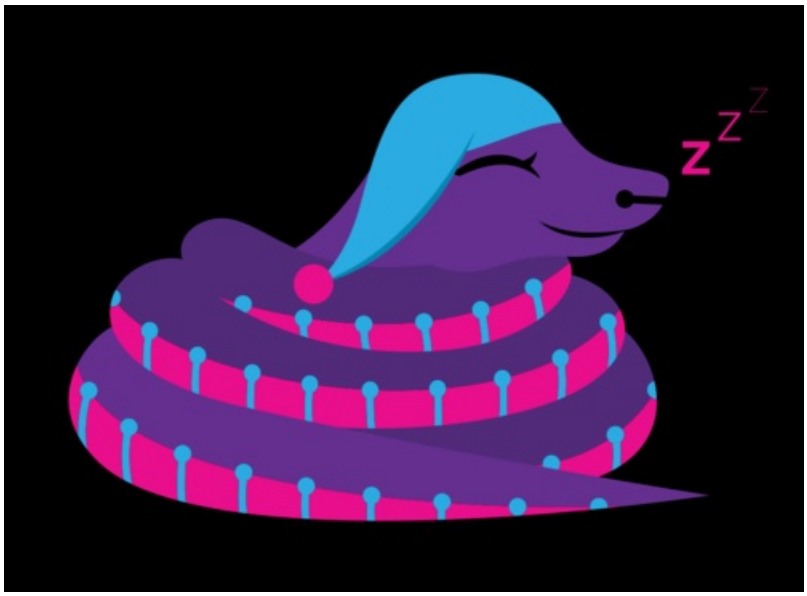
adafruit learning system

Deep Sleep with CircuitPython

Created by Dan Halbert



Last updated on 2021-01-19 03:26:28 PM EST

Guide Contents

Overview

If you'd like to maximize the battery life on a CircuitPython project, you need to be able to put your program to sleep when it's not doing something. For instance, you may want to read a temperature or fetch some data only every few minutes or hours. In between, your board can go to sleep and draw only a tiny amount of power from the battery.

If you're using a display that is visible even when powered off, such as the e-ink display on the Adafruit MagTag, then you can sleep between updates to the display.

This guide will talk about using the sleep and wake-up alarm capabilities that are available in CircuitPython.

Not all board families have this capability, but the developers will be adding more!

Alarms and Sleep

Terminology

We'll distinguish between *deep sleep* and *light sleep*:

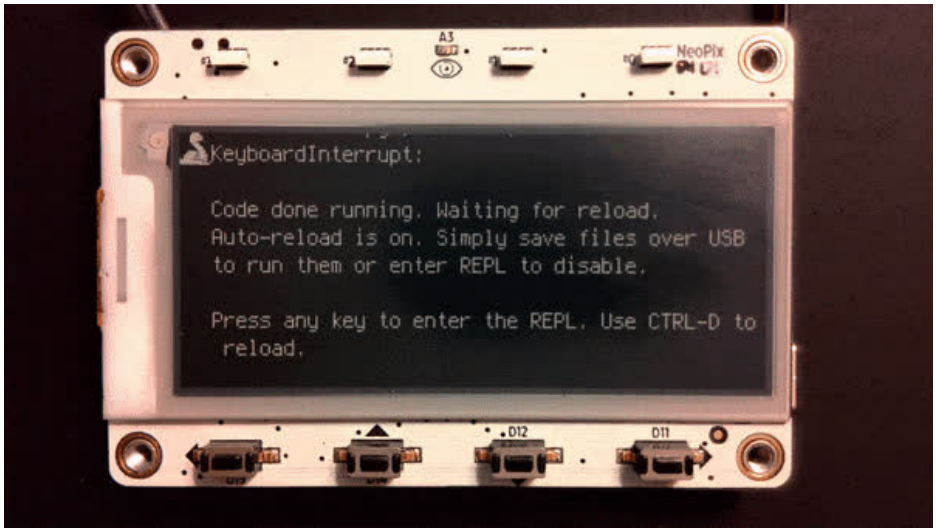
- If a program does a *deep sleep*, it first exits, and then the microcontroller goes to sleep, turning off as much as possible while still being able to wake up later. When the microcontroller wakes up, it will start your program (**code.py**) *from the beginning*.
- If a program does a *light sleep*, it still goes to sleep but continues running the program, resuming after the statement that did the light sleep. Power consumption will be minimized. However, on some boards, such as the ESP32-S2, light sleep does not save power compared with just using `time.sleep()`.

CircuitPython uses *alarms* to wake up from sleeping. An alarm can be triggered based on a specified time being reached, or based on an external event, such as a pin changing state. The pin might be attached to a button, so you would be able to wake up on a button press.

> Sleep and alarms are available only in CircuitPython 6.1.0-beta.2 or later, and right now are only provided in the ESP32-S2 port.

The `alarm` module

Alarms and sleep are available in the `alarm` module in CircuitPython. You create one or more alarms, and then go into a light sleep or deep sleep while waiting for them.



TimeAlarm Light Sleep

Here's a simple program that just blinks the status NeoPixel every 10 seconds, and does a light sleep in between, using a `TimeAlarm`. The video above demonstrates this program, eliding the 10-second sleeps.

```
import alarm
import board
import digitalio
import neopixel
import time

# On MagTag, enable power to NeoPixels.
# Remove these two lines on boards without board.NEOPIXEL_POWER.
np_power = digitalio.DigitalInOut(board.NEOPIXEL_POWER)
np_power.switch_to_output(value=False)

np = neopixel.NeoPixel(board.NEOPIXEL, 1)

while True:
    np[0] = (50, 50, 50)
    time.sleep(1)
    np[0] = (0, 0, 0)

    # Create a an alarm that will trigger 10 seconds from now.
    time_alarm = alarm.time.TimeAlarm(monotonic_time=time.monotonic() + 10)

    # Do a light sleep until the alarm wakes us.
    alarm.light_sleep_until_alarms(time_alarm)
    # Finished sleeping. Continue from here.
```

`TimeAlarm` Deep Sleep

Here's a similar program, which does a deep sleep. The video above is still what you'd see. Remember that for deep sleep, the program exits, and restarts when woken up. So in this program there's no `while True:` loop.

```
import alarm
import board
import digitalio
import neopixel
import time

# On MagTag, enable power to NeoPixels.
# Remove these two lines on boards without board.NEOPIXEL_POWER.
np_power = digitalio.DigitalInOut(board.NEOPIXEL_POWER)
np_power.switch_to_output(value=False)

np = neopixel.NeoPixel(board.NEOPIXEL, 1)

np[0] = (50, 50, 50)
time.sleep(1)
np[0] = (0, 0, 0)

# Create a an alarm that will trigger 20 seconds from now.
time_alarm = alarm.time.TimeAlarm(monotonic_time=time.monotonic() + 20)
# Exit the program, and then deep sleep until the alarm wakes us.
alarm.exit_and_deep_sleep_until_alarms(time_alarm)
# Does not return, so we never get here.
```

`PinAlarm` Deep Sleep

This example uses `PinAlarm` instead of `TimeAlarm`. It will deep sleep until the `D11` button on the lower right of the MagTag is pressed. On the MagTag, pressing a button connects a pin to ground, so we wait for a `False` value. We also enable a pull-up to hold the pin high (`True`) when the button is not pressed.

```
import alarm
import board
import digitalio
import neopixel
import time

# On MagTag, enable power to NeoPixels.
# Remove these two lines on boards without board.NEOPIXEL_POWER.
np_power = digitalio.DigitalInOut(board.NEOPIXEL_POWER)
np_power.switch_to_output(value=False)

np = neopixel.NeoPixel(board.NEOPIXEL, 1)

np[0] = (50, 50, 50)
time.sleep(1)
np[0] = (0, 0, 0)

pin_alarm = alarm.pin.PinAlarm(pin=board.D11, value=False, pull=True)

# Exit the program, and then deep sleep until the alarm wakes us.
alarm.exit_and_deep_sleep_until_alarms(pin_alarm)

# Does not return, so we never get here.
```

TouchAlarm is currently not working properly, as of CircuitPython 6.1.0-rc.1, but should be fixed in 6.1.0-rc.2 or later

`TouchAlarm` Deep Sleep

This example is for the Metro ESP32-S2. The MagTag has no pins that can be used for touch. (`D10` could theoretically be used, but protection components are connected to it that prevent it being used for touch.)

It will sleep until pin `IO5` is touched, or 10 seconds has elapsed, whichever comes first. The on-board LED blinks for one second at the beginning of the program.

```
import alarm
import board
import digitalio
import time

# Print out which alarm woke us up, if any.
print(alarm.wake_alarm)

led = digitalio.DigitalInOut(board.LED)
led.switch_to_output(value=True)
time.sleep(1)
led.value = False

# Create a an alarm that will trigger 10 seconds from now.
time_alarm = alarm.time.TimeAlarm(monotonic_time=time.monotonic() + 10)
# Create an alarm that will trigger if pin IO5 is touched.
touch_alarm = alarm.touch.TouchAlarm(pin=board.IO5)

# Exit the program, and then deep sleep until one of the alarms wakes us.
alarm.exit_and_deep_sleep_until_alarms(time_alarm, touch_alarm)
# Does not return, so we never get here.
```

Pretending to Sleep When Connected

When your board is connected to a host computer via USB, you don't want it to really do a light sleep or deep sleep, because that would break the USB connection and make it difficult to debug or edit your program. So when the board is connected, we **simulate** light and deep sleep. If CircuitPython can reduce power consumption while pretending to sleep and still remain connected, it will do so, but you will not be saving nearly as much power as when you're not connected.

So if you're trying to measure how much power you're saving while sleeping, you really need to do your measurements from battery power (or a power supply) while unconnected.

What Woke Me Up?

When a program awakens from light sleep or deep sleep, it's because of an alarm. You can find out what kind of alarm woke up the program by looking at `alarm.wake_alarm`.

If the program **did not wake up from sleep**, then `alarm.wake_alarm` will be `None`.

If the program woke up from a **light sleep**, then `alarm.wake_alarm` will be one of the alarm objects passed to `alarm.light_sleep_until_alarms(...)`. The triggered alarm is also returned by that function, so you can get the alarm value directly. Here are two ways to get the triggered alarm:

```
triggered_alarm = alarm.light_sleep_until_alarms(alarm1, alarm2)

# is the same as

alarm.light_sleep_until_alarms(alarm1, alarm2)
triggered_alarm = alarm.wake_alarm
```

If the program restarted after a deep sleep, then `alarm.wake_alarm` will be an alarm object of the same type as the original alarm, but it will not be exactly the same object. It's attributes may be different, or they may be incomplete in some way. For instance, for `TimeAlarm`, the .`monotonic_time` attribute may not contain the same value. But you can still do an `isinstance(alarm.wake_alarm, TimeAlarm)` to find out it was a `TimeAlarm` that woke up the program.

Sleep Memory

When a program goes into deep sleep, it exits and then sleeps. So all the information in its variables is lost. But the program might want to remember something for use when it restarts. It could write into a file onto the board **CIRCUITPY drive**, or it could write into internal flash using `microcontroller.nvm`. But flash has a limited lifetime, so it's better not to write it over and over. Instead, the program can write into a special part of memory (RAM) that is powered during deep sleep. In most microcontrollers, this kind of memory is called "backup RAM"; in CircuitPython, we call it `alarm.sleep_memory`. This memory requires very little power to maintain. If power is removed completely, then the memory is lost, but as long as USB power or a battery is connected, it will remember what is stored in it.

`alarm.sleep_memory` is just a byte array of a few thousand bytes. You can use it to store whatever you want, but you'll need to encode the data as bytes. You could use `struct.pack` and `struct.unpack`, or use JSON, or some other format that's convenient for you.

MagTag Example

Here's a simple example of using sleep memory, without any encoding. Each time the program wakes up, it increments a count kept in one byte in `alarm.sleep_memory`. This program displays the current battery voltage and the count on the MagTag display.

The very first time the program runs, we want to initialize the count. We can tell if this is the first time the program has run by checking `alarm.wake_alarm`. If it is `None`, then we know we have not done a deep sleep yet.

The video below shows the program running, with the long 60 second-sleeps elided.

```
import alarm
import microcontroller
import time
from adafruit_magtag.magtag import MagTag


magtag = MagTag()

magtag.add_text(
    text_scale=2,
    text_wrap=25,
    text_maxlen=300,
    text_position=(10, 10),
    text_anchor_point=(0, 0),
)

# Reset the count if we haven't slept yet.
if not alarm.wake_alarm:
    # Use byte 5 in sleep memory. This is just an example.
    alarm.sleep_memory[5] = 0

alarm.sleep_memory[5] = (alarm.sleep_memory[5] + 1) % 256

# Display the current battery voltage and the count.
magtag.set_text(
    "battery: {}V    count: {}".format(
        magtag.peripherals.battery, alarm.sleep_memory[5]
    )
)

magtag.refresh()

# Sleep for 60 seconds.
al = alarm.time.TimeAlarm(monotonic_time=time.monotonic() + 60)
alarm.exit_and_deep_sleep_until_alarms(al)
# Does not return. Exits, and restarts after 60 seconds.
```

Power Consumption

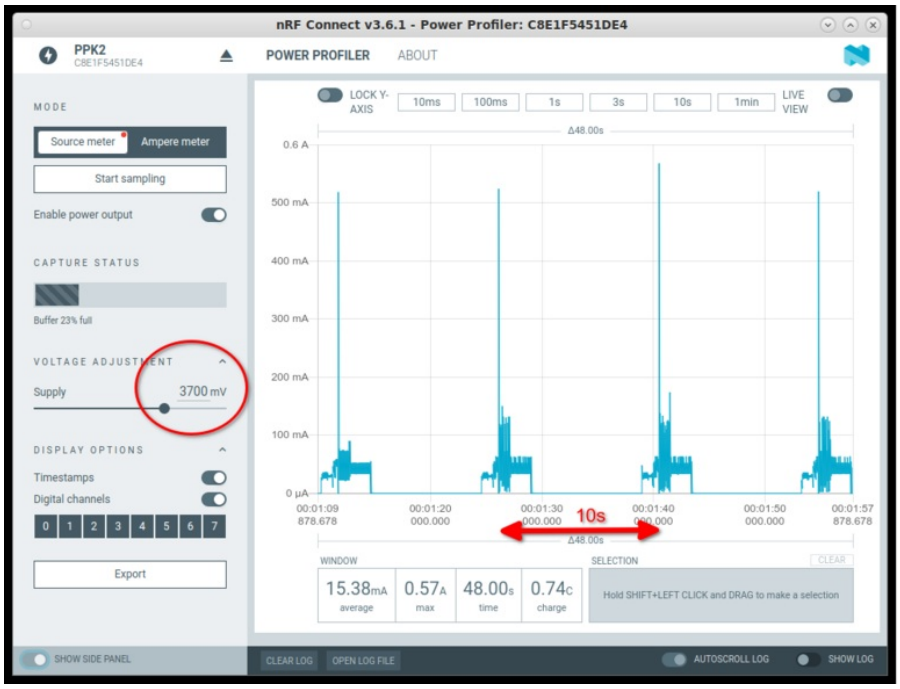
How much power would you actually save by using light sleep or deep sleep? It depends on which board you're using, what your program is doing when it's not sleeping, what is left turned on during light sleep, and how long you spend sleeping vs running (your program's *duty cycle*).

Given all those variables, let's look at power consumption using the sample programs in the Alarms and Sleep (https://adafru.it/PfF) section of this guide. If you recall, those programs just blink a NeoPixel for one second and then sleep for ten seconds.

We'll show a bunch of power-consumption graphs below. These screenshots were taken using the Nordic Semiconductor Power Profiler Kit II (https://adafru.it/PfG) (PPK2) hardware and software. The PPK2 is quite inexpensive (< $100) compared to the usual power meter, is easy to use, and works well.

ESP32-S2 TimeAlarm Deep Sleep Power Consumption

Here's a graph showing the `TimeAlarm` deep sleep demo program, running on a MagTag ESP32-S2, sleeping every ten seconds. There's a big short spike when the program starts up after sleeping. The program blinks the NeoPixel, and then sleeps. We are supplying 3.7V to the board, which is a typical LiPo battery voltage.



Let's zoom in on the power consumption while the program is actively running during one cycle. You can see it's using about 50mA. Note that the vertical axis scale has changed, because we're not including the big spike at the beginning of a run.

If this program were using WiFi, you'd see much higher power consumption, up to a few hundred mA during active WiFi use.

Now let's look at the power consumption during deep sleep. In the graphs above, that's where the power consumption line looks very close to 0. Again, the vertical axis has expanded, so we can see microamps accurately. You can see the board is using a little under 230uA when it's sleeping. About 25-30uA of this is the actual ESP32-S2 module on the MagTag; the rest is board overhead, like the voltage regulator, and the (dim) power LED.

ESP32-S2 TimeAlarm Light Sleep Sample Power Consumption

For comparison, here's a graph of the `TimeAlarm` light sleep demo program, which also cycles every ten seconds. On the ESP32-S2, as mentioned, light sleep is no better than `time.sleep` in terms of power consumption. For one second before the NeoPixel is turned on (marker 1), power consumption is about

33mA. Turning on the NeoPixel raises the current drawn to about 75mA. Then the program sleeps (marker 2), but the consumption is still about 33mA.

So there's not much reason to use light sleep on the ESP32-S2 if you're trying to save power.

ESP32-S2 PinAlarm Deep Sleep Power Consumption

Now let's look at deep sleep power consumption when using a `PinAlarm` on the ESP32-S2 MagTag. Here are several cycles of sleeping. Each red arrow points to when the D11 button was pressed. The intervals are different lengths because the time between button pushes was not the same.

Here is the start of a single `PinAlarm` sleep cycle. Unfortunately, the circuitry that needs to stay on to

detect pin changes takes a significant amount of current. Even when deep sleeping, the board is using about 1.65mA, much more than it would if we were just using `TimeAlarm`.

ESP32-S2 TouchAlarm Deep Sleep Power Consumption

ESP32-S2 `TouchAlarm` power consumption is similarly higher than you might like It's about 2.6mA, even higher than `PinAlarm`, about 2.6mA. So again, if you really want to save your battery, use `TimeAlarm`.

Sleep Power Summary

Here's a chart of deep sleep power consumption when different kinds of alarms are used. As we add sleep to other chip families, we'll add to this chart.

ESP32-S2 (MagTag) Deep Sleep

- `TimeAlarm`: 230uA
- `PinAlarm`: 1.65mA
- `TouchAlarm`: 2.6mA

ESP32-S2 Light Sleep

- Sleep current is the same as `time.sleep()`, so there's no advantage to using light sleep.

Do not measure true power consumption on USB power as CircuitPython only simulates sleep while connected via USB.

Measure to be Sure

As we mentioned, for any particular program and board, there are many things that can affect its power consumption. If you want to be know how long your battery will last, and whether you're consuming power unnecessarily, it's really helpful to have a power meter.

Don't Forget About Simulated Sleep

Don't forget that when your board is connected to a host computer via USB, it does not actually sleep

when sleep is requested, because we don't want to break the USB serial and mass storage (**CIRCUITPY**) connections. So you need to test power consumption when not connected. The board can be powered from the battery connector or via the USB port (via a power pack or a wall adapter), but **it can't be actively connected to a host computer.**