# CP210x/CP211x Device Customization Guide

**Relevant Devices**
This application note applies to the following devices:
CP2101, CP2102, CP2103, CP2104, CP2105, CP2108, CP2110, CP2112, CP2114

## 1. Introduction

This document explains the steps required to customize a fixed function USB device. It is intended for developers creating products based on the CP210x/CP211x USB Bridge Controllers. It contains information about obtaining a Vendor ID (VID) and Product ID (PID) for a CP210x/CP211x product and describes the steps necessary for customizing the device descriptors. Refer to www.silabs.com/interface for the latest revisions of this document and other application notes related to the CP210x/CP211x device families.

### 1.1. USB Logos and Certification Testing

USB is a widely used peripheral. The USB Implementers Forum, Inc. has introduced trademark-protected logos for use with qualified USB products. To use the logo, USB products are required to meet the standards of the USB Implementers Forum. For a product to have compliance and/or certification implies that the USB product has been tested by the USB-IF to meet the specification. Each type of USB product requires specific testing to be listed on the Integrators List. This is important not only to OEMs but to consumers because products tested and certified by the USB-IF are assured to work together. Compliance testing exists to help manufacturers measure how well their products match the respective USB specification. If a product has passed USB-IF compliance testing, the company can use the USB logo on the products.

### 1.2. USB Vendor IDs and Product IDs

Each device on a USB bus must have a unique Vendor ID (VID), Product ID (PID), and serial number combination. This ID system uniquely identifies the different devices on the bus to avoid conflicts. The PC uses the VID/PID to find the drivers (if any) to be used for the USB device. The VID/PID must be unique in that each USB device with the same VID/PID will use the same driver, and it is strongly recommended to make the PID unique to a particular design. The USB devices of a given VID/PID combination can be serialized, which allows the operating system to track not only a particular model, but also a specific board of that model.

Vendor IDs are owned by the vendor company and assigned by the USB Implementers Forum (USB-IF) only. Details about obtaining a unique VID can be found at www.usb.org/developers/vendor.

To obtain the right to license the USB-IF logo, register the product's VID and PID with USB-IF and submit the product to the USB-IF Compliance Program. USB-IF Compliance Program details are available at www.usb.org/developers/compliance. Once the product is certified, it can be added to the USB-IF Integrators List, and the “Certified USB” logo can be used on the product. The default Silicon Labs VID is 0x10C4 and the default Silicon Labs PID is dependent on the device. To obtain a unique PID for your CP210x/CP211x-based product, visit http://www.silabs.com/RequestPID. Note that customization of the USB strings is optional, but is strongly recommended. A unique VID/PID combination will prevent the driver from conflicting with any other USB driver.

# 2. Basic Device Customization

The steps to customize the CP210x family of devices is slightly different than customizing the CP211x devices. The CP210x devices require a driver, but the CP211x devices do not because they are of the HID class, which is natively supported by most operating systems. The next two sections describe the recommended steps for customizing the device based on the family, either the CP210x or the CP211x.

## 2.1. Summary of Steps for Customizing the CP210x Non-HID USB Devices

The CP210x family of devices provides communication from USB to UART. This requires a driver to interface to the device. There are two types of drivers provided. One is the Virtual COM port (VCP) driver which allows the device to appear to the PC's application software as a COM port. This driver is always used first to connect to the Device Customization Software program to change the PID since by default this driver has the same VID and PID as what is programmed in the default devices. After the Device Customization Software program is used to change the PID, the driver must match the new values that have been loaded in the device from the Device Customization Software. If the user wants to communicate to the device via a high level application program, a USBXpress driver can be used which provides this functionality. This driver must be downloaded and installed after using the Device Customization Software. Then the USBXpress driver or VCP driver (which ever one will be used) should be updated to make sure the driver matches the VID and PID in the device. A PC cannot have both the VCP and USBXpress drivers loaded with the same VID and PID, as this would cause USB device identification conflicts. In the end, only one driver can be used, either the VCP or the USBXpress.

There are VCP and USBXpress drivers for various operating systems, which are all listed on the website at http://www.silabs.com/products/mcu/pages/usbtouartbridgevcpdrivers.aspx for the VCP driver and at www.silabs.com/usbxpress for the USBXpress driver. The process described below must be followed each time a new operating system must be supported. If the driver has been certified for Windows 7 32-bit and then it is necessary to support Windows 7 64-bit, then the driver must be recertified. The Microsoft certification process must be initiated again, and the reseller fee must be paid to Microsoft for the 64-bit version of the driver. Microsoft requires this certification process which involves Windows hardware quality labs testing (WHQL). It certifies that the hardware or software has been tested by Microsoft to ensure compatibility. Device drivers that pass the WHQL tests are given a digitally signed certification file, which prevents Windows from displaying a warning message that the driver has not been certified by Microsoft.

Figure 1 shows the default VID and PID values for the device and drivers. To establish communication with the driver, the VID and PID of the device must match the driver. Notice that the default CP210x device VID and PID match the default VCP driver VID and PID numbers.

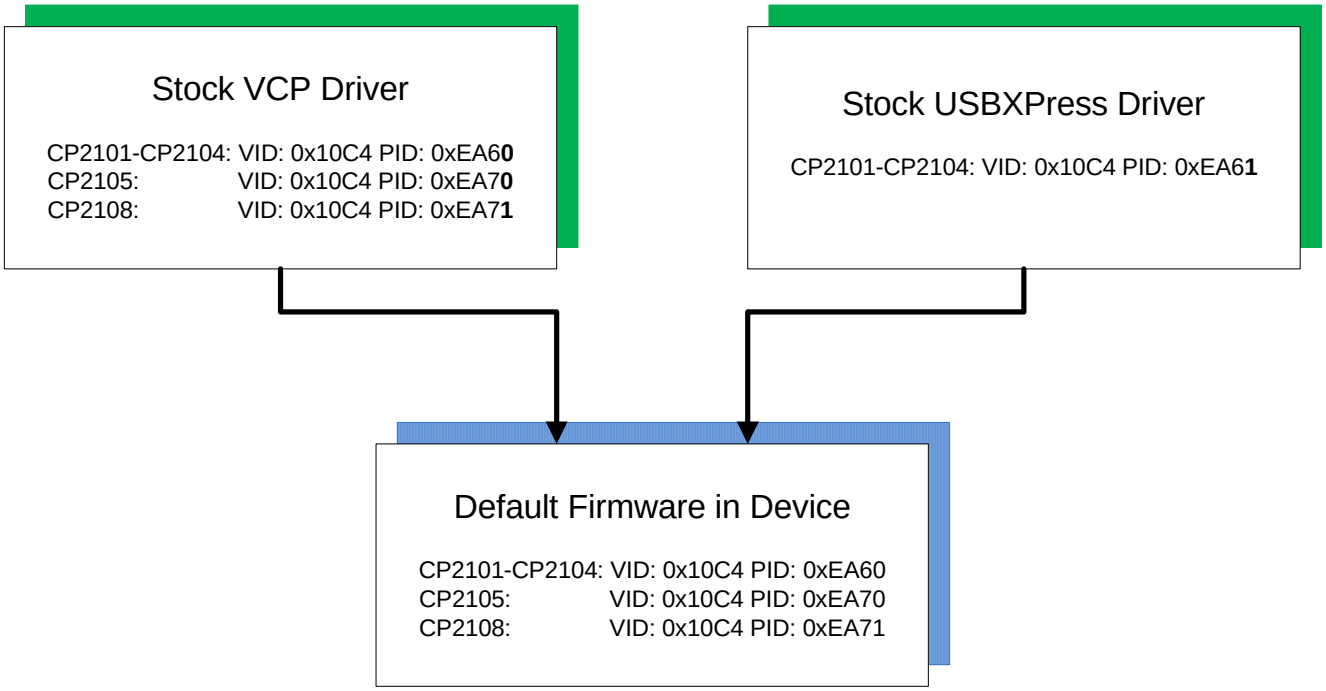



**Figure 1. Default VID and PID Values for Driver vs Firmware**

The steps to customize the CP210x USB devices are as follows:

1. Request a unique PID from Silicon Labs for your new product design: http://www.silabs.com/RequestPID, or obtain a VID/PID from usb.org.
2. Download the VCP driver appropriate for your operating system here: http://www.silabs.com/products/mcu/pages/usbtouartbridgevcpdrivers.aspx. The stock VCP driver from the website should match the default VID and PID in the CP210x. The VCP driver must be installed with matching VID and PID to communicate to the device.
3. Run the Device Customization Software program described in the next sections to change the descriptors in the device.
4. **(USBXpress Users Only):** If the desired driver is USBXpress, it can be download here: www.silabs.com/usbxpress. This driver allows direct access using Silicon Labs API commands to control the device. When this driver is initially downloaded it will not have the matching VID/PID of the CP210x devices. See Figure 1 for the default driver and device VID/PID.
5. Use the USB Driver Customization Wizard and instructions in AN220SW and AN220 “USB Driver Customization” https://www.silabs.com/Support%20Documents/TechnicalDocs/an220.pdf to update the driver for the new PID and any other descriptors that have been changed from Step 3. Be certain to use the version of AN220 software which corresponds to the correct driver when generating the modified drivers. Take care to verify that these customized drivers are completely correct, as none of the files in the driver package can change in any way once the driver has been certified. If you do not have the correct version of the AN220 software, please contact our support team www.silabs.com/contactsupport. Update either the VCP driver (COM port) or USBXpress driver (API commands) to match the device. The USB Driver Customization Wizard customizes the driver by changing the hardware installation files (.inf) in the driver package. The strings contained in the .inf files affect what is displayed in the “Found New Hardware Wizard” dialogs, Device Manager and the Registry. Any changes to the Windows® installation .inf files will require new Windows Hardware Quality Labs (WHQL) tests.
6. **(Microsoft Windows Only):** The customized driver is eligible for WHQL re-seller submissions to certify the driver. These submissions do not have the high cost and testing requirements of an original driver

submission. To certify a customized VCP or USBXpress driver, register at the WinQual site https://sysdev.microsoft.com to obtain a WinQual account with your company. Internet Explorer is the only web browser that can be used with the WHQL website. A Verisign ID is needed to register an account (instructions for obtaining one are available at Microsoft's website: http://msdn.microsoft.com/en-us/library/windows/hardware/hh801887?ppud=4. The correct Verisign ID is the CodeSigner Standard http://www.verisign.com/code-signing/microsoft-authenticode/index.html?sl=header.

7. **(Microsoft Windows Only):** After obtaining a WinQual account, notify the Silicon Labs support team www.silabs.com/contactsupport to be added as a registered Reseller. Provide the driver type (VCP or USBXpress) and version (e.g., v6.5) when requesting Reseller status.
8. **(Microsoft Windows Only):** Silicon Labs will add your company as a registered reseller. Your company must complete the provided directions to finish the recertification process.

**Note:** For further detailed instructions on Microsoft's submission process for recertification for a customized driver please view the document attached to the KnowledgeBase article called "How to Recertify a Customized Driver Package.pdf" located here: http://cp-siliconlabs.kb.net/article.aspx?article=89180&p=4120.

## 2.2. Summary of Steps for Customizing the CP211x HID USB Devices

The CP211x family does not require a driver because it is automatically recognized as part of the HID class, which simplifies the process. Most operating systems include native drivers. The CP211x will not fit a standard HID device type such as a keyboard or mouse. Any CP211x PC application will need to use the specific CP211x HID specification to communicate with it. This low-level HID specification is documented and provided by Silicon Labs in the form of a DLL.

The following are the steps to follow to customize the CP211x HID USB devices to ensure a unique VID/PID combination:

1. Request a unique PID from Silicon Labs for a new design: http://www.silabs.com/RequestPID.
2. Use the Device Customization Software program described in Section 3 below to change the descriptors in the firmware of the device.

## 3. Device Customization Software

The descriptors and other configurable options of the devices in the CP210x/CP211x families are modifiable using the Windows program CP21xx Device Customization Software.exe (Figure 2). The software program automatically recognizes the device that is plugged into the PC. When the Device Customization Software.exe is launched, the program searches the Windows registry for any CP21xx devices attached to the PC. The full path information for all of the devices found is inserted into the “Select Device” drop-down list, and the first device is selected automatically. This CP21xxDevice Customization Software.exe program is included in the software zip file in this application note. The descriptions of how to use the program is described in more detail in the following sections for the different families of devices. Be aware that one time programmable (OTP) devices can only be changed once. The program can access the fields but will not be able to program them more than once.

The CP21xx Device Customization Software uses the Windows Host API functions implemented by all the DLL files for the different families of devices mentioned. The Host API functions give read/write access to the descriptors contained in programmable areas of a connected device. Another option is implementing a custom application using the host API with the DLL suited to the individual needs of a particular production environment.

The descriptors can also be set in the factory at production time for large orders. Contact your Silicon Laboratories sales representative for details.

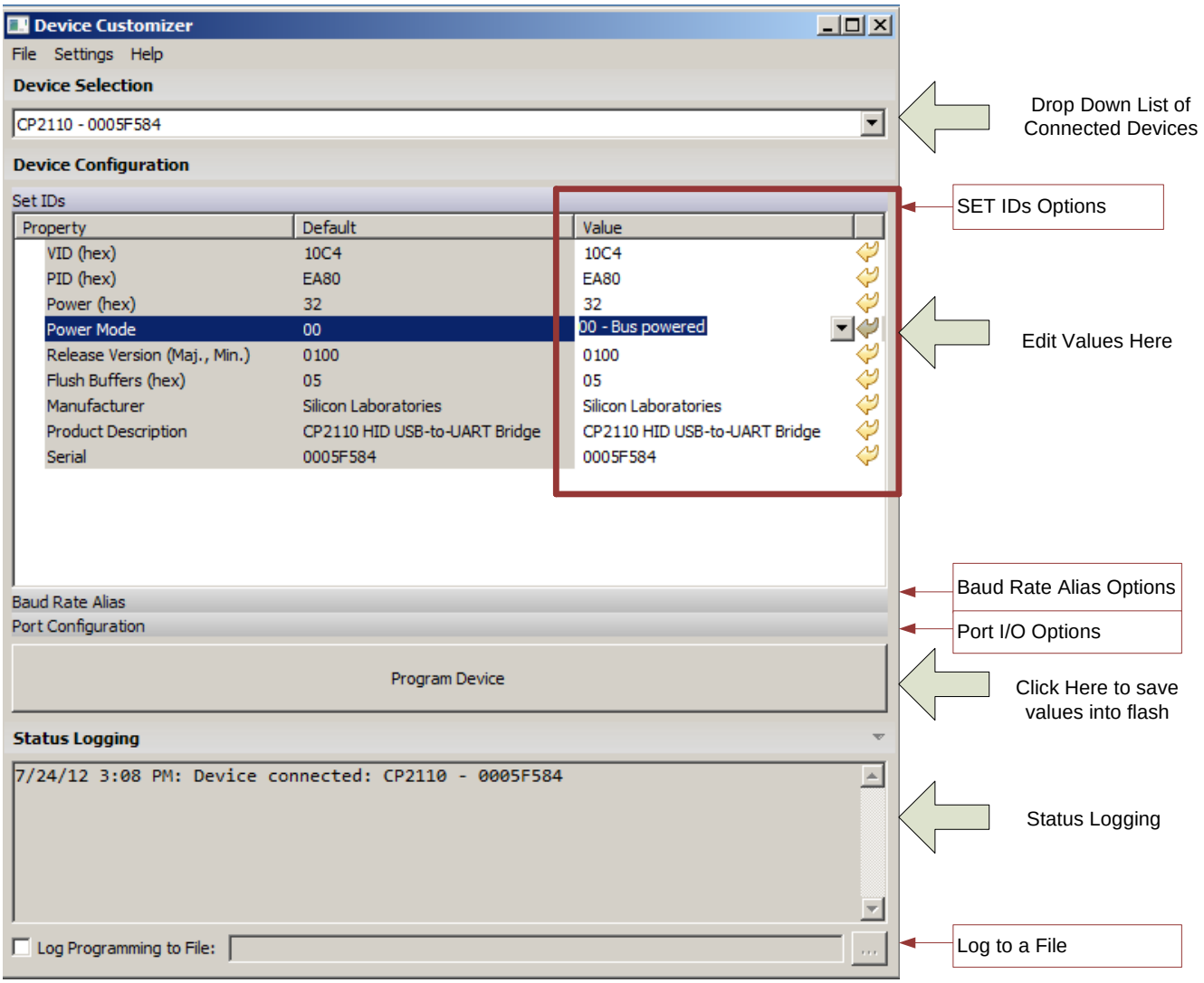



**Figure 2. Device Customization Software General Overview**

The above program is an example of the CP2110 device, but the program is similar for all devices. There are three main sections that can be edited. The first one is the Set IDs section as shown. Below that is the Baud Rate Alias section and then below that is the Port Configuration settings. Changes can be made in each of these sections and saved by clicking on the “Program Device” button. This will save the values into flash. If any of these three main sections are blank it is because there is no configuration for the Baud Rate Alias or Port Configuration. All devices will have the SETID Configuration. The Status Logging window updates and prints out all the transactions to verify the device has programmed correctly.

## 4. Changing Device Settings CP2110, CP2114

The CP2110 and CP2114 are both one time programmable devices. The various customizable fields of the CP2110/CP2114 devices are only programmable one time using the program. Be careful to change all settings under every tab before clicking the “Program Device” button.

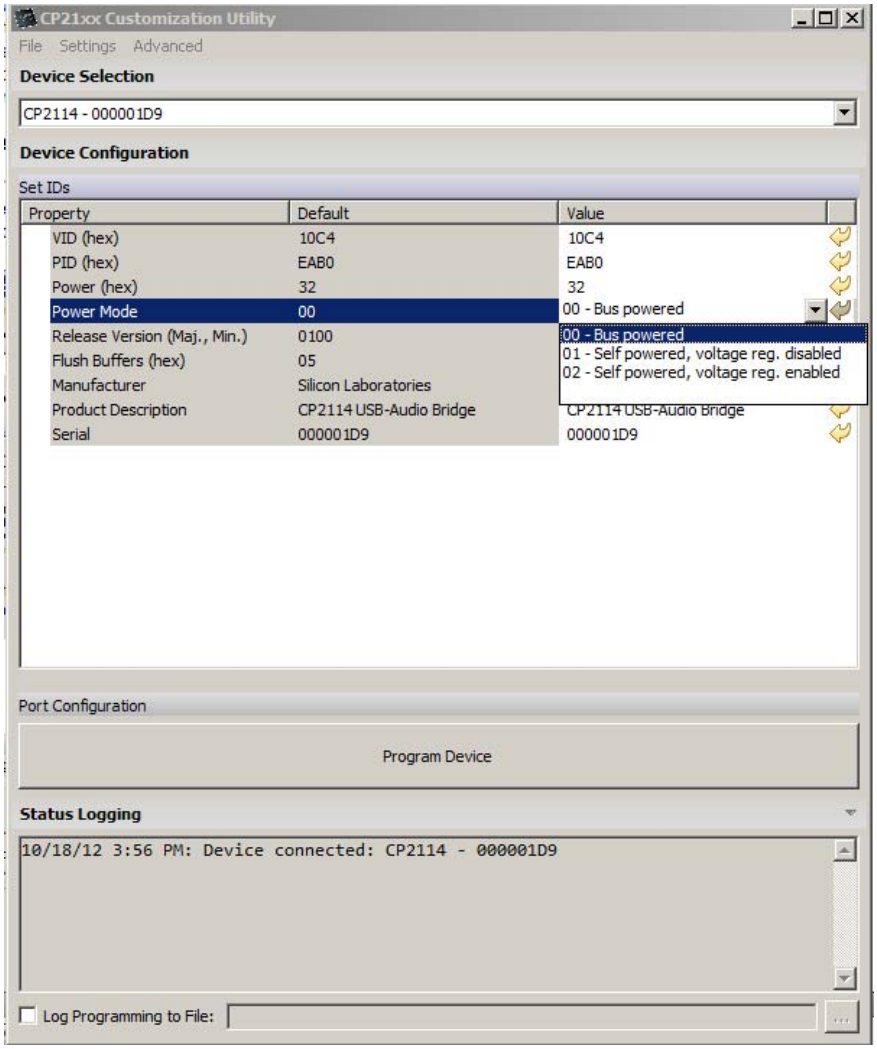



**Figure 3. CP2110, CP2114 Device Customization Software Set IDs Window**

When the Device Customization Software is launched, the program searches for any CP2110/CP2114 devices attached to the PC. The full path information for all of the devices found is inserted into the "Device Selection" drop-down list, and the first device is selected automatically. There is a SET IDs window and a Port Configuration Window for both these devices.

## 4.1. Set IDs Tab CP2110, CP2114 Devices

Figure 3 shows the following modifiable parameters for the CP2110/CP2114:

1. **VID**—The Vendor ID is a four hexadecimal digit number such as 10C4.
2. **PID**—The Product ID is a four hexadecimal digit number such as EAB0.
3. **Power**—This is a two hexadecimal digit number such as 10 with a maximum setting of 250. Note this number corresponds to units of current in 2 mA increments. Therefore a value of 10 corresponds to 32 mA.
4. **Power Mode**—There are three different power mode options. Click the field to see the options. 00 corresponds to Bus Powered, 01 is Self Powered with Voltage Regulator Disabled, 02 is Self Powered with Voltage Regulator Enabled.
5. **Release Version**—The release version is a numerical identifier of the device release version. Each field is a decimal number value 0–99. The first two digits are the major version number and the last two are the minor digits. 0100 corresponds to Major Version 01, Minor Version 00.

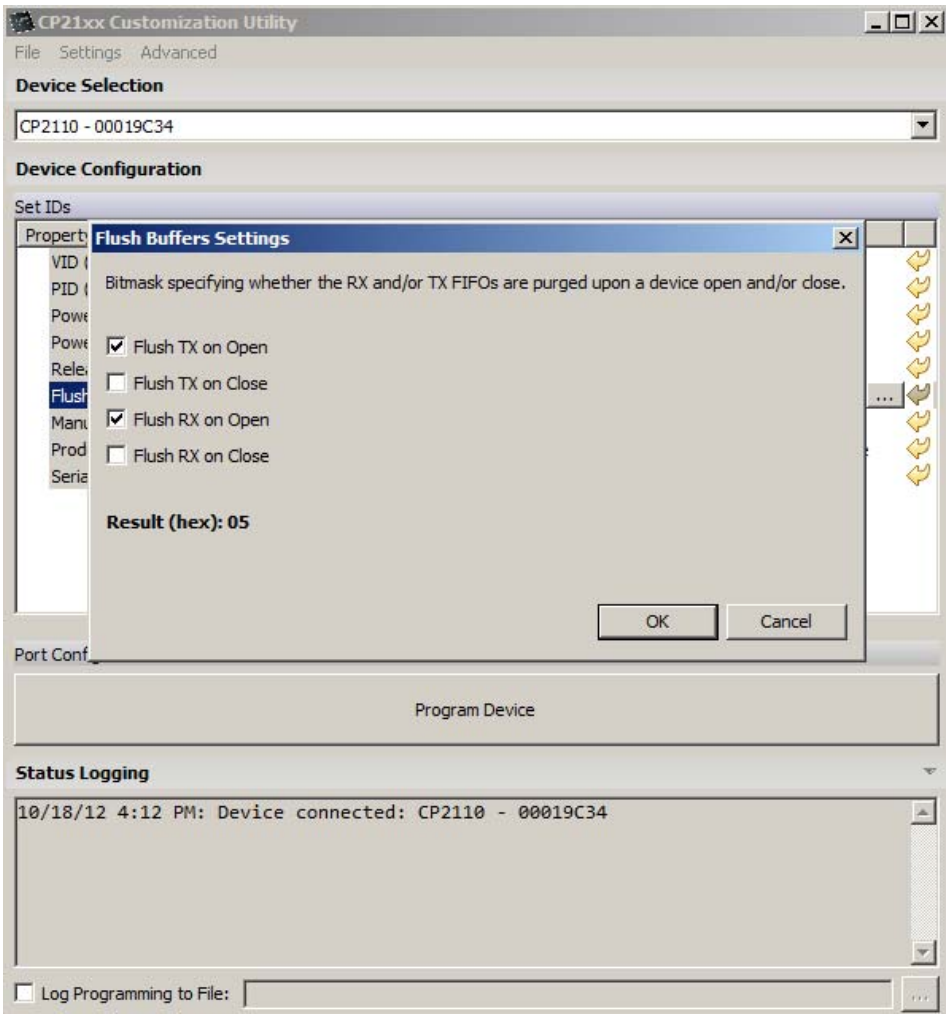



**Figure 4. CP2110/CP2114 Flush Buffers Options**

6. **Flush Buffers**—The flush buffers options shown above in Figure 4 allows open and close options on the RX and TX. Check the options required for the application.
7. **Manufacturer**—This is the name of the company manufacturing the product.
8. **Product Description**—The Product Description can be any sequence of up to 126 characters. Usually this is text which provides a description of the device, such as CP2103 USB to UART Bridge Controller.
9. **Serial Number**—The Serial Number can be any sequence of up to 63 characters.

## 4.2. Port Configuration Settings CP2110

The Port Configuration settings are available by clicking on the “Port Configuration” Tab. Click on the number in the value box to access a drop down menu of options for each pin configuration. Refer to the data sheet for further explanation of the different options available. After a value has been changed it will be highlighted in yellow. Click the Program Device button when complete.

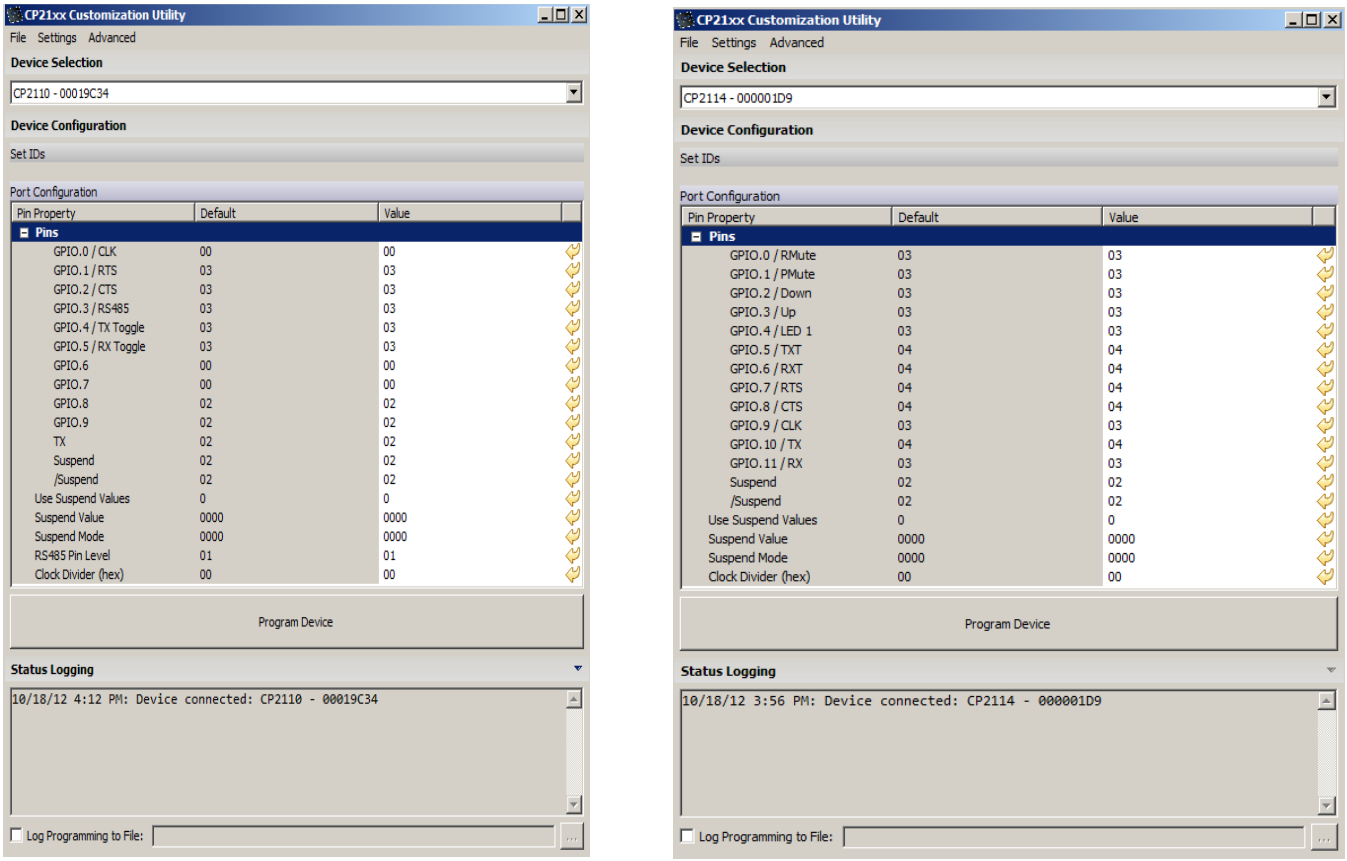



**Figure 5. CP2110 (LEFT) and CP2114 (RIGHT) Port Configuration Settings**

The GPIO pin settings in the CP2110/CP2114 can be modified to set the pins for input, Output Push-Pull, or Output Open Drain. Figure 5 shows the port configurations for the CP2110 (left) and the CP2114 (right). The alternate pin options are different and the default values are different. The GPIO pin configuration options and alternative functions are selectable from the drop down menu when clicking on the value. The Suspend Value setting allows the option for the latch to be enabled on any of the selected pins shown in Figure 6. A similar menu is available for the Suspend Mode allowing a selection between open-drain and push-pull for the different pins in suspend mode.

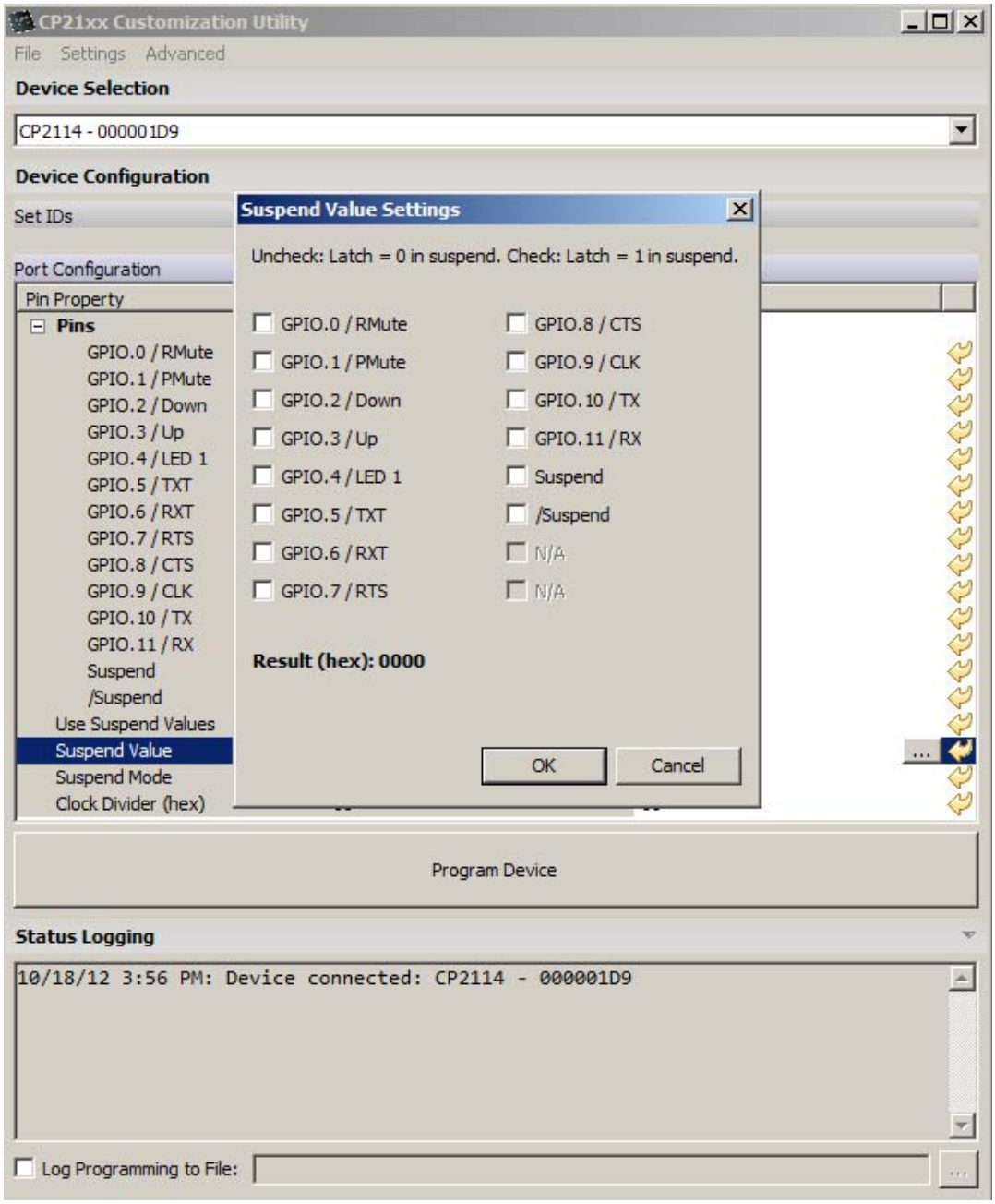



**Figure 6. CP2110/CP2114 Suspend Value Settings**

See “AN434: Interface Specification” for a full description of each of the customizable parameters.

If using the Device Customization Software to program multiple devices, the customized values can be saved to a text file using the File → Save and File → Save As commands. When connecting a new device, use the File → Open command to retrieve the saved settings, which are then directly programmable to the new device.

**Notes:**

1. Avoid connecting more than one device containing the same VID, PID, and serial number combination.
2. GPIO.0/CLK must be configured as a “CLK Output -Push Pull” before configuring the “CLK Output Divider” setting.

The Manufacturer String, Product String, and Serial Number are automatically converted to Unicode strings before programming.

## 4.3. DAC Configuration Application (CP2114 ONLY)

The DAC Configuration utility is accessed by clicking on the “Advanced” tab and selecting “DAC Configuration CP2114” shown in Figure 7. This is an application that allows audio configuration strings to be sent to the one-time programmable memory or RAM. The audio string is used by the CP2114 to configure the DAC/CODEC.

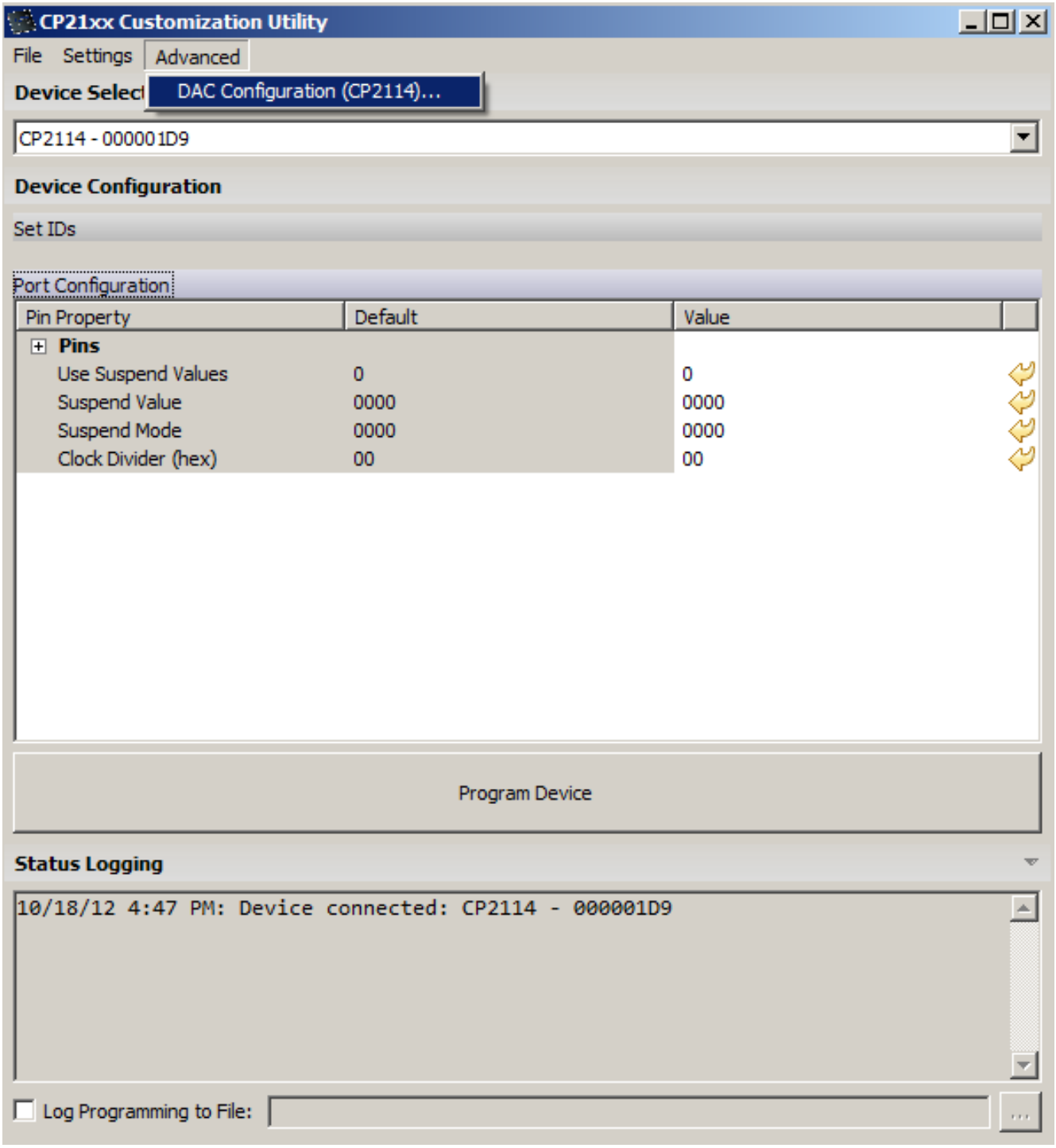



**Figure 7. Advanced Settings DAC Utility**

Figure 8 shows a screen shot of the DAC Utility. Section 4.4 details all the configuration settings for the utility.

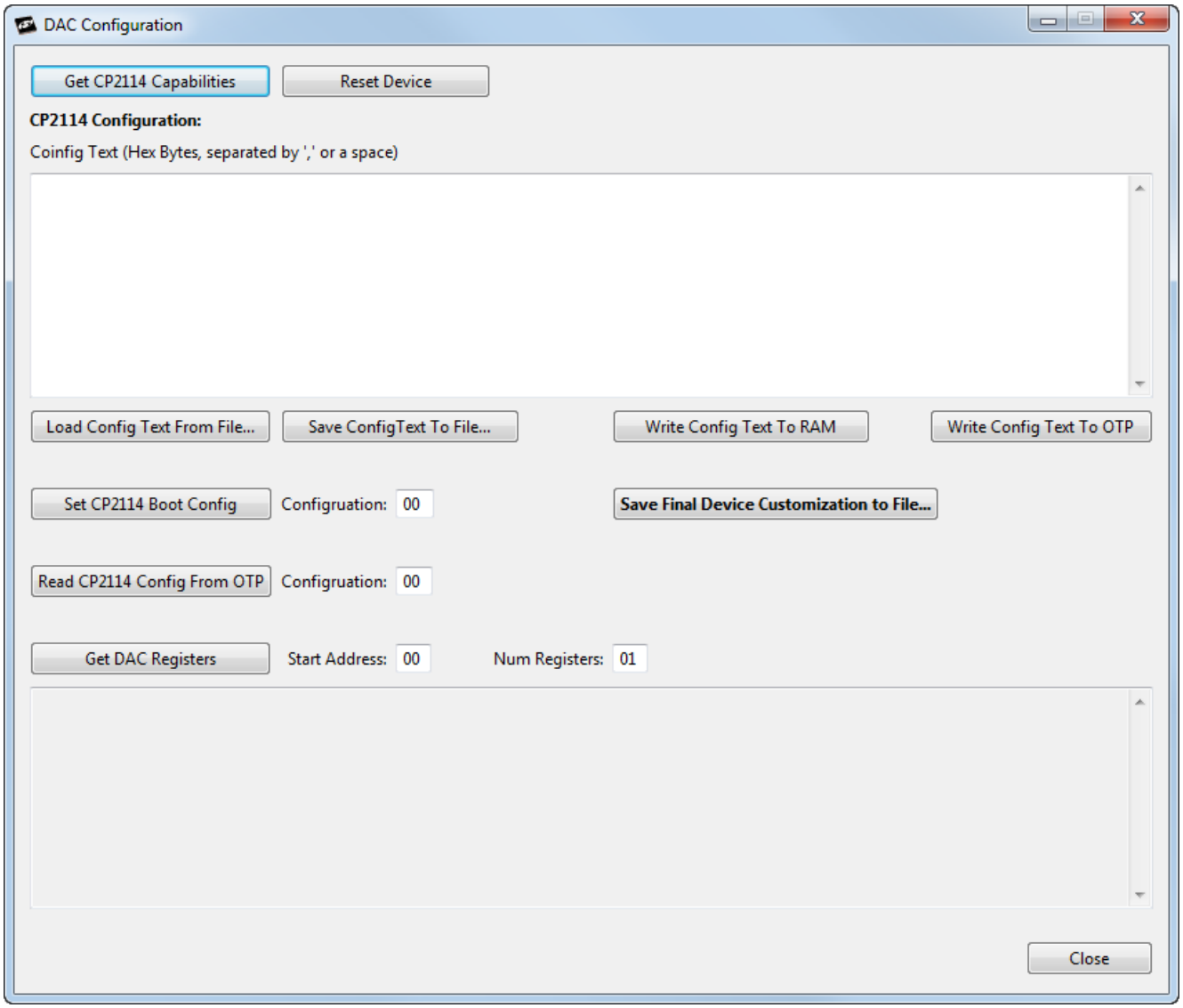



**Figure 8. DAC Configuration**

## 4.4. Application Command Descriptions

### 4.4.1. Get CP2114 Capabilities

This button will report the current configuration of the one-time programmable memory. With the default pin configuration and no jumpers installed on GPIO.5, GPIO.6, GPIO.7, or GPIO.8, the utility will display the following:

** CP2114 Caps*

*AvailableBootIndices: 0x20*

*AvailableOtpConfigs: 0x1D*

*CurrentBootConfig: 0xFF*

*AvailableOtpConfigSpace: 0x1532*

The **AvailableBootIndices: 0x20** parameter indicates that all 32 boot index slots are available for programming.

The **AvailableOtpConfigs: 0x1D** parameter indicates that there are 29 CP2114 configurations available in the one-time programmable memory.

The CP2114 EVB ships with three default configurations in OTP at the following indices:

Index 0: Audio out and audio in streams both set to Asynchronous with CS42L55 DAC settings.

Index 1: Audio out and audio in streams both set to Asynchronous with WM8523 DAC settings.

Index 2: Audio out and audio in streams both set to Asynchronous with PCM1774 DAC settings.

The **CurrentBootConfig** parameter with a value of 0xFF indicates the CP2114 will not configure any DAC devices on a reset or boot.This occurs when

- DAC select input feature is turned on and no jumpers are installed on GPIO.5, GPIO.6, GPIO.7, and GPIO.8.
- DAC select input feature is turned on and the GPIO value is set to “Use OTP Boot Config” while no valid boot config entry is found in the one-time programmable memory.
- DAC select input feature is turned off and there is no valid Boot Config entry in the one-time programmable memory.

The CP2114 has 32 programmable boot configuration entries by default. (The CP2114 boot configuration can be changed up to a total of 32 times.)

The final parameter, **AvailableOtPConfigSpace: 0x1532** indicates there are 0x1532 (5426) bytes of programmable memory available to support new configurations.

#### 4.4.2. Reset Device

This button forces a CP2114 reset.

#### 4.4.3. Load Config Text from File

Loads configuration text from a file into the **Config Text** window.

#### 4.4.4. Save Config Text to File

Saves the text in the **Config Text** window to a file.

#### 4.4.5. Write Config Text to RAM

Writes the configuration displayed in the **Config Text** window into the CP2114 RAM. The DAC configurations are also written to the DAC. This operation will cause the CP2114 to disconnect and reconnect on the USB bus. This configuration is not retained on device reset.

#### 4.4.6. Write Config Text to OTP

Writes the configuration in the **Config Text** window to the one-time programmable memory. The configuration does not become the active configuration in RAM unless it’s specified in CP2114 Boot Config or via DAC select GPIO pins followed by power cycling or resetting the device.

#### 4.4.7. Read CP2114 Config from OTP

After specifying the configuration number in the text box to the right of the button and clicking the button, the utility will display the one-time programmable memory configuration in the output window.

#### 4.4.8. Set CP2114 Boot Config

After specifying the configuration number in the text box to the right of the button and clicking the button, the boot configuration index is programmed into the one-time programmable memory space.

**Note:** The CP2114 boot configuration index can be changed up to 32 times.

#### 4.4.9. Get DAC Registers

Displays one or more DAC registers in the output window. To display a contiguous range of DAC registers enter the starting register address in the **Start Address** field and the number of registers to read in the **Num Registers** field. Click **Get DAC Registers** and the register values will be displayed in the status window as hex comma-separated values.

#### 4.4.10. Save Final Device Customization to File

Save the final 5.5 kB of customized data into a file for one-time programming in production.

## 4.5. Using the CP2114 DAC Application

Use the following steps to program the CP2114 using the DAC configuration utility:

1. Click **Get CP2114 Capabilities** and note the boot index, available OTP boot index slots, and available OTP config space.
2. Click **Load Config Text from File** if there is one available; otherwise write the config text into the Config Text window directly.
3. Click **Write Config Text to RAM**. This will cause CP2114 to re-enumerate over USB. Once CP2114 device is seen by the Host again, verify the device's audio functionalities including volume control and mute control. Adjust the value in **Config Text** if necessary and repeat **Write Config Text to RAM** until desired result is achieved.
4. Verify the DAC registers have the desired setting by entering the S**tart Address** and **Num Registers** and clicking **Get DAC Registers**.
5. Click **Save Config Text to File** to save a copy of the text in the **Config Text** window.
6. Click **Write Config Text to OTP** after all functionalities have been completely verified in RAM. The new config will consume one Boot Config index slot and some one-time programmable config space. The index of this new config should be the next available index. For example, if **Get CP2114 Capabilities** returned AvailableOtpConfigs of 0x1D, this Configuration index will be 3.
7. Enter the index of the new config and click **Read CP2114 Config from OTP**. Verify the data returned is as expected.
8. Click **Get CP2114 Capabilities** and verify the available OTP config space is (2+Config Data bytes) less than the previous value. The firmware inserts 2-byte length before the Config Data when writing to OTP. Available OTP Configs should be 1 less than the previous value.
9. Enter the Configuration index of the new Config and click **Set CP2114 Boot Config** to make CP2114 to boot from the new Config.
10. Reset the CP2114.
11. Verify CP2114 boots up properly with DAC functioning properly. Verify volume and mute integration with the Host. Click **Save Final Device Configuration to File** for OTP programming in production.

# 5. Changing Device Settings CP210X

There are six different CP210x devices. The CP2104 and CP2105 are one time programmable (OTP), therefore it is important to be careful when programming these since they cannot be programmed multiple times. The CP2101-CP2104 are all USB to single UART devices. The CP2105 is a USB to dual UART and the CP2108 is a USB to quad UART.

## 5.1. SET IDs CP210x Devices

All CP210x devices have strings that can be modified in the SETIDs window.

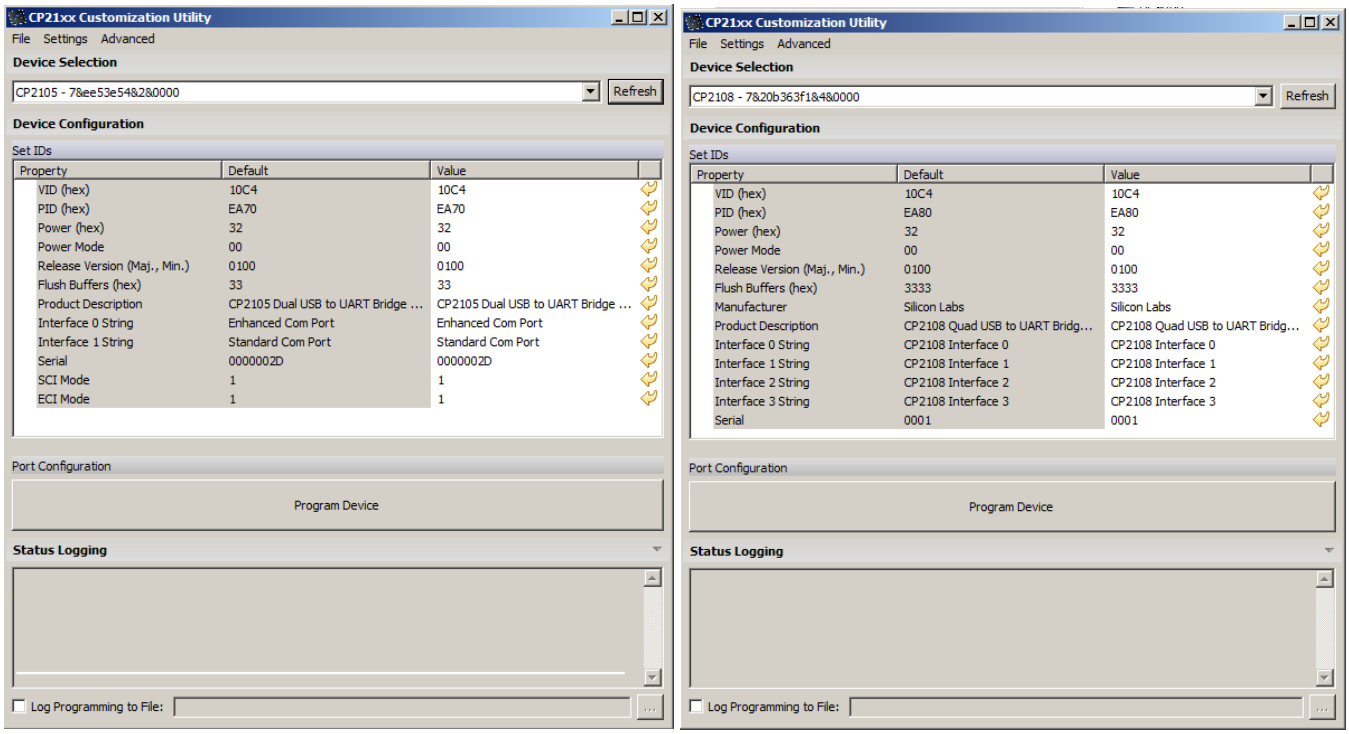



**Figure 9. CP210x SET IDs**

Figure 9, “CP210x SET IDs,”shows the modifiable parameters for all the CP210x devices. The CP2105 device shown on the left has 2 interface strings because it is a dual UART device. The CP2108 on the right has 4 interface strings because it is a quad UART device. There are 4 different interfaces. The single interface devices CP2101-CP2104 do not require an interface string since with only one interface there is only one option. The interface strings are unique to the CP2105 and CP2108 since they have multiple interfaces. All the devices contain the following parameters:

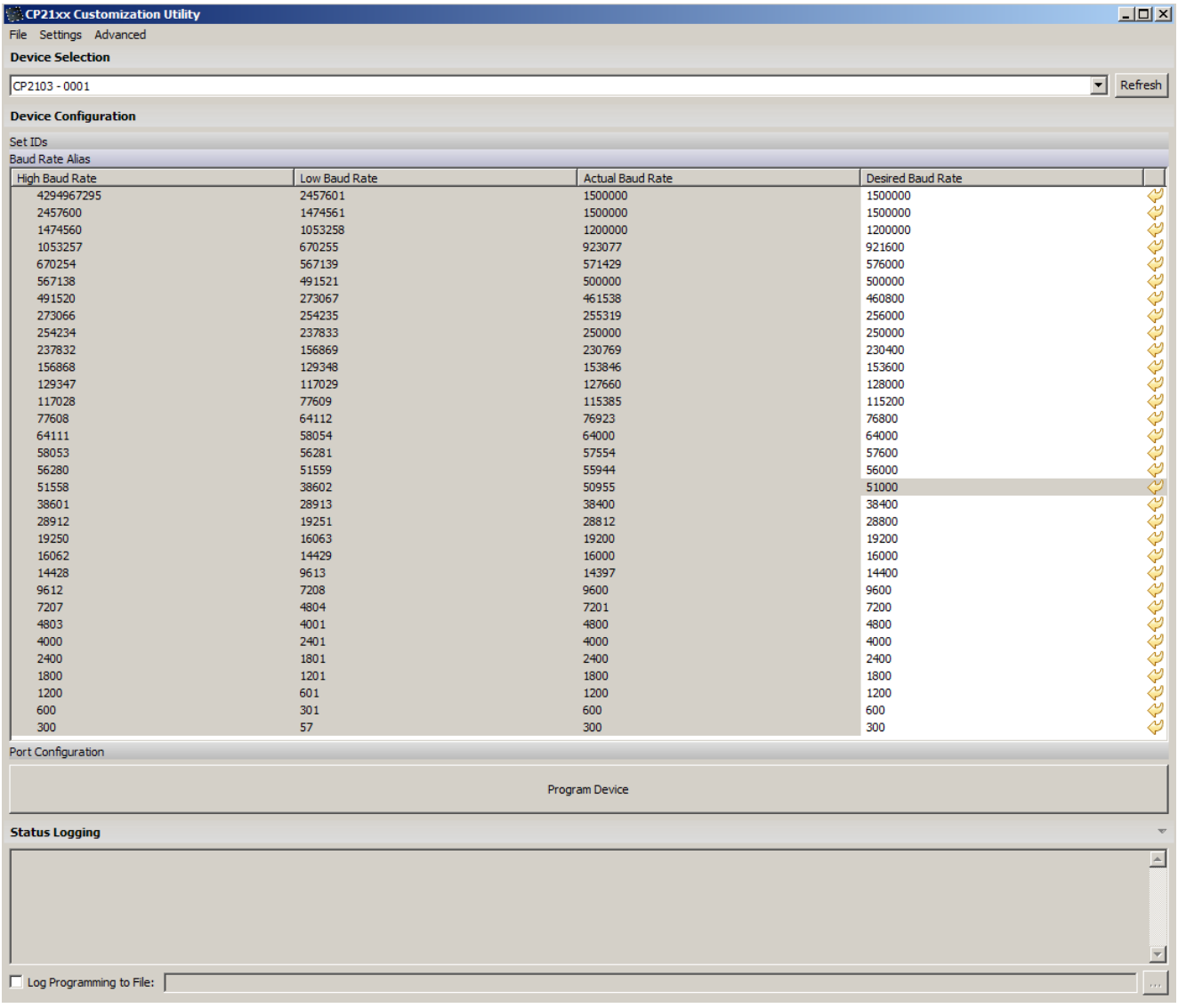
1. **VID**—The Vendor ID is a four hexadecimal digit number such as 10C4.
2. **PID**—The Product ID is a four hexadecimal digit number such as EA60.
3. **Power**—This is a two hexadecimal digit number such as 10 with a maximum setting of 250. Note this number corresponds to units of current in 2 mA increments. Therefore a value of 10 corresponds to 32 mA.
4. **Power Mode**—There are three different power mode options. Click the field to see the options. 00 corresponds to Bus Powered, 01 is Self Powered with Voltage Regulator Disabled, 02 is Self Powered with Voltage Regulator Enabled.
5. **Release Version**—The release version is a numerical identifier of the device release version. Each field is a decimal number value 0–99. The first two digits are the major version number and the last two are the minor digits. 0100 corresponds to Major Version 01, Minor Version 00.
6. **Product Description**—The Product Description can be any sequence of up to 126 characters. Usually this

is text which provides a description of the device, such as CP2103 USB to UART Bridge Controller.

7. **Interface String**—The string that identifies the different interface. This parameter is only present in the CP2105 and the CP2108, which have multiple interfaces.
8. **Serial Number** —The Serial Number can be any sequence of up to 63 characters.
9. **Device Mode**—The mode of the interface on the device, either GPIO or Modem Mode. This parameter is only present on CP2105.

## 5.2. Baud Rate Alias Configuration (CP2102, CP2103)

The CP2102 and CP2103 have Baud Rate Alias Configuration Settings. Changing the values in this table will cause the CP210x to translate 1 of 32 fixed UART baud rates to a desired baud rate shown in Figure 10. All ranges for a requested baud rate are shown.




| High Baud Rate | Low Baud Rate | Actual Baud Rate | Desired Baud Rate |
|---|---|---|---|
| 4294967295 | 2457601 | 1500000 | 1500000 |
| 2457600 | 1474561 | 1500000 | 1500000 |
| 1474560 | 1053258 | 1200000 | 1200000 |
| 1053257 | 670255 | 923077 | 921600 |
| 670254 | 567139 | 571429 | 576000 |
| 567138 | 491521 | 500000 | 500000 |
| 491520 | 273067 | 461538 | 460800 |
| 273066 | 254235 | 255319 | 256000 |
| 254234 | 237833 | 250000 | 250000 |
| 237832 | 156869 | 230769 | 230400 |
| 156868 | 129348 | 153846 | 153600 |
| 129347 | 117029 | 127660 | 128000 |
| 117028 | 77609 | 115385 | 115200 |
| 77608 | 64112 | 76923 | 76800 |
| 64111 | 58054 | 64000 | 64000 |
| 58053 | 56281 | 57554 | 57600 |
| 56280 | 51559 | 55944 | 56000 |
| 51558 | 38602 | 50955 | 51000 |
| 38601 | 28913 | 38400 | 38400 |
| 28912 | 19251 | 28812 | 28800 |
| 19250 | 16063 | 19200 | 19200 |
| 16062 | 14429 | 16000 | 16000 |
| 14428 | 9613 | 14397 | 14400 |
| 9612 | 7208 | 9600 | 9600 |
| 7207 | 4804 | 7201 | 7200 |
| 4803 | 4001 | 4800 | 4800 |
| 4000 | 2401 | 4000 | 4000 |
| 2400 | 1801 | 2400 | 2400 |
| 1800 | 1201 | 1800 | 1800 |
| 1200 | 601 | 1200 | 1200 |
| 600 | 301 | 600 | 600 |
| 300 | 57 | 300 | 300 |

**Figure 10. Baud Rate Configuration**

Baud rate aliasing refers to configuring a specific baud rate range to target a baud rate that is different from the default baud rate. Further support information can be found in the device data sheet for the CP2102/CP2103, which have this feature. The application-requested baud rate ranges are static and can never be changed. The

actual UART baud rate corresponding to a particular baud rate range is fully customizable. This customization is done using a Windows Dynamic Link Library (DLL) named CP210xManufacturing.DLL. Using the functions available in this API (GetBaudRateConfig() and SetBaudRateConfig()) the EEPROM settings can be changed using the USB connection.

Each line displays a range of baud rates that an application might request. There is a desired baud rate to use in that range and the actual baud rate that is used. To change the current configuration click on the desired baud rate number and enter in a new value. The Actual Baud Rate field will update to show the closest baud rate the hardware can achieve. Normally these two numbers will not be exactly the same, but as long as the actual baud rate is within 3% of the desired baud rate, the communication channel will work correctly.

## 5.3. Port Configuration (CP2103, CP2104, CP2105, CP2108)

The CP2103, CP2104, CP2105 and CP2108 all have modifiable port settings. These port settings have three types of interface pins. The UART/Modem interface pins consist for 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.

The following interface pin configuration options are available.

### 5.3.1. 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 5.3.8 and 5.3.9 for details concerning pin modes.

### 5.3.2. 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 5.3.11 for details concerning pin reset behavior.

### 5.3.3. 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.

### 5.3.4. 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. | | |

#### 5.3.5. 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.

#### 5.3.6. 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, $\overline{\text{SUSPEND}}$, and GPIO pins set for device-controlled function.

#### 5.3.7. 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 data sheet.

#### 5.3.8. 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 data sheet. This type of output is most often used to connect directly to another device.

#### 5.3.9. 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.

#### 5.3.10. 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.

#### 5.3.11. 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.

Figure 11 is a screenshot of the CP2103 and CP2104 Port Configuration Settings.Figure 12 is a screenshot of the CP2105 Port Configuration Settings and Figure 13 is a screenshot of the CP2108 Port Configuration Settings. Click on the value to modify any of the settings. In most cases a pull down menu is available. For the Suspend Value and the Reset Value Settings a pop up window allows the latch setting selections as shown in Figure 14.

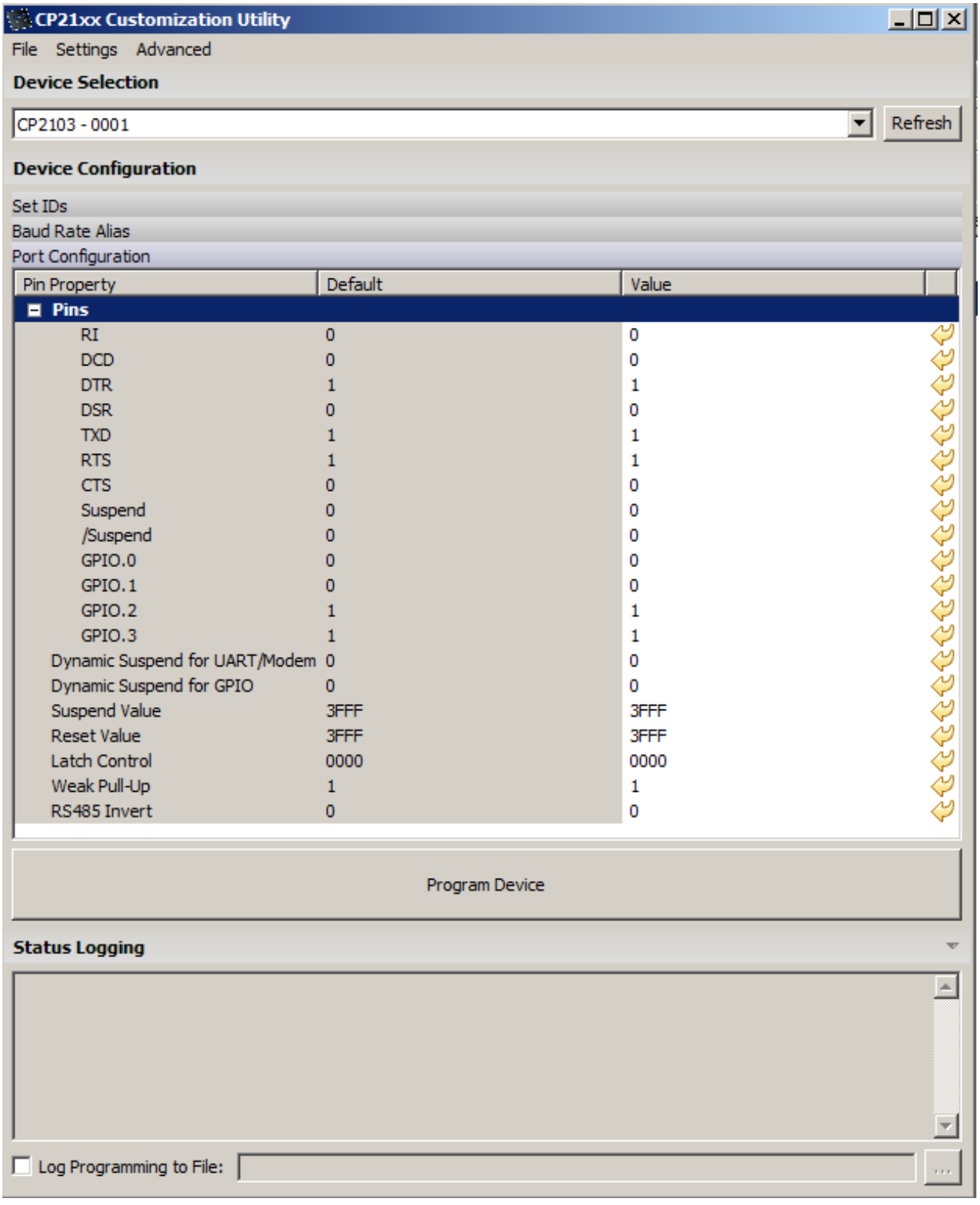



**Figure 11. CP2103/CP2104 Port Configuration**

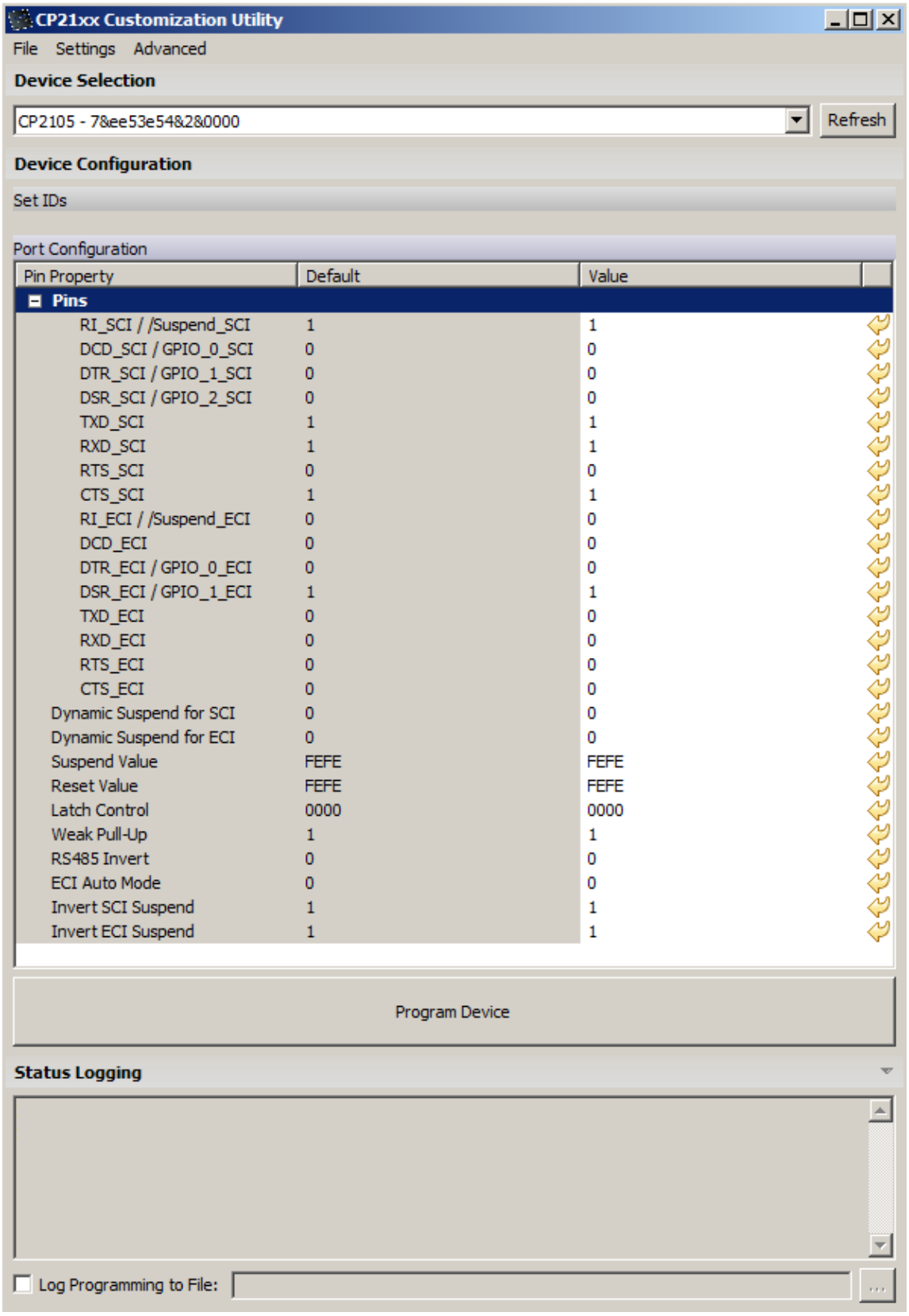



**Figure 12. CP2105 Port Configuration**

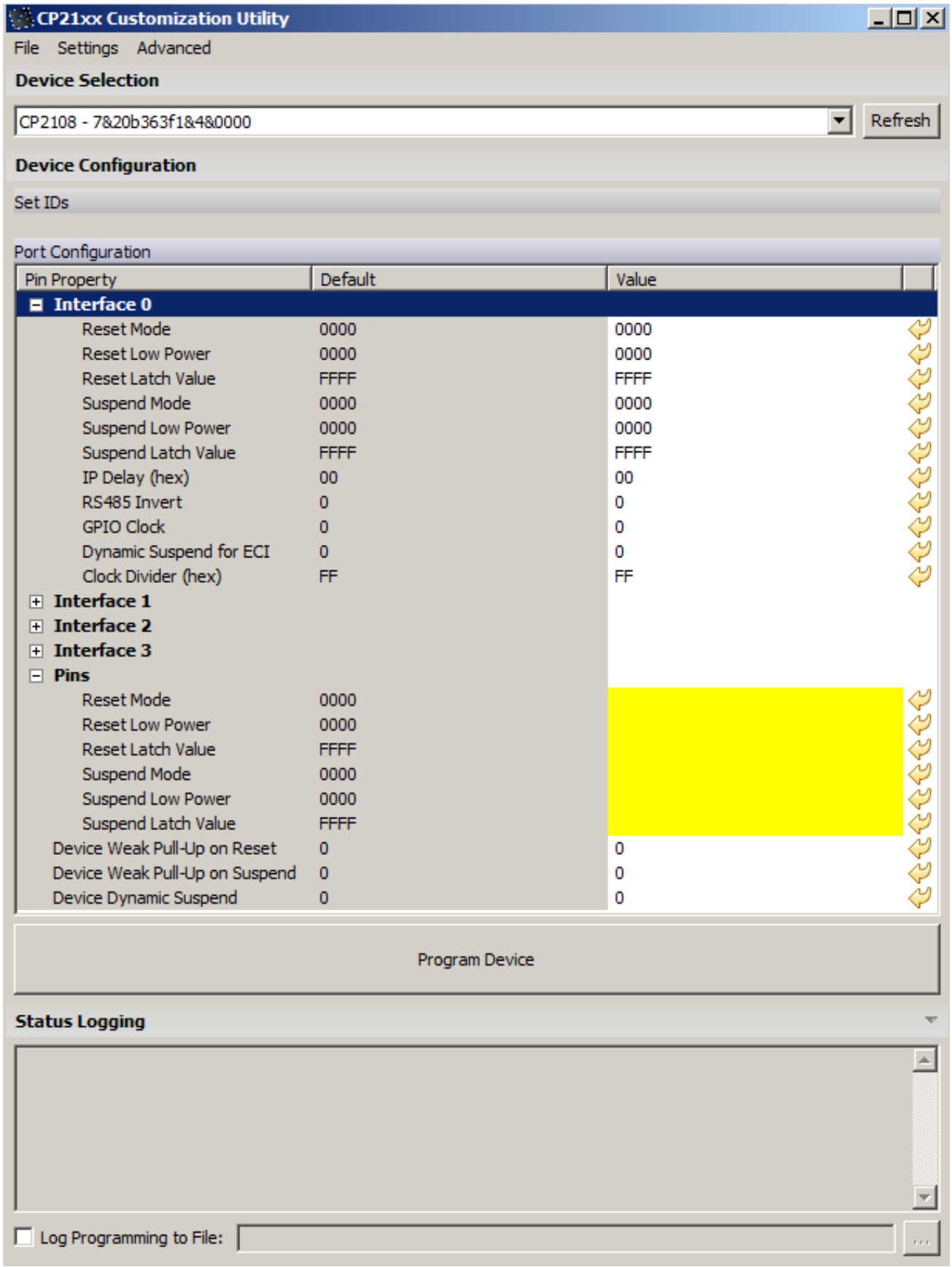



**Figure 13. CP2108 Port Configuration**

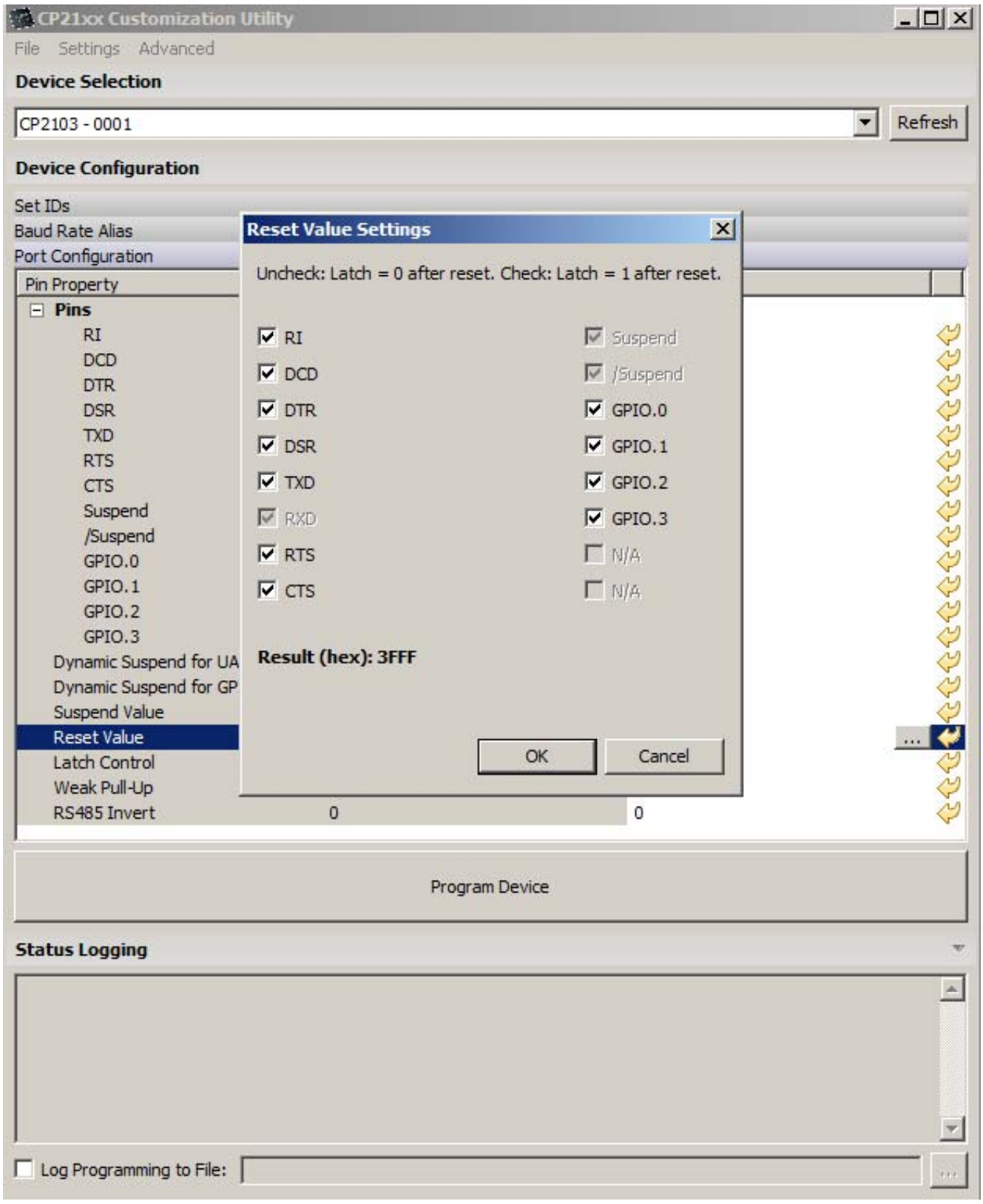



**Figure 14. Latch Settings**

# 6. CP210x Host API Functions

There are two DLL files CP210xManufacturing.DLL and CP210xRuntime.DLL which have several API functions that are described and listed.

## 6.1. CP210xManufacturing.DLL

The CP210x Host API is provided as a means to facilitate production of customized CP210x devices. The API allows access to the CP210x device for retrieving and setting the VID, PID, product string, serial number, self-power attribute, maximum power consumption, and device version.

The CP210x Host API is provided in the form of a Windows Dynamic Link Library (DLL), *CP210xManufacturing.DLL*. The host interface DLL communicates with the bridge controller device via the provided device driver and the operating system's USB stack. The following is a list of the available host API functions:

| | |
|---|---|
| `CP210x_GetNumDevices()` | Returns the number of CP210x devices connected. |
| `CP210x_GetProductString()` | Returns a descriptor from the registry for a CP210x USB device. |
| `CP210x_GetPartNumber()` | Returns the 1-byte Part Number of a CP210x device. |
| `CP210x_Open()` | Opens a CP210x device as a USB device and returns a handle. |
| `CP210x_Close()` | Closes a CP210x device handle. |
| | |
| `CP210x_SetVid()` | Sets the 2-byte vendor ID of a CP210x device. |
| `CP210x_SetPid()` | Sets the 2-byte product ID of a CP210x device. |
| `CP210x_SetProductString()` | Sets the product description string of a CP210x device. |
| `CP210x_SetInterfaceString()` | Sets the interface string of a CP2105 device. |
| `CP210x_SetSerialNumber()` | Sets the serial number string of a CP210x device. |
| `CP210x_SetSelfPower()` | Sets the self-power attribute of a CP210x device. |
| `CP210x_SetMaxPower()` | Sets the maximum power consumption of a CP210x device. |
| `CP210x_SetFlushBufferConfig()` | Sets the flush buffer configuration of CP2104/5 devices. |
| `CP210x_SetDeviceMode()` | Sets the operating modes of both interfaces of a CP2105 device. |
| `CP210x_SetDeviceVersion()` | Sets version number of the CP210x device. |
| `CP210x_SetBaudRateConfig()` | Sets the baud rate configuration data of a CP210x device. |
| `CP210x_SetLockValue()` | Sets the 1-byte Lock Value of a CP210x device. |
| `CP210x_SetPortConfig()` | Sets the port configuration of a CP2101/2/3/4 device. |
| `CP210x_SetDualPortConfig()` | Sets the port configuration of a CP2105 device. |
| `CP210x_SetQuadPortConfig()` | Sets the port configuration of a CP2108 device. |
| | |
| `CP210x_GetDeviceProductString()` | Gets the product description string of a CP210x device. |
| `CP210x_GetDeviceInterfaceString()` | Gets the interface string of a CP2105 device. |
| `CP210x_GetDeviceSerialNumber()` | Gets the serial number string of a CP210x device. |
| `CP210x_GetDeviceVid()` | Gets the vendor ID of a CP210x device. |
| `CP210x_GetDevicePid()` | Gets the product ID of a CP210x device. |
| `CP210x_GetSelfPower()` | Gets the self-power attribute of a CP210x device. |
| `CP210x_GetMaxPower()` | Gets the maximum power consumption value of a CP210x device. |
| `CP210x_GetFlushBufferConfig()` | Gets the flush buffer configuration of CP2104/5 devices. |
| `CP210x_GetDeviceMode()` | Gets the operating modes of interfaces of a CP2105 device. |
| `CP210x_GetDeviceVersion()` | Gets the version number of a CP210x device. |
| `CP210x_GetBaudRateConfig()` | Gets the baud rate configuration data of a CP210x device. |
| `CP210x_GetLockValue()` | Gets the 1-byte Lock Value of a CP210x device. |
| `CP210x_GetPortConfig()` | Gets the port configuration of a CP210x device. |
| `CP210x_GetDualPortConfig()` | Gets the port configuration of a CP2105 device. |
| `CP210x_GetQuadPortConfig()` | Gets the port configuration of a CP2108 device. |
| `CP210x_Reset()` | Resets a CP210x device. |

In general, the user initiates communication with the target CP210x device by making a call to *CP210x_GetNumDevices()*. This call returns the number of CP210x target devices. This number is used as a range when calling *CP210x_GetProductString()* to build a list of devices connected to the host machine.

A handle to the device must first be opened by a call to *CP210x_Open()* using an index determined from the call to *CP210x_GetNumDevices()*. The handle will be used for all subsequent accesses. When I/O operations are complete, the device handle is closed by a call to *CP210x_Close()*.

When programming a CP2105 device to configure the mode, the following functions must be called in the following order:

```
CP210x_SetDeviceMode()
CP210x_SetDualPortConfig()
```

The remaining functions are provided to allow access to customizable values contained in the CP210x programmable area.

#### 6.1.1. CP210x_GetNumDevices

**Description:** This function returns the number of CP210x devices connected to the host.

**Supported Devices:** CP2101, CP2102, CP2103, CP2104, CP2105, CP2108

**Location:** CP210x Manufacturing DLL

**Prototype:**
```
CP210x_STATUS CP210x_GetNumDevices( LPDWORD NumDevices )
```

**Parameters:**
1. NumDevices—Address of a DWORD that will contain the number of devices.

**Return Value:** CP210x_STATUS = CP210x_SUCCESS,
CP210x_DEVICE_NOT_FOUND,
CP210x_INVALID_PARAMETER

#### 6.1.2. CP210x_GetProductString

**Description:** This function returns a NULL-terminated serial number (S/N) string, product description string, or full path string for the device specified by an index passed in the DeviceNum parameter. The index of the first device is 0, and the index of the last device is the value (NumDevices) returned by CP210x_GetNumDevices() – 1.

**Supported Devices:** CP2101, CP2102, CP2103, CP2104, CP2105, CP2108

**Location:** CP210x Manufacturing DLL

**Prototype:**
```
CP210x_STATUS CP210x_GetProductString( DWORD DeviceNum,
LPVOID DeviceString, DWORD Options )
```

**Parameters:**
1. DeviceNum—Index of the device for which the product description string, serial number, or full path is desired.
2. DeviceString—Variable of type *CP210x_DEVICE_STRING* returning the NULL-terminated serial number, device description or full path string.
3. Options—Flag that determines if *DeviceString* contains the product description, serial number, or full-path string.

**Return Value:** CP210x_STATUS = CP210x_SUCCESS,
CP210x_DEVICE_NOT_FOUND,
CP210x_INVALID_PARAMETER

#### 6.1.3. CP210x_GetPartNumber

**Description:** Returns the 1-byte Part Number contained in a CP210x device.

**Supported Devices:** CP2101, CP2102, CP2103, CP2104, CP2105, CP2108

**Location:** CP210x Manufacturing DLL

**Prototype:**
```
CP210x_STATUS WINAPI CP210x_GetPartNumber(HANDLE cyHandle,
LPBYTE lpbPartNum);
```

**Parameters:**
1. Handle—Handle to the device returning a Part Number.
2. PartNum—Pointer to a 1-byte value returning the Part Number of the device. A *CP210x_CP2101_DEVICE* denotes a CP2101 device, and a *CP210x_CP2102_DEVICE* denotes a CP2102 device.

**Return Value:** CP210x_STATUS = CP210x_SUCCESS,
CP210x_INVALID_PARAMETER,
CP210x_INVALID_HANDLE,
CP210x_DEVICE_IO_FAILED

#### 6.1.4. CP210x_Open

**Description:** Opens and returns a handle to a device using a device number determined by the number returned from *CP210x_GetNumDevices()*.

**Supported Devices:** CP2101, CP2102, CP2103, CP2104, CP2105, CP2108

**Location:** CP210x Manufacturing DLL

**Prototype:**
```
CP210x_STATUS CP210x_Open( DWORD DeviceNum, HANDLE* Handle )
```

**Parameters:**
1. DeviceNum—Device index. 0 for the first device, 1 for the second, etc.
2. Handle—Pointer to a variable where the handle to the device will be stored. This handle will be used for all subsequent accesses to the device.

**Return Value:** CP210x_STATUS = CP210x_SUCCESS,
CP210x_DEVICE_NOT_FOUND,
CP210x_INVALID_PARAMETER

#### 6.1.5. CP210x_Close

**Description:** Closes an open device handle.

**Supported Devices:** CP2101, CP2102, CP2103, CP2104, CP2105, CP2108

**Location:** CP210x Manufacturing DLL

**Prototype:**
```
CP210x_STATUS CP210x_Close( HANDLE Handle )
```

**Parameters:**
1. Handle—Handle to the device to close as returned by *CP210x_Open()*.

**Return Value:** CP210x_STATUS = CP210x_SUCCESS,
CP210x_INVALID_HANDLE

#### 6.1.6. CP210x_SetVid

**Description:** Sets the 2-byte Vendor ID field of the Device Descriptor of a CP210x device.

**Supported Devices:** CP2101, CP2102, CP2103, CP2104, CP2105, CP2108

**Location:** CP210x Manufacturing DLL

**Prototype:** `CP210x_STATUS CP210x_SetVid( HANDLE Handle, WORD Vid )`

**Parameters:**
1. Handle—Handle to the device to close as returned by *CP210x_Open()*.
2. VID—2-byte Vendor ID value.

**Return Value:** CP210x_STATUS = CP210x_SUCCESS,
CP210x_INVALID_HANDLE,
CP210x_DEVICE_IO_FAILED

#### 6.1.7. CP210x_SetPid

**Description:** Sets the 2-byte Product ID field of the Device Descriptor of a CP210x device.

**Supported Devices:** CP2101, CP2102, CP2103, CP2104, CP2105, CP2108

**Location:** CP210x Manufacturing DLL

**Prototype:** `CP210x_STATUS CP210x_SetPid( HANDLE Handle, WORD Pid )`

**Parameters:**
1. Handle—Handle to the device to close as returned by *CP210x_Open()*.
2. PID—2-byte Product ID value.

**Return Value:** CP210x_STATUS = CP210x_SUCCESS,
CP210x_INVALID_HANDLE,
CP210x_DEVICE_IO_FAILED

#### 6.1.8. CP210x_SetProductString

**Description:** Sets the Product Description String of the String Descriptor of a CP210x device. If the string is not already in Unicode format, the function will convert the string to Unicode before committing it to programmable memory. The character size limit (in characters, not bytes), NOT including a NULL terminator, is *CP210x_MAX_PRODUCT_STRLEN or CP2105_MAX_PRODUCT_STRLEN*.

**Supported Devices:** CP2101, CP2102, CP2103, CP2104, CP2105, CP2108

**Location:** CP210x Manufacturing DLL

**Prototype:**
```
CP210x_STATUS CP210x_SetProductString( HANDLE Handle, LPVOID Product,
BYTE Length, BOOL ConvertToUnicode=TRUE )
```

**Parameters:**
1. Handle—Handle to the device to close as returned by *CP210x_Open()*.
2. Product—Buffer containing the Product String value.
3. Length—Length of the string in characters (not bytes), NOT including a NULL terminator.
4. ConvertToUnicode—Boolean flag that tells the function if the string needs to be converted to Unicode. The flag is set to TRUE by default (i.e., the string is in ASCII format and needs to be converted to Unicode).

**Return Value:** CP210x_STATUS = CP210x_SUCCESS,
CP210x_INVALID_PARAMETER,
CP210x_INVALID_HANDLE,
CP210x_DEVICE_IO_FAILED

### 6.1.9. CP210x_SetInterfaceString

**Description:** Sets the Interface String for the one of the interfaces available on the CP2105 or CP2108. If the string is not already in Unicode format, the function will convert the string to Unicode before committing it to programmable memory. The character size limit (in characters, not bytes), NOT including a NULL terminator, is *CP2105_MAX_INTERFACE_STRLEN*.

**Supported Devices:** CP2105, CP2108

**Location:** CP210x Manufacturing DLL

**Prototype:**
```
CP210x_STATUS CP210x_SetInterfaceString( HANDLE Handle, BYTE InterfaceNumber, LPVOID Interface, BYTE Length, BOOL ConvertToUnicode)
```

**Parameters:**

1. Handle—Handle to the device to close as returned by *CP210x_Open()*.
2. InterfaceNumber—Set to 0 for Enhanced Interface String, or 1 for Standard Interface String on the CP2105. 0-3 for the CP2108 which has 4 interfaces.
3. Interface—Buffer containing the Interface String.
4. Length—Length of the string in characters (not bytes), NOT including a NULL terminator.
5. ConvertToUnicode—Boolean flag that tells the function if the string needs to be converted to Unicode. The flag is set to TRUE by default (i.e., the string is in ASCII format and needs to be converted to Unicode).

**Return Value:** CP210x_STATUS = CP210x_SUCCESS,
CP210x_INVALID_PARAMETER,
CP210x_INVALID_HANDLE,
CP210x_DEVICE_IO_FAILED

### 6.1.10. CP210x_SetSerialNumber

**Description:** Sets the Serial Number String of the String Descriptor of a CP210x device. If the string is not already in Unicode format, the function will convert the string to Unicode before committing it to programmable memory. The character size limit (in characters, not bytes), NOT including a NULL terminator, is *CP210x_MAX_SERIAL_STRLEN*.

**Supported Devices:** CP2101, CP2102, CP2103, CP2104, CP2105, CP2108

**Location:** CP210x Manufacturing DLL

**Prototype:**
```
CP210x_STATUS CP210x_SetSerialNumber( HANDLE Handle,
LPVOID SerialNumber, BYTE Length, BOOL ConvertToUnicode=TRUE )
```

**Parameters:**

1. Handle—Handle to the device to close as returned by *CP210x_Open()*.
2. SerialNumber—Buffer containing the Serial Number String value.
3. Length—Length in characters (not bytes), NOT including a NULL terminator.
4. ConvertToUnicode—Boolean flag that tells the function if the string needs to be converted to Unicode. The flag is set to TRUE by default, i.e. the string is in ASCII format and needs to be converted to Unicode.

**Return Value:** CP210x_STATUS = CP210x_SUCCESS,
CP210x_INVALID_PARAMETER,
CP210x_INVALID_HANDLE,
CP210x_DEVICE_IO_FAILED

### 6.1.11. CP210x_SetSelfPower

**Description:** Sets or clears the Self-Powered bit of the Power Attributes field of the Configuration Descriptor of a CP210x device.

**Supported Devices:** CP2101, CP2102, CP2103, CP2104, CP2105, CP2108

**Location:** CP210x Manufacturing DLL

**Prototype:** `CP210x_STATUS CP210x_SetSelfPower( HANDLE Handle, BOOL SelfPower )`

**Parameters:**
1. Handle—Handle to the device to close as returned by *CP210x_Open()*.
2. SelfPower—Boolean flag where TRUE means set the Self-Powered bit, and FALSE means clear the Self-Powered bit.

**Return Value:** CP210x_STATUS = CP210x_SUCCESS,
CP210x_INVALID_HANDLE,
CP210x_DEVICE_IO_FAILED

### 6.1.12. CP210x_SetMaxPower

**Description:** Sets the Max Power field of the Configuration Descriptor of a CP210x device.

**Supported Devices:** CP2101, CP2102, CP2103, CP2104, CP2105, CP2108

**Location:** CP210x Manufacturing DLL

**Prototype:** `CP210x_STATUS CP210x_SetMaxPower( HANDLE Handle, BYTE MaxPower )`

**Parameters:**
1. Handle—Handle to the device to close as returned by *CP210x_Open()*.
2. MaxPower—1-byte value representing the maximum power consumption of the CP210x USB device, expressed in 2 mA units.

**Return Value:** CP210x_STATUS = CP210x_SUCCESS,
CP210x_INVALID_HANDLE,
CP210x_DEVICE_IO_FAILED

### 6.1.13. CP210x_SetFlushBufferConfig

**Description:** Sets the Flush Buffer configuration of a CP210x device.

**Supported Devices:** CP2104, CP2105, CP2108

**Location:** CP210x Manufacturing DLL

**Prototype:**
```
CP210x_STATUS CP210x_SetMaxPower( HANDLE Handle,
BYTE FlushBufferConfig )
```

**Parameters:**
1. Handle—Handle to the device to close as returned by *CP210x_Open()*.
2. FlushBufferConfig—Set to determine which buffer(s) to flush (TX and/or RX) and upon which event (Open and/or Close). See the header file for the bit defintions for this byte value.

**Return Value:** CP210x_STATUS = CP210x_SUCCESS,
CP210x_INVALID_HANDLE,
CP210x_FUNCTION_NOT_SUPPORTED
CP210x_DEVICE_NOT_FOUND

#### 6.1.14. CP210x_SetDeviceMode

**Description:** Sets the operating mode (GPIO or Modem) or each Interface of a CP210x device.

**Supported Devices:** CP2105

**Location:** CP210x Manufacturing DLL

**Prototype:**
```
CP210x_STATUS CP210x_SetMaxPower( HANDLE Handle,
BYTE DeviceModeECI, BYTE DeviceModeSCI)
```

**Parameters:**
1. Handle—Handle to the device to close as returned by *CP210x_Open()*.
2. DeviceModeECI—Set to 0 for modem mode for Enhanced interface. Set to 1 for GPIO mode.
3. DeviceModeSCI—Set to 0 for modem mode for Enhanced interface. Set to 1 for GPIO mode.

**Return Value:** CP210x_STATUS = CP210x_SUCCESS,
CP210x_INVALID_HANDLE,
CP210x_DEVICE_NOT_FOUND
CP210x_FUNCTION_NOT_SUPPORTED

#### 6.1.15. CP210x_SetDeviceVersion

**Description:** Sets the Device Release Version field of the Device Descriptor of a CP210x device.

**Supported Devices:** CP2101, CP2102, CP2103, CP2104, CP2105, CP2108

**Location:** CP210x Manufacturing DLL

**Prototype:**
```
CP210x_STATUS CP210x_SetDeviceVersion( HANDLE Handle, WORD Version )
```

**Parameters:**
1. Handle—Handle to the device to close as returned by *CP210x_Open()*.
2. Version—2-byte Device Release Version number in Binary-Coded Decimal (BCD) format with the upper two nibbles containing the two decimal digits of the major version and the lower two nibbles containing the two decimal digits of the minor version.

**Return Value:** CP210x_STATUS = CP210x_SUCCESS,
CP210x_INVALID_HANDLE,
CP210x_DEVICE_IO_FAILED

#### 6.1.16. CP210x_SetBaudRateConfig

**Description:** Sets the baud rate configuration data of a CP210x device.

**Supported Devices:** CP2102, CP2103

**Location:** CP210x Manufacturing DLL

**Prototype:**
```
CP210x_STATUS WINAPI CP210x_SetBaudRateConfig(HANDLE cyHandle, BAUD_CON-
FIG* baudConfigData);
```

**Parameters:**
1. Handle—Handle to the device from which to get the Part Number.

**2.** BaudConfigData—Pointer to a *BAUD_CONFIG* structure containing the Baud Config data to be set on the device.

**Return Value:** CP210x_STATUS = CP210x_SUCCESS,
CP210x_INVALID_PARAMETER,
CP210x_INVALID_HANDLE,
CP210x_DEVICE_IO_FAILED

### 6.1.17. CP210x_SetLockValue

**Description:** Sets the 1-byte Lock Value of a CP210x device.

**Supported Devices:** CP2102, CP2103, CP2104, CP2105, CP2108

**Location:** CP210x Manufacturing DLL

**Prototype:** `CP210x_STATUS WINAPI CP210x_SetLockValue(HANDLE cyHandle);`

**Parameters:** **1.** Handle—Handle of the device to lock. This will permanently set the lock value to 0x01.

**WARNING:** Setting the lock value locks ALL customizable data and cannot be reset; only use this function to keep all customizable data on the part permanently.

**Return Value:** CP210x_STATUS = CP210x_SUCCESS,
CP210x_INVALID_PARAMETER,
CP210x_INVALID_HANDLE,
CP210x_DEVICE_IO_FAILED

### 6.1.18. CP210xSetPortConfig

**Description:** Sets the current port pin configuration from the CP210x device.

**Supported Devices:** CP2103, CP2104, CP2105, CP2108

**Location:** CP210x Manufacturing DLL

**Prototype:** `CP210X_STATUS  CP210x_SetPortConfig(HANDLE Handle, LPVOID PortConfig)`

**Parameters:** **1.** Handle—Handle to the device as returned by CP210x_Open()

**2.** PortConfig—Pointer to a PORT_CONFIG structure

**Return Value:** CP210X_STATUS = CP210X_SUCCESS,
CP210X_INVALID_HANDLE,
CP210X_DEVICE_IO_FAILED,
CP210X_UNSUPPORTED_DEVICE

### 6.1.19. CP210xSetDualPortConfig

**Description:** Sets the current port pin configuration from the CP210x device. SetDeviceMode() must be called before calling this function.

**Supported Devices:** CP2105

**Location:** CP210x Manufacturing DLL

**Prototype:** `CP210X_STATUS CP210x_SetPortConfig(HANDLE Handle, LPVOID DualPortConfig)`

**Parameters:**
1. Handle—Handle to the device as returned by CP210x_Open()
2. DualPortConfig—Pointer to a DUAL_PORT_CONFIG structure

**Return Value:** CP210X_STATUS = CP210X_SUCCESS,
CP210X_INVALID_HANDLE,
CP210X_DEVICE_IO_FAILED,
CP210X_UNSUPPORTED_DEVICE

### 6.1.20. CP210x_GetDeviceProductString

**Description:** Returns the Product Description String of the String Descriptor of a CP210x device. If the ConvertToASCII parameter is set, the string will be converted to ASCII format before being returned to the caller. The character size limit (in characters, not bytes), NOT including a NULL terminator, is *CP210x_MAX_PRODUCT_STRLEN*.

**Supported Devices:** CP2101, CP2102, CP2103, CP2104, CP2105, CP2108

**Location:** CP210x Manufacturing DLL

**Prototype:**
```
CP210x_STATUS CP210x_GetDeviceProductString( HANDLE Handle,
LPVOID Product, LPBYTE Length, BOOL ConvertToASCII=TRUE )
```

**Parameters:**
1. Handle—Handle to the device to close as returned by CP210x_Open().
2. Product—Pointer to a buffer returning the Product String value.
3. Length—Pointer to a BYTE value returning the length of the string in characters (not bytes), NOT including a NULL terminator.
4. ConvertToASCII—Boolean flag that tells the function whether the string needs to be converted to ASCII before it is returned to the caller. The flag is set to TRUE by default (i.e., the caller is expecting the string in ASCII format).

**Return Value:** CP210x_STATUS = CP210x_SUCCESS,
CP210x_INVALID_PARAMETER,
CP210x_INVALID_HANDLE,
CP210x_DEVICE_IO_FAILED

### 6.1.21. CP210xSetQuadPortConfig

**Description:** Sets the current port pin configuration from the CP2108 device.

**Supported Devices:** CP2108

**Location:** CP210x Manufacturing DLL

**Prototype:** `CP210x_STATUS CP210x_SetQuadPortConfig( HANDLE Handle, LPVOID QuadPort-Config )`

**Parameters:**
1. Handle—Handle to the device to close as returned by CP210x_Open().
2. QuadPortConfig—Pointer to a QUAD_PORT_CONFIG structure.

**Return Value:** CP210x_STATUS = CP210x_SUCCESS,
CP210x_INVALID_HANDLE,
CP210x_DEVICE_IO_FAILED,
CP210x_UNSUPPORTED_DEVICE

### 6.1.22. CP210x_GetDeviceInterfaceString

**Description:** Gets the specified interface string from a CP210x device. If the ConvertToASCII parameter is set, the string will be converted to ASCII format before being returned to the caller. The character size limit (in characters, not bytes), NOT including a NULL terminator, is *CP210x_MAX_SERIAL_STR-LEN*.

**Supported Devices:** CP2105, CP2108

**Location:** CP210x Manufacturing DLL

**Prototype:**
```
CP210x_STATUS CP210x_GetDeviceInterfaceString( HANDLE Handle,
BYTE InterfaceNumber, LPVOID Interface, BYTE Length, BOOL ConvertToAS-CII)
```

**Parameters:**
1. Handle—Handle to the device to close as returned by *CP210x_Open()*.
2. InterfaceNumber —Set to 0 for Enhanced Interface. Set to 1 for Standard Interface.
3. Interface—Pointer to buffer returning the selected Interface String value.
4. Length—Pointer to a BYTE value returning the length of the string in characters (not bytes), NOT including a NULL terminator.
5. ConvertToASCII—Boolean flag that tells the function whether the string needs to be converted to ASCII before it is returned to the caller. The flag is set to TRUE by default (i.e., the caller is expecting the string in ASCII format).

**Return Value:** CP210x_STATUS = CP210x_SUCCESS,
CP210x_INVALID_PARAMETER,
CP210x_INVALID_HANDLE,
CP210x_DEVICE_IO_FAILED

### 6.1.23. CP210x_GetDeviceSerialNumber

**Description:** Gets the Serial Number String of the String Descriptor of a CP210x device. If the ConvertToASCII parameter is set, the string will be converted to ASCII format before being returned to the caller. The character size limit (in characters, not bytes), NOT including a NULL terminator, is *CP210x_-MAX_SERIAL_STRLEN*.

**Supported Devices:** CP2101, CP2102, CP2103, CP2104, CP2105, CP2108

**Location:** CP210x Manufacturing DLL

**Prototype:**
```
CP210x_STATUS CP210x_GetDeviceSerialNumber( HANDLE Handle,
LPVOID SerialNumber, LPBYTE Length, BOOL ConvertToASCII=TRUE )
```

**Parameters:**
1. Handle—Handle to the device to close as returned by *CP210x_Open()*.
2. SerialNumber —Pointer to a buffer returning the Serial Number String value.
3. Length—Pointer to a BYTE value returning the length of the string in characters (not bytes), NOT including a NULL terminator.
4. ConvertToASCII—Boolean flag that tells the function whether the string needs to be converted to ASCII before it is returned to the caller. The flag is set to TRUE by default (i.e., the caller is expecting the string in ASCII format).

**Return Value:** CP210x_STATUS = CP210x_SUCCESS,
CP210x_INVALID_PARAMETER,
CP210x_INVALID_HANDLE,
CP210x_DEVICE_IO_FAILED

### 6.1.24. CP210x_GetDeviceVid

**Description:** Returns the 2-byte Vendor ID field of the Device Descriptor of a CP210x device.

**Supported Devices:** CP2101, CP2102, CP2103, CP2104, CP2105, CP2108

**Location:** CP210x Manufacturing DLL

**Prototype:** `CP210x_STATUS CP210x_GetDeviceVid( HANDLE Handle, LPWORD Vid )`

**Parameters:**
1. Handle—Handle to the device to close as returned by *CP210x_Open()*.
2. VID—Pointer to a 2-byte value that returns the Vendor ID of the CP210x device.

**Return Value:** CP210x_STATUS = CP210x_SUCCESS,
CP210x_INVALID_PARAMETER,
CP210x_INVALID_HANDLE,
CP210x_DEVICE_IO_FAILED

### 6.1.25. CP210x_GetDevicePid

**Description:** Returns the 2-byte Product ID field of the Device Descriptor of a CP210x device.

**Supported Devices:** CP2101, CP2102, CP2103, CP2104, CP2105, CP2108

**Location:** CP210x Manufacturing DLL

**Prototype:** `CP210x_STATUS CP210x_GetDevicePid( HANDLE Handle, LPWORD Pid )`

**Parameters:**
1. Handle—Handle to the device to close as returned by *CP210x_Open()*.
2. PID—Pointer to a 2-byte value that returns the Product ID of the CP210x device.

**Return Value:** CP210x_STATUS = CP210x_SUCCESS,
CP210x_INVALID_PARAMETER,
CP210x_INVALID_HANDLE,
CP210x_DEVICE_IO_FAILED

#### 6.1.26. CP210x_GetSelfPower

**Description:** Returns the state of the Self-Powered bit of the Power Attributes field of the Configuration Descriptor of a CP210x device.

**Supported Devices:** CP2101, CP2102, CP2103, CP2104, CP2105, CP2108

**Location:** CP210x Manufacturing DLL

**Prototype:** `CP210x_STATUS CP210x_GetSelfPower( HANDLE Handle, LPBOOL SelfPower )`

**Parameters:**
1. Handle—Handle to the device to close as returned by *CP210x_Open()*.
2. SelfPower—Pointer to a boolean flag where TRUE means the Self-Powered bit is set, and FALSE means the Self-Powered bit is cleared.

**Return Value:** CP210x_STATUS = CP210x_SUCCESS,
CP210x_INVALID_PARAMETER,
CP210x_INVALID_HANDLE,
CP210x_DEVICE_IO_FAILED

#### 6.1.27. CP210x_GetMaxPower

**Description:** Returns the 1-byte Max Power field of the Configuration Descriptor of a CP210x device.

**Supported Devices:** CP2101, CP2102, CP2103, CP2104, CP2105, CP2108

**Location:** CP210x Manufacturing DLL

**Prototype:** `CP210x_STATUS CP210x_GetMaxPower( HANDLE Handle, LPBYTE MaxPower )`

**Parameters:**
1. Handle—Handle to the device to close as returned by *CP210x_Open()*.
2. MaxPower—Pointer to a 1-byte value returning the Maximum power consumption of the CP210x USB device expressed in 2 mA units.

**Return Value:** CP210x_STATUS = CP210x_SUCCESS,
CP210x_INVALID_PARAMETER,
CP210x_INVALID_HANDLE,
CP210x_DEVICE_IO_FAILED

#### 6.1.28. CP210x_GetMaxPower

**Description:** Returns the 1-byte Max Power field of the Configuration Descriptor of a CP210x device.

**Supported Devices:** CP2101, CP2102, CP2103, CP2104, CP2105, CP2108

**Location:** CP210x Manufacturing DLL

**Prototype:** `CP210x_STATUS CP210x_GetMaxPower( HANDLE Handle, LPBYTE MaxPower )`

**Parameters:**
1. Handle—Handle to the device to close as returned by *CP210x_Open()*.
2. MaxPower—Pointer to a 1-byte value returning the Maximum power consumption of the CP210x USB device expressed in 2 mA units.

**Return Value:** CP210x_STATUS = CP210x_SUCCESS,
CP210x_INVALID_PARAMETER,

CP210x_INVALID_HANDLE,
CP210x_DEVICE_IO_FAILED

### 6.1.29. CP210x_GetFlushBufferConfig

**Description:** Returns the flush buffer configuration of a CP210x device.

**Supported Devices:** CP2104, CP2105, CP2108

**Location:** CP210x Manufacturing DLL

**Prototype:**
```
CP210x_STATUS CP210x_GetFlushBufferConfig( HANDLE Handle,
LPWORD FlushBufferConfig )
```

**Parameters:**
1. Handle—Handle to the device to close as returned by *CP210x_Open()*.
2. FlushBufferConfig—Pointer to the values which indicates which buffer(s) are flushed (TX and/or RX) and upon which event (Open and/or Close). See the header file for the bit defintions for this byte value.

**Return Value:** CP210x_STATUS =
CP210x_SUCCESS,
CP210x_DEVICE_NOT_FOUND,
CP210x_INVALID_HANDLE,
CP210x_FUNCTION_NOT_SUPPORTED

### 6.1.30. CP210x_GetDeviceMode

**Description:** Gets the operating mode (GPIO or Modem) or each Interface of a CP210x device.

**Supported Devices:** CP2105

**Location:** CP210x Manufacturing DLL

**Prototype:**
```
CP210x_STATUS CP210x_SetMaxPower( HANDLE Handle,
BYTE DeviceModeECI, BYTE DeviceModeSCI)
```

**Parameters:**
1. Handle—Handle to the device to close as returned by *CP210x_Open()*.
2. DeviceModeECI—Pointer to a 1-byte value returning the 0 if interface is in Modem mode, or 1 if GPIO mode.
3. DeviceModeSCI—Pointer to a 1-byte value returning the 0 if interface is in Modem mode, or 1 if GPIO mode.

**Return Value:** CP210x_STATUS =
CP210x_SUCCESS,
CP210x_INVALID_HANDLE,
CP210x_DEVICE_NOT_FOUND
CP210x_FUNCTION_NOT_SUPPORTED

### 6.1.31. CP210x_GetBaudRateConfig

**Description:** Returns the baud rate configuration data of a CP210x device.

**Supported Devices:** CP2102, CP2103, CP2104, CP2105, CP2108

**Location:** CP210x Manufacturing DLL

**Prototype:** `CP210x_STATUS WINAPI CP210x_GetBaudRateConfig(HANDLE cyHandle, BAUD_CONFIG* baudConfigData);`

**Parameters:**
1. Handle—Handle to the device on which to determine the lock value.
2. BaudConfigData—Pointer to a *BAUD_CONFIG* structure returning the Baud Config data of the device.

**Return Value:** CP210x_STATUS = CP210x_SUCCESS,
CP210x_INVALID_PARAMETER,
CP210x_INVALID_HANDLE,
CP210x_DEVICE_IO_FAILED

#### 6.1.32. CP210x_GetLockValue

**Description:** Returns the 1-byte Lock Value of a CP210x device.

**Supported Devices:** CP2102, CP2103, CP2104, CP2105, CP2108

**Location:** CP210x Manufacturing DLL

**Prototype:** `CP210x_STATUS WINAPI CP210x_GetLockValue(HANDLE cyHandle, LPBYTE lpbLockValue);`

**Parameters:**
1. Handle—Handle to the device on which to determine the lock value.
2. LockValue—Pointer to a 1-byte value returning the Lock Value of the device. A 0x01 denotes that the device is locked, and a 0x00 denotes that the device is unlocked.

**Return Value:** CP210x_STATUS = CP210x_SUCCESS,
CP210x_INVALID_PARAMETER,
CP210x_INVALID_HANDLE,
CP210x_DEVICE_IO_FAILED

#### 6.1.33. CP210x_GetPortConfig

**Description:** Gets the current port pin configuration from the CP210x device.

**Supported Devices:** CP2103, CP2104, CP2105, CP2108

**Location:** CP210x Manufacturing DLL

**Prototype:** `CP210X_STATUS CP210x_GetPortConfig(HANDLE Handle, LPVOID PortConfig)`

**Parameters:**
1. Handle—Handle to the device as returned by CP210x_Open()
2. Port Config—Pointer to a PORT_CONFIG structure

**Return Value:** CP210X_STATUS = CP210X_SUCCESS,
CP210X_INVALID_HANDLE,
CP210X_DEVICE_IO_FAILED,
CP210X_UNSUPPORTED_DEVICE

### 6.1.34. CP210xGetDualPortConfig

**Description:** Gets the current port pin configuration from the CP210x device.

**Supported Devices:** CP2105

**Location:** CP210x Manufacturing DLL

**Prototype:** `CP210X_STATUS CP210x_SetPortConfig(HANDLE Handle, LPVOID DualPortConfig)`

**Parameters:**
1. Handle—Handle to the device as returned by CP210x_Open()
2. DualPortConfig—Pointer to a DUAL_PORT_CONFIG structure

**Return Value:** CP210X_STATUS = CP210X_SUCCESS,
CP210X_INVALID_HANDLE,
CP210X_DEVICE_IO_FAILED,
CP210X_UNSUPPORTED_DEVICE

### 6.1.35. CP210x_Reset

**Description:** Initiates a reset of the USB interface.

**Note:** There is a delay of ~1 second before the reset is initiated by the device firmware to give the application time to call CP210x_Close() to close the device handle. No further operations should be performed with the device until it resets, re-enumerates in Windows, and a new handle is opened.

**Supported Devices:** CP2101, CP2102, CP2103, CP2104, CP2105, CP2108

**Location:** CP210x Manufacturing DLL

**Prototype:** `CP210x_STATUS CP210x_Reset( HANDLE Handle )`

**Parameters:**
1. Handle—Handle to the device to close as returned by *CP210x_Open()*.

**Return Value:** CP210x_STATUS = CP210x_SUCCESS,
CP210x_INVALID_HANDLE,
CP210x_DEVICE_IO_FAILED

### 6.1.36. CP210xGetQuadPortConfig

**Description:** Gets the current port pin configuration from the CP210x device.

**Supported Devices:** CP2105

**Location:** CP210x Manufacturing DLL

**Prototype:** `CP210X_STATUS CP210x_SetPortConfig(HANDLE Handle, LPVOID DualPortConfig)`

**Parameters:**
1. Handle—Handle to the device as returned by CP210x_Open()
2. DualPortConfig—Pointer to a DUAL_PORT_CONFIG structure

**Return Value:** CP210X_STATUS = CP210X_SUCCESS,
CP210X_INVALID_HANDLE,
CP210X_DEVICE_IO_FAILED,
CP210X_UNSUPPORTED_DEVICE

### 6.1.37. CP210x Manufacturing.DLL Type Definitions and Constants

Type Definitions from C++ Header File CP210xManufacturingDLL.h

```
// GetProductString() function flags
#define     CP210x_RETURN_SERIAL_NUMBER             0x00
#define     CP210x_RETURN_DESCRIPTION               0x01
#define     CP210x_RETURN_FULL_PATH                 0x02

// GetDeviceVersion() return codes
#define     CP210x_CP2101_VERSION                   0x01
#define     CP210x_CP2102_VERSION                   0x02
#define     CP210x_CP2103_VERSION                   0x03
#define     CP210x_CP2104_VERSION                   0x04
#define     CP210x_CP2105_VERSION                   0x05
#define     CP210x_CP2108_VERSION                   0x06

// 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_FILE_ERROR                       0x06
#define     CP210x_COMMAND_FAILED                   0x08
#define     CP210x_INVALID_ACCESS_TYPE              0x09

// Type definitions
typedef     int         CP210x_STATUS;

// Buffer size limits
#define     CP210x_MAX_DEVICE_STRLEN                256
#define     CP210x_MAX_PRODUCT_STRLEN               126
#define     CP210x_MAX_SERIAL_STRLEN                63
#define     CP210x_MAX_MAXPOWER                     250


// Type definitions
typedef     char  CP210x_DEVICE_STRING[CP210x_MAX_DEVICE_STRLEN];
typedef     char  CP210x_PRODUCT_STRING[CP210x_MAX_PRODUCT_STRLEN];
typedef     char  CP210x_SERIAL_STRING[CP210x_MAX_SERIAL_STRLEN];


//Baud Rate Aliasing definitions
#define     NUM_BAUD_CONFIGS                        32
```

```
typedef     struct
{
      WORD  BaudGen;
      WORD  Timer0Reload;
      BYTE  Prescaler;
      DWORD       BaudRate;
} BAUD_CONFIG;

#define     BAUD_CONFIG_SIZE10

typedef     BAUD_CONFIG          BAUD_CONFIG_DATA[NUM_BAUD_CONFIGS];


//Port Config definitions
typedef     struct
{
      WORD Mode;        // Push-Pull = 1, Open-Drain = 0
      WORD Reset_Latch; // Logic High = 1, Logic Low = 0
      WORD Suspend_Latch;// Logic High = 1, Logic Low = 0
      unsigned char EnhancedFxn;
} PORT_CONFIG;

// Define bit locations for Mode/Latch for Reset and Suspend structures
#define PORT_RI_ON                  0x0001
#define PORT_DCD_ON                 0x0002
#define PORT_DTR_ON                 0x0004
#define PORT_DSR_ON                 0x0008
#define PORT_TXD_ON                 0x0010
#define PORT_RXD_ON                 0x0020
#define PORT_RTS_ON                 0x0040
#define PORT_CTS_ON                 0x0080

#define PORT_GPIO_0_ON              0x0100
#define PORT_GPIO_1_ON              0x0200
#define PORT_GPIO_2_ON              0x0400
#define PORT_GPIO_3_ON              0x0800

#define PORT_SUSPEND_ON             0x4000//  Can't configure latch value
#define PORT_SUSPEND_BAR_ON         0x8000//  Can't configure latch value

// Define bit locations for EnhancedFxn
#define EF_GPIO_0_TXLED                   0x01   //  Under device control
#define EF_GPIO_1_RXLED                   0x02   //  Under device control
#define EF_GPIO_2_RS485                   0x04   //  Under device control
#define EF_RESERVED_0                     0x08   //  Reserved, leave bit 3 cleared
#define EF_WEAKPULLUP                     0x10   //  Weak Pull-up on
#define EF_RESERVED_1                     0x20   //  Reserved, leave bit 5 cleared
#define EF_SERIAL_DYNAMIC_SUSPEND         0x40   //  For 8 UART/Modem signals
#define EF_GPIO_DYNAMIC_SUSPEND           0x80   //  For 4 GPIO signals
```

## 6.2. CP210xRuntime.DLL

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.

### 6.2.1. CP210xRT_ReadLatch

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

**Supported Devices:** CP2103, CP2104, CP2105, CP2108

**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

### 6.2.2. CP210xRT_WriteLatch

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

**Supported Devices:** CP2103, CP2104, CP2105, CP2108

**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

#### 6.2.3. CP210xRT_GetPartNumber

**Description:** Gets the part number of the current device.

**Supported Devices:** CP2101, CP2102, CP2103, CP2104, CP2105, CP2108

**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

#### 6.2.4. CP210xRT_GetDeviceProductString

**Description:** Gets the product string in the current device.

**Supported Devices:** CP2101, CP2102, CP2103, CP2104, CP2105, CP2108

**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

#### 6.2.5. CP210xRT_GetDeviceSerialNumber

**Description:** Gets the serial number in the current device.

**Supported Devices:** CP2101, CP2102, CP2103, CP2104, CP2105, CP2108

**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

### 6.2.6. CP210xRT_GetDeviceInterfaceString

**Description:** Gets the interface string of the current device.

**Supported Devices:** CP2105, CP2108

**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

## CONTACT INFORMATION

Silicon Laboratories Inc.
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.

**Patent Notice**
Silicon Labs invests in research and development to help our customers differentiate in the market with innovative low-power, small size, analog-intensive mixed-signal solutions. Silicon Labs' extensive patent portfolio is a testament to our unique approach and world-class engineering team.

Стандарт Электрон Связь

Мы молодая и активно развивающаяся компания в области поставок электронных компонентов. Мы поставляем электронные компоненты отечественного и импортного производства напрямую от производителей и с крупнейших складов мира.

Благодаря сотрудничеству с мировыми поставщиками мы осуществляем комплексные и плановые поставки широчайшего спектра электронных компонентов.

Собственная эффективная логистика и склад в обеспечивает надежную поставку продукции в точно указанные сроки по всей России.

Мы осуществляем техническую поддержку нашим клиентам и предпродажную проверку качества продукции. На все поставляемые продукты мы предоставляем гарантию .

Осуществляем поставки продукции под контролем ВП МО РФ на предприятия военно-промышленного комплекса России , а также работаем в рамках 275 ФЗ с открытием отдельных счетов в уполномоченном банке. Система менеджмента качества компании соответствует требованиям ГОСТ ISO 9001.

Минимальные сроки поставки, гибкие цены, неограниченный ассортимент и индивидуальный подход к клиентам являются основой для выстраивания долгосрочного и эффективного сотрудничества с предприятиями радиоэлектронной промышленности, предприятиями ВПК и научно-исследовательскими институтами России.

С нами вы становитесь еще успешнее!

**Наши контакты:**

**Телефон:** +7 812 627 14 35

**Электронная почта:** sales@st-electron.ru

**Адрес:** 198099, Санкт-Петербург, Промышленная ул, дом № 19, литера Н, помещение 100-Н Офис 331