



ghubcoder

[About](#)

[Posts](#)

[Categories](#)



Oct 27 2021

Deep sleeping the Raspberry Pico with Micropython

The [first](#) post of this series discussed how you can put your pico into a deep sleep state and wake it at some point later using the internal real time clock, using the c programming language. Here we will discuss a solution to allow the same technique to be used with Micropython.

TLDR

Skip to release and example [here](#)

Micropython

Whilst the first approach used C, Micropython however is heavily promoted by many of the getting started guides you will find for the Pico such as this one found [here](#). This makes sense as today Python is the first programming language many will encounter, it's widely used and has a huge library of modules available to leverage. It's very easy to get started with, especially when using Thonny, allowing you to run commands via the [REPL](#), or by uploading files onto the board to be ran on boot-up. Micropython itself can be used on numerous boards as a result of its support of a number of ARM based architectures (perhaps one slight downside as we will come to later).

Python and hence Micropython does have a performance drawback as a result of it being an interpreted language when compared to C, however it's possible to extend Python (and Micropython) with natively compiled C modules for critical paths of your code which can be imported as regular Python modules. An example of how to do this with Micropython can be seen [here](#).

Dormant sleep mode

This leads us onto [this](#) issue raised a few weeks ago around how to use deep sleep with Micropython. As mentioned previously, the Pico has two sleep modes, known as **dormant** and **sleep**, read more about these in the [previous](#) post which includes links to the official docs.

Micropython deepsleep

Micropython does support a [deepsleep](#) state. As mentioned above, Micropython supports a variety of board types:

“ The precise behaviour and power-saving capabilities of lightsleep and deepsleep is highly dependent on the underlying hardware

Which means that with a single command, `machine.deepsleep` they must be able to implement similar behavior across a wide number of boards.

Currently when `machine.deepsleep` is utilized on the Pico, the board sleeps and will then reset. If you have uploaded your Python main script file as `main.py`, the board will restart after sleeping and once again begin executing your code, until it sleeps once more (if that code path is triggered).

The power consumption was measured during this sleep state and unfortunately on the Pico it still pulls **25.6mA** (at least on the board used and with the ambient conditions for this particular test).

Dormant Sleep mode on the Pico

After a quick look around for a solution, it appears [tomjorquera](#) has done the hard work [already](#) and has been able to demonstrate how the dormant mode can be utilized from within Micropython. Unfortunately you need an external trigger as described [here](#) to bring your Pico back out of deep sleep. If you have an **DS3231** handy it's certainly a good option and is the lowest sleep state available drawing **0.8mA**.

Sleep mode on the Pico

The simplest method however is to make use of the internal real time clock as [demonstrated](#), as no external modules are required. We ask the Pico to sleep and then awake again after some time has past. This draws around **1.3mA**.

As discussed in [this](#) reply, Circuitpython, an alternative to Micropython does appear to support waking from the RTC. The following

code was tested:

```
import alarm
import board
import time
import digitalio

print("Waking up")

led = digitalio.DigitalInOut(board.LED)
led.direction = digitalio.Direction.OUTPUT

# Flash the led
led.value = True
time.sleep(2)
led.value = False
# Set an alarm for 60 seconds from now.
time_alarm = alarm.time.TimeAlarm(monotonic_time=time.monotonic() + 60)

print("Sleeping")
# Deep sleep until the alarm goes off. Then restart the program.
alarm.exit_and_deep_sleep_until_alarms(time_alarm)
```

The current draw was measured during sleep mode and was found to be around **7.6mA**, which whilst better than Micropython's deep sleep is still much more than the **1.3mA** of the sleep mode when used via C.

Possible solutions

There are perhaps 3 ways we can get this to work via Micropython.

1. As [tomjorquera](#) has demonstrated, we can try to work out what register calls must be made on the Pico to replicate the calls the `pico-extras` library is making from within C in order to get everything ready to sleep and wake from the RTC.
2. We could attempt to write an importable external C module as described [here](#).
3. We can extend Micropython directly to support deep sleeping.

Solution **1** would most likely be the most portable as nothing needs to be recompiled. This is not the solution chosen here, it's perhaps something that can be looked at in the future. Solution **2** may not be feasible, please see [this](#) post on the Raspberry Pico forums.

That leaves us with solution **3**. This could be broken down into two parts, we can try and submit a PR and try to ensure that when `machine.deepsleep` is called when running on the Pico, the correct calls are made to get everything ready to sleep and wake as we wish. Before we do that however, we can try and recompile Micropython and directly add a new module for ourselves that will be available to use when we run our custom version of Micropython on our own Raspberry Pico's.

This is not the best solution, as it means we are having to fork Micropython and will have to ensure our fork is kept up to date if we want to continue to bring in new features and bug fixes that the mainline codebase brings us. It does though demonstrate the feasibility of deep sleeping with the Pico using Micropython. We can open source this code and provide a custom `firmware.uf2` file which people can download and drop onto their boards for them to try it out.

The rest of this article will discuss this approach.

Required changes to Micropython

[This](#) forum post describes how we can add our own C extension to Micropython thanks to the most excellent [hippy](#) who provides so much insight on the Raspberry forums.

The basic idea is to make a copy of the `ports/rp2/` directory that we can work with to add our custom module, add our code and modify a number of files to instruct the compiler to include it along with the rest of the current code.

The example hippy gives is as follows:

```
#include "py/runtime.h"

STATIC mp_obj_t hack_test(void) {
    return MP_OBJ_NEW_SMALL_INT(1234);
}
MP_DEFINE_CONST_FUN_OBJ_0(hack_test_obj, hack_test);

STATIC const mp_rom_map_elem_t hack_module_globals_table[] = {
    { MP_ROM_QSTR(MP_QSTR__name__), MP_ROM_QSTR(MP_QSTR_hack) },
    { MP_ROM_QSTR(MP_QSTR_test), MP_ROM_PTR(&hack_test_obj) },
};
STATIC MP_DEFINE_CONST_DICT(hack_module_globals, hack_module_globals_table);

const mp_obj_module_t mp_module_hack = {
    .base = { &mp_type_module },
    .globals = (mp_obj_dict_t *)&hack_module_globals,
};
```

This can then be used as follows within Micropython:

```
>>> import hack
>>> hack.test()
1234
```

Which prints `1234` when run from the Micropython repl. In the above block of c code, the first part is what we really care about, everything else below is required to set everything up within Micropython for our new custom function to be called:

```
STATIC mp_obj_t hack_test(void) {
    return MP_OBJ_NEW_SMALL_INT(1234);
}
```

By following the same steps we can create a module which will run the same c code as used in the first Pico sleep demo post. The module will be called `picosleep`.

Including pico-extras

The Micropython rp2 port currently makes use of the `pico-sdk` git repo as a sub module, allowing it to make use of libraries provided by the Pico developers. That can be found [here](#). A git sub module allows you to include other git repositories within your project, without having to include a complete copy of the code.

The Pico deep sleep code we are going to add makes use of functions provided by the `pico-extras` git [repo](#) also provided by the Pico developers. We can add it using the following command under the `lib/` directory:

```
git submodule https://github.com/raspberrypi/pico-extras
```

A number of changes to `CMakeLists.txt` were also made to ensure `pico-extras` is correctly added for everything to compile correctly, such as [here](#) for example.

The full example code for how this looks once we add in the sleep code can be seen [here](#), looking very similar to the first [example](#).

Building

With the changes made to cmake, the custom module can be built as follows:

```
git clone https://github.com/ghubcoder/micropython-pico-deepsleep.git
cd micropython/
make -C mpy-cross/
git submodule update --init -- lib/pico-sdk
git submodule update --init -- lib/pico-extras
git submodule update --init -- lib/tinyusb
cd ports/rp2sleep
make -j4
```

This should result in a `build-PICO` folder being generated, inside which will contain a `firmware.uf2` file. By powering on the Pico and holding down the BOOTSEL button, this file can be dropped into place. Once the Pico restarts it will be running the new custom firmware, ready for the `picosleep` module to be imported and used.

Connecting via Thonny should show a message similar to:

```
MicroPython v1.17-94-gfab32dd88-dirty on 2021-10-27; Raspberry Pi Pico (sleep) with RP2040
```

Where `(sleep)` indicates the custom firmware is being executed as defined [here](#).

Using picosleep from micropython

Unfortunately attempting to use this from within the repl will result in the connection being lost and no further output will be seen from your script in the repl terminal. The best way is to write a script and upload this onto the Pico using Thonny. An example is as follows:

```
import picosleep
import time
from machine import Pin

led = Pin(25, Pin.OUT)
sleeptime = 1

while sleeptime <= 10:
    led.toggle()
    time.sleep(5)
    led.toggle()
    picosleep.seconds(sleeptime)
    sleeptime = sleeptime + 1
```

A quick example of how we can sleep within a loop. This will start your pico, enable the led for 5 seconds and then deepsleep and repeat until `sleeptime` gets to 10 seconds. Proving we can deepsleep from python and continue code execution.

Unfortunately when this was first ran, the Pico hung. It appears that when attempting to sleep for 1 second there is a timing issue, perhaps from the time the real time clock started to the time the sleep is initiated 1 second has already past (doubtful though!). To guard against this the following was added:

```
//Hangs if we attempt to sleep for 1 second...
//Guard against this and perform a normal sleep
if(seconds_to_sleep == 1){
    sleep_ms(1000);
    return;
}
```

So in the example above the first loop iteration doesn't actually deepsleep. Sleeping for 2 seconds or more works fine however and with the changes made to the code to make use of the c `<time.h>` library we can sleep for much longer periods. This is achieved by creating a struct representing the current time when the RTC is initialized, incrementing the seconds and then calling `mktime`, and using the resulting struct to create a Pico `datetime_t` struct which we can pass to the Pico to sleep :

```
t.tm_sec += seconds_to_sleep;
mktime(&t);

datetime_t t_alarm = {
    .year  = t.tm_year+1900,
    .month = t.tm_mon+1,
    .day   = t.tm_mday,
    .dotw  = t.tm_wday, // 0 is Sunday, so 5 is Friday
    .hour  = t.tm_hour,
    .min   = t.tm_min,
    .sec   = t.tm_sec
};

sleep_goto_sleep_until(&t_alarm, &sleep_callback);
```

Resulting Power consumption

By hooking up a multimeter to the Pico we can measure the current draw using different sleep methods:

Method	current (mA)
Micropython <code>time.sleep</code>	25.4mA
Micropython <code>machine.deepsleep</code>	25.6mA
Circuitpython <code>deepsleep</code>	7.6mA
Micropython (custom firmware) <code>picosleep.seconds</code>	1.4mA

Using the solution here we can get very close to the 1.3mA as advertised by the official docs for sleeping and waking using the RTC.

Code and release download location

Code can be found [here](#).

Pre-compiled uf2 file can be found [here](#)

Usage:

```
import picosleep
picosleep.seconds(60)
```

© ghubcoder

Powered by [Hugo](#) and [Devise](#)