class I2C – a two-wire serial protocol

I2C is a two-wire protocol for communicating between devices. At the physical level it consists of 2 wires: SCL and SDA, the clock and data lines respectively.

I2C objects are created attached to a specific bus. They can be initialised when created, or initialised later on.

Printing the I2C object gives you information about its configuration.

Both hardware and software I2C implementations exist via the machine.I2C and `machine.SoftI2C` classes. Hardware I2C uses underlying hardware support of the system to perform the reads/writes and is usually efficient and fast but may have restrictions on which pins can be used. Software I2C is implemented by bit-banging and can be used on any pin but is not as efficient. These classes have the same methods available and differ primarily in the way they are constructed.

Note

The I2C bus requires pull-up circuitry on both SDA and SCL for it's operation. Usually these are resistors in the range of 1 - 10 kOhm, connected from each SDA/SCL to Vcc. Without these, the behaviour is undefined and may range from blocking, unexpected watchdog reset to just wrong values. Often, this pull-up circuitry is built-in already to the MCU board or sensor breakout boards, but there is no rule for that. So please check in case of trouble. See also this excellent learning guide by Adafruit about I2C wiring.

Example usage:

```
from machine import I2C

i2c = I2C(freq=400000)          # create I2C peripheral at frequency of 400kHz
                                # depending on the port, extra parameters may be
required
                                # to select the peripheral and/or pins to use

i2c.scan()                      # scan for peripherals, returning a list of 7-bit
addresses

i2c.writeto(42, b'123')         # write 3 bytes to peripheral with 7-bit address 42
i2c.readfrom(42, 4)             # read 4 bytes from peripheral with 7-bit address 42

i2c.readfrom_mem(42, 8, 3)      # read 3 bytes from memory of peripheral 42,
                                #   starting at memory-address 8 in the peripheral
i2c.writeto_mem(42, 2, b'\x10') # write 1 byte to memory of peripheral 42
                                #   starting at address 2 in the peripheral
```

Constructors

class **machine.I2C**(*id, *, scl, sda, freq=400000, timeout=50000*)

Construct and return a new I2C object using the following parameters:

- *id* identifies a particular I2C peripheral. Allowed values for depend on the particular port/board
- *scl* should be a pin object specifying the pin to use for SCL.
- *sda* should be a pin object specifying the pin to use for SDA.
- *freq* should be an integer which sets the maximum frequency for SCL.
- *timeout* is the maximum time in microseconds to allow for I2C transactions. This parameter is not allowed on some ports.

Note that some ports/boards will have default values of *scl* and *sda* that can be changed in this constructor. Others will have fixed values of *scl* and *sda* that cannot be changed.

class **machine.SoftI2C**(*scl, sda, *, freq=400000, timeout=50000*)

Construct a new software I2C object. The parameters are:

- *scl* should be a pin object specifying the pin to use for SCL.
- *sda* should be a pin object specifying the pin to use for SDA.
- *freq* should be an integer which sets the maximum frequency for SCL.
- *timeout* is the maximum time in microseconds to wait for clock stretching (SCL held low by another device on the bus), after which an `OSError(ETIMEDOUT)` exception is raised.

General Methods

I2C.init(*scl, sda, *, freq=400000*)

Initialise the I2C bus with the given arguments:

- *scl* is a pin object for the SCL line
- *sda* is a pin object for the SDA line
- *freq* is the SCL clock rate

In the case of hardware I2C the actual clock frequency may be lower than the requested frequency. This is dependent on the platform hardware. The actual rate may be determined by printing the I2C object.

`I2C.deinit`()

Turn off the I2C bus.

Availability: WiPy.

`I2C.scan`()

Scan all I2C addresses between 0x08 and 0x77 inclusive and return a list of those that respond. A device responds if it pulls the SDA line low after its address (including a write bit) is sent on the bus.

Primitive I2C operations

The following methods implement the primitive I2C controller bus operations and can be combined to make any I2C transaction. They are provided if you need more control over the bus, otherwise the standard methods (see below) can be used.

These methods are only available on the `machine.SoftI2C` class.

`I2C.start`()

Generate a START condition on the bus (SDA transitions to low while SCL is high).

`I2C.stop`()

Generate a STOP condition on the bus (SDA transitions to high while SCL is high).

`I2C.readinto`(*buf, nack=True, /*)

Reads bytes from the bus and stores them into *buf*. The number of bytes read is the length of *buf*. An ACK will be sent on the bus after receiving all but the last byte. After the last byte is received, if *nack* is true then a NACK will be sent, otherwise an ACK will be sent (and in this case the peripheral assumes more bytes are going to be read in a later call).

`I2C.write`(*buf*)

Write the bytes from *buf* to the bus. Checks that an ACK is received after each byte and stops transmitting the remaining bytes if a NACK is received. The function returns the number of ACKs that were received.

Standard bus operations

The following methods implement the standard I2C controller read and write operations that target a given peripheral device.

I2C.readfrom(*addr*, *nbytes*, *stop=True*, */*)

Read *nbytes* from the peripheral specified by *addr*. If *stop* is true then a STOP condition is generated at the end of the transfer. Returns a `bytes` object with the data read.

I2C.readfrom_into(*addr*, *buf*, *stop=True*, */*)

Read into *buf* from the peripheral specified by *addr*. The number of bytes read will be the length of *buf*. If *stop* is true then a STOP condition is generated at the end of the transfer.

The method returns `None`.

I2C.writeto(*addr*, *buf*, *stop=True*, */*)

Write the bytes from *buf* to the peripheral specified by *addr*. If a NACK is received following the write of a byte from *buf* then the remaining bytes are not sent. If *stop* is true then a STOP condition is generated at the end of the transfer, even if a NACK is received. The function returns the number of ACKs that were received.

I2C.writevto(*addr*, *vector*, *stop=True*, */*)

Write the bytes contained in *vector* to the peripheral specified by *addr*. *vector* should be a tuple or list of objects with the buffer protocol. The *addr* is sent once and then the bytes from each object in *vector* are written out sequentially. The objects in *vector* may be zero bytes in length in which case they don't contribute to the output.

If a NACK is received following the write of a byte from one of the objects in *vector* then the remaining bytes, and any remaining objects, are not sent. If *stop* is true then a STOP condition is generated at the end of the transfer, even if a NACK is received. The function returns the number of ACKs that were received.

Memory operations

Some I2C devices act as a memory device (or set of registers) that can be read from and written to. In this case there are two addresses associated with an I2C transaction: the peripheral address and the memory address. The following methods are convenience functions to communicate with such devices.

I2C.readfrom_mem(*addr*, *memaddr*, *nbytes*, *, *addrsize=8*)

Read *nbytes* from the peripheral specified by *addr* starting from the memory address specified by *memaddr*. The argument *addrsize* specifies the address size in bits. Returns a `bytes` object with the data read.

I2C.readfrom_mem_into(*addr*, *memaddr*, *buf*, *, *addrsize=8*)

Read into *buf* from the peripheral specified by *addr* starting from the memory address specified by *memaddr*. The number of bytes read is the length of *buf*. The argument *addrsize* specifies the address size in bits (on ESP8266 this argument is not recognised and the address size is always 8 bits).

The method returns `None`.

I2C.writeto_mem(*addr*, *memaddr*, *buf*, *, *addrsize=8*)

Write *buf* to the peripheral specified by *addr* starting from the memory address specified by *memaddr*. The argument *addrsize* specifies the address size in bits (on ESP8266 this argument is not recognised and the address size is always 8 bits).

The method returns `None`.