Super Simple Sunrise Lamp

Created by Carter Nelson

https://learn.adafruit.com/super-simple-sunrise-lamp

Last updated on 2025-09-22 06:14:46 PM EDT

Table of Contents

Overview 3

- RPI Dark Times
- Other Options

CircuitPython 5

- CircuitPython Quickstart

CircuitPython Internet Test 8

- The settings.toml File
- IPv6 Networking

Lamp Hardware 15

- Wiring
- Lamp Assembly

Code Using NTP 17

Code Using AIO 19

- Add AIO info to settings.toml

Using the Alarm 21

- NTP Version Only
- Arming and Resetting
- Troubleshooting

Overview

This guide was motivated by a really cool project we saw go by online. The idea is to create an alarm clock for kids that uses colors instead of an actual clock face. A night time color means "keep sleeping" and a sunrise color means "wake up! time to play!". Great for youngsters who are not quite ready to read a clock (or struggle to in the morning)

Here's a link to the original project which is based on a Raspberry Pi:

A Raspberry Pi-powered Snapology Origami Sunrise Lamp

https://adafru.it/106a

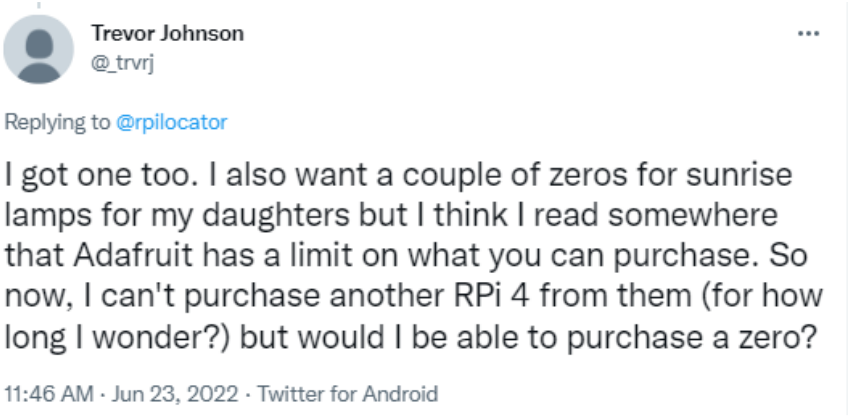
The origami used to make the lamp structure is amazing!

The lamp is really nothing more than: (1) grab current time from internet, (2) wait for wake up time, (3) and then do something with NeoPixels. Can an Internet connected Raspberry Pi running a full linux operating system do this? Yes. Easily. Without even breaking a sweat. Might even be overkill to use a Pi. But when a wifi capable full linux compy can be had for $10, why not?

RPI Dark Times

Throwing a $10 linux compy at a trivial task is fine, assuming said hardware is actually available. However, Raspberry Pi's are currently very difficult to come by.

There's an unfortunate combination of limited supply with ongoing high demand. Bot driven resellers are buying up stock and reselling them for crazy marked-up prices. Resellers (like Adafruit) are trying to thwart these efforts with various purchasing limits and requirements. For people that just want to make fun Pi based projects, all this just leads to anguish and heart ache trying to get hands on a Pi. The original tweet that caught our eye bemoans this as well:



Remember when Pi's were given away free on the cover of magazines? (https://adafru.it/106b) Ah...good memories from the before times.

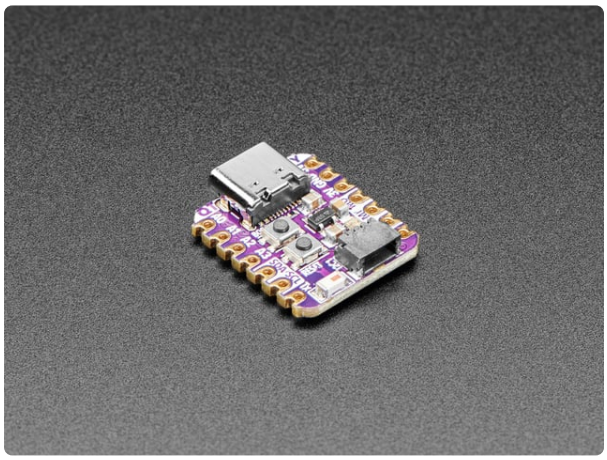
Other Options

But wait! There is hope!

A Raspberry Pi is not the only option for "connecting to the Internet" and doing stuff. There are simpler microcontrollers, like the various ESP chips, that can do this. Sure, they won't be pulling down terabytes of data and running numpy. But they can do relatively simple things - like grab time from the Internet. And they can also readily drive NeoPixels.

In this guide we show how to recreate the Sunrise Lamp project using an ESP32-S2 based board. There are numerous ESP32-S2 options, but to keep things extra smol, we use a QT Py.

There's one with a built in antenna:

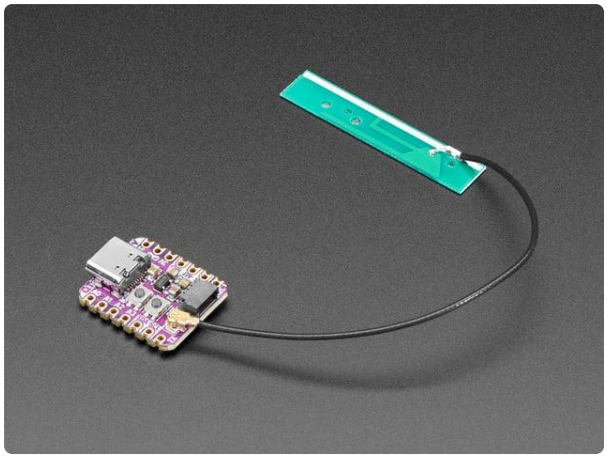


Adafruit QT Py ESP32-S2 WiFi Dev Board with STEMMA QT

What has your favorite Espressif WiFi microcontroller, comes with our favorite connector - the STEMMA QT, a chainable I2C port, and has...

https://www.adafruit.com/product/5325

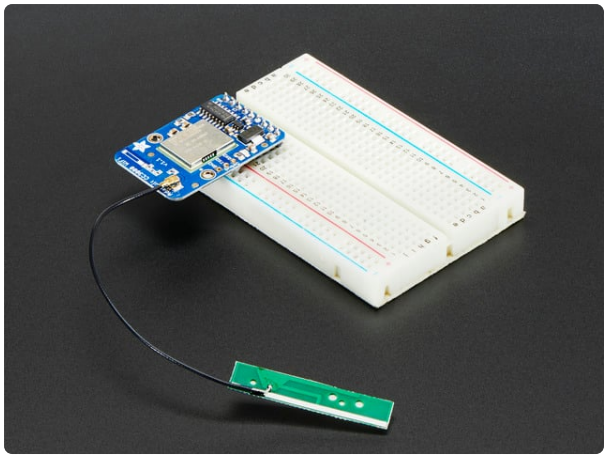
Or with a uFL connector for attaching an external antenna:



Adafruit QT Py ESP32-S2 WiFi Dev Board with uFL Antenna Port

What has your favorite Espressif WiFi microcontroller, comes with our favorite connector - the STEMMA QT, a chainable I2C port, and has...

https://www.adafruit.com/product/5348

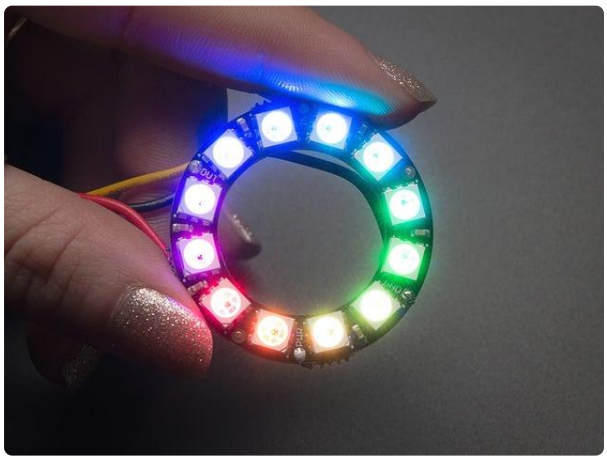


2.4GHz Mini Flexible WiFi Antenna with uFL Connector

This 4" / 100mm long flexible uFL 2.4GHz antenna has approx 4DBi gain and a 50Ω impedance so it will work fantastically with just about any 2.4-2.5GHz...

https://www.adafruit.com/product/2308

Just like the original project, we use a 12 NeoPixel ring:



NeoPixel Ring - 12 x 5050 RGB LED with Integrated Drivers

Round and round and round they go! 12 ultra bright smart LED NeoPixels are arranged in a circle with 1.5" (37mm) outer diameter. The rings are 'chainable' - connect the...

https://www.adafruit.com/product/1643

But any NeoPixels could be used.

CircuitPython

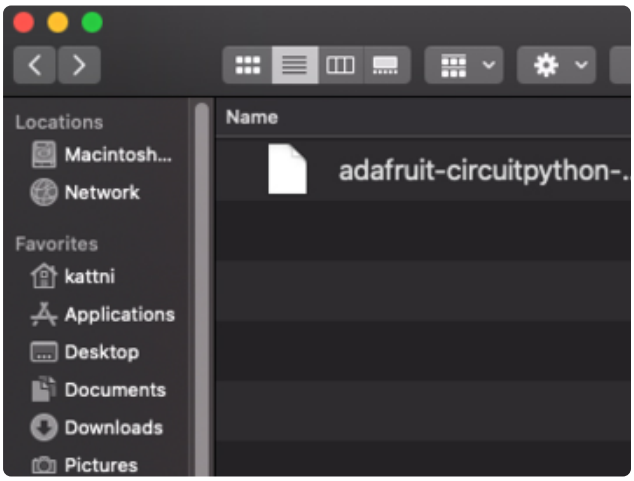
CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/BeZ) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files on the **CIRCUITPY** drive to iterate.

CircuitPython Quickstart

Follow this step-by-step to quickly get CircuitPython running on your board.

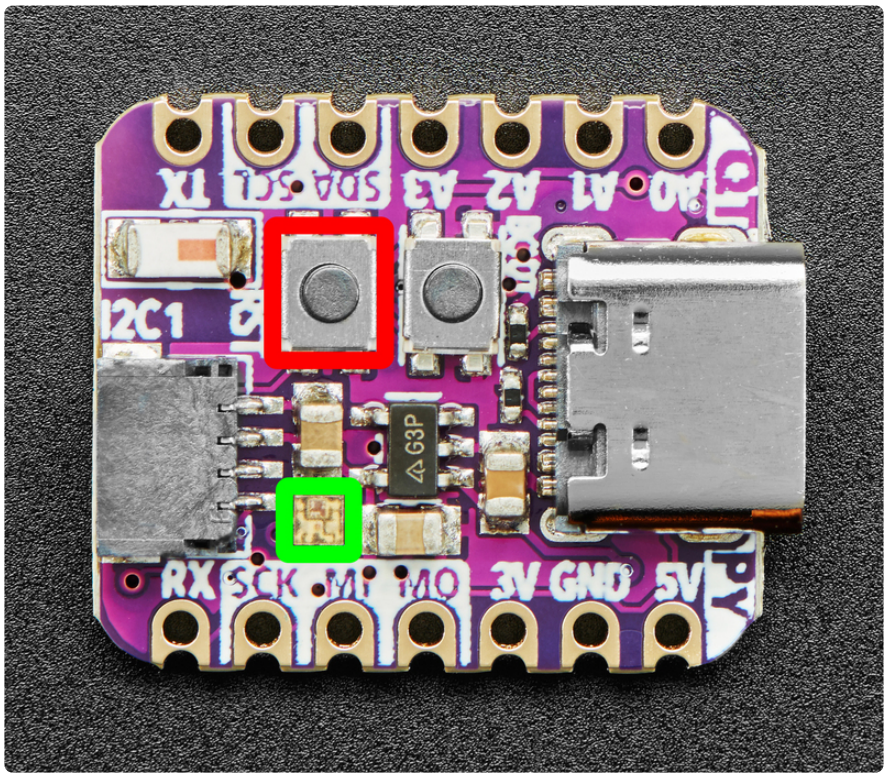
Download the latest version of CircuitPython for this board via circuitpython.org

https://adafru.it/XCk



Click the link above to download the latest CircuitPython UF2 file.

Save it wherever is convenient for you.



The board above has a chip antenna, not the u.Fl connector, but the process is the same.

Plug your board into your computer, using a known-good data-sync cable, directly, or via an adapter if needed.

Click the **reset** button once (highlighted in red above), and then click it again when you see the **RGB status LED(s)** (highlighted in green above) turn purple (approximately half a second later). Sometimes it helps to think of it as a "slow double-click" of the reset button.

If you do not see the LED turning purple, you will need to reinstall the UF2 bootloader. See the **Factory Reset** page in this guide for details.

On some very old versions of the UF2 bootloader, the status LED turns red instead of purple.

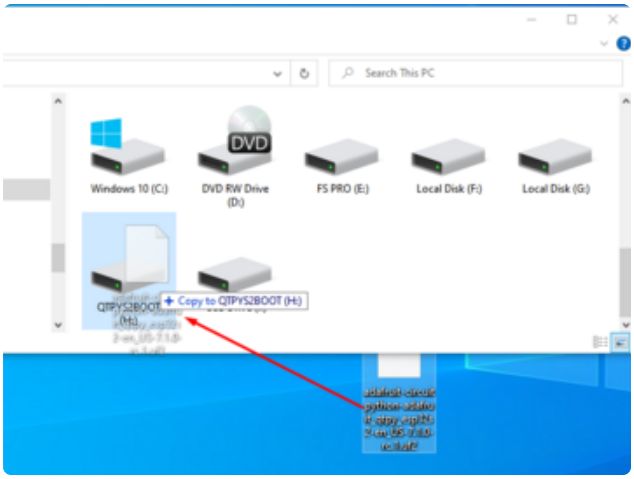
For this board, tap reset and wait for the LED to turn purple, and as soon as it turns purple, tap reset again. The second tap needs to happen while the LED is still purple.

Once successful, you will see the **RGB status LED(s)** turn green (highlighted in green above), and a disk drive ending in "**...BOOT**" should appear on your host computer. If you see red, try another port, or if you're using an adapter or hub, try without the hub, or different adapter or hub.

If double-clicking doesn't work the first time, try again. Sometimes it can take a few tries to get the rhythm right!

A lot of people end up using charge-only USB cables and it is very frustrating! **Make sure you have a USB cable you know is good for data sync.**

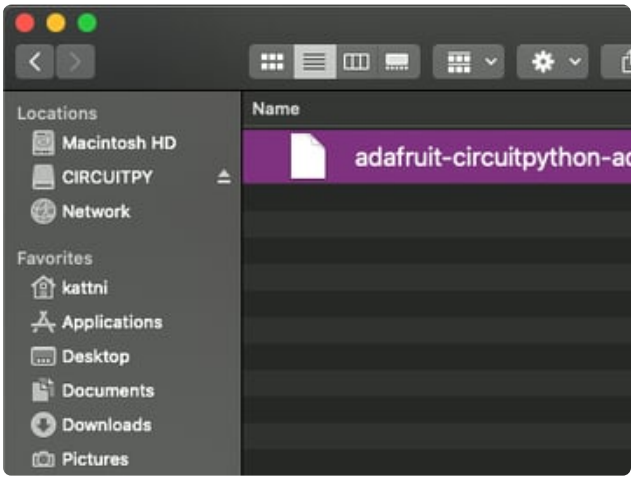
If after several tries, and verifying your USB cable is data-ready, you still cannot get to the bootloader, it is possible that the bootloader is missing or damaged. Check out the Factory Reset page for details on resolving this issue.



You will see a new disk drive appear called **QTPYS2BOOT**.

Drag the **adafruit_circuitpython_etc.uf2** file to **QTPYS2BOOT**.

Copy or drag the UF2 file you downloaded to the **BOOT** drive.



The **BOOT** drive will disappear and a new disk drive called **CIRCUITPY** will appear.

That's it!

CircuitPython Internet Test

One of the great things about most Espressif microcontrollers are their built-in WiFi capabilities. This page covers the basics of getting connected using CircuitPython.

The first thing you need to do is update your **code.py** to the following (it will error until WiFi details are added). Click the **Download Project Bundle** button to download the necessary libraries and the **code.py** file in a zip file. Extract the contents of the zip file, and copy the **entire lib folder** and the **code.py** file to your **CIRCUITPY** drive.

```
# SPDX-FileCopyrightText: 2020 Brent Rubell for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import os
import ipaddress
import ssl
import wifi
import socketpool
import adafruit_requests

# URLs to fetch from
```

```
TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"
JSON_QUOTES_URL = "https://www.adafruit.com/api/quotes.php"
JSON_STARS_URL = "https://api.github.com/repos/adafruit/circuitpython"

print("ESP32-S2 WebClient Test")

print(f"My MAC address: {[hex(i) for i in wifi.radio.mac_address]}")

print("Available WiFi networks:")
for network in wifi.radio.start_scanning_networks():
    print("\t%s\t\tRSSI: %d\tChannel: %d" % (str(network.ssid, "utf-8"),
                                             network.rssi, network.channel))
wifi.radio.stop_scanning_networks()

print(f"Connecting to {os.getenv('CIRCUITPY_WIFI_SSID')}")
wifi.radio.connect(os.getenv("CIRCUITPY_WIFI_SSID"),
os.getenv("CIRCUITPY_WIFI_PASSWORD"))
print(f"Connected to {os.getenv('CIRCUITPY_WIFI_SSID')}")
print(f"My IP address: {wifi.radio.ipv4_address}")

ping_ip = ipaddress.IPv4Address("8.8.8.8")
ping = wifi.radio.ping(ip=ping_ip)

# retry once if timed out
if ping is None:
    ping = wifi.radio.ping(ip=ping_ip)

if ping is None:
    print("Couldn't ping 'google.com' successfully")
else:
    # convert s to ms
    print(f"Pinging 'google.com' took: {ping * 1000} ms")

pool = socketpool.SocketPool(wifi.radio)
requests = adafruit_requests.Session(pool, ssl.create_default_context())

print(f"Fetching text from {TEXT_URL}")
response = requests.get(TEXT_URL)
print("-" * 40)
print(response.text)
print("-" * 40)

print(f"Fetching json from {JSON_QUOTES_URL}")
response = requests.get(JSON_QUOTES_URL)
print("-" * 40)
print(response.json())
print("-" * 40)

print()

print(f"Fetching and parsing json from {JSON_STARS_URL}")
response = requests.get(JSON_STARS_URL)
print("-" * 40)
print(f"CircuitPython GitHub Stars: {response.json()['stargazers_count']}")
print("-" * 40)

print("Done")
```

Your **CIRCUITPY** drive should resemble the following.

To get connected, the next thing you need to do is update the **settings.toml** file.

The settings.toml File

We expect people to share tons of projects as they build CircuitPython WiFi widgets. What we want to avoid is people accidentally sharing their passwords or secret tokens and API keys. So, we designed all our examples to use a **settings.toml** file, that is on your **CIRCUITPY** drive, to hold secret/private/custom data. That way you can share your main project without worrying about accidentally sharing private stuff.

If you have a fresh install of CircuitPython on your board, the initial **settings.toml** file on your **CIRCUITPY** drive is empty.

To get started, you can update the **settings.toml** on your **CIRCUITPY** drive to contain the following code.

```
# SPDX-FileCopyrightText: 2023 Adafruit Industries
#
# SPDX-License-Identifier: MIT

# This is where you store the credentials necessary for your code.
# The associated demo only requires WiFi, but you can include any
# credentials here, such as Adafruit IO username and key, etc.
CIRCUITPY_WIFI_SSID = "your-wifi-ssid"
CIRCUITPY_WIFI_PASSWORD = "your-wifi-password"
```

This file should contain a series of Python variables, each assigned to a string. Each variable should describe what it represents (say `wifi_ssid`), followed by an = (equals sign), followed by the data in the form of a Python string (such as `"my-wifi-password"` including the quote marks).

At a minimum you'll need to add/update your WiFi SSID and WiFi password, so do that now!

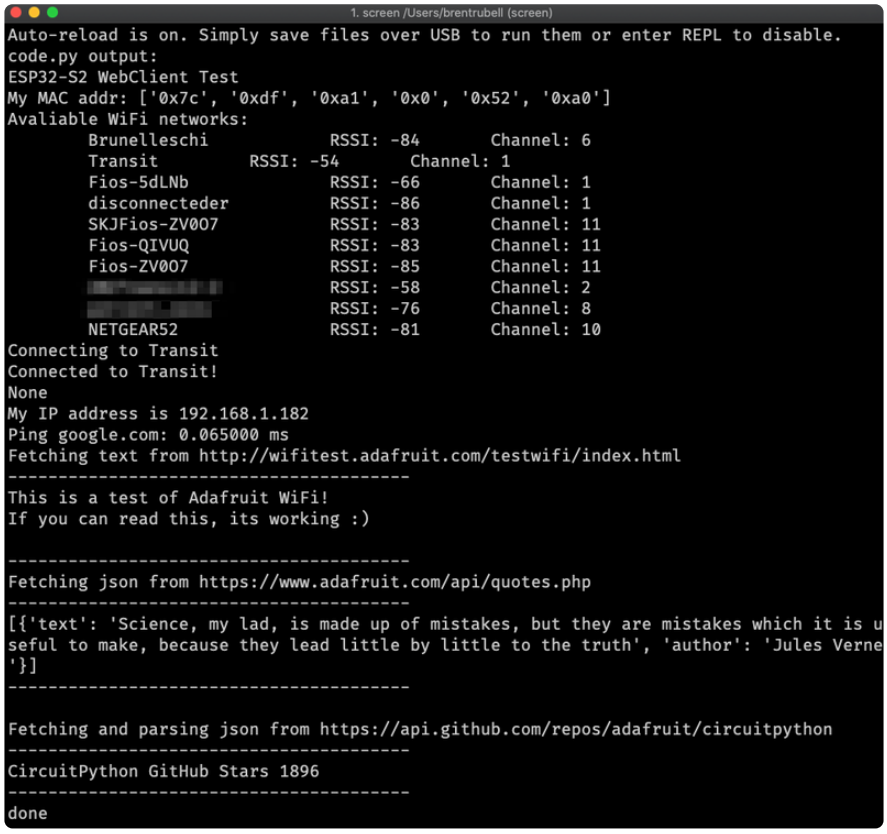
As you make projects you may need more tokens and keys, just add them one line at a time. See for example other tokens such as one for accessing GitHub or the Hackaday API. Other non-secret data like your timezone can also go here.

For the correct time zone string, look at http://worldtimeapi.org/timezones (https://adafru.it/EcP) and remember that if your city is not listed, look for a city in the same time zone, for example Boston, New York, Philadelphia, Washington DC, and Miami are all on the same time as New York.

Of course, don't share your **settings.toml** - keep that out of GitHub, Discord or other project-sharing sites.

> Don't share your settings.toml file! It has your passwords and API keys in it!

If you connect to the serial console, you should see something like the following:



In order, the example code...

Checks the ESP32's MAC address.

```
print(f"My MAC address: {[hex(i) for i in wifi.radio.mac_address]}")
```

Performs a scan of all access points and prints out the access point's name (SSID), signal strength (RSSI), and channel.

```
print("Available WiFi networks:")
for network in wifi.radio.start_scanning_networks():
    print("\t%s\t\tRSSI: %d\tChannel: %d" % (str(network.ssid, "utf-8"),
                                              network.rssi, network.channel))
wifi.radio.stop_scanning_networks()
```

Connects to the access point you defined in the **settings.toml** file, and prints out its local IP address.

```
print(f"Connecting to {os.getenv('WIFI_SSID')}")
wifi.radio.connect(os.getenv("WIFI_SSID"), os.getenv("WIFI_PASSWORD"))
print(f"Connected to {os.getenv('WIFI_SSID')}")
print(f"My IP address: {wifi.radio.ipv4_address}")
```

Attempts to ping a Google DNS server to test connectivity. If a ping fails, it returns `None`. Initial pings can sometimes fail for various reasons. So, if the initial ping is successful (`is not None`), it will print the echo speed in ms. If the initial ping fails, it will try one more time to ping, and then print the returned value. If the second ping fails, it will result in `"Ping google.com: None ms"` being printed to the serial console. Failure to ping does not always indicate a lack of connectivity, so the code will continue to run.

```
ping_ip = ipaddress.IPv4Address("8.8.8.8")
ping = wifi.radio.ping(ip=ping_ip) * 1000
if ping is not None:
    print(f"Ping google.com: {ping} ms")
else:
    ping = wifi.radio.ping(ip=ping_ip)
    print(f"Ping google.com: {ping} ms")
```

The code creates a socketpool using the wifi radio's available sockets. This is performed so we don't need to re-use sockets. Then, it initializes a a new instance of the requests (https://adafru.it/E9o) interface - which makes getting data from the internet really really easy.

```
pool = socketpool.SocketPool(wifi.radio)
requests = adafruit_requests.Session(pool, ssl.create_default_context())
```

To read in plain-text from a web URL, call `requests.get` - you may pass in either a http, or a https url for SSL connectivity.

```
print(f"Fetching text from {TEXT_URL}")
response = requests.get(TEXT_URL)
print("-" * 40)
print(response.text)
print("-" * 40)
```

Requests can also display a JSON-formatted response from a web URL using a call to `requests.get`.

```
print(f"Fetching json from {JSON_QUOTES_URL}")
response = requests.get(JSON_QUOTES_URL)
print("-" * 40)
print(response.json())
print("-" * 40)
```

Finally, you can fetch and parse a JSON URL using `requests.get`. This code snippet obtains the `stargazers_count` field from a call to the GitHub API.

```
print(f"Fetching and parsing json from {JSON_STARS_URL}")
response = requests.get(JSON_STARS_URL)
print("-" * 40)
print(f"CircuitPython GitHub Stars: {response.json()['stargazers_count']}")
print("-" * 40)
```

OK you now have your ESP32 board set up with a proper **settings.toml** file and can connect over the Internet. If not, check that your **settings.toml** file has the right SSID and password and retrace your steps until you get the Internet connectivity working!

IPv6 Networking

Starting in CircuitPython 9.2, IPv6 networking is available on most Espressif wifi boards. Socket-using libraries like **adafruit_requests** and **adafruit_ntp** will need to be updated to use the new APIs and for now can only connect to services on IPv4.

IPv6 connectivity & privacy

IPv6 addresses are divided into many special kinds, and many of those kinds (like those starting with **FC**, **FD**, **FE**) are private or local; Addresses starting with other prefixes like **2002:** and **2001:** are globally routable. In 2024, far from all ISPs and home networks support IPv6 internet connectivity. For more info consult resources like Wikipedia (https://adafru.it/1a4z). If you're interested in global IPv6 connectivity you can use services like Hurricane Electric (https://adafru.it/1a4A) to create an "IPv6 tunnel" (free as of 2024, but requires expertise and a compatible router or host computer to set up)

It's also important to be aware that, as currently implemented by Espressif, there are privacy concerns especially when these devices operate on the global IPv6 network: The device's unique identifier (its EUI-64 or MAC address) is used by default as part of its IPv6 address. This means that the device identity can be tracked across multiple networks by any service it connects to.

Enable IPv6 networking

Due to the privacy consideration, IPv6 networking is not automatically enabled. Instead, it must be explicitly enabled by a call to `start_dhcp_client` with the `ipv6=True` argument specified:

```
wifi.start_dhcp_client(ipv6=True)
```

Check IP addresses

The read-only `addresses` property of the `wifi.radio` object holds all addresses, including IPv4 and IPv6 addresses:

```
&gt;&gt;&gt; wifi.radio.addresses
('FE80::7EDF:A1FF:FE00:518C', 'FD5F:3F5C:FE50:0:7EDF:A1FF:FE00:518C', '10.0.3.96')
```

The `wifi.radio.dns` servers can be IPv4 or IPv6:

```
&gt;&gt;&gt; wifi.radio.dns
('FD5F:3F5C:FE50::1',)
&gt;&gt;&gt; wifi.radio.dns = ("1.1.1.1",)
&gt;&gt;&gt; wifi.radio.dns
('1.1.1.1',)
```

Ping v6 networks

`wifi.radio.ping` accepts v6 addresses and names:

```
&gt;&gt;&gt; wifi.radio.ping("google.com")
0.043
&gt;&gt;&gt; wifi.radio.ping("ipv6.google.com")
0.048
```

Create & use IPv6 sockets

Use the address family `socket.AF_INET6`. After the socket is created, use methods like `connect`, `send`, `recfrom_into`, etc just like for IPv4 sockets. This code snippet shows communicating with a private-network NTP server; this IPv6 address will not work on your network:

```
&gt;&gt;&gt; ntp_addr = ("fd5f:3f5c:fe50::20e", 123)
&gt;&gt;&gt; PACKET_SIZE = 48
&gt;&gt;&gt;
&gt;&gt;&gt; buf = bytearray(PACKET_SIZE)
&gt;&gt;&gt; with socket.socket(socket.AF_INET6, socket.SOCK_DGRAM) as s:
...     s.settimeout(1)
...     buf[0] = 0b0010_0011
...     s.sendto(buf, ntp_addr)
...     print(s.recvfrom_into(buf))
...     print(buf)
...
48
(48, ('fd5f:3f5c:fe50::20e', 123))
bytearray(b'$\x01\x03\xeb\x00\x00\x00\x00\x00\x00\x00GGPS\x00\xeaA0h\x07s;
\xc0\x00\x00\x00\x00\x00\x00\x00\xeaA0n\xeb4\x82-\xeaA0n\xebAU\xb1')
```

Lamp Hardware

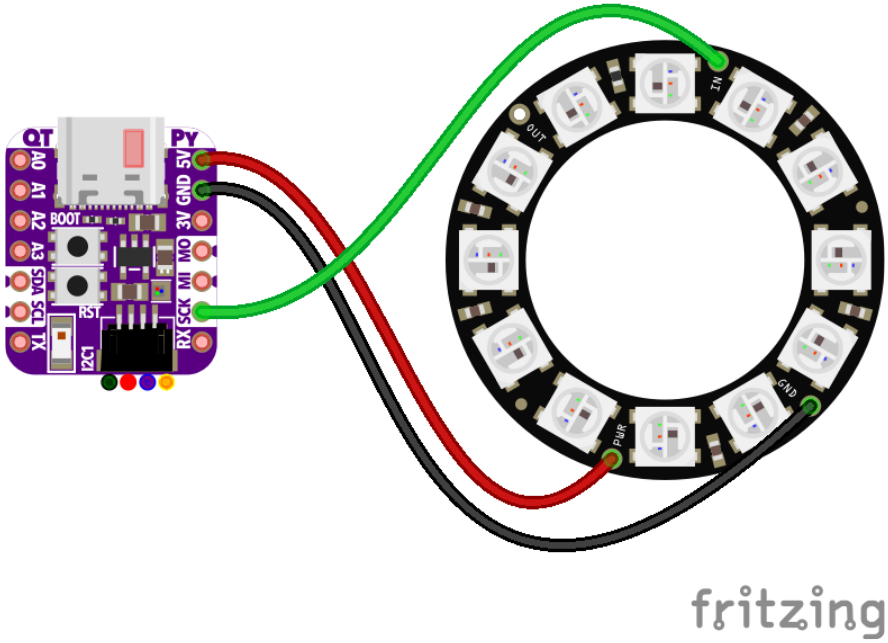
Be creative here! Other NeoPixel products could be used - like strips. And the lamp housing can be anything fun. We document what we used just to provide the info.

Wiring

To wire up the NeoPixels:

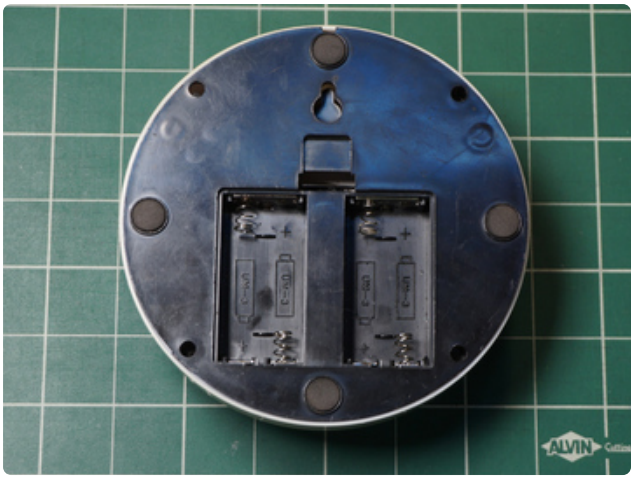
- **QT Py 5V** to **NeoPixel PWR**
- **QT Py GND** to **NeoPixel GND**
- **QT Py SCK*** to **NeoPixel IN**

* or any available GPIO pin

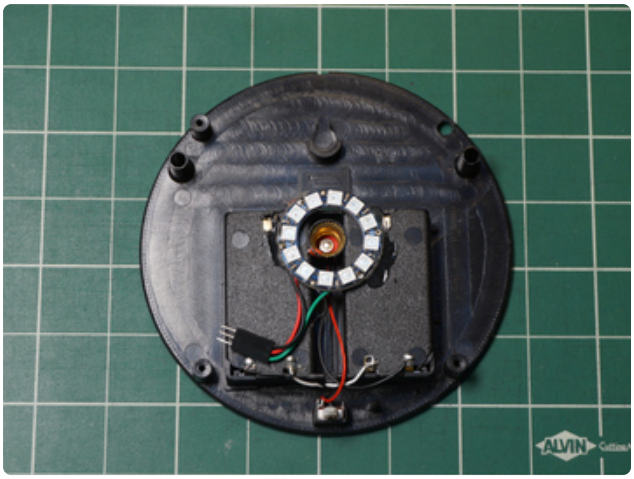


Lamp Assembly

Honestly, we were hoping to find something a little more fun than a generic dome. Anything with good neutral diffusion would work, like a plastic dinosaur! But when we went looking, the thrift store gods only offered up this 50 cent battery powered closet light. It'll work to demonstrate the general idea though, which is to basically just shove everything into whatever is used for the lamp.

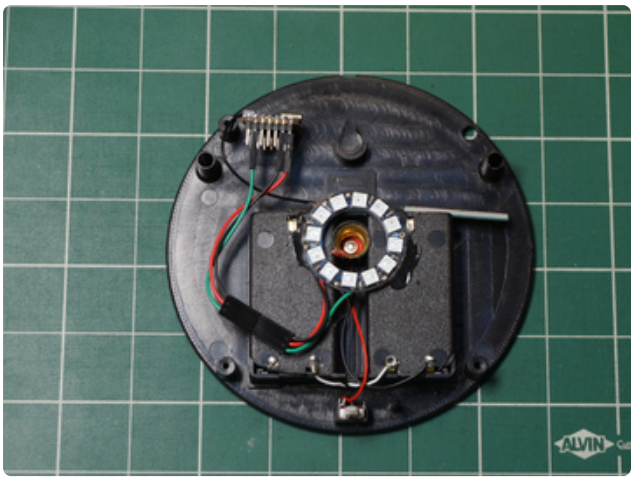


Four screws were removed to allow removing the bottom plate (black plastic).



The NeoPixel ring was hot glued on the inside of the bottom plate in the center.

The original lamp socket can be seen in the center of the NeoPixel ring.



Then the QT Py ESP32-S2 was wired to the NeoPixels.

These could be direct wired. The wiring harness shown here is not necessary.

And then the top was put back on. Done!

For power, a USB cable was fed through the open battery compartment and connected to the QT Py ESP32-S2's USB C connector.

Code Using NTP

There are a couple of ways to grab time from the Internet. First we'll show how this can be done using Network Time Protocol (https://adafru.it/Igf) (NTP). This has the benefit of not requiring any accounts or logins - just grab the time from an NTP server. There's even an NTP CircuitPython library (https://adafru.it/106c) that can be used.

However, the offset to local time must be manually set and adjusted for daylight savings (if you are affected)

To get the NTP code, click on the **Download Project Bundle** button in the window below. It will download as a zipped folder.

```
# SPDX-FileCopyrightText: 2022 Carter Nelson for Adafruit Industries
#
# SPDX-License-Identifier: MIT

from os import getenv
import time
import board
import rtc
import socketpool
import wifi
import adafruit_ntp
import neopixel

# --| User Config |-------------------------------
TZ_OFFSET = -7  # time zone offset in hours from UTC
WAKE_UP_HOUR = 7  # alarm time hour (24hr)
WAKE_UP_MIN = 30  # alarm time minute
SLEEP_COLOR = (0, 25, 150)  # sleepy time color as tuple
WAKEUP_COLOR = (255, 100, 0)  # wake up color as tuple
FADE_STEPS = 100  # wake up fade animation steps
FADE_DELAY = 0.1  # wake up fade animation speed
NEO_PIN = board.SCK  # neopixel pin
NEO_CNT = 12  # neopixel count
# ----------------------------------------------
```

```
# Set up NeoPixels
pixels = neopixel.NeoPixel(NEO_PIN, NEO_CNT)

# Get WiFi details, ensure these are setup in settings.toml
ssid = getenv("CIRCUITPY_WIFI_SSID")
password = getenv("CIRCUITPY_WIFI_PASSWORD")

# Connect to local network
try:
    wifi.radio.connect(ssid, password)
except ConnectionError:
    print("Wifi failed to connect.")
    while True:
        pixels.fill(0)
        time.sleep(0.5)
        pixels.fill(0x220000)
        time.sleep(0.5)

print("Wifi connected.")

# Get current time using NTP
print("Fetching time from NTP.")
pool = socketpool.SocketPool(wifi.radio)
ntp = adafruit_ntp.NTP(pool, tz_offset=TZ_OFFSET)
rtc.RTC().datetime = ntp.datetime

# Fill with sleepy time colors
pixels.fill(SLEEP_COLOR)

# Wait for wake up time
now = time.localtime()
print("Current time: {:2}:{:02}".format(now.tm_hour, now.tm_min))
print("Waiting for alarm {:2}:{:02}".format(WAKE_UP_HOUR, WAKE_UP_MIN))
while not (now.tm_hour == WAKE_UP_HOUR and now.tm_min == WAKE_UP_MIN):
    # just sleep until next time check
    time.sleep(30)
    now = time.localtime()

# Sunrise animation
print("Waking up!")
r1, g1, b1 = SLEEP_COLOR
r2, g2, b2 = WAKEUP_COLOR
dr = (r2 - r1) / FADE_STEPS
dg = (g2 - g1) / FADE_STEPS
db = (b2 - b1) / FADE_STEPS

for _ in range(FADE_STEPS):
    r1 += dr
    g1 += dg
    b1 += db
    pixels.fill((int(r1), int(g1), int(b1)))
    time.sleep(FADE_DELAY)

print("Done.")
```

Open the zip file and copy **code.py** to your **CIRCUITPY** folder. Also copy the contents of the **lib** folder to the **CIRCUITPY/lib** folder.

Now checkout the **Using the Alarm** section for how to configure things.

Code Using AIO

Using Adafruit IO (https://adafru.it/Pla) makes things even easier for getting local time. All the time zone, day light saving time, etc. details are taken care of by the AIO server. This does require setting up an AIO account. However, the time query service is available with even the free level account.

Add AIO info to settings.toml

In addition to the wifi information that was previously added to the **settings.toml** file, be sure it also includes your AIO username and API key. These should be added in the secrets dictionary as `ADAFRUIT_AIO_USERNAME` and `ADAFRUIT_AIO_KEY`.

Checkout this Learn Guide page for more information:

Getting Date and Time

https://adafru.it/RXe

Then, to get the AIO based code, click on the **Download Project Bundle** button in the window below. It will download as a zipped folder.

```
# SPDX-FileCopyrightText: 2022 Carter Nelson for Adafruit Industries
#
# SPDX-License-Identifier: MIT

from os import getenv
import time
import ssl
import board
import rtc
import wifi
import socketpool
import adafruit_requests
import neopixel

# --| User Config |-------------------------------
WAKE_UP_HOUR = 7  # alarm time hour (24hr)
WAKE_UP_MIN = 30  # alarm time minute
SLEEP_COLOR = (0, 25, 150)  # sleepy time color as tuple
WAKEUP_COLOR = (255, 100, 0)  # wake up color as tuple
FADE_STEPS = 100  # wake up fade animation steps
FADE_DELAY = 0.1  # wake up fade animation speed
NEO_PIN = board.SCK  # neopixel pin
NEO_CNT = 12  # neopixel count
# ----------------------------------------------

# Get WiFi details and Adafruit IO keys, ensure these are setup in settings.toml
# (visit io.adafruit.com if you need to create an account, or if you need your
Adafruit IO key.)
ssid = getenv("CIRCUITPY_WIFI_SSID")
password = getenv("CIRCUITPY_WIFI_PASSWORD")
aio_username = getenv("ADAFRUIT_AIO_USERNAME")
aio_key = getenv("ADAFRUIT_AIO_KEY")

# Set up NeoPixels
pixels = neopixel.NeoPixel(NEO_PIN, NEO_CNT)
```

```
# Setup AIO time query URL
TIME_URL = "https://io.adafruit.com/api/v2/"
TIME_URL += aio_username
TIME_URL += "/integrations/time/strftime?x-aio-key="
TIME_URL += aio_key
TIME_URL += "&fmt=%25Y%3A%25m%3A%25d%3A%25H%3A%25M%3A%25S"

# Connect to local network
try:
    wifi.radio.connect(ssid, password)
except ConnectionError:
    print("Wifi failed to connect.")
    while True:
        pixels.fill(0)
        time.sleep(0.5)
        pixels.fill(0x220000)
        time.sleep(0.5)

print("Wifi connected.")

# Get current time using AIO
print("Fetching time.")
pool = socketpool.SocketPool(wifi.radio)
requests = adafruit_requests.Session(pool, ssl.create_default_context())
response = [int(x) for x in requests.get(TIME_URL).text.split(":")]
response += [0, 0, -1]
rtc.RTC().datetime = time.struct_time(response)

# Fill with sleepy time colors
pixels.fill(SLEEP_COLOR)

# Wait for wake up time
now = time.localtime()
print("Current time: {}:{}".format(now.tm_hour, now.tm_min))
print("Waiting for alarm {}:{}".format(WAKE_UP_HOUR, WAKE_UP_MIN))
while not (now.tm_hour == WAKE_UP_HOUR and now.tm_min == WAKE_UP_MIN):
    # just sleep until next time check
    time.sleep(30)
    now = time.localtime()

# Sunrise animation
print("Waking up!")
r1, g1, b1 = SLEEP_COLOR
r2, g2, b2 = WAKEUP_COLOR
dr = (r2 - r1) / FADE_STEPS
dg = (g2 - g1) / FADE_STEPS
db = (b2 - b1) / FADE_STEPS

for _ in range(FADE_STEPS):
    r1 += dr
    g1 += dg
    b1 += db
    pixels.fill((int(r1), int(g1), int(b1)))
    time.sleep(FADE_DELAY)

print("Done.")
```

Open the zip file and copy **code.py** to your **CIRCUITPY** folder. Also copy the contents of the **lib** folder to the **CIRCUITPY/lib** folder.

Now checkout the **Using the Alarm** section for how to configure things.

Using the Alarm

Both versions of the code work in essentially the same way. The only difference is in how they grab time from the Internet.

Connect the QT Py to a PC with a good data capable USB cable so that the **CIRCUITPY** folder shows up. Then open up **code.py** in a text editor.

Look for these lines near the top of the code listing:

```
# --| User Config |------------------------------
WAKE_UP_HOUR = 7  # alarm time hour (24hr)
WAKE_UP_MIN = 30  # alarm time minute
SLEEP_COLOR = (0, 25, 150)  # sleeping time color as tuple
WAKEUP_COLOR = (255, 100, 0)  # wake up color as tuple
FADE_STEPS = 100  # wake up fade animation steps
FADE_DELAY = 0.1  # wake up fade animation speed
NEO_PIN = board.SCK  # neopixel pin
NEO_CNT = 12  # neopixel count
# ------------------------------------------------
```

To set the alarm time, change `WAKE_UP_HOUR` and `WAKE_UP_MIN` as desired. Note that the hour uses 24 hour format.

While waiting for the alarm time, the `SLEEP_COLOR` will be shown on the NeoPixels. Once the alarm time is reached, the color will fade to `WAKEUP_COLOR`. Use (red, green, blue) tuples to change either of these colors.

The speed of the fade effect can be altered by changing `FADE_STEPS` and `FADE_DELAY`.

If you end up using different NeoPixels or attach them to a different pin, change `NEO_PIN` and `NEO_CNT` as needed.

NTP Version Only

For the NTP version of the alarm, there is also a `TZ_OFFSET` parameter. Use that to set your local time in terms of hourly offset from UTC.

List of UTC Offsets

https://adafru.it/106f

Arming and Resetting

Once the alarm time is set, simply reset the board or cycle power. The board will fetch the current time and then wait for the alarm time.

Once the alarm triggers, it will stay lit. To re-arm the alarm, simply power cycle the board. Pressing the reset button would also work, if the reset button is reachable.

Troubleshooting

If the board fails to connect to the local wifi network, the NeoPixels should blink red. If that happens, double check the `ssid` and `password` settings in your **secrets.py** file.

For other issues, connecting to the serial console is the best option. There are several print statements in the code. Being able to see these, and any other error messages, will help determine where and why things are not working. For details on connecting to the serial console, see here:

Connecting to the Serial Console

https://adafru.it/Bec