# `microcontroller` - Pin references and cpu functionality

The `microcontroller` module defines the pins from the perspective of the microcontroller. See `board` for board-specific pin mappings.

`microcontroller.`**`cpu`** *:Processor*

CPU information and control, such as `cpu.temperature` and `cpu.frequency` (clock frequency). This object is the sole instance of `microcontroller.Processor`.

`microcontroller.`**`delay_us`**(*delay: int*) → None

Dedicated delay method used for very short delays. **Do not** do long delays because this stops all other functions from completing. Think of this as an empty `while` loop that runs for the specified `(delay)` time. If you have other code or peripherals (e.g audio recording) that require specific timing or processing while you are waiting, explore a different avenue such as using `time.sleep()`.

`microcontroller.`**`disable_interrupts`**() → None

Disable all interrupts. Be very careful, this can stall everything.

`microcontroller.`**`enable_interrupts`**() → None

Enable the interrupts that were enabled at the last disable.

`microcontroller.`**`on_next_reset`**(*run_mode: microcontroller.RunMode*) → None

Configure the run mode used the next time the microcontroller is reset but not powered down.

**Parameters:** **run_mode** (*RunMode*) – The next run mode

`microcontroller.`**`reset`**() → None

Reset the microcontroller. After reset, the microcontroller will enter the run mode last set by `on_next_reset`.

**Warning:** This may result in file system corruption when connected to a host computer. Be very careful when calling this! Make sure the device “Safely removed” on Windows or “ejected” on Mac OSX and Linux.

`microcontroller.`**`nvm`** *:Optional[ByteArray]*

Available non-volatile memory. This object is the sole instance of `nvm.ByteArray` when available or `None` otherwise.

**Type:** nvm.ByteArray or None

`microcontroller.`**`watchdog`** *:Optional[WatchDogTimer]*

Available watchdog timer. This object is the sole instance of `watchdog.WatchDogTimer` when available or `None` otherwise.

*class* `microcontroller.`**`Pin`**

Identifies an IO pin on the microcontroller.

Identifies an IO pin on the microcontroller. They are fixed by the hardware so they cannot be constructed on demand. Instead, use `board` or `microcontroller.pin` to reference the desired pin.

*class* `microcontroller.`**`Processor`**

Microcontroller CPU information and control

Usage:

```
import microcontroller
print(microcontroller.cpu.frequency)
print(microcontroller.cpu.temperature)
```

You cannot create an instance of `microcontroller.Processor`. Use `microcontroller.cpu` to access the sole instance available.

**`frequency`** *:int*

The CPU operating frequency in Hertz. (read-only)

**`temperature`** *:Optional[float]*

The on-chip temperature, in Celsius, as a float. (read-only)

Is `None` if the temperature is not available.

**`uid`** *:bytearray*

The unique id (aka serial number) of the chip as a `bytearray`. (read-only)

**`voltage`** *:Optional[float]*

The input voltage to the microcontroller, as a float. (read-only)

Is `None` if the voltage is not available.

*class* `microcontroller.`**`RunMode`**

run state of the microcontroller

Enum-like class to define the run mode of the microcontroller and CircuitPython.

**`NORMAL`** *:RunMode*

Run CircuitPython as normal.

**`SAFE_MODE`** *:RunMode*

Run CircuitPython in safe mode. User code will not be run and the file system will be writeable over USB.

**`BOOTLOADER`** *:RunMode*

Run the bootloader.