# AdafruitMCP230xx Library Documentation

*Release 1.0*

**Tony DiCola**

**Aug 03, 2021**

# Contents

CircuitPython module for the MCP23017/08 I2C and MCP23S17/08 SPI I/O extenders.

CHAPTER 1

# Dependencies

This driver depends on:

- Adafruit CircuitPython
- Bus Device

Please ensure all dependencies are available on the CircuitPython filesystem. This is easily achieved by downloading the Adafruit library and driver bundle.

CHAPTER 2

# Installing from PyPI

On supported GNU/Linux systems like the Raspberry Pi, you can install the driver locally from PyPI. To install for current user:

```
pip3 install adafruit-circuitpython-mcp230xx
```

To install system-wide (this may be required in some cases):

```
sudo pip3 install adafruit-circuitpython-mcp230xx
```

To install in a virtual environment in your current project:

```
mkdir project-name && cd project-name
python3 -m venv .env
source .env/bin/activate
pip3 install adafruit-circuitpython-mcp230xx
```

CHAPTER 3

# Usage Example

See examples/mcp230xx_simpletest.py for a demo of the usage.

CHAPTER 4

# Contributing

Contributions are welcome! Please read our Code of Conduct before contributing to help this project stay welcoming.

CHAPTER 5

# Documentation

For information on building library documentation, please check out this guide.

CHAPTER 6

# Table of Contents

## 6.1 Simple test

Ensure your device works with this simple test.

Listing 1: examples/mcp230xx_simpletest.py

```
# SPDX-FileCopyrightText: 2017 Tony DiCola for Adafruit Industries
#
# SPDX-License-Identifier: MIT

# Simple demo of reading and writing the digital I/O of the MCP2300xx as if
# they were native CircuitPython digital inputs/outputs.
# Author: Tony DiCola
import time

import board
import busio
import digitalio

from adafruit_mcp230xx.mcp23008 import MCP23008

# from adafruit_mcp230xx.mcp23017 import MCP23017


# Initialize the I2C bus:
i2c = busio.I2C(board.SCL, board.SDA)

# Create an instance of either the MCP23008 or MCP23017 class depending on
# which chip you're using:
mcp = MCP23008(i2c)  # MCP23008
# mcp = MCP23017(i2c)  # MCP23017

# Optionally change the address of the device if you set any of the A0, A1, A2
```

(continues on next page)

(continued from previous page)

```
# pins.  Specify the new address with a keyword parameter:
# mcp = MCP23017(i2c, address=0x21)  # MCP23017 w/ A0 set

# Now call the get_pin function to get an instance of a pin on the chip.
# This instance will act just like a digitalio.DigitalInOut class instance
# and has all the same properties and methods (except you can't set pull-down
# resistors, only pull-up!).  For the MCP23008 you specify a pin number from 0
# to 7 for the GP0...GP7 pins.  For the MCP23017 you specify a pin number from
# 0 to 15 for the GPIOA0...GPIOA7, GPIOB0...GPIOB7 pins (i.e. pin 12 is GPIOB4).
pin0 = mcp.get_pin(0)
pin1 = mcp.get_pin(1)

# Setup pin0 as an output that's at a high logic level.
pin0.switch_to_output(value=True)

# Setup pin1 as an input with a pull-up resistor enabled.  Notice you can also
# use properties to change this state.
pin1.direction = digitalio.Direction.INPUT
pin1.pull = digitalio.Pull.UP

# Now loop blinking the pin 0 output and reading the state of pin 1 input.
while True:
    # Blink pin 0 on and then off.
    pin0.value = True
    time.sleep(0.5)
    pin0.value = False
    time.sleep(0.5)
    # Read pin 1 and print its state.
    print("Pin 1 is at a high level: {0}".format(pin1.value))
```

Listing 2: examples/mcp23Sxx_simpletest.py

```
# SPDX-FileCopyrightText: 2017 Tony DiCola for Adafruit Industries
# SPDX-FileCopyrightText: 2021 Red_M
#
# SPDX-License-Identifier: MIT

# Simple demo of reading and writing the digital I/O of the MCP2300xx as if
# they were native CircuitPython digital inputs/outputs.
# Author: Tony DiCola
import time

import board
import busio
import digitalio

from adafruit_mcp230xx.mcp23s08 import MCP23S08

# from adafruit_mcp230xx.mcp23s17 import MCP23S17


# Initialize the SPI bus with a chip select pin:
spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)
cs = digitalio.DigitalInOut(board.A1)

# Create an instance of either the MCP23S08 or MCP23S17 class depending on
```

(continues on next page)

(continued from previous page)

```
# which chip you're using:
mcp = MCP23S08(spi, cs)  # MCP23S08
# mcp = MCP23S17(spi, cs)  # MCP23S17

# Optionally change the address of the device if you set any of the A0, A1, A2
# pins.  Specify the new address with a keyword parameter:
# mcp = MCP23S17(spi, cs, address=0x21)  # MCP23S17 w/ A0 set

# Now call the get_pin function to get an instance of a pin on the chip.
# This instance will act just like a digitalio.DigitalInOut class instance
# and has all the same properties and methods (except you can't set pull-down
# resistors, only pull-up!).  For the MCP23S08 you specify a pin number from 0
# to 7 for the GP0...GP7 pins.  For the MCP23S17 you specify a pin number from
# 0 to 15 for the GPIOA0...GPIOA7, GPIOB0...GPIOB7 pins (i.e. pin 12 is GPIOB4).
pin0 = mcp.get_pin(0)
pin1 = mcp.get_pin(1)

# Setup pin0 as an output that's at a high logic level.
pin0.switch_to_output(value=True)

# Setup pin1 as an input with a pull-up resistor enabled.  Notice you can also
# use properties to change this state.
pin1.direction = digitalio.Direction.INPUT
pin1.pull = digitalio.Pull.UP

# Now loop blinking the pin 0 output and reading the state of pin 1 input.
while True:
    # Blink pin 0 on and then off.
    pin0.value = True
    time.sleep(0.5)
    pin0.value = False
    time.sleep(0.5)
    # Read pin 1 and print its state.
    print("Pin 1 is at a high level: {0}".format(pin1.value))
```

## 6.2 mcp230xx

CircuitPython module for the MCP23017 and MCP23008 I2C I/O extenders.

- Author(s): Tony DiCola, Red_M (2021)

**class** adafruit_mcp230xx.mcp230xx.**MCP230XX**(*bus_device*, *address*, *chip_select=None*, *baudrate=100000*)

Base class for MCP230xx devices.

## 6.3 mcp23008

CircuitPython module for the MCP23008 I2C I/O extenders.

- Author(s): Tony DiCola

**class** adafruit_mcp230xx.mcp23008.**MCP23008**(*i2c*, *address=<sphinx.ext.autodoc.importer._MockObject object>*, *reset=True*)

Supports MCP23008 instance on specified I2C bus and optionally at the specified I2C address.

**get_pin**(*pin*)
Convenience function to create an instance of the DigitalInOut class pointing at the specified pin of this MCP23008 device.

**gpio**
The raw GPIO output register. Each bit represents the output value of the associated pin (0 = low, 1 = high), assuming that pin has been configured as an output previously.

**gppu**
The raw GPPU pull-up register. Each bit represents if a pull-up is enabled on the specified pin (1 = pull-up enabled, 0 = pull-up disabled). Note pull-down resistors are NOT supported!

**iodir**
The raw IODIR direction register. Each bit represents direction of a pin, either 1 for an input or 0 for an output mode.

## 6.4 mcp23017

CircuitPython module for the MCP23017 I2C I/O extenders.

- Author(s): Tony DiCola

**class** adafruit_mcp230xx.mcp23017.**MCP23017**(*i2c*, *address=<sphinx.ext.autodoc.importer._MockObject object>*, *reset=True*)
Supports MCP23017 instance on specified I2C bus and optionally at the specified I2C address.

**clear_inta**()
Clears port A interrupts.

**clear_intb**()
Clears port B interrupts.

**clear_ints**()
Clears interrupts by reading INTCAP.

**default_value**
The raw DEFVAL interrupt control register. The default comparison value is configured in the DEFVAL register. If enabled (via GPINTEN and INTCON) to compare against the DEFVAL register, an opposite value on the associated pin will cause an interrupt to occur.

**get_pin**(*pin*)
Convenience function to create an instance of the DigitalInOut class pointing at the specified pin of this MCP23017 device.

**gpio**
The raw GPIO output register. Each bit represents the output value of the associated pin (0 = low, 1 = high), assuming that pin has been configured as an output previously.

**gpioa**
The raw GPIO A output register. Each bit represents the output value of the associated pin (0 = low, 1 = high), assuming that pin has been configured as an output previously.

**gpiob**
The raw GPIO B output register. Each bit represents the output value of the associated pin (0 = low, 1 = high), assuming that pin has been configured as an output previously.

**gppu**
The raw GPPU pull-up register. Each bit represents if a pull-up is enabled on the specified pin (1 = pull-up enabled, 0 = pull-up disabled). Note pull-down resistors are NOT supported!

**gppua**
The raw GPPU A pull-up register. Each bit represents if a pull-up is enabled on the specified pin (1 = pull-up enabled, 0 = pull-up disabled). Note pull-down resistors are NOT supported!

**gppub**
The raw GPPU B pull-up register. Each bit represents if a pull-up is enabled on the specified pin (1 = pull-up enabled, 0 = pull-up disabled). Note pull-down resistors are NOT supported!

**int_cap**
Returns a list with the pin values at time of interrupt port A —-> pins 0-7 port B —-> pins 8-15

**int_capa**
Returns a list of pin values at time of interrupt pins: 0-7

**int_capb**
Returns a list of pin values at time of interrupt pins: 8-15

**int_flag**
Returns a list with the pin numbers that caused an interrupt port A —-> pins 0-7 port B —-> pins 8-15

**int_flaga**
Returns a list of pin numbers that caused an interrupt in port A pins: 0-7

**int_flagb**
Returns a list of pin numbers that caused an interrupt in port B pins: 8-15

**interrupt_configuration**
The raw INTCON interrupt control register. The INTCON register controls how the associated pin value is compared for the interrupt-on-change feature. If a bit is set, the corresponding I/O pin is compared against the associated bit in the DEFVAL register. If a bit value is clear, the corresponding I/O pin is compared against the previous value.

**interrupt_enable**
The raw GPINTEN interrupt control register. The GPINTEN register controls the interrupt-on-change feature for each pin. If a bit is set, the corresponding pin is enabled for interrupt-on-change. The DEFVAL and INTCON registers must also be configured if any pins are enabled for interrupt-on-change.

**io_control**
The raw IOCON configuration register. Bit 1 controls interrupt polarity (1 = active-high, 0 = active-low). Bit 2 is whether irq pin is open drain (1 = open drain, 0 = push-pull). Bit 3 is unused. Bit 4 is whether SDA slew rate is enabled (1 = yes). Bit 5 is if I2C address pointer auto-increments (1 = no). Bit 6 is whether interrupt pins are internally connected (1 = yes). Bit 7 is whether registers are all in one bank (1 = no), this is silently ignored if set to `1`.

**iodir**
The raw IODIR direction register. Each bit represents direction of a pin, either 1 for an input or 0 for an output mode.

**iodira**
The raw IODIR A direction register. Each bit represents direction of a pin, either 1 for an input or 0 for an output mode.

**iodirb**
The raw IODIR B direction register. Each bit represents direction of a pin, either 1 for an input or 0 for an output mode.

**ipol**
The raw IPOL output register. Each bit represents the polarity value of the associated pin (0 = normal, 1 = inverted), assuming that pin has been configured as an input previously.

**ipola**
The raw IPOL A output register. Each bit represents the polarity value of the associated pin (0 = normal, 1 = inverted), assuming that pin has been configured as an input previously.

**ipolb**
The raw IPOL B output register. Each bit represents the polarity value of the associated pin (0 = normal, 1 = inverted), assuming that pin has been configured as an input previously.

## 6.5 mcp23sxx

CircuitPython module for the MCP23S17 SPI I/O extenders.

- Author(s): Romy Bompart (2020), Red_M (2021)

**class** adafruit_mcp230xx.mcp23sxx.**MCP23SXX**(*spi*, *address*, *chip_select*, *baudrate=100000*)
Base class for MCP23Sxx devices.

## 6.6 MCP23S08

CircuitPython module for the MCP23S08 I2C I/O extenders.

- Author(s): Tony DiCola, Romy Bompart (2020), Red_M (2021)

**class** adafruit_mcp230xx.mcp23s08.**MCP23S08**(*spi*, *chip_select*, *address=<sphinx.ext.autodoc.importer._MockObject object>*, *reset=True*, *baudrate=100000*)
Supports MCP23S08 instance on specified I2C bus and optionally at the specified I2C address.

**get_pin**(*pin*)
Convenience function to create an instance of the DigitalInOut class pointing at the specified pin of this MCP23S08 device.

**gpio**
The raw GPIO output register. Each bit represents the output value of the associated pin (0 = low, 1 = high), assuming that pin has been configured as an output previously.

**gppu**
The raw GPPU pull-up register. Each bit represents if a pull-up is enabled on the specified pin (1 = pull-up enabled, 0 = pull-up disabled). Note pull-down resistors are NOT supported!

**iodir**
The raw IODIR direction register. Each bit represents direction of a pin, either 1 for an input or 0 for an output mode.

## 6.7 MCP23S17

CircuitPython module for the MCP23S17 SPI I/O extenders.

- Author(s): Tony DiCola, Romy Bompart (2020), Red_M (2021)

**class** adafruit_mcp230xx.mcp23s17.**MCP23S17**(*spi*, *chip_select*, *address=<sphinx.ext.autodoc.importer._MockObject object>*, *reset=True*, *baudrate=100000*)
Supports MCP23S17 instance on specified SPI bus and optionally at the specified SPI address.

**clear_inta**()
Clears port A interrupts.

**clear_intb**()
Clears port B interrupts.

**clear_ints**()
Clears interrupts by reading INTCAP.

**default_value**
The raw DEFVAL interrupt control register. The default comparison value is configured in the DEFVAL register. If enabled (via GPINTEN and INTCON) to compare against the DEFVAL register, an opposite value on the associated pin will cause an interrupt to occur.

**get_pin**(*pin*)
Convenience function to create an instance of the DigitalInOut class pointing at the specified pin of this MCP23S17 device.

**gpio**
The raw GPIO output register. Each bit represents the output value of the associated pin (0 = low, 1 = high), assuming that pin has been configured as an output previously.

**gpioa**
The raw GPIO A output register. Each bit represents the output value of the associated pin (0 = low, 1 = high), assuming that pin has been configured as an output previously.

**gpiob**
The raw GPIO B output register. Each bit represents the output value of the associated pin (0 = low, 1 = high), assuming that pin has been configured as an output previously.

**gppu**
The raw GPPU pull-up register. Each bit represents if a pull-up is enabled on the specified pin (1 = pull-up enabled, 0 = pull-up disabled). Note pull-down resistors are NOT supported!

**gppua**
The raw GPPU A pull-up register. Each bit represents if a pull-up is enabled on the specified pin (1 = pull-up enabled, 0 = pull-up disabled). Note pull-down resistors are NOT supported!

**gppub**
The raw GPPU B pull-up register. Each bit represents if a pull-up is enabled on the specified pin (1 = pull-up enabled, 0 = pull-up disabled). Note pull-down resistors are NOT supported!

**int_flag**
Returns a list with the pin numbers that caused an interrupt port A —-> pins 0-7 port B —-> pins 8-15

**int_flaga**
Returns a list of pin numbers that caused an interrupt in port A pins: 0-7

**int_flagb**
Returns a list of pin numbers that caused an interrupt in port B pins: 8-15

**interrupt_configuration**
The raw INTCON interrupt control register. The INTCON register controls how the associated pin value is compared for the interrupt-on-change feature. If a bit is set, the corresponding I/O pin is compared against the associated bit in the DEFVAL register. If a bit value is clear, the corresponding I/O pin is compared against the previous value.

**interrupt_enable**
The raw GPINTEN interrupt control register. The GPINTEN register controls the interrupt-on-change feature for each pin. If a bit is set, the corresponding pin is enabled for interrupt-on-change. The DEFVAL and INTCON registers must also be configured if any pins are enabled for interrupt-on-change.

**io_control**
The raw IOCON configuration register. Bit 1 controls interrupt polarity (1 = active-high, 0 = active-low). Bit 2 is whether irq pin is open drain (1 = open drain, 0 = push-pull). Bit 3 is unused. Bit 4 is whether SDA slew rate is enabled (1 = yes). Bit 5 is if SPI address pointer auto-increments (1 = no). Bit 6 is whether interrupt pins are internally connected (1 = yes). Bit 7 is whether registers are all in one bank (1 = no), this is silently ignored if set to `1`.

**iodir**
The raw IODIR direction register. Each bit represents direction of a pin, either 1 for an input or 0 for an output mode.

**iodira**
The raw IODIR A direction register. Each bit represents direction of a pin, either 1 for an input or 0 for an output mode.

**iodirb**
The raw IODIR B direction register. Each bit represents direction of a pin, either 1 for an input or 0 for an output mode.

**ipol**
The raw IPOL output register. Each bit represents the polarity value of the associated pin (0 = normal, 1 = inverted), assuming that pin has been configured as an input previously.

**ipola**
The raw IPOL A output register. Each bit represents the polarity value of the associated pin (0 = normal, 1 = inverted), assuming that pin has been configured as an input previously.

**ipolb**
The raw IPOL B output register. Each bit represents the polarity value of the associated pin (0 = normal, 1 = inverted), assuming that pin has been configured as an input previously.

## 6.8 digital_inout

Digital input/output of the MCP230xx.

- Author(s): Tony DiCola

**class** adafruit_mcp230xx.digital_inout.**DigitalInOut**(*pin_number*, *mcp230xx*)
Digital input/output of the MCP230xx. The interface is exactly the same as the digitalio.DigitalInOut class, however:

- MCP230xx family does not support pull-down resistors;
- MCP23016 does not support pull-up resistors.

Exceptions will be thrown when attempting to set unsupported pull configurations.

**direction**
The direction of the pin, either True for an input or False for an output.

**invert_polarity**
The polarity of the pin, either True for an Inverted or False for an normal.

**pull**
Enable or disable internal pull-up resistors for this pin. A value of digitalio.Pull.UP will enable a pull-up resistor, and None will disable it. Pull-down resistors are NOT supported!

**switch_to_input**(*pull=None*, *invert_polarity=False*, ***kwargs*)
Switch the pin state to a digital input with the provided starting pull-up resistor state (optional, no pull-up by default) and input polarity. Note that pull-down resistors are NOT supported!

**switch_to_output**(*value=False*, ***kwargs*)
Switch the pin state to a digital output with the provided starting value (True/False for high or low, default is False/low).

**value**
The value of the pin, either True for high or False for low. Note you must configure as an output or input appropriately before reading and writing this value.

CHAPTER 7

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## a

# Index

## M



## P



## S



## V