
PORT CONFIGURATION AND GPIO FOR CP210X

Relevant Devices

This application note applies to the following devices:
CP2103, CP2104, CP2105

1. Introduction

The devices discussed in this application note have added pin configurability for improved connectivity. These added configuration options are accessed through the USB interface using a Dynamic Link Library (DLL).

This document details configuration options for the CP210x device port pins, which are used to connect to external circuitry. These pins fall into three types of interface pins. First, the UART/Modem interface pins consist of the signals RI, DCD, DTR, DSR, TXD, RXD, RTS, and CTS. These signals are used for UART communication and the associated handshaking.

The second type of interface pin is for General Purpose Input/Output (GPIO). This type consists of all signals named GPIO.x, where x is a number. These signals are available for any user-defined function.

The final type of interface pin is for power control and consists of SUSPEND and $\overline{\text{SUSPEND}}$ signals. These signals are used to gate power consumption of external circuitry for bus-powered USB products.

For information on ordering preconfigured parts, contact your Silicon Laboratories sales representative.

2. Customizable Options

The following interface pin configuration options are available.

2.1. Mode

The mode setting controls whether the interface pin operates in push-pull or open-drain mode. This setting is not available on the RXD pin. See Sections 3.2 and 3.3 for details concerning pin modes.

2.2. Reset Latch Value

This setting controls the initial value of the interface pin latch, after a device reset. Not available on RXD, TXD, SUSPEND, $\overline{\text{SUSPEND}}$, and GPIO pins set for device-controlled function. See Section 3.5 for details concerning pin reset behavior.

2.3. Weak Pull-Ups

This setting enables a weak pull-up for all interface pins. This setting applies to the device as a whole and cannot be configured for each pin independently. Upon reset, weak pull-ups are enabled.

2.4. GPIO Pin Function

By default, the GPIO pins are controlled manually by host-based software using the *CP210xRuntime.DLL* and an open handle to the COM port to read and write the latch.

Alternatively, the CP210x device can automatically control certain GPIO pin latches for a predetermined function. When operating in this mode, the GPIO pin will no longer be available using the *CP210xRuntime.DLL*. Host writes will have no effect, and host reads will be logic high. This device-controlled function is available as shown in Table 1.

Table 1. GPIO Pin Function

Pin Name	Function	Behavior
GPIO.0	Transmit LED	Toggles when there is UART data to transmit; otherwise logic high
GPIO.1	Receive LED	Toggles when there is data on the UART receive buffer; otherwise logic high.
GPIO.2	RS-485	Logic low while transmitting UART data; otherwise logic high.
Note: On the CP2105, RS-485 mode is available on the Enhanced Communication Interface (ECI) on GPIO.1. Because of this, the alternate function GPIO.1_ECI can either be Receive LED or RS-485 mode. The Standard Communication Interface (SCI) does not have an RS-485 mode.		

2.5. Dynamic Suspend

By default the latch values for all interface pins remains static during USB suspend.

Alternatively, the dynamic suspend feature sets the interface pin latch to a predefined state when the CP210x device moves from the configured USB state to the suspend USB state (see chapter nine of USB 2.0 specification for more information on USB device states). When the device exits the suspend USB state the interface pin latch is restored to the previous value before entering the suspend state. Dynamic Suspend is configured separately for the GPIO pins and UART/Modem Control pins.

2.6. Latch Value (SUSPEND)

When dynamic suspend is enabled, this value is written to the interface pin latch when the CP210x device moves from the configured USB state to the suspend USB state (see chapter nine of USB 2.0 specification for more information on USB device states). Not available on RXD, SUSPEND, SUSPEND, and GPIO pins set for device-controlled function.

3. Intended Use and Limitations

Using the various customizable options in conjunction with one another can achieve a broad range of device characteristics. The following are some of the most common uses of those options and their limitations.

3.1. High-Impedance Input

By configuring for open-drain operation and writing logic high (1) to the latch, an interface pin assumes a high impedance state. This input pin will have electrical characteristics as listed in table 3 of the device datasheet.

3.2. Push-Pull Output

By configuring for push-pull operation, an interface pin operates as a push-pull output. The output voltage is determined by pin's latch value. This output pin will have electrical characteristics as listed in table 3 of the device datasheet. This type of output is most often used to connect directly to another device.

3.3. Open-Drain Output

By configuring for open-drain operation, an interface pin operates as an open-drain output. The output voltage is determined by the pin's latch value. If the pin latch value is 1, the pin is pulled up to VDD (CP2102) or VIO (CP2103, CP2104, CP2105) through an on-chip pull-up resistor. The pin can also be safely pulled up to 5 V if an external pull-up resistor is added. This output pin will have electrical characteristics as listed in Table 3 of the device data sheet.

3.4. Low Power State

By writing logic low to the latch, an interface pin is grounded and consumes minimal power with weak pull-ups disabled. This setting is best for unused interface pins that are not connected to external circuitry.

3.5. Reset Behavior

All interface pins temporarily float high during a device reset. If this behavior is undesirable, a strong pull-down (10 k Ω) can be used to ensure the pin remains low during reset.

3.6. DLL Organization

The configuration options presented in this document are included in the *CP210xManufacturing.DLL* and the *CP210xRuntime.DLL*. The functions defined in the *CP210xManufacturing.DLL* are meant for use when the product is manufactured, and should not be distributed with the end product. Configuration options set using this DLL do not take effect until the CP210x device is reset.

The *CP210xRuntime.DLL* is meant for distribution with the end product. This library includes functions to read and write the GPIO pins only. These functions take effect immediately, instead of at the next device reset.

4. Default Device Behavior

Table 4 defines the default device behavior programmed at the factory.

Table 2. CP2103 Default Settings

Signal	Pin #	Mode	Latch Value (Reset)	Controlled By
RI	1	Open-Drain	1	Win32 COM API or USBXpress
DCD	28	Open-Drain	1	Win32 COM API or USBXpress
DTR	27	Push-Pull	1	Win32 COM API or USBXpress
DSR	26	Open-Drain	1	Win32 COM API or USBXpress
TXD	25	Push-Pull	1	Win32 COM API or USBXpress
RXD	24	Open-Drain	1	Win32 COM API or USBXpress
RTS	23	Push-Pull	1	Win32 COM API or USBXpress
CTS	22	Open-Drain	1	Win32 COM API or USBXpress
SUSPEND	12	Push-Pull	1	USB Device State
$\overline{\text{SUSPEND}}$	11	Push-Pull	0	USB Device State
GPIO.0	19	Open-Drain	1	Manually by Host
GPIO.1	18	Open-Drain	1	Manually by Host
GPIO.2	17	Open-Drain	1	Manually by Host
GPIO.3	16	Open-Drain	1	Manually by Host
Note: Global settings: Weak Pull-up = ON; UART Dynamic Suspend = OFF; GPIO Dynamic Suspend = OFF				

Table 3. CP2104 Default Settings

Signal	Pin #	Mode	Latch Value (Reset)	Controlled By
RI	1	Open-Drain	1	Win32 COM API or USBXpress
DCD	24	Open-Drain	1	Win32 COM API or USBXpress
DTR	23	Push-Pull	1	Win32 COM API or USBXpress
DSR	22	Open-Drain	1	Win32 COM API or USBXpress
TXD	21	Push-Pull	1	Win32 COM API or USBXpress
RXD	20	Open-Drain	1	Win32 COM API or USBXpress
RTS	19	Push-Pull	1	Win32 COM API or USBXpress
CTS	18	Open-Drain	1	Win32 COM API or USBXpress
SUSPEND	17	Push-Pull	1	USB Device State
$\overline{\text{SUSPEND}}$	15	Push-Pull	0	USB Device State
GPIO_0	14	Open-Drain	1	Manually by Host
GPIO_1	13	Open-Drain	1	Manually by Host
GPIO_2	12	Open-Drain	1	Manually by Host
GPIO_3	11	Open-Drain	1	Manually by Host
Note: Global settings: Weak Pull-up = ON; UART Dynamic Suspend = OFF; GPIO Dynamic Suspend = OFF				

Table 4. CP2105 Default Settings

Signal	Pin #	Mode	Latch Value (Reset)	Controlled By
TXD_SCI	21	Push-Pull	1	Win32 COM API or USBXpress
RXD_SCI	20	Open-Drain	1	Win32 COM API or USBXpress
RTS_SCI	19	Push-Pull	1	Win32 COM API or USBXpress
CTS_SCI	18	Open-Drain	1	Win32 COM API or USBXpress
GPIO_0_SCI	24	Open-Drain	1	Manually by Host
GPIO_1_SCI	23	Open-Drain	1	Manually by Host
GPIO_2_SCI	22	Push-Pull	1	Manually by Host
$\overline{\text{SUSPEND_SCI}}$	1	Push-Pull	0	USB Device State
TXD_ECI	13	Open-Drain	1	Win32 COM API or USBXpress
RXD_ECI	12	Open-Drain	1	Win32 COM API or USBXpress
RTS_ECI	11	Open-Drain	1	Win32 COM API or USBXpress
CTS_ECI	10	Open-Drain	1	Win32 COM API or USBXpress
GPIO_0_ECI	15	Push-Pull	1	Manually by Host
GPIO_1_ECI	14	Open-Drain	1	Manually by Host
$\overline{\text{SUSPEND_ECI}}$	17	Push-Pull	0	USB Device State
<ol style="list-style-type: none"> Both interfaces on the CP2105 are configured for GPIO mode by default. Global settings: Weak Pull-up = ON; UART Dynamic Suspend = OFF; GPIO Dynamic Suspend = OFF 				

5. CP210x Port Configuration Utility

The CP210x Port Configuration Utility uses the *CP210xManufacturing.DLL* to customize the CP210x EEPROM settings related to port pin behavior. The functions in this API (*CP210x_SetPortConfig()* and *CP210x_GetPortConfig()*) give software access to these settings across the USB connection. For more information on this API and corresponding functions, refer to “AN144: CP21xx Device Customization Guide”.

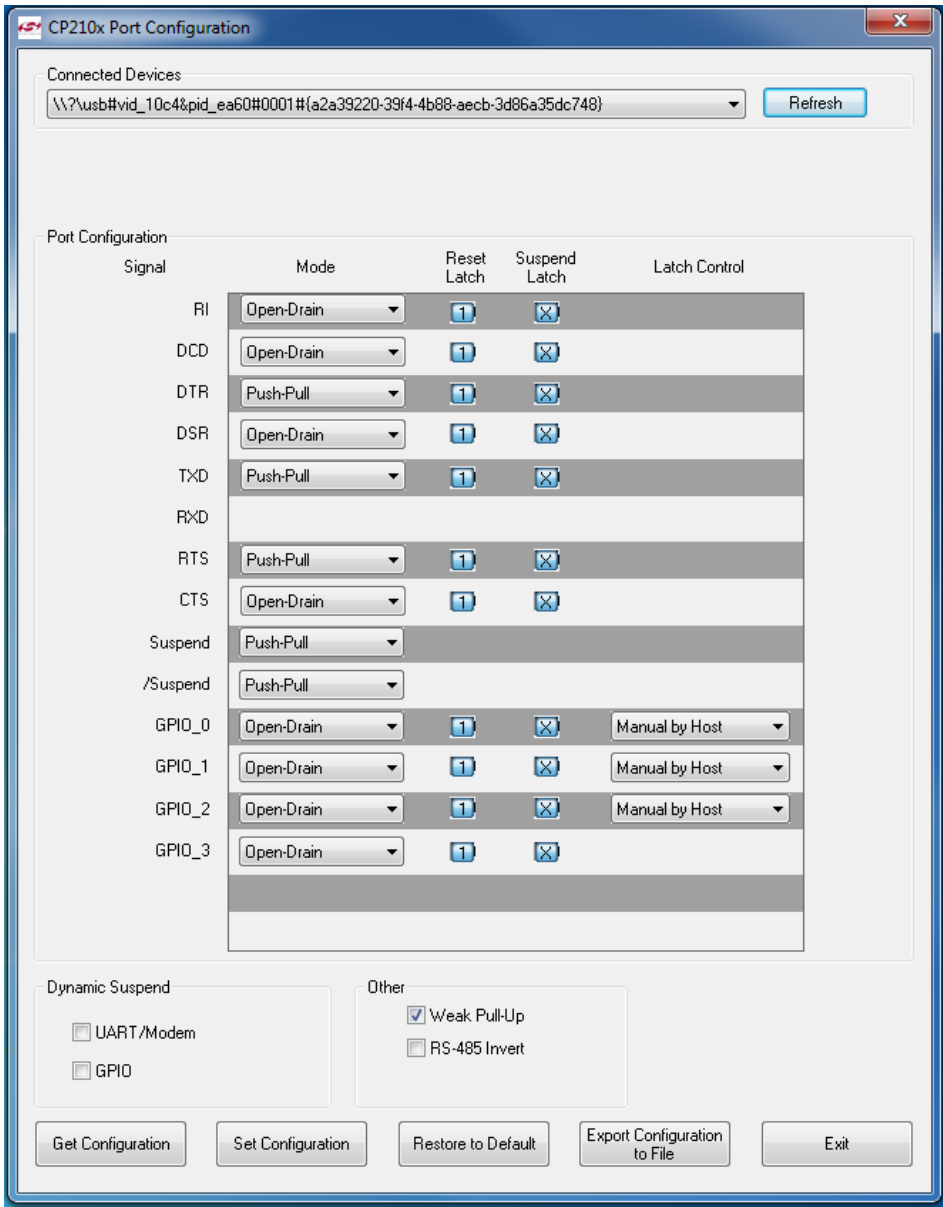


Figure 1. Main Window

The main window of the CP210x Port Configuration Utility (Figure 2) has a drop down list at the top showing all of the connected CP210x devices; this is where the CP210x device to be programmed can be selected. If a CP210x device is plugged in after the program has been started, click the Refresh button to obtain the most recent list of connected CP210x devices. The grid in the middle contains all the port configuration options available. Located at the bottom of the window are buttons labeled “Get Configuration”, “Set Configuration”, “Restore to Default”, and “Export Configuration”. The next sections will further explain how and when to use these buttons.

5.1. Obtaining the Current Configuration of a Device

Click the Refresh button to obtain the most current list of connected CP210x devices. Clicking the “Get Configuration” button will obtain the current configuration of the device that is selected.

Note: If a CP210x device is automatically detected on application start it will automatically retrieve and display this device's configuration. However, when a new CP210x device is selected the application does not automatically retrieve and display this device's configuration. Pressing the “Get Configuration” button will retrieve and display the newly selected device's configuration in the window. Because the configuration remains static while switching devices, multiple devices can be programmed without having to re-enter the desired settings when a new device is selected.

5.2. Customizing the Current Configuration of a Device

Once the current configuration has been retrieved from the device, it can be modified as desired and then programmed back into the device.

5.3. Programming the Current Configuration to a Device

Click on the “Set Configuration” button at any time to program the displayed configuration to the currently selected device. Because the displayed configuration remains static while switching devices, multiple devices can be programmed without having to re-enter the desired settings when a new device is selected.

Note: Parameters on the CP2104 and CP2105 can only be modified once and cannot be changed back to their default values.

5.4. Restoring the Default Baud Rate Configuration (CP2103)

Clicking the “Restore to Default” button will restore the default settings shown in Table 2 on page 3 to the device selected.

5.5. Exporting the Configuration to a File

To export all the custom settings in the EEPROM or EPROM into a file, click the “Export Configuration” to File button. A common dialog will come up that allows a file to be specified for the output, as shown in Figure 2.

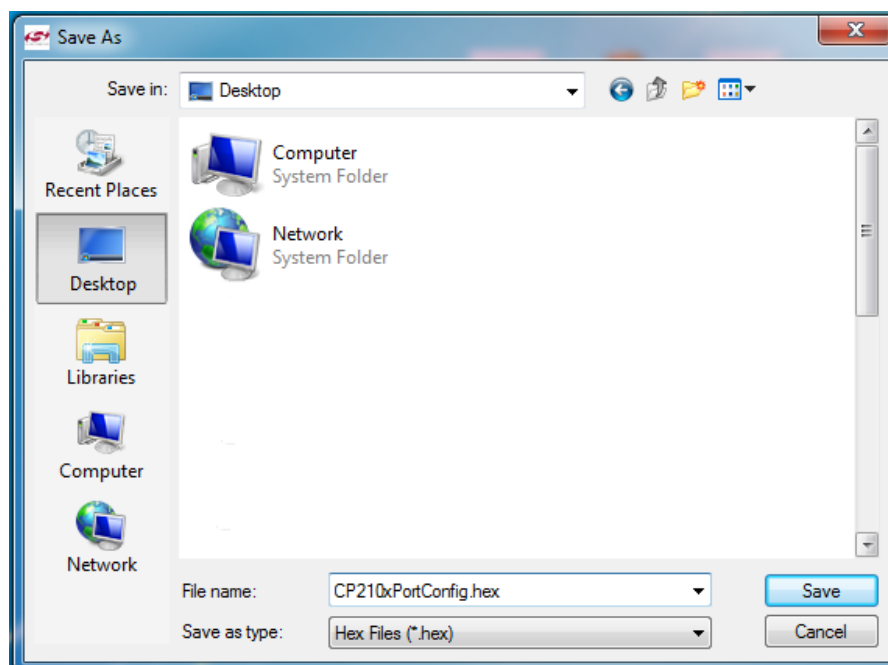


Figure 2. File Dialog

5.6. CP2105 GPIO Mode and Modem Mode

Each interface on the CP2105 can be configured for either GPIO mode or modem mode. This allows the SCI and ECI to have either modem control signals or GPIO signals available at various pins. Table 5 shows the functions that are available in each mode. By default, both interfaces are configured for GPIO mode.

Table 5. CP2105 Modem Mode and GPIO Mode

Interface	Pin #	Modem Mode	GPIO Mode
Standard Communications Interface	24	DCD_SCI	GPIO_0_SCI
	23	DTR_SCI	GPIO_1_SCI
	22	DSR_SCI	GPIO_2_SCI
	1	RI_SCI	$\overline{\text{SUSPEND_SCI}}$
Enhanced Communications Interface	15	DTR_ECI	GPIO_0_ECI
	14	DSR_ECI	GPIO_1_ECI
	17	RI_ECI	$\overline{\text{SUSPEND_SCI}}$

Only one mode can be selected for each interface. Also, the mode of the CP2105 can only be configured once and cannot be reset to the default configuration after being programmed. When changing the mode of the device, these functions must be called in the following order:

CP210x_SetDeviceMode ()

CP210x_SetDualPortConfig ()

Both of the functions above can be found in CP210xManufacturingDLL.h. Refer to “AN144: CP21xx Device Customization Guide” for more information on the two functions *CP210x_SetDeviceMode ()*, *CP210x_SetDualPortConfig ()*, and the corresponding AN144SW package with CP210xManufacturingDLL.h.

6. CP210x Port Read/Write Example

The CP210x Port Read/Write Example illustrates how the GPIO latch can be read from and written to using the *CP210xRuntime.DLL*. The functions in this API (*CP210xRT_ReadLatch()* and *CP210xRT_WriteLatch()*) give host-based software access to the CP210x device's GPIO latch using the USB connection as shown in Figure 3.

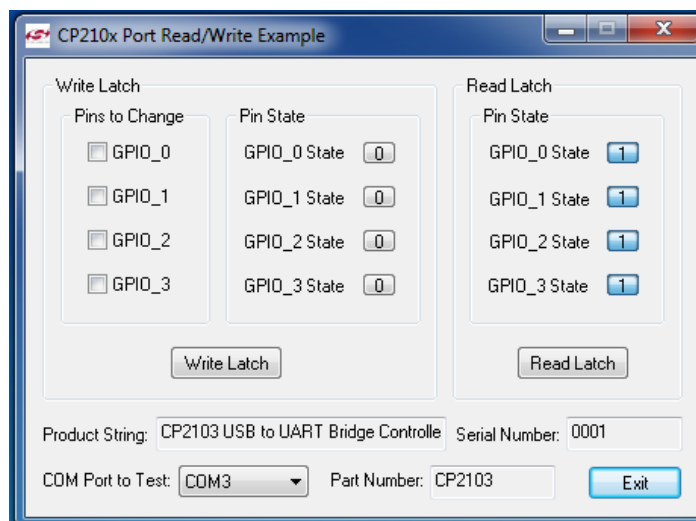


Figure 3. Main Window

The main window of the CP210x Port Read/Write Example contains one section to write the GPIO latch, and another to read the GPIO latch. Below that is a list to select a COM port, and display boxes for the device part number, product string, and serial number.

To write new values to the latch, select which GPIO pins to update, and set the pin state for each. Any GPIO pins not selected to change will remain static when the “Write Latch” button is pressed. The “Write Latch” button calls the *CP210xRT_WriteLatch()* function followed by the *CP210xRT_ReadLatch()* function, which updates the values displayed in the Read Latch portion of the dialog. At any time, the “Read Latch” button can be pressed to read in the current GPIO pin state of the device.

7. Creating Custom Applications using *CP210xRuntime.DLL*

Custom applications can use the CP210x Runtime API implemented in *CP210xRuntime.DLL*. To use functions implemented in *CP210xRuntime.DLL*, link *CP210xRuntime.LIB* with your Visual C++ 6.0 application. Include *CP210xRuntimeDLL.h* in any file that calls functions implemented in *CP210xRuntime.DLL*.

8. CP210x Runtime API Functions

The CP210x Runtime API provides access to the GPIO port latch, and is meant for distribution with the product containing a CP210x device.

- *CP210xRT_ReadLatch()*—Returns the GPIO port latch of a CP210x device.
- *CP210xRT_WriteLatch()*—Sets the GPIO port latch of a CP210x device.
- *CP210xRT_GetPartNumber()*—Returns the 1-byte Part Number of a CP210x device.
- *CP210xRT_GetProductString()*—Returns the product string programmed to the device.
- *CP210xRT_GetDeviceSerialNumber()*—Returns the serial number programmed to the device.
- *CP210xRT_GetDeviceInterfaceString()*—Returns the interface string programmed to the device.

Typically, the user initiates communication with the target CP210x device by opening a handle to a COM port using *CreateFile()* (See AN197: Serial Communication Guide for CP210x). The handle returned allows the user to call the API functions listed above. Each of these functions are described in the following sections. Type definitions and constants are defined in the file CP210xRuntimeDLL.h.

Note: Functions calls into this API are blocked until completed. This can take several milliseconds depending on USB traffic.

8.1. CP210xRT_ReadLatch

Description: Gets the current port latch value from the device.

Supported Devices: CP2103, CP2104, CP2105

Location: CP210x Runtime DLL

Prototype: CP210x_STATUS CP210xRT_ReadLatch(HANDLE Handle, LPBYTE Latch)

Parameters:

1. Handle—Handle to the Com port returned by *CreateFile()*.
2. Latch—Pointer for 1-byte return GPIO latch value [Logic High = 1, Logic Low = 0].

Return Value: CP210x_STATUS = CP210x_SUCCESS,
CP210x_INVALID_HANDLE,
CP210x_DEVICE_IO_FAILED
CP210x_FUNCTION_NOT_SUPPORTED

8.2. CP210xRT_WriteLatch

Description: Sets the current port latch value for the device.

Supported Devices: CP2103, CP2104, CP2105

Location: CP210x Runtime DLL

Prototype: CP210x_STATUS CP210xRT_WriteLatch(HANDLE Handle, BYTE Mask, BYTE Latch)

Parameters:

1. Handle—Handle to the Com port returned by *CreateFile()*.
2. Mask—Determines which pins to change [Change = 1, Leave = 0].
3. Latch—1-byte value to write to GPIO latch [Logic High = 1, Logic Low = 0]

Return Value: CP210x_STATUS = CP210x_SUCCESS,
CP210x_INVALID_HANDLE,
CP210x_DEVICE_IO_FAILED
CP210x_FUNCTION_NOT_SUPPORTED

8.3. CP210xRT_GetPartNumber

Description: Gets the part number of the current device.

Supported Devices: CP2101, CP2102, CP2103, CP2104, CP2105

Location: CP210x Runtime DLL

Prototype: `CP210x_STATUS CP210xRT_GetPartNumber(HANDLE Handle, LPBYTE PartNum)`

Parameters:

1. Handle—Handle to the Com port returned by *CreateFile()*.
2. PartNum—Pointer to a byte containing the return code for the part number.

Return Value: CP210x_STATUS = CP210x_SUCCESS,
CP210x_INVALID_HANDLE,
CP210x_DEVICE_IO_FAILED

8.4. CP210xRT_GetDeviceProductString

Description: Gets the product string in the current device.

Supported Devices: CP2101, CP2102, CP2103, CP2104, CP2105

Location: CP210x Runtime DLL

Prototype: `CP210x_STATUS CP210xRT_GetDeviceProductString(HANDLE cyHandle, LPVOID lpProduct, LPBYTE lpbLength, BOOL bConvertToASCII = TRUE)`

Parameters:

1. Handle—Handle to the Com port returned by *CreateFile()*.
2. lpProduct—Variable of type CP210x_PRODUCT_STRING returning the NULL terminated product string.
3. lpbLength—Length in characters (not bytes) not including a NULL terminator.
4. ConvertToASCII—Boolean that determines whether the string should be left in Unicode, or converted to ASCII. This parameter is true by default, and will convert to ASCII.

Return Value: CP210x_STATUS = CP210x_SUCCESS,
CP210x_INVALID_HANDLE,
CP210x_DEVICE_IO_FAILED
CP210x_INVALID_PARAMETER

8.5. CP210xRT_GetDeviceSerialNumber

Description: Gets the serial number in the current device.

Supported Devices: CP2101, CP2102, CP2103, CP2104, CP2105

Location: CP210x Runtime DLL

Prototype: `CP210x_STATUS CP210xRT_GetDeviceSerialNumber(HANDLE cyHandle, LPVOID lpProduct, LPBYTE lpbLength, BOOL bConvertToASCII = TRUE)`

Parameters:

1. Handle—Handle to the Com port returned by *CreateFile()*.
2. lpProduct—Variable of type CP210x_SERIAL_STRING returning the NULL terminated serial string.
3. lpbLength—Length in characters (not bytes) not including a NULL terminator.
4. ConvertToASCII—Boolean that determines whether the string should be left in Unicode, or converted to ASCII. This parameter is true by default, and will convert to ASCII.

Return Value: CP210x_STATUS = CP210x_SUCCESS,
CP210x_INVALID_HANDLE,
CP210x_DEVICE_IO_FAILED
CP210x_INVALID_PARAMETER

8.6. CP210xRT_GetDeviceInterfaceString

Description: Gets the interface string of the current device.

Supported Devices: CP2105

Location: CP210x Runtime DLL

Prototype: `CP210x_STATUS CP210xRT_GetDeviceInterfaceString(HANDLE cyHandle, LPVOID lpInterfaceString, LPBYTE lpbLength, BOOL bConvertToASCII = TRUE)`

Parameters:

1. Handle—Handle to the Com port returned by *CreateFile()*.
2. lpInterfaceString—Variable of type CP210x_SERIAL_STRING returning the NULL terminated interface string.
3. lpbLength—Length in characters (not bytes) not including a NULL terminator.
4. ConvertToASCII—Boolean that determines whether the string should be left in Unicode, or converted to ASCII. This parameter is true by default, and will convert to ASCII.

Return Value: CP210x_STATUS = CP210x_SUCCESS,
CP210x_INVALID_HANDLE,
CP210x_DEVICE_IO_FAILED
CP210x_INVALID_PARAMETER

DOCUMENT CHANGE LIST

Revision 0.1 to Revision 0.2

- Reworded Open Drain Output description for clarity.

Revision 0.2 to Revision 0.3

- Added CP210xRT_GetProductString
- Added CP210xRT_GetDeviceSerialNumber
- Added CP210xRT_GetDeviceProductString

Revision 0.3 to Revision 0.4

- Added support for CP2104 and CP2105
- Added CP210xRT_GetDeviceInterfaceString
- Added Table 3 and Table 4
- Added Section 5.6. CP2105 GPIO Mode and Modem Mode
- Removed the Appendix

CONTACT INFORMATION

400 West Cesar Chavez
Austin, TX 78701
Tel: 1+(512) 416-8500
Fax: 1+(512) 416-9669
Toll Free: 1+(877) 444-3032

Please visit the Silicon Labs Technical Support web page:
<https://www.silabs.com/support/pages/contacttechnicalsupport.aspx>
and register to submit a technical support request.

The information in this document is believed to be accurate in all respects at the time of publication but is subject to change without notice. Silicon Laboratories assumes no responsibility for errors and omissions, and disclaims responsibility for any consequences resulting from the use of information included herein. Additionally, Silicon Laboratories assumes no responsibility for the functioning of undescribed features or parameters. Silicon Laboratories reserves the right to make changes without further notice. Silicon Laboratories makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Silicon Laboratories assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. Silicon Laboratories products are not designed, intended, or authorized for use in applications intended to support or sustain life, or for any other application in which the failure of the Silicon Laboratories product could create a situation where personal injury or death may occur. Should Buyer purchase or use Silicon Laboratories products for any such unintended or unauthorized application, Buyer shall indemnify and hold Silicon Laboratories harmless against all claims and damages.

Silicon Laboratories, Silicon Labs, and USBXpress are trademarks of Silicon Laboratories Inc.

Other products or brandnames mentioned herein are trademarks or registered trademarks of their respective holders.