Contact us

MCP2200

USB 2.0 to UART Protocol Converter with GPIO

Features

Universal Serial Bus (USB)

- Supports Full-Speed USB (12 Mb/s)
- Implements USB Protocol Composite Device:
  - Communication Device Class (CDC) for Communications and Configuration
  - Human Interface Device (HID) for I/O control
- 128-Byte Buffer to Handle Data Throughput at Any UART Baud Rate:
  - 64-byte transmit
  - 64-byte receive
- Fully Configurable VID and PID Assignments and String Descriptors
- Bus-Powered or Self-Powered
- USB 2.0 Compliant: TID 40001150

USB Driver and Software Support

- Uses Standard Windows® Drivers for Virtual Com Port (VCP): Windows XP (SP2 or later), Windows Vista, Windows 7, Windows 8, Windows 8.1 and Windows 10
- Configuration Utility for Initial Configuration

Universal Asynchronous Receiver/Transmitter (UART)

- Responds to `SET_LINE_CODING` Commands to Dynamically Change Baud Rates
- Supports Baud Rates: 300-1000k
- Hardware Flow Control
- UART Signal Polarity Option

General Purpose Input/Output (GPIO) Pins

- Eight General Purpose I/O pins

EEPROM

- 256 Bytes of User EEPROM

Other

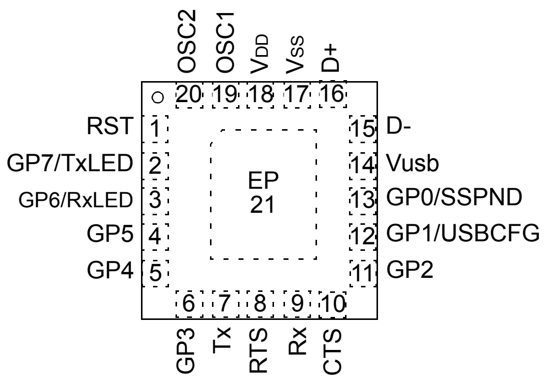
- USB Activity LED Outputs (TxLED and RxLED)
- SSPND Output Pin
- USBCFG Output Pin (indicates when the enumeration is completed)
- Operating Voltage: 3.0V-5.5V
- Oscillator Input: 12 MHz
- Electrostatic Discharge (ESD) Protection: >4 kV Human Body Model (HBM)
- Industrial (I) Operating Temperature: –40°C to +85°C

Package Types

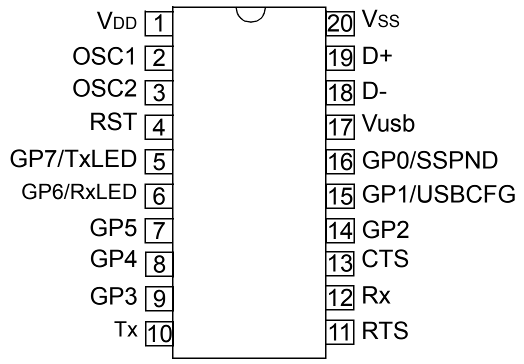
The device is offered in the following packages:

- 20-lead VQFN (5x5 mm)
- 20-lead SOIC
- 20-lead SSOP

MCP2200
5x5 VQFN*

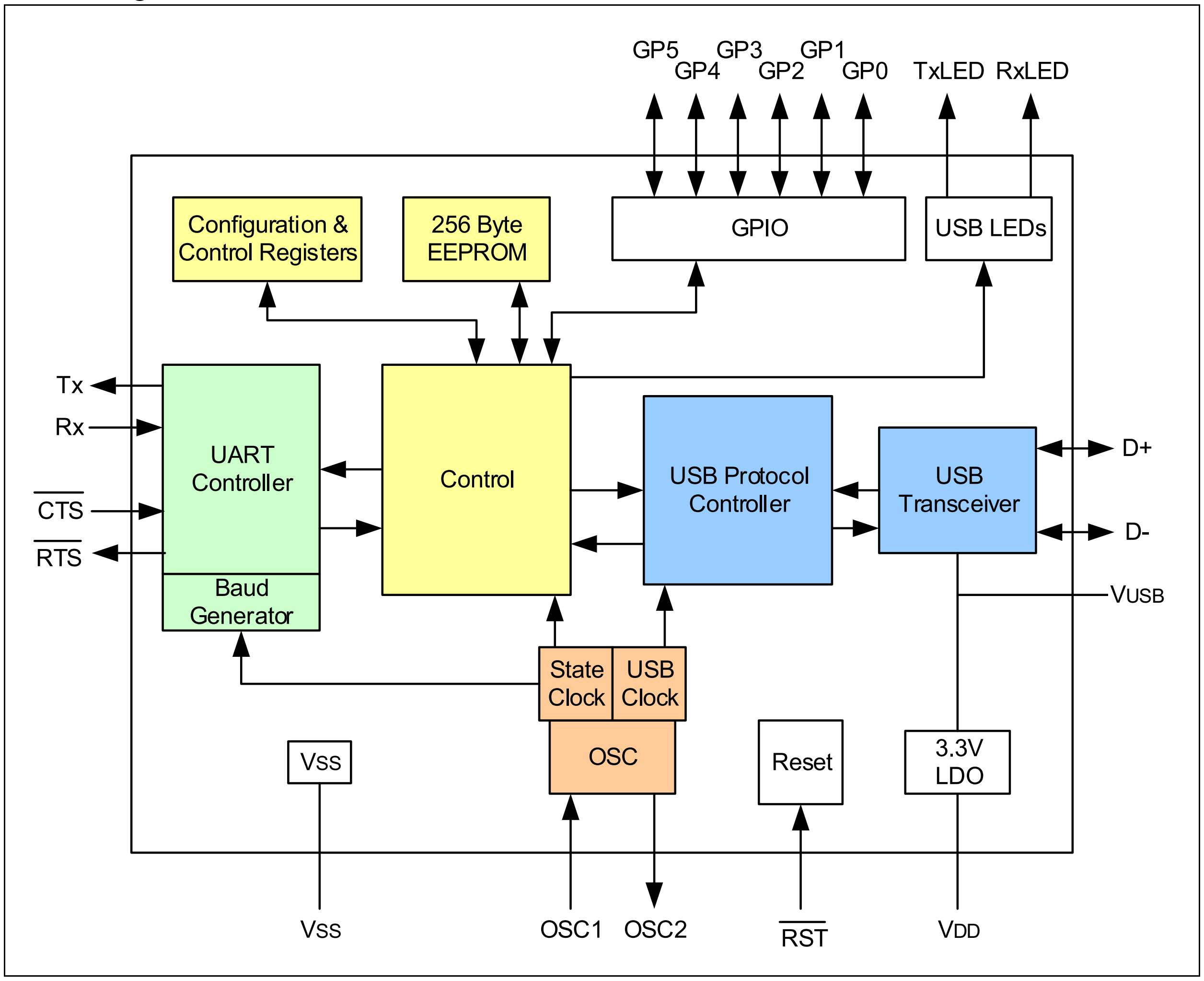


MCP2200
SOIC, SSOP



* Includes Exposed Thermal Pad (EP); see Table 1-1.

Block Diagram



1.0 FUNCTIONAL DESCRIPTION

The MCP2200 is a USB-to-UART serial converter that enables USB connectivity in applications that have a UART interface. The device reduces external components by integrating the USB termination resistors. The MCP2200 also has 256 bytes of integrated user EEPROM.

The MCP2200 has eight general purpose input/output pins. Four pins have alternate functions to indicate USB and communication status. See Table 1-1 and Section 1.6 "GPIO Module" for details about the pin functions.

TABLE 1-1: PINOUT DESCRIPTION

Pin Name	VQFN	SSOP, SOIC	Pin Type	Standard Function	Alternate Function
GP0/SSPND	13	16	I/O	General purpose I/O	USB suspend status pin (refer to Section 1.6.1.1 "SSPND Pin Function")
GP1/USB-CFG	12	15	I/O	General purpose I/O	USB configuration status pin (refer to Section 1.6.1.2 "USBCFG Pin Function")
GP2	11	14	I/O	General purpose I/O	
GP3	6	9	I/O	General purpose I/O	
GP4	5	8	I/O	General purpose I/O	
GP5	4	7	I/O	General purpose I/O	
GP6/RxLED	3	6	I/O	General purpose I/O	USB receive activity LED output (refer to Section 1.6.1.3 "RxLED Pin Function (IN Message)")
GP7/TxLED	2	5	I/O	General purpose I/O	USB transmit activity LED output (refer to Section 1.6.1.4 "TxLED Pin Function (OUT Message)")
CTS	10	13	I	Hardware flow control "Clear to Send" input signal	
RTS	8	11	O	Hardware flow control "Request to Send" output signal	
Rx	9	12	I	USART RX input	
Tx	7	10	O	USART TX output	
RST	1	4	I	Reset input must be externally biased	
VDD	18	1	P	Power	
Vss	17	20	P	Ground	
OSC1	19	2	I	Oscillator input	
OSC2	20	3	O	Oscillator output	
D+	16	19	I/O	USB D+	
D-	15	18	I/O	USB D-	
Vusb	14	17	P	USB power pin (internally connected to 3.3V). Should be locally bypassed with a high-quality ceramic capacitor.	
EP	21	—	—	Exposed Thermal Pad (EP). Do not electrically connect.	

1.1 Supported Operating Systems

Windows XP (SP2 and later), Windows Vista, Windows 7, Windows 8, Windows 8.1 and Windows 10 operating systems are supported.

1.1.1 ENUMERATION

The MCP2200 will enumerate as a USB device after Power-on Reset (POR). The device enumerates as both a Human Interface Device (HID) for I/O control, and a Virtual Com Port (VCP).

1.1.1.1 Human Interface Device (HID)

The MCP2200 enumerates as an HID, so the device can be configured and the I/O can be controlled. A DLL that facilitates I/O control through a custom interface is supplied by Microchip.

1.1.1.2 Virtual Com Port (VCP)

The VCP enumeration implements the USB-to-UART data translation.

1.2 Control Module

The control module is the heart of the MCP2200. All other modules are tied together and controlled via the control module. The control module manages the data transfers between the USB and the UART, as well as the command requests generated by the USB host controller and the commands for controlling the function of the UART and I/O.

1.2.1 SERIAL INTERFACE

The control module interfaces to the UART and USB modules.

1.2.2 INTERFACING TO THE DEVICE

The MCP2200 can be accessed for reading and writing via USB host commands. The device cannot be accessed and controlled via the UART interface.

1.3 UART Interface

The MCP2200 UART interface consists of the Tx and Rx data signals and the $\overline{\text{RTS}}/\overline{\text{CTS}}$ flow control pins.

The UART is configurable for several baud rates. The available baud rates are listed in Table 1-3.

1.3.1 INITIAL CONFIGURATION

The default UART configuration is 19200, 8, N, 1. The default start-up baud rate can be changed using the Microchip-supplied configuration PC tool.

Alternatively, a custom configuration tool can be created using the Microchip-supplied DLL to set the baud rate as well as other parameters. See Section 2.0 “Configuration” for details.

TABLE 1-2: UART CONFIGURATIONS

Parameter	Configuration
Primary Baud Rates	See Table 1-3
Data Bits	8
Parity	N
Stop Bits	1

1.3.2 GET/SET LINE CODING

The `GET_LINE_CODING` and `SET_LINE_CODING` commands are used to read and set the UART parameters while in operation. For example, HyperTerminal sends the `SET_LINE_COMMAND` when connecting to the port. The MCP2200 responds by setting the baud rate only. The other parameters (data bits, parity, stop bits) remain unchanged.

1.3.2.1 Rounding Errors

The primary baud rate setting (with the rounding errors) is shown in Table 1-3. If baud rates other than the ones shown in the table are used, the error percentage can be calculated using Equation 1-1 to find the actual baud rate.

TABLE 1-3: UART PRIMARY BAUD RATES

Desired Rate	Actual rate	% Error
300	300	0.00%
1200	1200	0.00%
2400	2400	0.00%
4800	4800	0.00%
9600	9600	0.00%
19200	19200	0.00%
38400	38339	0.16%
57600	57692	0.16%
115200	115385	0.16%
230400	230769	0.16%
460800	461538	0.16%
921600	923077	0.16%

EQUATION 1-1: SOLVING FOR ACTUAL BAUD RATE

$$ActualRate = \frac{12\ MHz}{int(x)}$$

Where:

$$x = \frac{12\ MHz}{Desired\ Baud}$$

1.3.3 CUSTOM BAUD RATES

Custom baud rates are configured by sending the `SET_LINE_CODING` USB command, or by using the DLL. See Section 2.0 "Configuration" for more information.

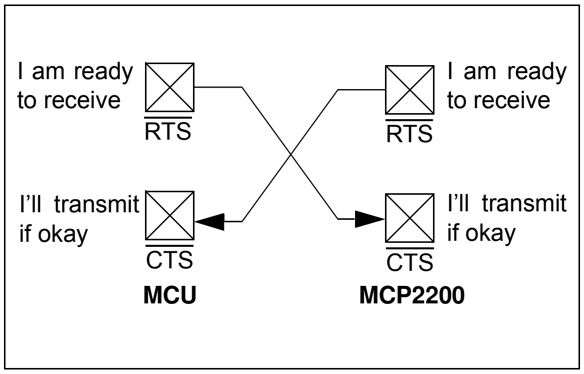
1.3.4 HARDWARE FLOW CONTROL

Hardware flow control uses the $\overline{\text{RTS}}$ and $\overline{\text{CTS}}$ pins as a handshake between two devices. The $\overline{\text{RTS}}$ pin of one device is typically connected to the $\overline{\text{CTS}}$ of the other device.

$\overline{\text{RTS}}$ is an active-low output that notifies the other device when it is ready to receive data by driving the pin low. The MCP2200 trip point for deasserting $\overline{\text{RTS}}$ (high) is 63 characters. This is one character short of "buffer full".

$\overline{\text{CTS}}$ is an active-low input that notifies the MCP2200 when it is ready to send data. The MCP2200 will check $\overline{\text{CTS}}$ just before loading and sending UART data. If the pin is asserted during a transfer, the transfer will continue. Refer to Figure 1-1.

FIGURE 1-1: $\overline{\text{RTS}}$/$\overline{\text{CTS}}$ CONNECTIONS EXAMPLE



1.3.4.1 Flow Control Disabled

The buffer pointer does not increment (or reset to zero) if the buffer is full. Therefore, if hardware flow control is not enabled and an overflow occurs (i.e., 65 unprocessed characters received), the new data overwrites the last position in the buffer.

1.4 USB Protocol Controller

The USB controller in the MCP2200 is full-speed USB 2.0 compliant.

- Composite device (CDC + HID):
  - CDC: USB-to-UART communications
  - HID: I/O control, EEPROM access and initial configuration
- 128-byte buffer to handle data throughput at any UART baud rate:
  - 64-byte transmit
  - 64-byte receive
- Fully configurable VID and PID assignments and descriptors (stored on-chip)
- Bus-powered or self-powered

1.4.1 DESCRIPTORS

During configuration, the supplied PC interface stores the descriptors in the MCP2200.

1.4.2 SUSPEND AND RESUME

The USB Suspend and Resume signals are supported for power management of the MCP2200. The device enters Suspend mode when "suspend signaling" is detected on the bus.

The MCP2200 exits Suspend mode when any of the following events occur:

1. "Resume signaling" is detected or generated.
2. A USB "Reset" signal is detected.
3. A device reset occurs.

1.5 USB Transceiver

The MCP2200 has a built-in, full-speed USB 2.0 transceiver internally connected to the USB module.

The USB transceiver obtains power from the V_{USB} pin, which is internally connected to the 3.3V regulator. The best electrical signal quality is obtained when V_{USB} is locally bypassed with a high-quality ceramic capacitor.

1.5.1 INTERNAL PULL-UP RESISTORS

The MCP2200 devices have built-in pull-up resistors designed to meet the requirements for full-speed USB.

1.5.2 MCP2200 POWER OPTIONS

The following are the main power options for the MCP2200:

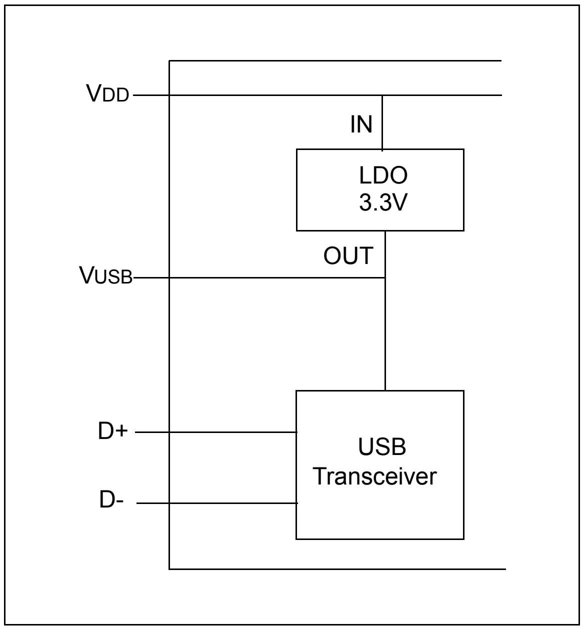
- USB Bus-Powered (5V)
- 3.3V Self-Powered

1.5.2.1 Internal Power Supply Details

MCP2200 offers various options for power supply. To meet the required USB signaling levels, the MCP2200 device incorporates an internal LDO used solely by the USB transceiver in order to present the correct D+/D- voltage levels.

Figure 1-2 shows the internal connections of the USB transceiver LDO in relation to the VDD power supply rail. The output of the USB transceiver LDO is tied to the VUSB line. A capacitor connected to the VUSB pin is required if the USB transceiver LDO provides the 3.3V supply to the transceiver.

FIGURE 1-2: MCP2200 INTERNAL POWER SUPPLY DETAILS



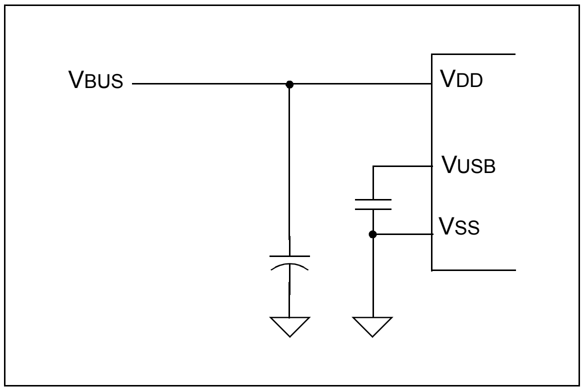
The provided VDD voltage has a direct influence on the voltage levels present on the GPIO pins (Rx/Tx and $\overline{\text{RTS}}$/$\overline{\text{CTS}}$). When VDD is 5V, all of these pins will have a logical '1' around 5V with the variations specified in Section 3.1 "DC Characteristics".

For applications that require a 3.3V logical '1' level, VDD must be connected to a power supply providing 3.3V voltage. In this case, the internal USB transceiver LDO cannot provide the required 3.3V of power. It is necessary to also connect the VUSB pin to the 3.3V power supply rail. This way, the USB transceiver is powered-up directly from the 3.3V power supply.

1.5.2.2 USB Bus-Powered (5V)

In Bus Power Only mode, all power for the application is drawn from the USB (Figure 1-3). This is effectively the simplest power method for the device.

FIGURE 1-3: BUS POWER ONLY



In order to meet the inrush current requirements of the USB 2.0 specifications, the total effective capacitance appearing across VBUS and ground must be no more than 10 µF. If it is not more than 10 µF, some kind of inrush current limiting is required. For more details on inrush current limiting, consider the latest version of the *"Universal Serial Bus Specification"*.

According to the USB 2.0 specification, all USB devices must also support a low-power Suspend mode. In the USB Suspend mode, devices must consume no more than 500 µA (or 2.5 mA for high-powered devices that are remote wake-up capable) from the 5V VBUS line of the USB cable.

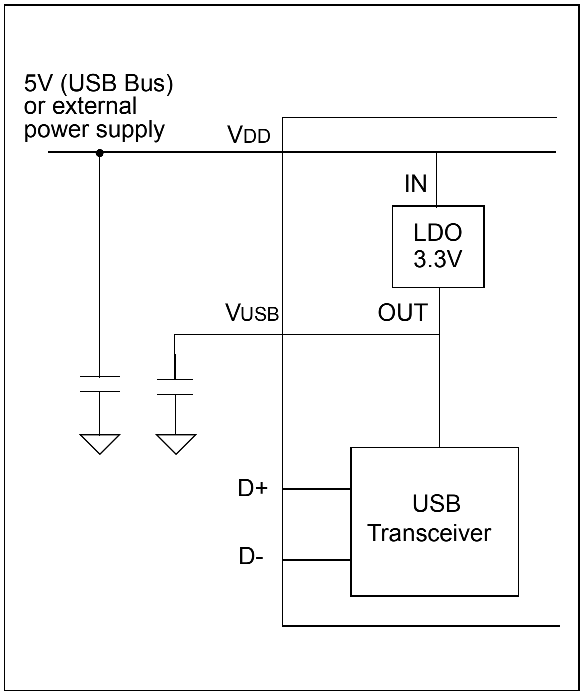
The host signals the USB device to enter Suspend mode by stopping all USB traffic to that device for more than 3 ms.

The USB bus provides a 5V voltage. However, the USB transceiver requires 3.3V for the signaling (on the D+ and D- lines).

During USB Suspend mode, the D+ or D- pull-up resistor must remain active, which will consume some of the allowed suspended current budget (500 µA/ 2.5 mA). The VUSB pin is required to have an external bypass capacitor. It is recommended that the capacitor be a ceramic capacitor between 0.22 µF. and 0.47 µF.

Figure 1-4 shows a circuit where MCP2200's internal LDO is used to provide 3.3V to the USB transceiver. The voltage on the VDD affects the voltage levels onto the GPIO pins (Rx/Tx and $\overline{\text{RTS}}$/$\overline{\text{CTS}}$). With VDD at 5V, these pins will have a logic '1' of 5V with the variations specified in Section 3.1 "DC Characteristics".

FIGURE 1-4: TYPICAL POWER SUPPLY OPTION USING THE 5V PROVIDED BY THE USB

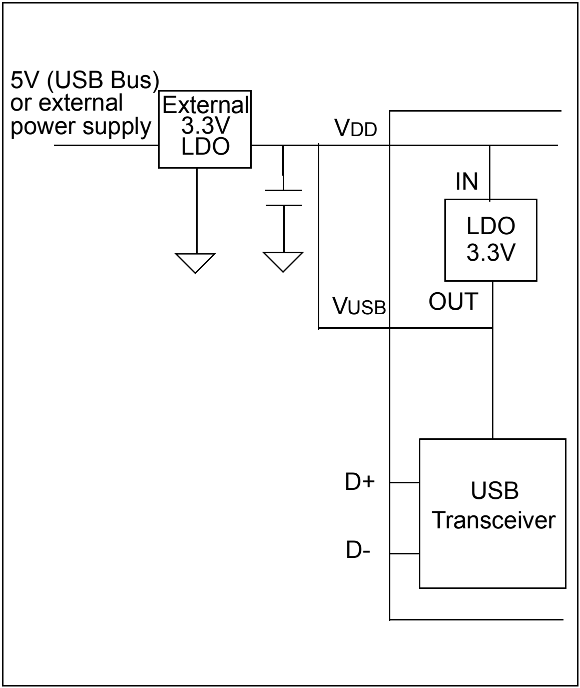


1.5.2.3 3.3V Self-Powered

Typically, many embedded applications are using 3.3V power supplies. When such an option is available in the target system, MCP2200 can be powered up from the existing 3.3V power supply rail. The typical connections for the MCP2200 are shown in Figure 1-5.

In this example, the MCP2200 has both VDD and VUSB lines tied to the 3.3V rail. These tied connections disable the internal USB transceiver LDO of the MCP2200 to regulate the power supply on VUSB pin. Another consequence is that the '1' logical level on the GPIO pins will be at the 3.3V level, in accordance with the variations specified in Section 3.1 "DC Characteristics".

FIGURE 1-5: USING AN EXTERNALLY PROVIDED 3.3V POWER SUPPLY



1.6 GPIO Module

The GPIO Module is a standard 8-bit I/O port.

1.6.1 CONFIGURABLE PIN FUNCTIONS

The pins can be configured as:

- GPIO – individually configurable general purpose input or output
- SSPND – USB Suspend state
- USBCFG – indicates USB configuration status
- RxLED – indicates USB receive traffic
- TxLED – indicates USB transmit traffic

1.6.1.1 SSPND Pin Function

The SSPND pin (if enabled) reflects the USB state (Suspend/Resume). The pin is active-low when the Suspend state has been issued by the USB host. Likewise, the pin drives 'high' after the Resume state is achieved.

This pin allows the application to go into low power mode when USB communication is suspended, and switches to a full active state when USB activity is resumed.

1.6.1.2 USBCFG Pin Function

The USBCFG pin (if enabled) starts out 'low' during power-up or after Reset, and goes 'high' after the device successfully configures to the USB. The pin will go 'low' when in Suspend mode and 'high' when the USB resumes.

1.6.1.3 RxLED Pin Function (IN Message)

The ‘Rx’ in the pin name refers to the USB host. The RxLED pin is an indicator for USB ‘IN’ messages.

This pin will either pulse low for a period of time (configurable for ~100 ms or ~200 ms), or toggle to the opposite state for every message received (IN message) by the USB host. This allows the application to count messages or provide a visual indication of USB traffic.

1.6.1.4 TxLED Pin Function (OUT Message)

The ‘Tx’ in the pin name refers to the USB host. The TxLED pin is an indicator for USB ‘OUT’ messages.

This pin will either pulse low for a period of time (configurable for ~100 ms or ~200 ms), or toggle to the opposite state for every message transmitted (OUT message) by the USB host. This allows the application to count messages or provide a visual indication of USB traffic.

1.7 EEPROM Module

The EEPROM module is a 256-byte array of nonvolatile memory. The memory locations are accessed for read/write operations via USB host commands. Refer to Section 2.0 “Configuration” for details on accessing the EEPROM. The memory cells for data EEPROM are rated to endure thousands of erase/write cycles, up to 100K for EEPROM.

Data retention without refresh is conservatively estimated to be greater than 40 years.

The host should wait for the write cycle to complete and then verify the write by reading the byte(s).

1.8 RESET/POR

1.8.1 RESET PIN

The $\overline{\text{RST}}$ pin provides a method for triggering an external Reset of the device. A Reset is generated by holding the pin low. These devices have a noise filter in the Reset path which detects and ignores small pulses.

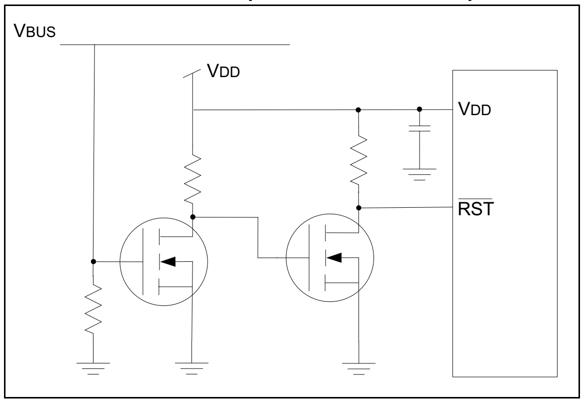
1.8.2 POWER-ON RESET (POR)

A POR pulse is generated on-chip whenever VDD rises above a certain threshold. This allows the device to start in the initialized state when VDD is adequate for operation.

To take advantage of the POR circuitry, tie the $\overline{\text{RST}}$ pin through a resistor (1 kΩ to 10 kΩ) to VDD. This will eliminate external RC components usually needed to create a POR delay.

In the self-powered configuration, it is recommended to tie the $\overline{\text{RST}}$ pin to the VBUS line of the USB connector, as in Figure 1-6.

FIGURE 1-6: CONNECTING THE $\overline{\text{RST}}$ PIN IN A SELF-POWERED CONFIGURATION (RECOMMENDED)



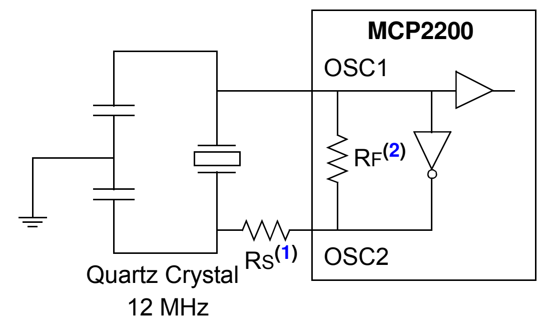
When the device starts normal operation (i.e., exits the Reset condition), device operating parameters (voltage, frequency, temperature, etc.) must be met to ensure operation. If these conditions are not achieved, the device must be held in Reset until the operating conditions are met.

1.9 Oscillator

The input clock must be 12 MHz to provide the proper frequency for the USB module.

USB full speed is defined as 12 Mb/s. The clock input accuracy is ±0.25% (2,500 ppm maximum).

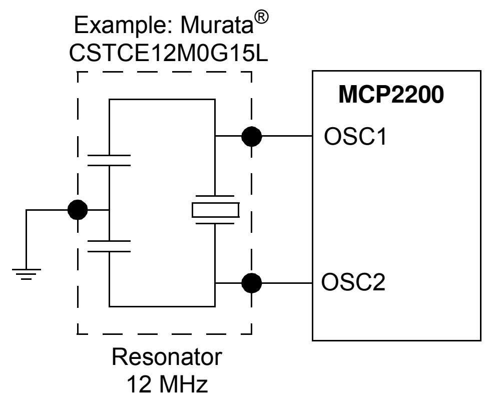
FIGURE 1-7: QUARTZ CRYSTAL OPERATION



Note 1: A series resistor (RS) may be required for quartz crystals with high drive level.

2: The value of RF is typically between 2 MΩ to 10 MΩ..

FIGURE 1-8: CERAMIC RESONATOR OPERATION



2.0 CONFIGURATION

The MCP2200 is configured by writing special commands using the HID interface. Configuration can be achieved using the configuration utility provided by Microchip. Alternatively, a custom utility can be developed by using the DLL available on the MCP2200 product page.

2.1 Configuration Utility

The configuration utility provided by Microchip allows the user to configure the MCP2200 to custom defaults. The configuration utility (shown in Figure 2-1) connects to the device's HID interface, where all of the configurable features can be set.

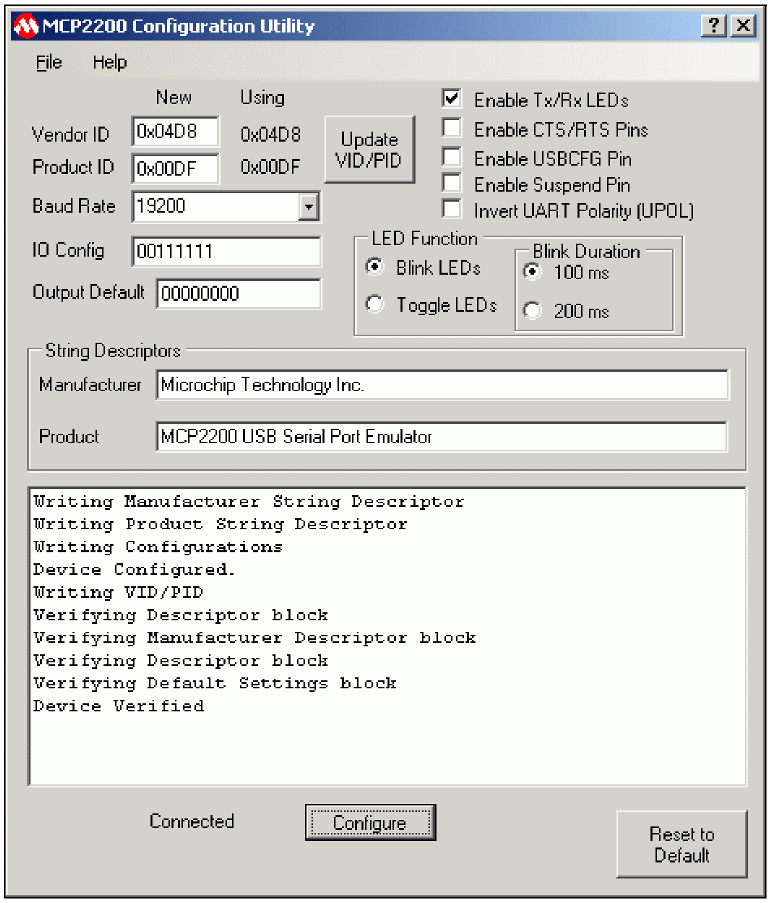
2.2 Serial String

The MCP2200 is supplied from the factory with a serialized USB serial string.

TABLE 2-1: CONFIGURATION DESCRIPTIONS

Configuration Name	Description
Vendor ID (0x04D8)	The USB vendor identification assigned to Microchip by the USB consortium.
Product ID (0x00DF)	Device ID assigned by Microchip. The device can be used as-is, or Microchip can assign a custom PID by request.
Baud Rate	Sets the UART baud rate using a list of primary baud rates. See the UART section for details on setting non-primary baud rates.
IO Config	Individually configures the I/O to inputs or outputs.
IO Default	Individually configures the output default state for pins configured as outputs.
Tx/Rx LEDs	Enables/disables the GP6 and GP7 pins to function as USB traffic indicators. Pins are active-low when configured as traffic indicators.
Hardware Flow Control	Enables/disables $\overline{\text{CTS}}$ and $\overline{\text{RTS}}$ flow control.
USBCFG Pin	Enables/disables the GP1 pin as a USB configuration status indicator.
Suspend Pin	Enables/disables the GP0 pin as a USB suspend status pin.
Invert Sense	Enables/disables the UART lines states: - Normal – Tx/Rx idle-high; $\overline{\text{CTS}}$/$\overline{\text{RTS}}$ active-low - Inverted – Tx/Rx idle-low; $\overline{\text{CTS}}$/$\overline{\text{RTS}}$ active-high
Manufacturer String	USB manufacturer string.
Product String	USB product string.

FIGURE 2-1: CONFIGURATION UTILITY



	New	Using
Vendor ID	0x04D8	0x04D8
Product ID	0x00DF	0x00DF

Baud Rate: 19200

IO Config: 00111111

Output Default: 00000000

- [✓] Enable Tx/Rx LEDs
- [ ] Enable CTS/RTS Pins
- [ ] Enable USBCFG Pin
- [ ] Enable Suspend Pin
- [ ] Invert UART Polarity (UPOL)

LED Function: [✓ Blink LEDs] Toggle LEDs

Blink Duration: [✓ 100 ms] 200 ms

Manufacturer: Microchip Technology Inc.

Product: MCP2200 USB Serial Port Emulator

```
Writing Manufacturer String Descriptor
Writing Product String Descriptor
Writing Configurations
Device Configured.
Writing VID/PID
Verifying Descriptor block
Verifying Manufacturer Descriptor block
Verifying Descriptor block
Verifying Default Settings block
Device Verified
```

2.3 Simple Configuration and I/O DLL

To help the user develop a custom configurator, Microchip provides a DLL that uses Microsoft®.NET Framework 3.5. There is documentation about drivers and utilities on the MCP2200 product page at www.microchip.com (in the Software section) with information on associating the DLL with a Visual C++ project.

2.3.1 SIMPLE I/O DLL CALLS

Table 2-2 lists the functions provided by the DLL to allow the configuration of the device and control of the I/O.

TABLE 2-2: CONFIGURATION FUNCTIONS

Category and Function Name	
Initialization (Note 1)	
`void InitMCP2200(VID, PID)`	
Configuration (Note 2)	
`bool ConfigureIO(mask)`	
`bool ConfigureIoDefaultOutput(mask, defaultGpioOutputValue)`	
`bool fnRxLED (OFF/TOGGLE/BLINKSLOW/BLINKFAST)`	
`bool fnTxLED (OFF/TOGGLE/BLINKSLOW/BLINKFAST)`	
`bool fnHardwareFlowControl (ON/OFF)`	
`bool fnULoad(ON/OFF)`	
`bool fnSuspend (ON/OFF)`	
`bool ConfigureMCP2200(mask, baudrate, RxLedMode, TxLedMode, flowCtrl, ULoad, suspend)`	
`bool ConfigureIO(mask)`	
Miscellaneous	
`String^ GetDeviceInfo(deviceIndex)`	
`unsigned int GetNoOfDevices()`	
`int GetSelectedDevice()`	
`String^ GetSelectedDeviceInfo()`	
`bool IsConnected()`	
`int SelectDevice(uiDeviceNo)`	
`int ReadEEPROM(uiEEPAddress)`	
`int WriteEEPROM(uiEEPAddress, ucValue)`	
I/O Control	
`bool ClearPin(pinnumber)`	
`bool SetPin(pinnumber)`	
`bool ReadPin(pinnumber, *pinvalue)`	
`int ReadPinValue(pinnumber)`	
`bool ReadPort(*portValue)`	
`int ReadPortValue()`	
`bool WritePort(portValue)`	
Summary	
`bool SimpleIOClass::ClearPin(unsigned int pin)`	Section 2.3.1.1
`bool SimpleIOClass::ConfigureIO (unsigned char IOMap)`	Section 2.3.1.2
`bool SimpleIOClass::ConfigureIoDefaultOutput(unsigned char ucIoMap, unsigned char ucDefValue )`	Section 2.3.1.3
`bool SimpleIOClass::ConfigureMCP2200 (unsigned char IOMap, unsigned long BaudRateParam, unsigned int RxLEDMode, unsigned int TxLEDMode, bool FLOW, bool ULOAD,bool SSPND)`	Section 2.3.1.4
`bool SimpleIOClass::fnHardwareFlowControl (unsigned int onOff)`	Section 2.3.1.5

Note 1: Prior to any DLL API usage, a call to the `InitMCP2200()` function is needed. This function is the only initialization function in the presented DLL.

2: The configuration only needs to be set a single time – it is stored in NVM.

TABLE 2-2: CONFIGURATION FUNCTIONS (CONTINUED)

Category and Function Name	
Summary (Continued)	
`bool SimpleIOClass::fnRxLED (unsigned int mode)`	Section 2.3.1.6
`bool SimpleIOClass::fnSetBaudRate (unsigned long BaudRateParam)`	Section 2.3.1.7
`bool SimpleIOClass::fnSuspend(unsigned int onOff)`	Section 2.3.1.8
`bool SimpleIOClass::fnTxLED (unsigned int mode)`	Section 2.3.1.9
`bool SimpleIOClass::fnULoad(unsigned int onOff)`	Section 2.3.1.10
`String^ SimpleIOClass::GetDeviceInfo(unsigned int uiDeviceNo)`	Section 2.3.1.11
`unsigned int SimpleIOClass::GetNoOfDevices(void)`	Section 2.3.1.12
`int SimpleIOClass::GetSelectedDevice(void)`	Section 2.3.1.13
`String^ SimpleIOClass::GetSelectedDeviceInfo(void)`	Section 2.3.1.14
`void SimpleIOClass::InitMCP2200 (unsigned int VendorID, unsigned int ProductID)`	Section 2.3.1.15
`bool SimpleIOClass::IsConnected()`	Section 2.3.1.16
`int SimpleIOClass::ReadEEPROM(unsigned int uiEEPAddress)`	Section 2.3.1.17
`bool SimpleIOClass::ReadPin(unsigned int pin, unsigned int *returnvalue)`	Section 2.3.1.18
`int SimpleIOClass::ReadPinValue(unsigned int pin)`	Section 2.3.1.19
`bool SimpleIOClass::ReadPort(unsigned int *returnvalue)`	Section 2.3.1.20
`int SimpleIOClass::ReadPortValue()`	Section 2.3.1.21
`int SimpleIOClass::SelectDevice(unsigned int uiDeviceNo)`	Section 2.3.1.22
`bool SimpleIOClass::SetPin(unsigned int pin)`	Section 2.3.1.23
`int SimpleIOClass::WriteEEPROM(unsigned int uiEEPAddress, unsigned char ucValue)`	Section 2.3.1.24
`bool SimpleIOClass::WritePort(unsigned int portValue)`	Section 2.3.1.25
Constants	
`const unsigned int OFF = 0;`	
`const unsigned int ON = 1;`	
`const unsigned int TOGGLE = 3;`	
`const unsigned int BLINKSLOW = 4;`	
`const unsigned int BLINKFAST = 5;`	

Note 1: Prior to any DLL API usage, a call to the `InitMCP2200()` function is needed. This function is the only initialization function in the presented DLL.

2: The configuration only needs to be set a single time – it is stored in NVM.

2.3.1.1 ClearPin

Function:
`bool SimpleIOClass::ClearPin (unsigned int pin)`

Summary:	Clears the specified pin.
Description:	Clears the specified pin to logic ‘`0`’.
Precondition:	This pin must be previously configured as an output via a `ConfigureIO` or `ConfigureIoDefaultOutput` call.
Parameters:	`pin` - The pin number to set (0-7).
Returns:	This function returns True if the transmission is successful and returns False if the transmission fails.
Remarks:	None

EXAMPLE 2-1:

```
if (SimpleIOClass::ClearPin (2))
    {
        lblStatusBar->Text = “Success”;
    }
    else
        lblStatusBar->Text = “Invalid command ” +  SimpleIOClass::LastError;
```

2.3.1.2 ConfigureIO

Function:

`bool SimpleIOClass::ConfigureIO (unsigned char IOMap)`

Summary:	Configures the GPIO pins for Digital Input or Digital Output.
Description:	GPIO Pins can be configured as Digital Input or Digital Output.
Precondition:	VID and PID must be previously set via a call to `InitMCP2200(VID, PID)`.

Parameters:

`IOMap` - a byte that represents a bitmap of the GPIO configuration:

- a bit set to '`1`' will be a digital input
- a bit set to '`0`' will be a digital output
- MSB – – – – – – LSB
  GP7 GP6 GP5 GP4 GP3 GP2 GP1 GP0

Returns:	This function returns True if the transmission is successful and returns False if the transmission fails.
Remarks:	Error code is returned in `LastError`.

EXAMPLE 2-2:

```
    if (SimpleIOClass::ConfigureIO(0xA5) == SUCCESS)
        lblStatusBar->Text = "Success";
    else
        lblStatusBar->Text = "Invalid command " + SimpleIOClass::LastError;
```

2.3.1.3 ConfigureIODefaultOutput

Function:

`bool SimpleIOClass::ConfigureIoDefaultOutput (unsigned char ucIoMap, unsigned char ucDefValue)`

Summary:	Configures the IO pins for Digital Input, Digital Output and also the default output latch value.
Description:	IO Pins can be configured as Digital Input or Digital Output. The default output latch value is received as a parameter.
Precondition:	VID and PID must be previously set via a call to `InitMCP2200(VID, PID)`.

Parameters:

1. `ucIoMap` - a byte that represents a bitmap used to set the GPIOs as either input or output.
   •'`1`' configures GPIO as input
   •'`0`' configures GPIO as output
   •MSB – – – – – – LSB
   GP7 GP6 GP5 GP4 GP3 GP2 GP1 GP0
2. `ucDefValue` - the default value that will be loaded to the output latch (affects only the pins configured as outputs).

Returns:	This function returns True if the transmission is successful and returns False if the transmission fails.
Remarks:	Error code is returned in `LastError`.

EXAMPLE 2-3:

```
if (SimpleIOClass::ConfigureIoDefaultOutput(IoMap, DefValue) == SUCCESS)
        lblStatusBar->Text = "Success";
    else
        lblStatusBar->Text = "Invalid command " + SimpleIOClass::LastError;
```

2.3.1.4 ConfigureMCP2200

Function:

`bool SimpleIOClass::ConfigureIoDefaultOutput (unsigned long BaudRateParam, unsigned int RxLEDMode, unsigned int TxLEDMode, bool FLOW, bool ULOAD, bool SSPND)`

Summary: Configures the device.

Description: Sets the default GPIO designation, baud rate, TX/RX LED modes, flow control.

Precondition: VID and PID must be previously set via a call to `InitMCP2200(VID, PID)`.

Parameters:

1. `IOMap` - A byte that represents the input/output state of the pins (each bit may be either a '`1`' for input or '`0`' for output.
2. `BaudRateParam` - the default communication baud rate.
3. `RxLEDMode` - can take one of the constant values (`OFF`, `ON`, `TOGGLE`, `BLINKSLOW`, `BLINKFAST`) to define the behavior of the Rx LED.
   - `OFF` = 0
   - `ON` = 1
   - `TOGGLE` = 3
   - `BLINKSLOW` = 4
   - `BLINKFAST` = 5
4. `TxLEDMode` - can take one of the defined values (`OFF`, `ON`, `TOGGLE`, `BLINKSLOW`, `BLINKFAST`) in order to define the behavior of the Tx LED.
5. `FLOW` - this parameter establishes the default flow control method (False - no HW flow control, True - $\overline{\text{RTS}}$/$\overline{\text{CTS}}$ flow control).
6. `ULOAD` - this parameter establishes when the USB has loaded the configuration.
7. `SSPND` - this parameter establishes when the USB sends the Suspend mode signal.

Returns: This function returns True if the transmission is successful and returns False if the transmission fails.

Remarks: None.

EXAMPLE 2-4:

```
    if (SimpleIOClass::ConfigureMCP2200(0x43, 9600, BLINKSLOW, BLINKFAST, false, false, false)
== SUCCESS)
        lblStatusBar->Text = "Success";
    else
        lblStatusBar->Text = "Invalid command "
```

2.3.1.5 fnHardwareFlowControl

Function:

`bool SimpleIOClass::fnHardwareFlowControl (unsigned int onOff)`

Summary: Configures the flow control of the MCP2200. The flow control configuration will be stored in NVRAM.

Description: Sets the flow control to HW flow control ($\overline{\text{RTS}}$/$\overline{\text{CTS}}$) or no flow control.

Precondition: VID and PID must be previously set via a call to `InitMCP2200(VID, PID)`.

Parameters: `onOff`:
- '`1`' if HW flow control is required
- '`0`' if no flow control is required

Returns: This function returns True if the transmission is successful and returns False if the transmission fails.

Remarks: Error code is returned in `LastError`.

EXAMPLE 2-5:

```
if (SimpleIOClass::fnHardwareFlowControl(1) == SUCCESS)
        lblStatusBar->Text = "Success";
    else
        lblStatusBar->Text = "Invalid command " + SimpleIOClass::LastError;
```

2.3.1.6 fnRxLED

Function:
`bool SimpleIOClass::fnRxLED (unsigned int mode)`

Summary:	Configures the Rx LED mode. Rx LED configuration will be stored in NVRAM.
Description:	Sets the Rx LED mode to one of the possible values.
Precondition:	VID and PID must be previously set via a call to `InitMCP2200(VID, PID)`.
Parameters:	`mode (constant)`: `OFF`, `TOGGLE`, `BLINKSLOW`, `BLINKFAST`
Returns:	This function returns True if the transmission is successful and returns False if the transmission fails.
Remarks:	Error code is returned in `LastError`.

EXAMPLE 2-6:

```
if (SimpleIOClass::fnRxLED (BLINKFAST) == SUCCESS)
        lblStatusBar->Text = “Success”;
    else
        lblStatusBar->Text = “Invalid command ” + SimpleIOClass::LastError;
```

2.3.1.7 fnSetBaudRate

Function:
`bool SimpleIOClass::fnSetBaudRate (unsigned long BaudRateParam)`

Summary:	Configures the device’s default baud rate. The baud rate value will be stored in NVRAM.
Description:	Sets the desired baud rate and it stores it into the device’s NVRAM.
Precondition:	VID and PID must be previously set via a call to `InitMCP2200(VID, PID)`.
Parameters:	`BaudRateParam` - the desired baud rate value
Returns:	This function returns True if the transmission is successful and returns False if the transmission fails.
Remarks:	Error code is returned in `LastError`. This function is used only to set the default power-up baud rate value. When used with a terminal program, there is no need to call this function to change the baud rate. Changing the baud rate from the terminal program will send the appropriate CDC packet that will change the communication’s baud rate without the need to call this function.

EXAMPLE 2-7:

```
    if (SimpleIOClass::fnSetBaudRate(9600) == SUCCESS)
        lblStatusBar->Text = “Success”;
    else
        lblStatusBar->Text = “Invalid command ” + SimpleIOClass::LastError;
```

2.3.1.8 fnSuspend

Function:
`bool SimpleIOClass::fnSuspend (unsigned int onOff)`

Summary:	Configures the GP0 pin of the MCP2200 to show the status of the USB Suspend/Resume states.
Description:	When the GP0 is designated to show the USB Suspend/Resume states, the pin will go ‘low’ when the Suspend state is issued, or will go ‘high’ when the Resume state is on.
Precondition:	VID and PID must be previously set via a call to `InitMCP2200(VID, PID)`.
Parameters:	`onOff`: • ‘`1`’ GP0 will reflect the USB Suspend/Resume states • ‘`0`’ GP0 will not reflect the USB Suspend/Resume states (can be used as GPIO)
Returns:	This function returns True if the transmission is successful and returns False if the transmission fails.
Remarks:	Error code is returned in `LastError`.

EXAMPLE 2-8:

```
if (SimpleIOClass::fnSuspend(1) == SUCCESS)
        lblStatusBar->Text = “Success”;
    else
        lblStatusBar->Text = “Invalid command ” + SimpleIOClass::LastError;
```

2.3.1.9 fnTxLED

Function:
`bool SimpleIOClass::fnTxLED (unsigned int mode)`

Summary:	Configures the Tx LED mode. Tx LED configuration will be stored in NVRAM.
Description:	Sets the Tx LED mode to one of the possible values.
Precondition:	VID and PID must be previously set via a call to `InitMCP2200(VID, PID)`.
Parameters:	`mode (constant)`: `OFF`, `TOGGLE`, `BLINKSLOW`, `BLINKFAST`
Returns:	This function returns True if the transmission is successful and returns False if the transmission fails.
Remarks:	Error code is returned in `LastError`.

EXAMPLE 2-9:

```
if (SimpleIOClass::fnTxLED (BLINKSLOW) == SUCCESS)
        lblStatusBar->Text = “Success”;
    else
        lblStatusBar->Text = “Invalid command ” + SimpleIOClass::LastError;
```

2.3.1.10 fnULoad

Function:
`bool SimpleIOClass::fnULoad (unsigned int onOff)`

Summary:	Configures the GP1 pin of the MCP2200 to show the configuration status of the USB.
Description:	When the GP1 is designated to show the USB configuration status, the pin will start ‘low’ (during power-up or after Reset), and it will go ‘high’ after the MCP2200 is successfully configured by the host.
Precondition:	VID and PID must be previously set via a call to `InitMCP2200(VID, PID)`.
Parameters:	`onOff`: • ‘`1`’ GP1 will reflect the USB configuration status • ‘`0`’ GP1 will not reflect the USB configuration status (can be used as GPIO)
Returns:	This function returns True if the transmission is successful and returns False if the transmission fails.
Remarks:	Error code is returned in `LastError`.

EXAMPLE 2-10:

```
    if (SimpleIOClass::fnULoad(1) == SUCCESS)
        lblStatusBar->Text = “Success”;
    else
        lblStatusBar->Text = “Invalid command ” + SimpleIOClass::LastError;
```

2.3.1.11 GetDeviceInfo

Function:
`String^ SimpleIOClass::GetDeviceInfo (unsigned int uiDeviceNo)`

Summary:	Returns the path name for one of the connected devices.
Description:	The function will return the path name for the given device ID.
Precondition:	At least one call to the `InitMCP2200()` is required in order to initiate a DLL search for the compatible devices. VID and PID must be previously set via a call to `InitMCP2200(VID, PID)`.
Parameters:	`uiDeviceNo`: The device ID for which the path information is needed. Can have a value between 0 and the number of devices minus 1.
Returns:	This function returns a string containing the path name of the given device id. • In the case the given ID is out of range, the function will return the "`Device Index Error`" string. • In the case the device for which the path name is required is not connected anymore, the return string will be "`Device Not Connected`".
Remarks:	None.

EXAMPLE 2-11:

```
lblStatusBar->Text = SimpleIOClass::GetDeviceInfo(0);
```

2.3.1.12 GetNoOfDevices

Function:
`unsigned int SimpleIOClass::GetNoOfDevices(void)`

Summary:	The function returns the number of available devices present in the system.
Description:	The function returns the number of HID devices (with the given VID/PID) connected to the system.
Precondition:	At least one call to the `InitMCP2200()` is required in order to initiate a DLL search for the compatible devices. Also, in order to know the actual number of devices connected to the system, call the `SimpleIOClass::IsConnected()` function. VID and PID must be previously set via a call to `InitMCP2200(VID, PID)`.
Parameters:	None.
Returns:	This function returns the number of HID devices with the given VID/PID (as parameters of the `SimpleIOClass::InitMCP2200()` function).
Remarks:	Call the `SimpleIOClass::IsConnected()` function prior to the call of this function in order to have the most recent number of devices that are present in the system.

EXAMPLE 2-12:

```
SimpleIOClass::IsConnected(); //call this function to refresh the number of
//the devices present in the system
lblStatusBar->Text = SimpleIOClass::GetNoOfDevices();
```

2.3.1.13 GetSelectedDevice

Function:
`int SimpleIOClass::GetSelectedDevice(void)`

Summary:	Returns the ID of the selected device.
Description:	The function returns the ID of the current selected device.
Precondition:	At least one call to the `InitMCP2200()` is required in order to initiate a DLL search for the compatible devices. VID and PID must be previously set via a call to `InitMCP2200(VID, PID)`.
Parameters:	None.
Returns:	This function returns the ID of the current selected device. Its value can range from 0 to the number of devices minus 1.
Remarks:	None.

EXAMPLE 2-13:

```
lblStatusBar->Text = SimpleIOClass::GetSelectedDevice();
```

2.3.1.14 GetSelectedDeviceInfo

Function:
`String^ SimpleIOClass::GetSelectedDeviceInfo(void)`

Summary:	Returns the selected device path name.
Description:	The function returns a string containing the unique path name of the selected device.
Precondition:	At least one call to the `InitMCP2200()` is required in order to initiate a DLL search for the compatible devices. VID and PID must be previously set via a call to `InitMCP2200(VID, PID)`.
Parameters:	None.
Returns:	This function returns a string containing the unique path name of the selected device.
Remarks:	The default selected device is the first one that the DLL finds. If the user wants to retrieve other devices path names (assuming more than one device is present in the system), a call to `SimpleIOClass::SelectDevice(deviceNo)` is required.

EXAMPLE 2-14:

```
    lblStatusBar->Text = SimpleIOClass::GetSelectedDeviceInfo(void)
```

2.3.1.15 InitMCP2200

Function:
`void SimpleIOClass::InitMCP2200 (unsigned int VendorID, unsigned int ProductID)`

Summary:	Configures the Simple IO class for a specific Vendor and Product ID.
Description:	Sets the Vendor and Product ID used for the project.
Precondition:	None.
Parameters:	1. `Vendor ID` - assigned by USB IF (www.usb.org) 2. `Product ID` - assigned by the Vendor ID Holder
Returns:	None.
Remarks:	Call this function before any other calls, to set the Vendor and Product IDs.

EXAMPLE 2-15:

```
  InitMCP2200 (0x4D8, 0x00DF);
```

2.3.1.16 IsConnected

Function:
`bool SimpleIOClass::IsConnected()`

Summary:	Checks with the OS if the current VID/PID device is connected.
Description:	Checks if a MCP2200 device is connected to the computer. If so, it returns True; otherwise, the result will be False.
Precondition:	VID and PID must be previously set via a call to `InitMCP2200(VID, PID)`.
Parameters:	None.
Returns:	True - if at least one device is connected to the host. False - if there are no devices connected to the host.
Remarks:	No actual communication with the end device is occurring. The function inquires the OS if the specified VID/PID was enumerated.

EXAMPLE 2-16:

```
    unsigned int rv;
    if (SimpleIOClass::IsConnected ())
    {
        lblStatusBar->Text = “Device connected”;
    }
    else
        lblStatusBar->Text = “Device Disconnected”;
```

2.3.1.17 ReadEEPROM

Function:
`int SimpleIOClass::ReadEEPROM (unsigned int uiEEPAddress)`

Summary:	Reads a byte from the EEPROM.
Description:	Reads a byte from the EEPROM at the given address.
Precondition:	At least one call to the `InitMCP2200()` is required in order to initiate a DLL search for the compatible devices. VID and PID must be previously set via a call to `InitMCP2200(VID, PID)`.
Parameters:	`uiEEPAddress` - the EEPROM address location we need to write to (must be from 0 to 255, inclusively).
Returns:	This function returns any positive value as being the EEPROM’s location value: • `E_WRONG_ADDRESS (-3)` - in case the given EEPROM address is out of range • `E_CANNOT_SEND_DATA (-4)` - in case the function cannot send the command to the device
Remarks:	None.

EXAMPLE 2-17:

```
int iRetValue = SimpleIOClass::ReadEEPROM(0x01);
    if (iRetValue >= 0)
    {
        lblStatusBar->Text = “Success”;
    }
    else
        lblStatusBar->Text = “Error reading to EEPROM” +  SimpleIOClass::LastError;
```

2.3.1.18 ReadPin

Function:
`bool SimpleIOClass::ReadPin (unsigned int pin, unsigned int *returnvalue)`

Summary:	Reads the specified pin.
Description:	Reads the specified pin and returns the value in `returnvalue`. If the pin has been configured as a digital input, the return value will be either ‘`0`’ or ‘`1`’.
Precondition:	Must be previously configured as an input via a `ConfigureIO` call. VID and PID must be previously set via a call to `InitMCP2200(VID, PID)`.
Parameters:	• `pin` - the pin number to set (0-7) • `returnvalue` - the value read on the pin (‘`0`’ or ‘`1`’)
Returns:	This function returns True if the transmission is successful and returns False if the transmission fails.
Remarks:	None.

EXAMPLE 2-18:

```
unsigned int rv;
    if (SimpleIOClass::ReadGPIOn (0, &rv))
    {
        lblStatusBar->Text = “Success”;
    }
    else
        lblStatusBar->Text = “Invalid command ” +  SimpleIOClass::LastError;
```

2.3.1.19 ReadPinValue

Function:

`int SimpleIOClass::ReadPinValue(unsigned int pin)`

Summary:	Reads the specified pin.
Description:	Reads the specified pin and returns the value as the return value. If the pin has been configured as a digital input, the return value will be either '`0`' or '`1`'. If an error occurs, the function will return a value of 0x8000.
Precondition:	Must be previously configured as an input via a `ConfigureIO` call. VID and PID must be previously set via a call to `InitMCP2200(VID, PID).`
Parameters:	`pin` - the pin number to set (0-7)
Returns:	This function returns the read value of the pin, or returns a value of 0x8000, if an error occurs.
Remarks:	None.

EXAMPLE 2-19:

```
    unsigned int rv;
    if (SimpleIOClass::ReadPinValue(0) != 0x8000)
    {
        lblStatusBar->Text = "Success";
    }
    else
        lblStatusBar->Text = "Invalid command " +  SimpleIOClass::LastError;
```

2.3.1.20 ReadPort

Function:

`bool SimpleIOClass::ReadPort(unsigned int *returnvalue)`

Summary:	Reads the GPIO port as digital input.
Description:	Reads the GPIO port and returns the value in `returnvalue`. This provides a means to read all pins simultaneously, instead of one-by-one.
Precondition:	Must be previously configured as an input via a `ConfigureIO` call. VID and PID must be previously set via a call to `InitMCP2200(VID, PID).`
Parameters:	• `pin` - the pin number to set (0-7) • `returnvalue` - the value read on the pin ('`0`' or '`1`')
Returns:	This function returns True if the transmission is successful and returns False if the transmission fails.
Remarks:	Pins configured for output return the current state of the port. Pins configured as input read as zero.

EXAMPLE 2-20:

```
unsigned int rv;
    if (SimpleIOClass::ReadGPIOPort (0, &rv))
    {
        lblStatusBar->Text = "Success";
    }
    else
        lblStatusBar->Text = "Invalid command " +  SimpleIOClass::LastError;
```

2.3.1.21 ReadPortValue

Function:
`int SimpleIOClass::ReadPortValue()`

Summary:	Reads the GPIO port as digital input.
Description:	Reads the GPIO port and returns the value of the port. This provides a method to read all pins simultaneously, instead of one-by-one. In case of an error, the returned value will be 0x8000.
Precondition:	Must be previously configured as an input via a `ConfigureIO` call. VID and PID must be previously set via a call to `InitMCP2200(VID, PID).`
Parameters:	None.
Returns:	This function returns True if the transmission is successful and returns False if the transmission fails.
Remarks:	Pins configured for output return the current state of the port. Pins configured as input read as zero.

EXAMPLE 2-21:

```
    int rv;
    rv = SimpleIOClass::ReadPortValue()
    if (rv != 0x8000)
    {
        lblStatusBar->Text = “Success”;
    }
    else
        lblStatusBar->Text = “Invalid command ” +  SimpleIOClass::LastError;
```

2.3.1.22 SelectDevice

Function:
`int SimpleIOClass::SelectDevice(unsigned int uiDeviceNo)`

Summary:	Selects one of the active devices in the system.
Description:	The function is used to select one of the detected devices in the system as the “active device”.
Precondition:	At least one call to the `InitMCP2200()` is required in order to initiate a DLL search for the compatible devices. Also, in order to know the actual number of devices in the system, call the `SimpleIOClass::IsConnected()` function. VID and PID must be previously set via a call to `InitMCP2200(VID, PID).`
Parameters:	`uiDeviceNo` - the ID of the device to be selected (can have a value between 0 and the number of devices minus 1).
Returns:	This function returns ‘`0`’ in case of selection success, otherwise it will return: • `E_WRONG_DEVICE_ID (-1)` for a device ID that is out of range • `E_INACTIVE_DEVICE (-2)` for an inactive device.
Remarks:	Call the `SimpleIOClass::IsConnected()` prior to the call of this function in order to have the most recent number of devices that are present in the system.

EXAMPLE 2-22:

```
int iResult;
    iResult = SimpleIOClass::SelectDevice(1)
    if (iResult == 0)
    {
        lblStatusBar->Text = “Success”;
    }
    else
        lblStatusBar->Text = “Error selecting device”;
```

2.3.1.23 SetPin

Function:
`bool SimpleIOClass::SetPin(unsigned int pin)`

Summary:	Sets the specified pin.
Description:	Sets the specified pin to logic '1'.
Precondition:	Must be previously configured as an output via a `ConfigureIO` or `ConfigureIoDefaultOutput` call. VID and PID must be previously set via a call to `InitMCP2200(VID, PID)`.
Parameters:	`pin` - the pin number to set (0-7)
Returns:	This function returns True if the transmission is successful and returns False if the transmission fails.
Remarks:	None.

EXAMPLE 2-23:

```
    if (SimpleIOClass::SetPin (2))
    {
        lblStatusBar->Text = "Success";
    }
    else
        lblStatusBar->Text = "Invalid command " +  SimpleIOClass::LastError;
```

2.3.1.24 WriteEEPROM

Function:
`int SimpleIOClass::WriteEEPROM(unsigned int uiEEPAddress, unsigned char ucValue)`

Summary:	Writes a byte into the MCP2200 device's EEPROM.
Description:	Writes a byte at the given address into the internal 256 bytes EEPROM.
Precondition:	At least one call to the `InitMCP2200()` is required in order to initiate a DLL search for the compatible devices. VID and PID must be previously set via a call to `InitMCP2200(VID, PID)`.
Parameters:	• `uiEEPAddress` - the EEPROM address location to write to (must be from 0 to 255 inclusively). • `ucValue` - the byte value required for writing to the given location.
Returns:	This function returns '0' if the write command was successfully sent to the device, otherwise it returns: • `E_WRONG_ADDRESS (-3)` in case the given EEPROM address is out of range • `E_CANNOT_SEND_DATA (-4)` in case the function cannot send the command to the device.
Remarks:	The function will send the write EEPROM command, but has no confirmation whether the EEPROM location was actually written. In order to verify the correctness of the EEPROM write, the user can issue a `SimpleIOClass::ReadEEPROM()` and check if the returned value matches the written one.

EXAMPLE 2-24:

```
int iRetValue = SimpleIOClass::WriteEEPROM(0x01, 0xAB);

    if (iRetValue == 0)
    {
        lblStatusBar->Text = "Success";
    }
    else
        lblStatusBar->Text = "Error writting to EEPROM" +  SimpleIOClass::LastError;
```

2.3.1.25 WritePort

Function:

`bool SimpleIOClass::WritePort(unsigned int portValue)`

Summary:	Writes a value to the GPIO port.
Description:	Writes the GPIO port. This provides a means to write all pins simultaneously, instead of one-by-one.
Precondition:	Must be previously configured as an output via a `ConfigureIO` call. VID and PID must be previously set via a call to `InitMCP2200(VID, PID)`.
Parameters:	`portValue` - byte value to set on the port.
Returns:	This function returns True if the transmission is successful and returns False if the transmission fails.
Remarks:	None.

EXAMPLE 2-25:

```
if (SimpleIOClass::WritePort (0x5A))
    {
        lblStatusBar->Text = "Success";
    }
    else
        lblStatusBar->Text = "Invalid command " +  SimpleIOClass::LastError;
```

NOTES: