AdafruitMCP230xx Library Documentation

Release 1.0

Tony DiCola

May 17, 2019

Contents

CircuitPython module for the MCP23017 and MCP23008 I2C 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

Usage Example

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

CHAPTER 3

Contributing

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

CHAPTER 4

Building locally

4.1 Zip release files

To build this library locally you'll need to install the circuitpython-build-tools package.

```
python3 -m venv .env
source .env/bin/activate
pip install circuitpython-build-tools
```

Once installed, make sure you are in the virtual environment:

```
source .env/bin/activate
```

Then run the build:

```
circuitpython-build-bundles --filename_prefix adafruit-circuitpython-mcp230xx --
↪library_location .
```

4.2 Sphinx documentation

Sphinx is used to build the documentation based on rST files and comments in the code. First, install dependencies (feel free to reuse the virtual environment from above):

```
python3 -m venv .env
source .env/bin/activate
pip install Sphinx sphinx-rtd-theme
```

Now, once you have the virtual environment activated:

```
cd docs
sphinx-build -E -W -b html . _build/html
```

This will output the documentation to `docs/_build/html`. Open the index.html in your browser to view them. It will also (due to -W) error out on any warning like Travis will. This is a good way to locally verify it will pass.

CHAPTER 5

Table of Contents

5.1 Simple test

Ensure your device works with this simple test.

Listing 1: examples/mcp230xx_simpletest.py

```
# 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
# 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
```

(continues on next page)

(continued from previous page)

```
# 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))
```

5.2 mcp230xx

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

- Author(s): Tony DiCola

class adafruit_mcp230xx.mcp230xx.**MCP230XX**(*i2c*, *address*)
Base class for MCP230xx devices.

5.3 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>*)
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.

5.4 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>*)
Supports MCP23017 instance on specified I2C bus and optionally at the specified I2C address.

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!

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).

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.

5.5 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 the MCP230xx does not support pull-down resistors and an exception will be thrown attempting to set one).

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

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*, ***kwargs*)
Switch the pin state to a digital input with the provided starting pull-up resistor state (optional, no pull-up by default). 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 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index