AN223

PORT CONFIGURATION AND GPIO FOR CP210X

Relevant Devices

This application note applies to the following devices:
CP2103

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 /Suspend signals. These signals are used to gate power consumption of external circuitry for bus-powered USB products.

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, Suspend, /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	Logic low when there is UART data to transmit; otherwise logic high
GPIO.1	Receive LED	Logic low 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.

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) 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 kΩ) 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 2 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
/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

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: CP210x Device Customization Guide”.

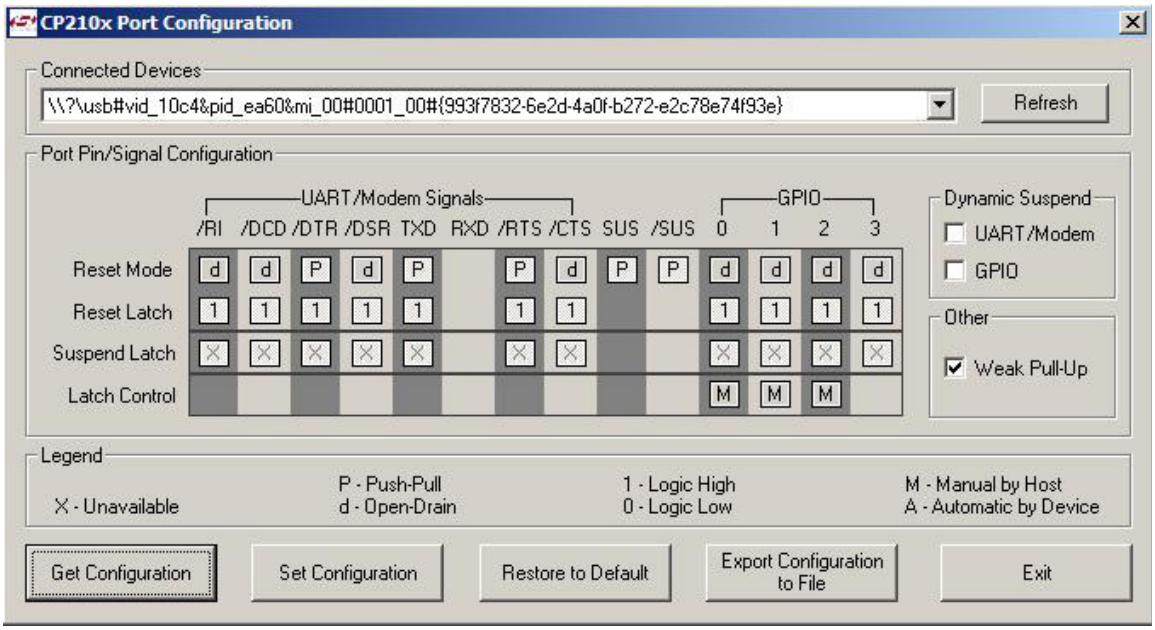

Figure 1. Main Window

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.

5.4. Restoring the Default Baud Rate Configuration

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 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. This file is used by Silicon Laboratories to mass-produce parts with the same settings. Please contact a sales representative for availability and restrictions for this feature.

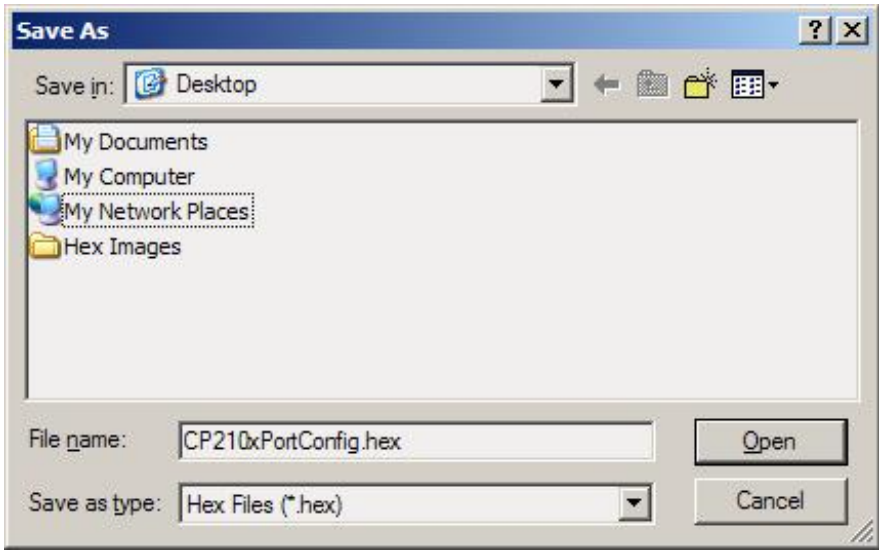

Figure 2. File Dialog

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.

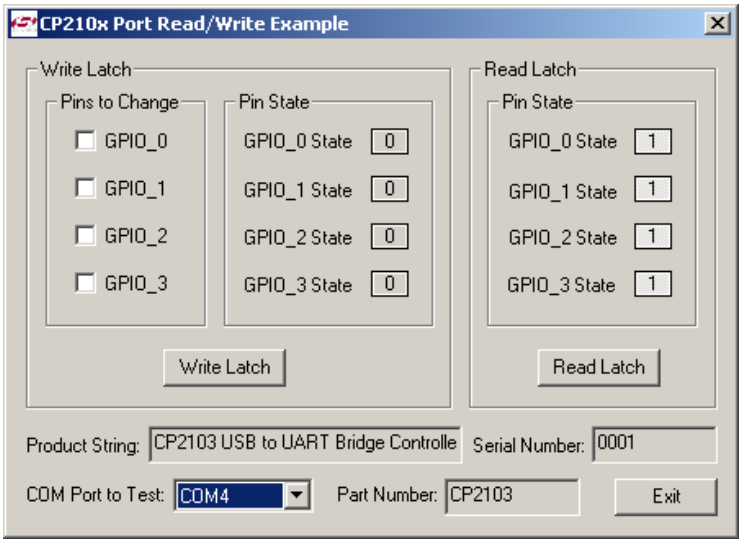

Figure 3. Main Window

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.

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 "Appendix—Type Definitions and Constants" section.

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

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

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

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

Location: `CP210x Runtime DLL`

Prototype:
```
CP210x_STATUS CP210xRT_GetDeviceProductString(HANDLEcyHandle,
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

Location: `CP210x Runtime DLL`

Prototype: CP210x_STATUS CP210xRT_GetDeviceSerialNumber (HANDLE cyHandle, LPVOID pProduct, 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

APPENDIX—TYPE DEFINITIONS AND CONSTANTS

Type Definitions from C++ Header File *CP210xRuntimeDLL.h*

```
// GetDeviceVersion() return codes
#define     CP210x_CP2101_VERSION           0x01
#define     CP210x_CP2102_VERSION           0x02
#define     CP210x_CP2103_VERSION           0x03

// Return codes
#define     CP210x_SUCCESS                  0x00
#define     CP210x_DEVICE_NOT_FOUND         0xFF
#define     CP210x_INVALID_HANDLE           0x01
#define     CP210x_INVALID_PARAMETER        0x02
#define     CP210x_DEVICE_IO_FAILED         0x03
#define     CP210x_FUNCTION_NOT_SUPPORTED   0x04
#define     CP210x_GLOBAL_DATA_ERROR        0x05
#define     CP210x_COMMAND_FAILED           0x08
#define     CP210x_INVALID_ACCESS_TYPE      0x09

// Type definitions
typedef     int         CP210x_STATUS;

// Buffer size limits
#define                 CP210x_MAX_PRODUCT_STRLEN       126
#define                 CP210x_MAX_SERIAL_STRLEN        63

// Type definitions
typedef     char CP210x_PRODUCT_STRING[CP210x_MAX_PRODUCT_STRLEN];
typedef     char CP210x_SERIAL_STRING[CP210x_MAX_SERIAL_STRLEN];

// Mask and Latch value bit definitions
#define     CP210x_GPIO_0                   0x01
#define     CP210x_GPIO_1                   0x02
#define     CP210x_GPIO_2                   0x04
#define     CP210x_GPIO_3                   0x08
```

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

NOTES:

CONTACT INFORMATION

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
Email: MCUinfo@silabs.com
Internet: www.silabs.com