machine — functions related to the hardware

The machine module contains specific functions related to the hardware on a particular board. Most functions in this module allow to achieve direct and unrestricted access to and control of hardware blocks on a system (like CPU, timers, buses, etc.). Used incorrectly, this can lead to malfunction, lockups, crashes of your board, and in extreme cases, hardware damage.

A note of callbacks used by functions and class methods of machine module: all these callbacks should be considered as executing in an interrupt context. This is true for both physical devices with IDs >= 0 and "virtual" devices with negative IDs like -1 (these "virtual" devices are still thin shims on top of real hardware and real hardware interrupts). See Writing interrupt handlers.

Reset related functions

machine.reset()

Resets the device in a manner similar to pushing the external RESET button.

machine.reset_cause()

Get the reset cause. See constants for the possible return values.

Interrupt related functions

machine.disable_irq()

Disable interrupt requests. Returns the previous IRQ state which should be considered an opaque value. This return value should be passed to the enable_irq() function to restore interrupts to their original state, before disable_irq() was called.

machine.enable_irq(*state*)

Re-enable interrupt requests. The *state* parameter should be the value that was returned from the most recent call to the disable_irq() function.

Power related functions

machine.freq()

Returns CPU frequency in hertz.

machine.idle()

Gates the clock to the CPU, useful to reduce power consumption at any time during short or long periods. Peripherals continue working and execution resumes as soon as any interrupt is triggered (on many ports this includes system timer interrupt occurring at regular intervals on the order of millisecond).

machine.sleep()

Note

This function is deprecated, use `lightsleep()` instead with no arguments.

machine.lightsleep([*time_ms*])

machine.deepsleep([*time_ms*])

Stops execution in an attempt to enter a low power state.

If *time_ms* is specified then this will be the maximum time in milliseconds that the sleep will last for. Otherwise the sleep can last indefinitely.

With or without a timout, execution may resume at any time if there are events that require processing. Such events, or wake sources, should be configured before sleeping, like `Pin` change or `RTC` timeout.

The precise behaviour and power-saving capabilities of lightsleep and deepsleep is highly dependent on the underlying hardware, but the general properties are:

- A lightsleep has full RAM and state retention. Upon wake execution is resumed from the point where the sleep was requested, with all subsystems operational.
- A deepsleep may not retain RAM or any other state of the system (for example peripherals or network interfaces). Upon wake execution is resumed from the main script, similar to a hard or power-on reset. The `reset_cause()` function will return `machine.DEEPSLEEP` and this can be used to distinguish a deepsleep wake from other resets.

machine.wake_reason()

Get the wake reason. See constants for the possible return values.

Availability: ESP32, WiPy.

Miscellaneous functions

machine.unique_id()

Returns a byte string with a unique identifier of a board/SoC. It will vary from a board/SoC instance to another, if underlying hardware allows. Length varies by hardware (so use substring of a full value if you expect a short ID). In some MicroPython ports, ID corresponds to the network MAC address.

machine.time_pulse_us(*pin, pulse_level, timeout_us=1000000*)

Time a pulse on the given *pin*, and return the duration of the pulse in microseconds. The *pulse_level* argument should be 0 to time a low pulse or 1 to time a high pulse.

If the current input value of the pin is different to *pulse_level*, the function first (*) waits until the pin input becomes equal to *pulse_level*, then (**) times the duration that the pin is equal to *pulse_level*. If the pin is already equal to *pulse_level* then timing starts straight away.

The function will return -2 if there was timeout waiting for condition marked (*) above, and -1 if there was timeout during the main measurement, marked (**) above. The timeout is the same for both cases and given by *timeout_us* (which is in microseconds).

machine.rng()

Return a 24-bit software generated random number.

Availability: WiPy.

Constants

machine.IDLE

machine.SLEEP

machine.DEEPSLEEP

IRQ wake values.

machine.PWRON_RESET

machine.HARD_RESET

machine.WDT_RESET

machine.DEEPSLEEP_RESET

machine.SOFT_RESET

Reset causes.

machine.WLAN_WAKE

machine.PIN_WAKE

`machine.RTC_WAKE`

Wake-up reasons.

Classes

- class Pin – control I/O pins
- class Signal – control and sense external I/O devices
- class ADC – analog to digital conversion
- class ADCChannel — read analog values from internal or external sources
- class UART – duplex serial communication bus
- class SPI – a Serial Peripheral Interface bus protocol (master side)
- class I2C – a two-wire serial protocol
- class RTC – real time clock
- class Timer – control hardware timers
- class WDT – watchdog timer
- class SD – secure digital memory card (cc3200 port only)
- class SDCard – secure digital memory card