esp — functions related to the ESP8266 and ESP32

The esp module contains specific functions related to both the ESP8266 and ESP32 modules. Some functions are only available on one or the other of these ports.

Functions

esp.sleep_type([*sleep_type*])

Note: ESP8266 only

Get or set the sleep type.

If the *sleep_type* parameter is provided, sets the sleep type to its value. If the function is called without parameters, returns the current sleep type.

The possible sleep types are defined as constants:

- SLEEP_NONE – all functions enabled,
- SLEEP_MODEM – modem sleep, shuts down the WiFi Modem circuit.
- SLEEP_LIGHT – light sleep, shuts down the WiFi Modem circuit and suspends the processor periodically.

The system enters the set sleep mode automatically when possible.

esp.deepsleep(*time=0*)

Note: ESP8266 only - use machine.deepsleep() on ESP32

Enter deep sleep.

The whole module powers down, except for the RTC clock circuit, which can be used to restart the module after the specified time if the pin 16 is connected to the reset pin. Otherwise the module will sleep until manually reset.

esp.flash_id()

Note: ESP8266 only

Read the device ID of the flash memory.

esp.flash_size()

Read the total size of the flash memory.

esp.flash_user_start()

Read the memory offset at which the user flash space begins.

esp.flash_read(*byte_offset, length_or_buffer*)

esp.flash_write(*byte_offset, bytes*)

esp.flash_erase(*sector_no*)

esp.set_native_code_location(*start, length*)

Note: ESP8266 only

Set the location that native code will be placed for execution after it is compiled. Native code is emitted when the `@micropython.native`, `@micropython.viper` and `@micropython.asm_xtensa` decorators are applied to a function. The ESP8266 must execute code from either iRAM or the lower 1MByte of flash (which is memory mapped), and this function controls the location.

If *start* and *length* are both `None` then the native code location is set to the unused portion of memory at the end of the iRAM1 region. The size of this unused portion depends on the firmware and is typically quite small (around 500 bytes), and is enough to store a few very small functions. The advantage of using this iRAM1 region is that it does not get worn out by writing to it.

If neither *start* nor *length* are `None` then they should be integers. *start* should specify the byte offset from the beginning of the flash at which native code should be stored. *length* specifies how many bytes of flash from *start* can be used to store native code. *start* and *length* should be multiples of the sector size (being 4096 bytes). The flash will be automatically erased before writing to it so be sure to use a region of flash that is not otherwise used, for example by the firmware or the filesystem.

When using the flash to store native code *start+length* must be less than or equal to 1MByte. Note that the flash can be worn out if repeated erasures (and writes) are made so use this feature sparingly. In particular, native code needs to be recompiled and rewritten to flash on each boot (including wake from deepsleep).

In both cases above, using iRAM1 or flash, if there is no more room left in the specified region then the use of a native decorator on a function will lead to `MemoryError` exception being raised during compilation of that function.