class SPI – a Serial Peripheral Interface bus protocol (master side)

SPI is a synchronous serial protocol that is driven by a master. At the physical level, a bus consists of 3 lines: SCK, MOSI, MISO. Multiple devices can share the same bus. Each device should have a separate, 4th signal, SS (Slave Select), to select a particular device on a bus with which communication takes place. Management of an SS signal should happen in user code (via machine.Pin class).

Constructors

class **machine.SPI**(*id, ...*)

Construct an SPI object on the given bus, `id`. Values of `id` depend on a particular port and its hardware. Values 0, 1, etc. are commonly used to select hardware SPI block #0, #1, etc. Value -1 can be used for bitbanging (software) implementation of SPI (if supported by a port).

With no additional parameters, the SPI object is created but not initialised (it has the settings from the last initialisation of the bus, if any). If extra arguments are given, the bus is initialised. See `init` for parameters of initialisation.

Methods

SPI.init(*baudrate=1000000, *, polarity=0, phase=0, bits=8, firstbit=SPI.MSB, sck=None, mosi=None, miso=None, pins=(SCK, MOSI, MISO)*)

Initialise the SPI bus with the given parameters:

- `baudrate` is the SCK clock rate.
- `polarity` can be 0 or 1, and is the level the idle clock line sits at.
- `phase` can be 0 or 1 to sample data on the first or second clock edge respectively.
- `bits` is the width in bits of each transfer. Only 8 is guaranteed to be supported by all hardware.
- `firstbit` can be `SPI.MSB` or `SPI.LSB`.
- `sck`, `mosi`, `miso` are pins (machine.Pin) objects to use for bus signals. For most hardware SPI blocks (as selected by `id` parameter to the constructor), pins are fixed and cannot be changed. In some cases, hardware blocks allow 2-3 alternative pin sets for a hardware SPI block. Arbitrary pin assignments are possible only for a bitbanging SPI driver (`id` = -1).
- `pins` - WiPy port doesn't `sck`, `mosi`, `miso` arguments, and instead allows to specify them as a tuple of `pins` parameter.

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

SPI.deinit()

Turn off the SPI bus.

SPI.read(*nbytes, write=0x00*)

Read a number of bytes specified by `nbytes` while continuously writing the single byte given by `write`. Returns a `bytes` object with the data that was read.

SPI.readinto(*buf, write=0x00*)

Read into the buffer specified by `buf` while continuously writing the single byte given by `write`. Returns `None`.

Note: on WiPy this function returns the number of bytes read.

SPI.write(*buf*)

Write the bytes contained in `buf`. Returns `None`.

Note: on WiPy this function returns the number of bytes written.

SPI.write_readinto(*write_buf, read_buf*)

Write the bytes from `write_buf` while reading into `read_buf`. The buffers can be the same or different, but both buffers must have the same length. Returns `None`.

Note: on WiPy this function returns the number of bytes written.

Constants

`SPI.MASTER`

for initialising the SPI bus to master; this is only used for the WiPy

`SPI.MSB`

set the first bit to be the most significant bit

`SPI.LSB`

set the first bit to be the least significant bit