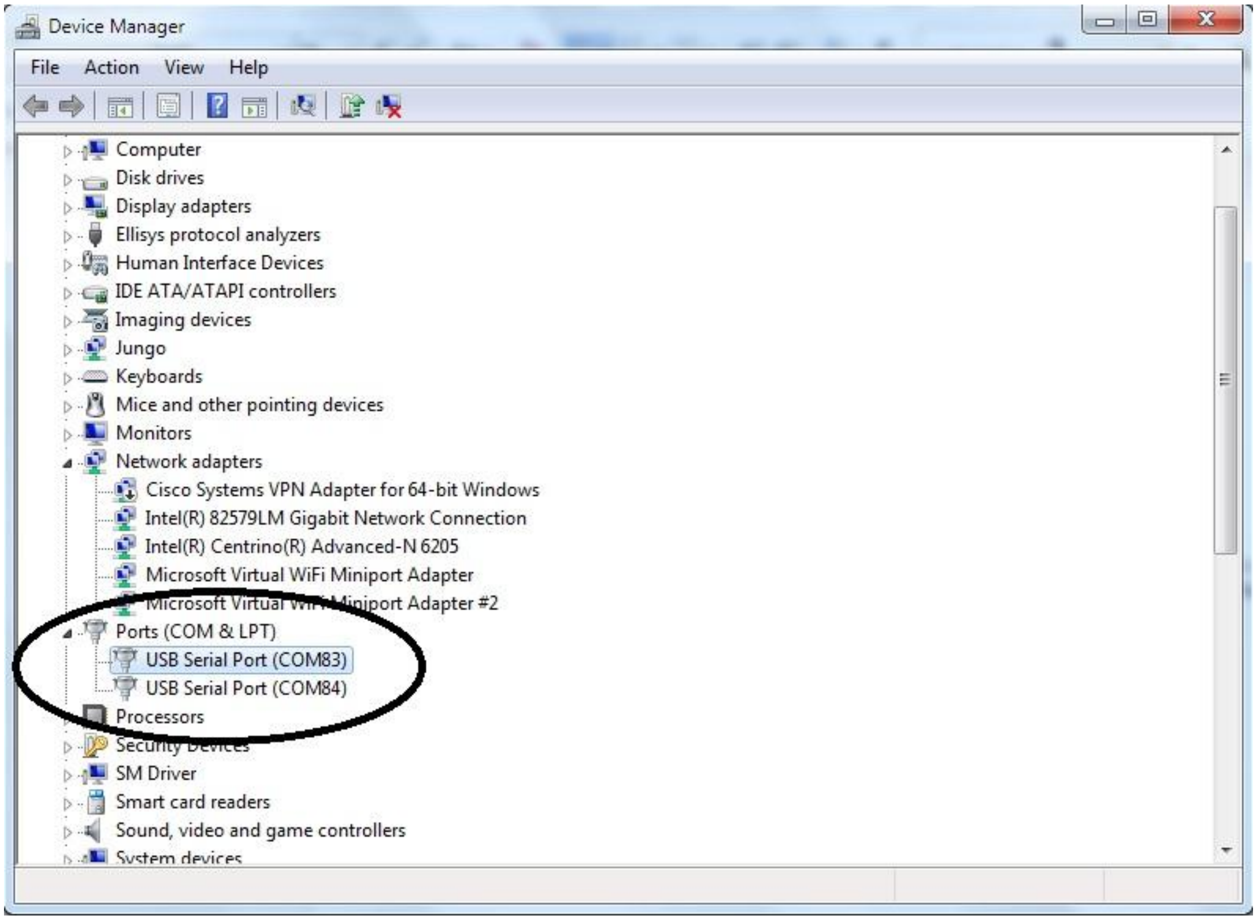


For the CY7C65215A Serial-Bridge Controller, Windows **Device Manager** displays two different VCP numbers (devices). So, two options are available for you to see which COM port is mapped to which SCB functionality.

2.1.1 Option 1

1. Open Windows **Device Manager**. Run *Devmgmt.msc* and locate the virtual COM port number assigned to the device, as shown in Figure 2-2.

Figure 2-2. COM Port Location

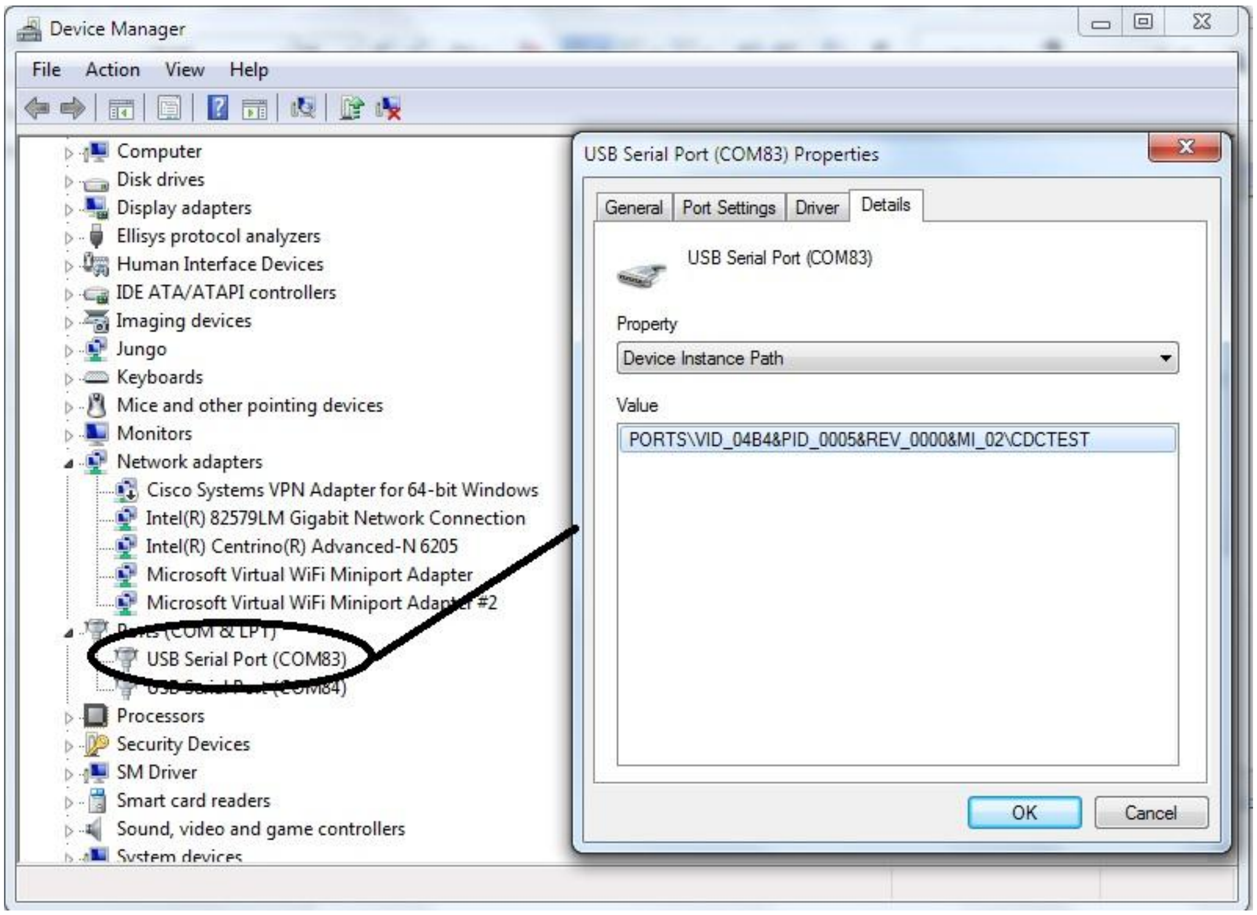

Device Manager
File Action View Help
Computer
Disk drives
Display adapters
Ellisys protocol analyzers
Human Interface Devices
IDE ATA/ATAPI controllers
Imaging devices
Jungo
Keyboards
Mice and other pointing devices
Monitors
Network adapters
Cisco Systems VPN Adapter for 64-bit Windows
Intel(R) 82579LM Gigabit Network Connection
Intel(R) Centrino(R) Advanced-N 6205
Microsoft Virtual WiFi Miniport Adapter
Microsoft Virtual WiFi Miniport Adapter #2
Ports (COM & LPT)
USB Serial Port (COM83)
USB Serial Port (COM84)
Processors
Security Devices
SM Driver
Smart card readers
Sound, video and game controllers
System devices

2. For this example, right-click on the COM83 port. A new dialog window will pop up. Click the **Details** tab, and navigate to the **Device Instance Path** property of the device, as shown in Figure 2-3:

   PORTS\VID_04B4&PID_0005&REV_0000&**MI_02**\CDCTEST

   Here, “MI_02” represents the SCB1 data path. Any communication that happens through COM83 will reach the SCB1 device.

Figure 2-3. SCB1 to COM Port Mapping


Device Manager
File Action View Help
Computer
Disk drives
Display adapters
Ellisys protocol analyzers
Human Interface Devices
IDE ATA/ATAPI controllers
Imaging devices
Jungo
Keyboards
Mice and other pointing devices
Monitors
Network adapters
Cisco Systems VPN Adapter for 64-bit Windows
Intel(R) 82579LM Gigabit Network Connection
Intel(R) Centrino(R) Advanced-N 6205
Microsoft Virtual WiFi Miniport Adapter
Microsoft Virtual WiFi Miniport Adapter #2
Ports (COM & LPT)
USB Serial Port (COM83)
USB Serial Port (COM84)
Processors
Security Devices
SM Driver
Smart card readers
Sound, video and game controllers
System devices
USB Serial Port (COM83) Properties
General
Port Settings
Driver
Details
USB Serial Port (COM83)
OK
Cancel

Property: Device Instance Path

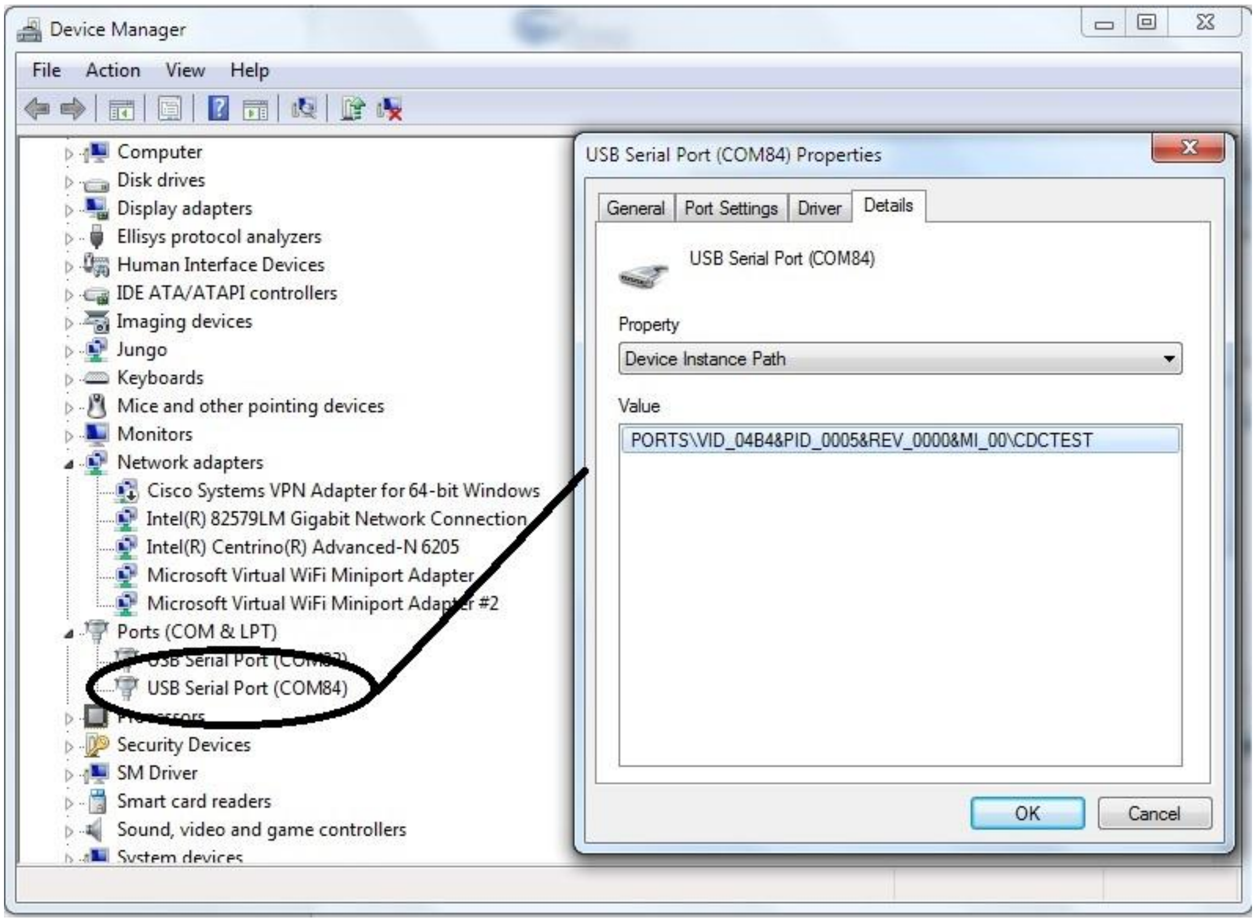
Value: PORTS\VID_04B4&PID_0005&REV_0000&MI_02\CDCTEST

3. Now right-click on the second available port, COM84. Figure 2-4 displays the COM84 **Device Instance Path** as follows:

   PORTS\VID_04B4&PID_0005&REV_0000&**MI_00**\CDCTEST

   Here, “MI_00” represents the SCB0 data path. Any communication that happens through COM84 will reach the SCB0 device.

Figure 2-4. SCB0 to COM Port Mapping


Device Manager
File Action View Help
Computer
Disk drives
Display adapters
Ellisys protocol analyzers
Human Interface Devices
IDE ATA/ATAPI controllers
Imaging devices
Jungo
Keyboards
Mice and other pointing devices
Monitors
Network adapters
Cisco Systems VPN Adapter for 64-bit Windows
Intel(R) 82579LM Gigabit Network Connection
Intel(R) Centrino(R) Advanced-N 6205
Microsoft Virtual WiFi Miniport Adapter
Microsoft Virtual WiFi Miniport Adapter #2
Ports (COM & LPT)
USB Serial Port (COM83)
USB Serial Port (COM84)
Processors
Security Devices
SM Driver
Smart card readers
Sound, video and game controllers
System devices
USB Serial Port (COM84) Properties
General
Port Settings
Driver
Details
OK
Cancel

USB Serial Port (COM84)

Property: Device Instance Path

Value: PORTS\VID_04B4&PID_0005&REV_0000&MI_00\CDCTEST

2.1.2 Option 2

The Windows native Setup API includes support to query the device instance string using an API called "SetupDiGetDeviceInstanceId." The following code is useful to query the device instance ID. It needs the *setupapi.lib* file for the linker to complete the linking job.

```
#include <Windows.h>
#include <Cfgmgr32.h>
#include <SetupAPI.h>
#include <stdio.h>
#include <Ntddser.h>

int QueryDeviceInstanceID()
{
    HDEVINFO hdevinfo;

    SP_DEVINFO_DATA devinfo;
    wchar_t instance_id[4096];
    DWORD n;

hdevinfo = SetupDiGetClassDevs(&GUID_DEVINTERFACE_SERENUM_BUS_ENUMERATOR,  NULL, NULL,
(DIGCF_PRESENT|DIGCF_ALLCLASSES));

    if (hdevinfo == INVALID_HANDLE_VALUE)
    {
        DWORD err = GetLastError();
        printf("SetupDiGetClassDevs: %u\n", err);
        return 1;
    }

    for (n = 0;; n++)
    {
        devinfo.cbSize = sizeof(devinfo);
        if (!SetupDiEnumDeviceInfo(hdevinfo, n, &devinfo))
        {
            DWORD err = GetLastError();
            printf("SetupDiEnumDeviceInfo: %u\n", err);
            break;
        }

        if (!SetupDiGetDeviceInstanceId(hdevinfo, &devinfo,
              instance_id, _countof(instance_id), NULL))
        {
            DWORD err = GetLastError();
            printf("SetupDiGetDeviceInstanceId: %u\n", err);
        }
        else
        {
            CString strData(instance_id);
            if (strData.Find(L"PORTS\\VID_04B4") != -1 )
            {
                BYTE friendlyName[300];
                    SetupDiGetDeviceRegistryProperty(hdevinfo, &devinfo,
                    SPDRP_FRIENDLYNAME, NULL, friendlyName, sizeof(friendlyName), NULL);

                printf("DevicePath: %ws\n", instance_id);
                printf("Friendly Name: %ws\n", friendlyName);
            }
        }
    }

    return 0;
}
```

2.2 Using the USB Serial Number

For any COM port device, Windows assigns the COM port number. For the USB VCP device, though, switching between the USB ports with the same hardware can result in a different COM port number being assigned to the same USB-Serial hardware.

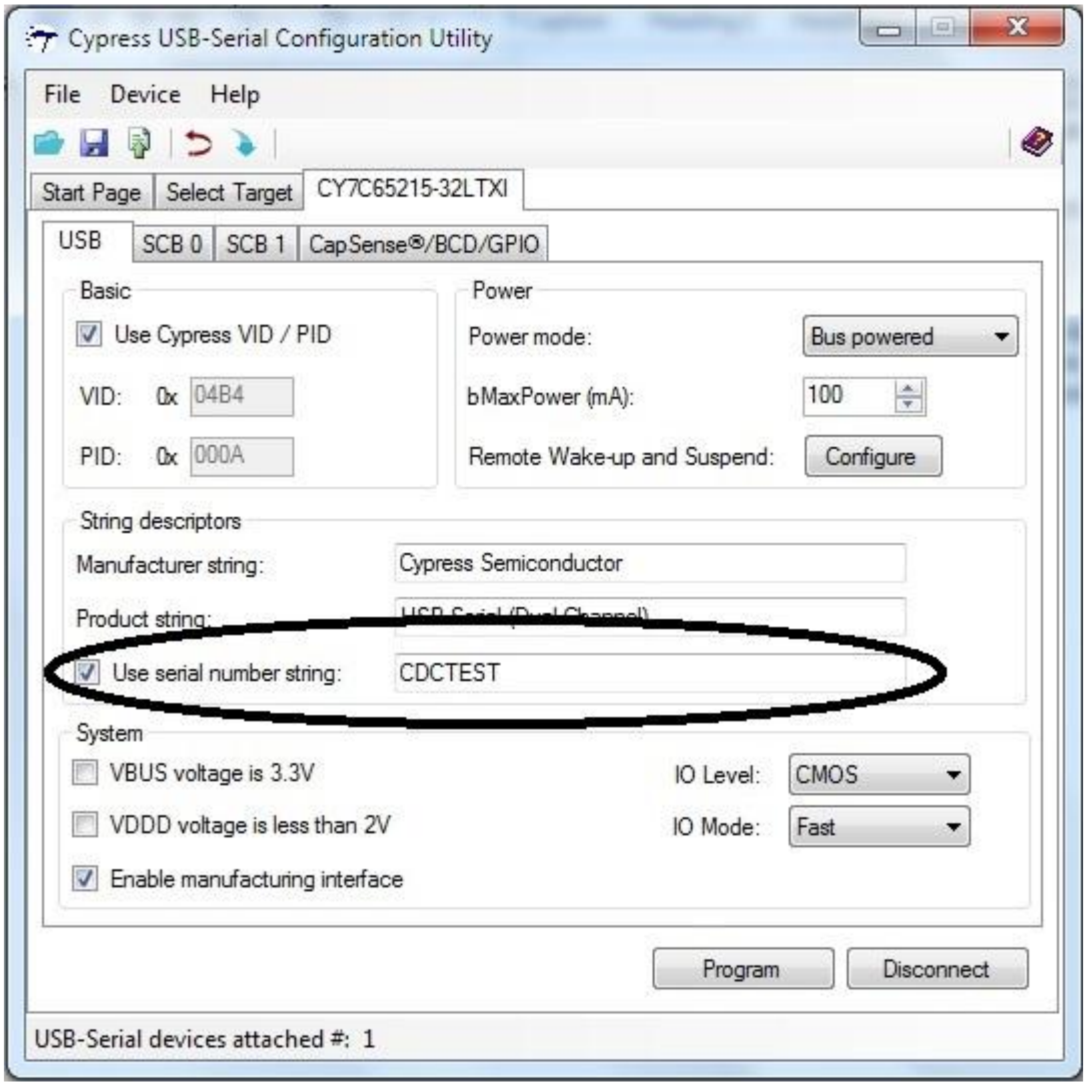
Configuring a USB serial number for the given USB-Serial device freezes the COM port number for a given PC and the given USB-Serial hardware. However, in Windows, two or more devices cannot share a USB serial number. This number is expected to be unique for all the connected USB-Serial hardware.

Figure 2-4 shows how to display the **Device Instance Path** for a given device. The device instance path appears as follows:

PORTS\VID_04B4&PID_0005&REV_0000&MI_00\CDCTEST

Here, the “CDCTEST” string is a USB serial number string coming from the device, so, the **Device Instance Path** reflects this number. Use the **USB** tab of the **USB-Serial Configuration Utility** to assign a serial number to the device, as shown in Figure 2-5. For more information about configuring the USB serial number using the Configuration Utility, refer to the USB-Serial Configuration Utility User Guide.

Figure 2-5. USB-Serial Configuration Utility



Basic

[✓] Use Cypress VID / PID

VID: 0x 04B4

PID: 0x 000A

Power

Power mode: Bus powered

bMaxPower (mA): 100

Remote Wake-up and Suspend:

String descriptors

Manufacturer string: Cypress Semiconductor

Product string: USB-Serial (Dual Channel)

[✓] Use serial number string: CDCTEST

System

[ ] VBUS voltage is 3.3V

[ ] VDDD voltage is less than 2V

[✓] Enable manufacturing interface

IO Level: CMOS

IO Mode: Fast

USB-Serial devices attached #: 1

3. USB-Serial VCP API

The VCP library was created to handle nonstandard COM device (UART) requirements. The USB-Serial VCP API contains two main sections and one generic section: I²C communication and SPI communication using the VCP.

3.1 Data Structure for Communication

A few data structure definitions are available to use with the API.

3.1.1 Data Structure CY_VCP_LIBRARY_VERSION

```
/* Summary
This structure is used to hold version information of the library.

Description
This structure can be used to retrieve the version information of the library.

See Also
* CyGetLibraryVersion
*/
typedef struct _CY_VCP_LIBRARY_VERSION {

    UINT8 majorVersion;     /*The major version of the library*/
    UINT8 minorVersion;     /*The minor version of the library*/
    UINT16 patch;           /*The patch number of the library*/
    UINT8 buildNumber;      /*The build number of the library*/

} CY_VCP_LIBRARY_VERSION, *PCY_VCP_LIBRARY_VERSION;
```

3.1.2 Data Structure CY_VCP_I2C_DATA_CONFIG

```
/* Summary
This structure is used to configure each I2C data transaction.

Description
This structure defines parameters that are used for configuring
I2C module during each data transaction. These parameters are slave address, I2C
Master Read/Write Internal Address and Internal Address Byte Length.

Slave address value, Internal address and Internal Address length are needed for
I2C master communication.

A valid internal address with internal length helps I2C master to introduce an I2C
Restart condition during a I2C Master read or write operations.
During I2C slave mode all these fields will be ignored.

See Also
* CyI2cWrite_VCP
* CyI2cRead_VCP
*/
typedef struct _CY_VCP_I2C_DATA_CONFIG
{
    UINT8   slaveAddress;     /*Slave address to communicate*/
    UINT32  internalAddress;  /*Internal address in Master Mode*/
    UINT8   internalAddressLength;
/*Master's Read or Write Internal Address Length (Valid values are 1 byte to 4
bytes)*/

} CY_VCP_I2C_DATA_CONFIG, *PCY_VCP_I2C_DATA_CONFIG;
```

3.1.3 Data Structure CY_VCP_SPI_CONFIG

```
/* Summary
This structure is used to configure the SPI module of USB Serial device.

Description
This structure defines configuration parameters that are used for configuring the
SPI module .

See Also
* CyGetSpiConfig_VCP
* CySetSpiConfig_VCP
*/
typedef struct _CY_VCP_SPI_CONFIG
{

    UINT32 frequency;
/*SPI clock frequency. The frequency range supported by SPI module is between
1000(1 KHz) to 3000000(3 MHz) */

    UCHAR dataWidth;
/*Data width in bits. The valid values are from 4 to 16.*/

}CY_VCP_SPI_CONFIG,*PCY_VCP_SPI_CONFIG;
```

3.1.4 C Enumerator CY_VCP_RETURN_STATUS

```
/* Summary
Enumeration defining return status of  USB serial library APIs

Description
The enumeration CY_RETURN_STATUS holds the different return status of all the APIs
supported by USB Serial VCP library.
*/

typedef enum _CY_VCP_RETURN_STATUS{

    CY_VCP_SUCCESS = 0,
/*API returned successfully without any errors.*/
    CY_VCP_ERROR_ACCESS_DENIED,
/*Access of the API is denied for the application */
    CY_VCP_ERROR_DRIVER_INIT_FAILED,
/*Driver Initialization failed*/
    CY_VCP_ERROR_DEVICE_INFO_FETCH_FAILED,
/*Device information fetch failed */
    CY_VCP_ERROR_DRIVER_OPEN_FAILED,
/*Failed to open a device in the library */
    CY_VCP_ERROR_INVALID_PARAMETER,
/*One or more parameters sent to the API was invalid*/
    CY_VCP_ERROR_REQUEST_FAILED,
/*Request sent to USB Serial device failed */
    CY_VCP_ERROR_DOWNLOAD_FAILED,
/*Firmware download to the device failed */
    CY_VCP_ERROR_FIRMWARE_INVALID_SIGNATURE,
/*Invalid Firmware signature in firmware file*/
    CY_VCP_ERROR_INVALID_FIRMWARE,
/*Invalid firmware */
```

```
    CY_VCP_ERROR_DEVICE_NOT_FOUND,
/*Device disconnected */
    CY_VCP_ERROR_IO_TIMEOUT,
/*Timed out while processing a user request*/
    CY_VCP_ERROR_PIPE_HALTED,
/*Pipe halted while trying to transfer data*/
    CY_VCP_ERROR_BUFFER_OVERFLOW,
/*Overflow of buffer while trying to read/write data */
    CY_VCP_ERROR_INVALID_HANDLE,
/*Device handle is invalid */
    CY_VCP_ERROR_ALLOCATION_FAILED,
/*Error in Allocation of the resource inside the library*/
    CY_VCP_ERROR_I2C_DEVICE_BUSY,
/*I2C device busy*/
    CY_VCP_ERROR_I2C_NAK_ERROR,
/*I2C device NAK*/
    CY_VCP_ERROR_I2C_ARBITRATION_ERROR,
/*I2C bus arbitration error*/
    CY_VCP_ERROR_I2C_BUS_ERROR,
/*I2C bus error*/
    CY_VCP_ERROR_I2C_BUS_BUSY,
/*I2C bus is busy*/
    CY_VCP_ERROR_I2C_STOP_BIT_SET,
/*I2C master has sent a stop bit during a transaction*/
    CY_VCP_ERROR_SPI_SLAVE_DESELECT,
/*SPI Slave received unexpected deselection.*/
    CY_VCP_ERROR_STATUS_MONITOR_EXIST
/*API Failed because the SPI/UART status monitor thread already exists*/
} CY_VCP_RETURN_STATUS;
```

3.2 Generic APIs

These APIs are common to both I^{2}C and SPI communication over the COM port.

3.2.1 CyOpenVCP

```
/*
   Summary
   This API opens the USB Serial device attached to Virtual COM Port.

   Description
   This API opens the USB Serial device based on the COM Port Number.

   Note: COM Port argument is a 2 Byte argument but in reality the Port number
won't exceed 255.

   Return Value
   * CY_VCP_SUCCESS on success
   * CY_VCP_ERROR_INVALID_PARAMETER if the input parameters are invalid.
   * CY_VCP_ERROR_DRIVER_OPEN_FAILED if open was unsuccessful.
   * CY_VCP_ERROR_ALLOCATION_FAILED if memory allocation was failed.
   * CY_VCP_ERROR_DEVICE_NOT_FOUND if specified VCP number is invalid.

   See Also
   * CyClose
*/

CYWINEXPORTVCP CY_VCP_RETURN_STATUS CyOpenVCP (
    UINT16  vcpNumber,  /*VCP Number of device that needs to be opened*/
    HANDLE  *handle     /*OS Context Handle returned by the API*/
    );
```

3.2.2 CyCloseVCP

```
/*
   Summary
   This API closes the specified device handle and releases all resources
associated with it.

   Description
   This API closes the device handle and releases all the resources allocated
internally in the
   library. This API should be invoked using a valid device handle and upon
successful return
   of CyOpen.

   Return Value
   * CY_VCP_SUCCESS on success.
   * CY_VCP_ERROR_INVALID_PARAMETER if handle is invalid.

   See Also
   * CyOpenVCP
*/

CYWINEXPORTVCP CY_VCP_RETURN_STATUS CyCloseVCP (
    HANDLE handle    /*Handle of the device that needs to be closed*/
    );
```

3.2.3 CyGetLibraryVersion_VCP

```
/*
   Summary
   This API retrieves the version of USB Serial library.

   Description
   This API retrieves the version of USB Serial library.

   Return Value
   * CY_VCP_SUCCESS
   * CY_VCP_ERROR_INVALID_PARAMETER if PCY_VCP_LIBRARY_VERSION parameter is
invalid.

*/
CYWINEXPORTVCP CY_VCP_RETURN_STATUS CyGetLibraryVersion_VCP (
    PCY_VCP_LIBRARY_VERSION version
/*Library version of the current library*/
    );
```

3.3 I^{2}C Communication APIs

The following APIs are designed to perform I^{2}C communication.

3.3.1 CyGetI2cFreq_VCP

```
/*
Summary
This API retrieves the current Operating Frequency of I2C module in USB Serial
device.

Description
This API retrieves the current Operating Frequency of I2C module in USB Serial
device.

Return Value
   * CY_VCP_SUCCESS
   * CY_VCP_ERROR_INVALID_HANDLE
   * CY_VCP_ERROR_INVALID_PARAMETER
   * CY_VCP_ERROR_IO_TIMEOUT
   * CY_VCP_ERROR_REQUEST_FAILED

See Also
   * CySetI2cFreq_VCP
*/
CYWINEXPORTVCP CY_VCP_RETURN_STATUS CyGetI2cFreq_VCP (
    HANDLE handle,           /*Valid device handle*/
    UINT32 *i2cFreq          /*Pointer to I2C Operating Frequency Variable*/
    );
```

3.3.2 CySetI2cFreq_VCP

```
/*
Summary:
This API sets the new Operating Frequency of I2C module in USB Serial  device.

Description:
This API sets the new Operating Frequency of I2C module in USB Serial device.

Return Value:
   * CY_VCP_SUCCESS on success
   * CY_VCP_ERROR_INVALID_HANDLE
   * CY_VCP_ERROR_INVALID_PARAMETER
   * CY_VCP_ERROR_IO_TIMEOUT
   * CY_VCP_ERROR_REQUEST_FAILED

   See Also
   * CyGetI2cFreq_VCP
*/
CYWINEXPORTVCP CY_VCP_RETURN_STATUS CySetI2cFreq_VCP (
    HANDLE handle,                          /*Valid device handle*/
    UINT32 i2cFreq                          /*New I2C Freq value for setup*/
    );
```

3.3.3 CyI2cRead_VCP

```
/*
Summary
   This API reads data from the USB Serial I2C module.

Description
This API provides an interface to read data from the I2C device
connected to USB Serial. The readBuffer parameter needs to be initialized with buffer
pointer, number of bytes to be read before invoking the API. On return, the
transferCount field will contain the number of bytes read back from device.

CY_VCP_I2C_DATA_CONFIG structure specifies parameters such as setting stop bit, NAK
and slave address of the I2C device.

Return Value
   * CY_VCP_SUCCESS on success.
   * CY_VCP_ERROR_INVALID_HANDLE
   * CY_VCP_ERROR_INVALID_PARAMETER
   * CY_VCP_ERROR_REQUEST_FAILED
   * CY_VCP_ERROR_IO_TIMEOUT
   * CY_VCP_ERROR_DEVICE_NOT_FOUND
   * CY_VCP_ERROR_BUFFER_OVERFLOW
   * CY_VCP_ERROR_ALLOCATION_FAILED
   * CY_VCP_ERROR_I2C_DEVICE_BUSY
   * CY_VCP_ERROR_I2C_NAK_ERROR
   * CY_VCP_ERROR_I2C_ARBITRATION_ERROR
   * CY_VCP_ERROR_I2C_BUS_ERROR
   * CY_VCP_ERROR_I2C_STOP_BIT_SET

See Also
   * CY_VCP_DATA_BUFFER
   * CY_DATA_CONFIG
   * CyI2cCWrite
*/


CYWINEXPORTVCP CY_VCP_RETURN_STATUS CyI2cRead_VCP (
    HANDLE handle,                              /*Valid device handle*/
    CY_VCP_I2C_DATA_CONFIG *dataConfig,             /*I2C data config*/
    CY_VCP_DATA_BUFFER *readBuffer,                 /*Read buffer details*/
    UINT32 timeout                              /*API timeout value*/
    );
```

3.3.4 CyI2cWrite_VCP

```
/*
Summary
   This API writes data to USB Serial I2C module .

Description
This API provides an interface to write data to the I2C device connected to USB
Serial. The writeBuffer parameter needs to be initialized with buffer pointer,
number of bytes to be written before invoking the API. On return, transferCount
field contains number of bytes actually written to the device.

CY_VCP_I2C_DATA_CONFIG structure specifies parameter such as setting stop bit, Nak
and slave address of the I2C device being communicated when USB Serial is master.

Return Value
   * CY_VCP_SUCCESS on success.
   * CY_VCP_ERROR_INVALID_HANDLE
   * CY_VCP_ERROR_INVALID_PARAMETER
   * CY_VCP_ERROR_REQUEST_FAILED
   * CY_VCP_ERROR_IO_TIMEOUT
   * CY_VCP_ERROR_DEVICE_NOT_FOUND
   * CY_VCP_ERROR_BUFFER_OVERFLOW
   * CY_VCP_ERROR_ALLOCATION_FAILED
   * CY_VCP_ERROR_I2C_DEVICE_BUSY
   * CY_VCP_ERROR_I2C_NAK_ERROR
   * CY_VCP_ERROR_I2C_ARBITRATION_ERROR
   * CY_VCP_ERROR_I2C_BUS_ERROR
   * CY_VCP_ERROR_I2C_STOP_BIT_SET

   See Also
   * CY_VCP_DATA_BUFFER
   * CY_DATA_CONFIG
   * CyI2cRead_VCP
*/

CYWINEXPORTVCP CY_VCP_RETURN_STATUS  WINCALLCONVENVCP CyI2cWrite_VCP (
    HANDLE handle,                        /*Valid device handle*/
    CY_VCP_I2C_DATA_CONFIG *dataConfig, /*I2C Data Config */
    CY_VCP_DATA_BUFFER *writeBuffer,      /*Write buffer details*/
    BOOL bIsI2CRestartNeeded,

/*Applicable in Master mode. By I2C Protocol, Write Internal address Followed by
I2C Restart is supported by fewer slave device. Set this variable to TRUE, if the
slave device requires a I2C Restart. Otherwise, leave this variable as FALSE*/
```

```
    UINT32 timeout                        /*API timeout value*/
    );
```

3.4 SPI Communication APIs

This set of APIs provides an interface to configure the SPI module and perform read/write operations with the SPI device connected to the USB-Serial device.

3.4.1 CyGetSpiConfig_VCP

```
/*
   Summary
   This API retrieves the configuration of SPI module of USB Serial device.

   Description
   This API retrieves the configuration of SPI module of USB Serial device.

   Return Value
   * CY_VCP_SUCCESS
   * CY_VCP_ERROR_INVALID_HANDLE
   * CY_VCP_ERROR_INVALID_PARAMETER
   * CY_VCP_ERROR_IO_TIMEOUT
   * CY_VCP_ERROR_REQUEST_FAILED

   See Also
   * CY_VCP_SPI_CONFIG
   * CySetSpiConfig_VCP
*/
CYWINEXPORTVCP CY_VCP_RETURN_STATUS CyGetSpiConfig_VCP (
    HANDLE handle,
/*Valid device handle*/
    CY_VCP_SPI_CONFIG *spiConfig
/*SPI configuration structure value read back*/
    );
```

3.4.2 CySetSpiConfig_VCP

```
Summary
   This API sets the configuration of the SPI module on USB Serial device.

Description;
   This API sets the configuration of the SPI module in USB Serial device.

NOTE: Using this API during an active transaction of SPI may result in    data
loss.


Return Value
   * CY_VCP_SUCCESS
   * CY_VCP_ERROR_INVALID_HANDLE
   * CY_VCP_ERROR_INVALID_PARAMETER
   * CY_VCP_ERROR_IO_TIMEOUT
   * CY_VCP_ERROR_REQUEST_FAILED



See Also
   * CY_VCP_SPI_CONFIG
   * CyGetSpiConfig_VCP
*/
```

```
CYWINEXPORTVCP CY_VCP_RETURN_STATUS CySetSpiConfig_VCP (
    HANDLE handle,
/*Valid device handle*/
    CY_VCP_SPI_CONFIG *spiConfig
 /*SPI configuration structure value*/
    );
```

3.4.3 CySpiReadWrite_VCP

```
/*
Summary
This API reads and writes data to SPI device connected to USB Serial device.

Description
This API provides an interface to do data transfer with the SPI slave/master
connected to USB Serial device. To perform read only operation, pass NULL as
argument for writeBuffer and to perform write only operation pass NULL as an
argument for readBuffer.  On return, the transferCount field will contain the
number of bytes read and/or written.

Return Value
   * CY_VCP_SUCCESS
   * CY_VCP_ERROR_INVALID_HANDLE
   * CY_VCP_ERROR_INVALID_PARAMETER
   * CY_VCP_ERROR_IO_TIMEOUT
   * CY_VCP_ERROR_REQUEST_FAILED
   * CY_VCP_ERROR_PIPE_HALTED
   * CY_VCP_ERROR_DEVICE_NOT_FOUND
   * CY_VCP_ERROR_BUFFER_OVERFLOW


See Also
   * CY_VCP_DATA_BUFFER
   * CyGetSpiConfig_VCP
   * CySetSpiConfig_VCP
*/

CYWINEXPORTVCP CY_VCP_RETURN_STATUS CySpiReadWrite_VCP (
    HANDLE handle,                          /*Valid device handle*/
    CY_VCP_DATA_BUFFER* readBuffer,         /*Read data buffer*/
    CY_VCP_DATA_BUFFER* writeBuffer,        /*Write data buffer*/
    UINT32 timeout                          /*Timeout value of the API*/
    );
```

Revision History

Document Revision History

Document Title: Cypress USB-Serial VCP I^{2}C/SPI API Guide Document Number: 001-99398			
Revision	**Issue Date**	**Origin of Change**	**Description of Change**
**	07/31/2015	JEGA	New API guide