adafruit learning system

MagTag Dishwasher Status

Created by Carter Nelson

Last updated on 2021-05-19 05:19:47 PM EDT

Guide Contents

Overview

This project was inspired by this tweet (https://adafru.it/PQb) from @kevcody (https://adafru.it/PQc). They used an Adafruit MagTag (https://adafru.it/OMb) to help with Modern Problem #4836:

> There are dishes in the dishwasher, but has it been run yet? Are the ones in there dirty or clean?

This is an excellent use of the MagTag's EPD display. By displaying either DIRTY or CLEAN, the current state of the dishwasher is known. In their approach, they used the MagTag's button to change the indication - with nice little icons to let you know button function.

We thought this idea would make a good example for using **pin alarms** in conjunction with **deep sleep**, which are new features recently added to CircuitPython (https://adafru.it/EFq). Instead of looping forever reading button state and consuming power, we can instead go to sleep and only wake up when needed. This will let your MagTag run for days, or even weeks, on a single battery charge.

Parts

Here are the parts you'll need for this project. You can get them individually:

Adafruit MagTag - 2.9" Grayscale E-Ink WiFi Display

The Adafruit MagTag combines the new ESP32-S2 wireless module and a 2.9" grayscale E-Ink display to make a low-power IoT display that can show data on its screen even when power...

Out of Stock

Out of Stock

Lithium Ion Polymer Battery with Short Cable - 3.7V 420mAh

Lithium ion polymer (also known as 'lipo' or 'lipoly') batteries are thin, light and powerful. The output ranges from 4.2V when completely charged to 3.7V. This battery...

$6.95

In Stock

Add to Cart

Mini Magnet Feet for RGB LED Matrices (Pack of 4)

Got a glorious RGB Matrix project you want to mount and display in your workspace or home? If you have one of the matrix panels listed below, you'll need a pack of these...

Out of Stock

Out of Stock

OR these items are also available together as a kit:

Adafruit MagTag Starter Kit - 2.9" Grayscale E-Ink WiFi Display

The Adafruit MagTag combines the new ESP32-S2 wireless module and a 2.9" grayscale E-Ink display to make a low-power IoT display that can show data on its screen...

Out of Stock

Out of Stock

And you'll need a USB cable to connect the MagTag to your computer to upload the code:

USB Type A to Type C Cable - approx 1 meter / 3 ft long

As technology changes and adapts, so does Adafruit. This USB Type A to Type C cable will help you with the transition to USB C, even if you're still...

$4.95

In Stock

Add to Cart

Install CircuitPython

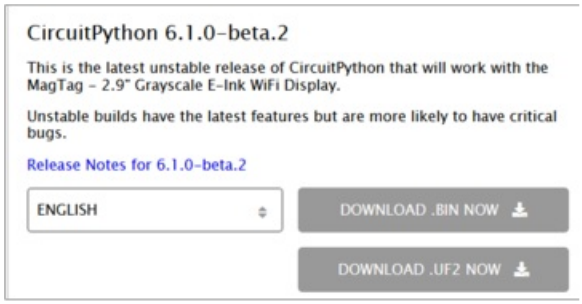
CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/BeZ) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files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

Set Up CircuitPython

Follow the steps to get CircuitPython installed on your MagTag.

https://adafru.it/OBd

https://adafru.it/OBd



Click the link above and download the latest .BIN and .UF2 file

(depending on how you program the ESP32S2 board you may need one or the other, might as well get both)

Download and save it to your desktop (or wherever is handy).

Plug your MagTag into your computer using a known-good USB cable.

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

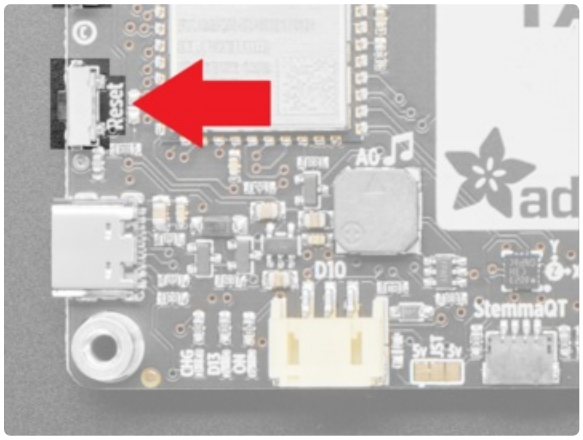
Option 1 - Load with UF2 Bootloader

This is by far the easiest way to load CircuitPython. **However it requires your board has the UF2 bootloader installed. Some early boards do not (we hadn't written UF2 yet!) - in which case you can load using the built in ROM bootloader.**

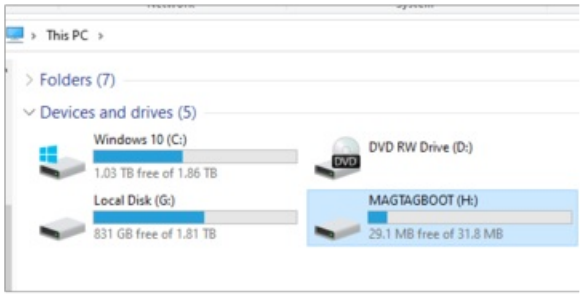
Still, try this first!

Try Launching UF2 Bootloader

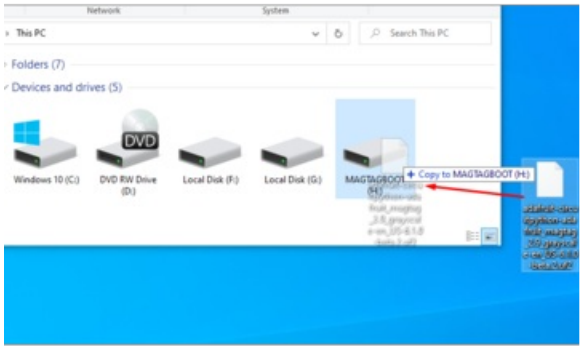
Loading CircuitPython by drag-n-drop UF2 bootloader is the easier way and we recommend it. If you have a MagTag where the front of the board is black, your MagTag came with UF2 already on it.



Launch UF2 by **double-clicking** the Reset button (the one next to the USB C port). You may have to try a few times to get the timing right.



If the UF2 bootloader is installed, you will see a new disk drive appear called **MAGTAGBOOT**



Copy the **UF2** file you downloaded at the first step of this tutorial onto the **MAGTAGBOOT** drive

If you're using Windows and you get an error at the end of the file copy that says **Error from the file copy, Error 0x800701B1: A device which does not exist was specified.** You can ignore this error, the bootloader sometimes disconnects without telling Windows, the install completed just fine and you can continue. If its really annoying, you can also upgrade the bootloader (the latest version of the UF2 bootloader fixes this warning) (https://adafru.it/Pfk)

Your board should auto-reset into CircuitPython, or you may need to press reset. A **CIRCUITPY** drive will appear. You're done! Go to the next pages.

Option 2 - Use esptool to load BIN file

If you have an original MagTag with while soldermask on the front, we didn't have UF2 written for the ESP32S2 yet so it will not come with the UF2 bootloader.

You can upload with **esptool** to the ROM (hardware) bootloader instead!

Follow the initial steps found in the Run esptool and check connection section of the ROM Bootloader page (https://adafru.it/OBc) to verify your environment is set up, your board is successfully connected, and which port it's using.

In the final command to write a binary file to the board, replace the port with your port, and replace "firmware.bin" with the the file you downloaded above.

The output should look something like the output in the image.

Press reset to exit the bootloader.

Your **CIRCUITPY** drive should appear!

You're all set! Go to the next pages.

Option 3 - Use Chrome Browser To Upload BIN file

If for some reason you cannot get esptool to run, you can always try using the Chrome-browser version of esptool we have written. This is handy if you don't have Python on your computer, or something is really weird with your setup that makes esptool not run (which happens sometimes and isn't worth debugging!) You can follow along on the Web Serial ESPTool (https://adafru.it/Pdq) page and either load the UF2 bootloader and then come back to Option 1 on this page, or you can download the CircuitPython BIN file directly using the tool in the same manner as the bootloader.

Project Specific Software Setup

Bitmaps

This project uses the following BMP image files. Make sure you have a copy of them to the **CIRCUITPY** drive which appears when you connect your MagTag to your computer via a known good USB cable. You can download them here or via the project zip link in the code sections.

https://adafru.it/PQd

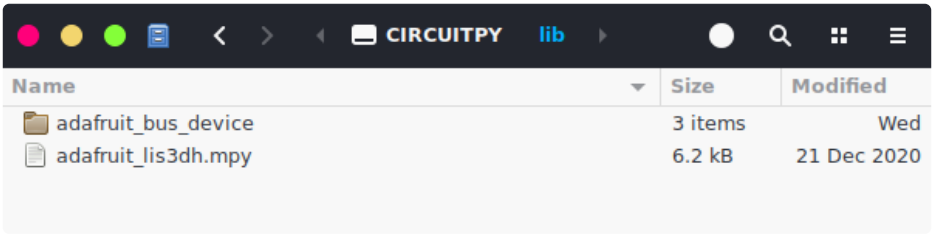
https://adafru.it/PQd

https://adafru.it/PQe

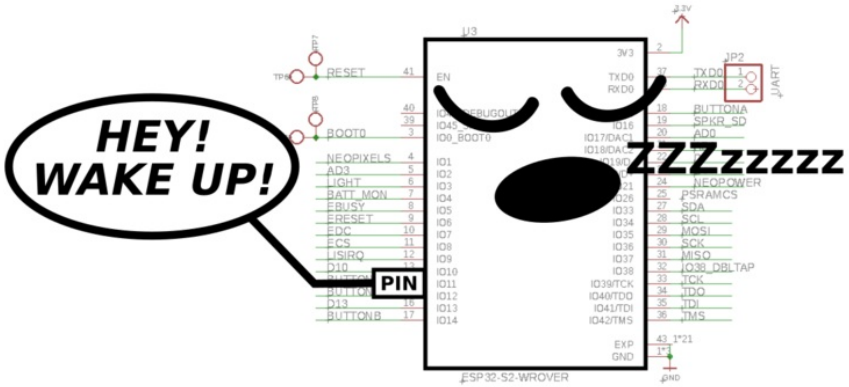
https://adafru.it/PQe

Libraries

Since we end up only using a few of the MagTag's features, this project does not use the MagTag specific library. Therefore, you only need the following in your **CIRCUITPY/lib** folder:



Pin Alarm Basics



Putting the MagTag to sleep is pretty easy, it's just a single line of code. Once you've imported the alarm module (https://adafru.it/PQf), you can put the MagTag into **deep sleep** (the other option is **light sleep**) with something like:

```
alarm.exit_and_deep_sleep_until_alarms(alarm)
```

The `alarm` parameter, which also comes from the alarm module, is one or more of several available alarm types:

- **time**
- **pin**
- **touch**

The difference between these is the actual trigger source for the alarm. However, the general idea is the same with any of them - when the alarm condition is met, they wake up the MagTag from sleep.

For example, the `time` based alarm is very much like the alarms we humans use to wake up in the morning. We set a time and after it has elapsed, the alarm goes off. This is good for scheduling something to run at a certain time of the day. Checkout these projects for some example usages:

- MagTag Tides Viewer (https://adafru.it/PcN)
- MagTag Daily Weather Forecast Display (https://adafru.it/PB-)

In this project we use the `pin` based alarm. This refers to an input **pin** on the processor and the alarm will trigger when the pin becomes **HIGH** (`True`) or **LOW** (`False`). This is analogous to you being woken up by a random noise instead of a time based alarm clock. The processor sleeps until this occurs, be it 5 seconds or 5 days.

Sources of Pin Change

The pin based alarm is general purpose. From the point of view of the processor, it only cares about the change in the pin state - like high or low. It does not care what actually caused that.

In this guide we'll look at two examples for generating a pin alarm:

- button(s)
- interrupt from external sensor (accelerometer)

Buttons are a good first example, since they are conceptually simple - you press a button and the pin changes. The sensor based interrupt is more complex since it requires the additional steps of properly configuring the sensor.

So....let's start with the simple button example.

Wake On Button

OK, here's our first example which uses button presses to wake from sleep.

The MagTag has four buttons on the front. They are wired to the MagTag in such a way that their associated pins read as follows:

- **HIGH** = `True` = not pressed
- **LOW** = `False` = pressed

So we want to configure a pin alarm that triggers when its value is `False` so it goes off when a button is pressed.

We can actually have more than one alarm. So one idea would be to wake up when any of the buttons are pressed. However, for technical reasons, we are limited to a maximum of two LOW pin alarms. So we just pick any two buttons - like the two on the left.

These are the lines of code that set up the pin alarms:

```
# set up pin alarms
buttons = (board.BUTTON_A, board.BUTTON_B)  # pick any two
pin_alarms = [alarm.pin.PinAlarm(pin=pin, value=False, pull=True) for pin in buttons]
```

Sleep Memory

The MagTag's EPD (eInk Display) will always be showing one of two indications. If the current indication is what you want, there's no need to do anything. When a button is pressed and it wakes up, we want the MagTag to show the other indication. But when it wakes up, how does the MagTag know what the current indication is? It doesn't.

To get around this we can use the `sleep_memory` feature of the alarm module. This is general purpose

memory that persists between sleep cycles. So we store a simple toggle value there that we change each time we wake up. Then we base the current indication on this toggle value.

That's what this line of code does:

```
# toggle saved state
alarm.sleep_memory[0] = not alarm.sleep_memory[0]
```

Code

Here's the code. Download this and save it as **code.py** into your **CIRCUITPY** folder so it will run automatically. If you click Download: Project Zip, you will get a zip file with the bitmap images used in the guide also.

```
import time
import board
import alarm
import displayio

# get the display
epd = board.DISPLAY
epd.rotation = 270

# set up pin alarms
buttons = (board.BUTTON_A, board.BUTTON_B)  # pick any two
pin_alarms = [alarm.pin.PinAlarm(pin=pin, value=False, pull=True) for pin in buttons]

# toggle saved state
alarm.sleep_memory[0] = not alarm.sleep_memory[0]

# set bitmap
bmp_file = "clean.bmp" if alarm.sleep_memory[0] else "dirty.bmp"

# show bitmap
with open(bmp_file, "rb") as fp:
    bitmap = displayio.OnDiskBitmap(fp)
    tile_grid = displayio.TileGrid(bitmap, pixel_shader=displayio.ColorConverter())
    group = displayio.Group(max_size=1)
    group.append(tile_grid)
    epd.show(group)
    time.sleep(epd.time_to_refresh + 0.01)
    epd.refresh()
    while epd.busy:
        pass

# go to sleep
alarm.exit_and_deep_sleep_until_alarms(*pin_alarms)
```

Wake On Flip

Now let's get a little fancier and use a sensor generated interrupt.

Configuring The Sensor

The MagTag includes a LIS3DH accelerometer, which has the ability to generate an interrupt for various events. There are numerous options, like shake, tap, free fall, etc. There are also numerous registers that come into play, so this requires additional setup. You can checkout the datasheet for all the details.

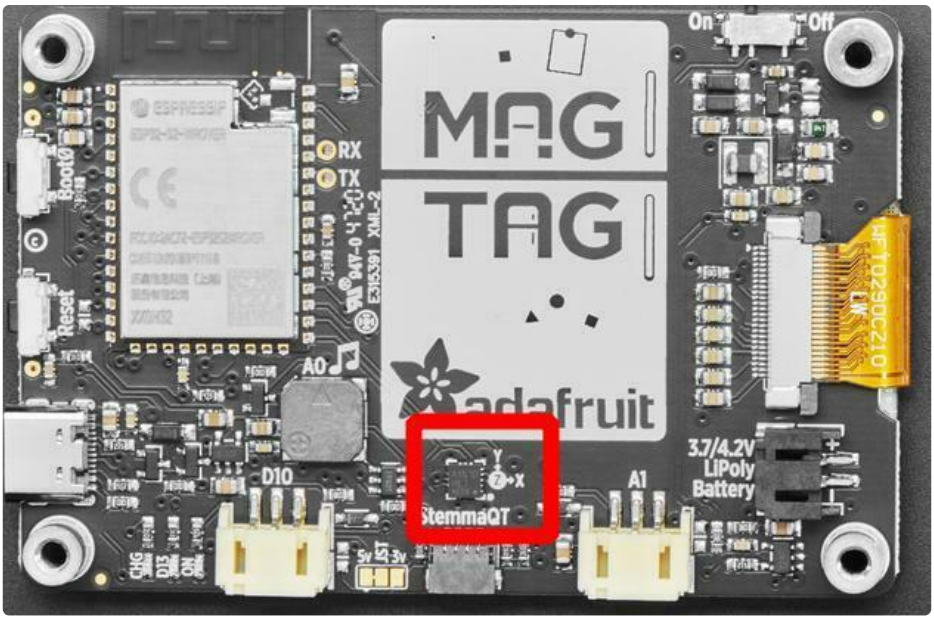
We ended up just copying the recommended setup we found in this ST Design Tip memo:

https://adafru.it/PQA

That's what these lines of code do:

```
lis._write_register_byte(0x20, 0x3F)  # low power mode with ODR = 25Hz
lis._write_register_byte(0x22, 0x40)  # AOI1 interrupt generation is routed to INT1 pin
lis._write_register_byte(0x23, 0x80)  # FS = ±2g low power mode with BDU bit enabled
lis._write_register_byte(
    0x24, 0x0C
)  # Interrupt signal on INT1 pin is latched with D4D_INT1 bit enabled
lis._write_register_byte(
    0x32, 0x20
)  # Threshold = 32LSBs * 15.625mg/LSB = 500mg. (~30 deg of tilt)
lis._write_register_byte(0x33, 0x01)  # Duration = 1LSBs * (1/25Hz) = 0.04s
```

The one difference for our case is we only care about two orientations - up and down. If you look on the back of the MagTag, you'll see the LIS3DH along with a little axis diagram:



So we can see that only the Y axis is involved for determining up / down orientation. We don't care about Z or X. Therefore, we only enable those bits in the associated register:

INT1_CFG (30h)

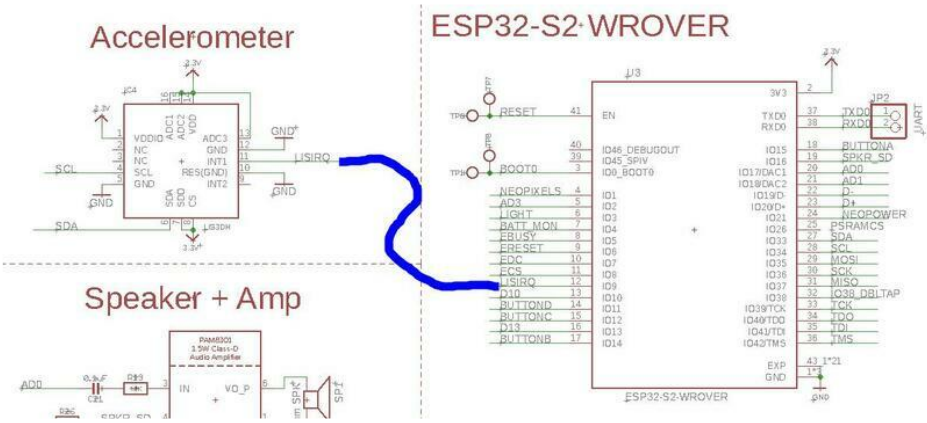
Table 46. INT1_CFG register

AOI	6D	ZHIE/ ZUPE	ZLIE/ ZDOWNE	YHIE/ YUPE	YLIE/ YDOWNE	XHIE/ XUPE	XLIE/ XDOWNE

Look at the variable `irq_config` in the code to see how this is done.

Pin Alarm from Sensor

OK, so the LIS3DH is configured to generate (output) an interrupt on its INT1 pin. How does the ESP32-S2 see that (input) on its end? Very simple. The LIS3DH INT1 pin is routed to IO9 on the ESP32-S2:



Now it's just like the button example. We set up a pin alarm on the ESP32-S2 pin connected to the LIS3DH INT1 pin. In CircuitPython, this pin is accessed using `board.ACCELEROMETER_INTERRUPT`. The interrupt pin goes HIGH when it fires, so we set up the alarm to trigger on a value of `True`.

That's what this line of code does:

```
pin_alarm = alarm.pin.PinAlarm(pin=board.ACCELEROMETER_INTERRUPT, value=True)
```

Code

Here's the complete code for the version that will change (wake) when flipped. Download this and save it as **code.py** into your **CIRCUITPY** folder so it will run automatically. Use Download: Project Zip to get a zip file with the bitmap images also, if not downloaded earlier.

```
import time
import board
import alarm
import displayio
import adafruit_lis3dh

# get the display
epd = board.DISPLAY

# set up accelerometer
lis = adafruit_lis3dh.LIS3DH_I2C(board.I2C(), address=0x19)

# See: ST Design Tip DT0008 - Simple screen rotation using
#      the accelerometer built-in 4D detection interrupt
# pylint: disable=protected-access
lis._write_register_byte(0x20, 0x3F)  # low power mode with ODR = 25Hz
```

```
lis._write_register_byte(0x22, 0x40)  # AOI1 interrupt generation is routed to INT1 pin
lis._write_register_byte(0x23, 0x80)  # FS = ±2g low power mode with BDU bit enabled
lis._write_register_byte(
    0x24, 0x0C
)  # Interrupt signal on INT1 pin is latched with D4D_INT1 bit enabled
lis._write_register_byte(
    0x32, 0x20
)  # Threshold = 32LSBs * 15.625mg/LSB = 500mg. (~30 deg of tilt)
lis._write_register_byte(0x33, 0x01)  # Duration = 1LSBs * (1/25Hz) = 0.04s

# read to clear
_ = lis._read_register_byte(0x31)

# get current accel values
_, y, _ = lis.acceleration

# update based on orientation
if y > 0:
    # upside up
    bmp_file = "clean.bmp"
    rotation = 270
    irq_config = 0b01000100
else:
    # upside down
    bmp_file = "dirty.bmp"
    rotation = 90
    irq_config = 0b01001000

# show bitmap
epd.rotation = rotation
with open(bmp_file, "rb") as fp:
    bitmap = displayio.OnDiskBitmap(fp)
    tile_grid = displayio.TileGrid(bitmap, pixel_shader=displayio.ColorConverter())
    group = displayio.Group(max_size=1)
    group.append(tile_grid)
    epd.show(group)
    time.sleep(epd.time_to_refresh + 0.01)
    epd.refresh()
    while epd.busy:
        pass

# config accelo irq
lis._write_register_byte(0x30, irq_config)

# go to sleep
pin_alarm = alarm.pin.PinAlarm(pin=board.ACCELEROMETER_INTERRUPT, value=True)
alarm.exit_and_deep_sleep_until_alarms(pin_alarm)
```