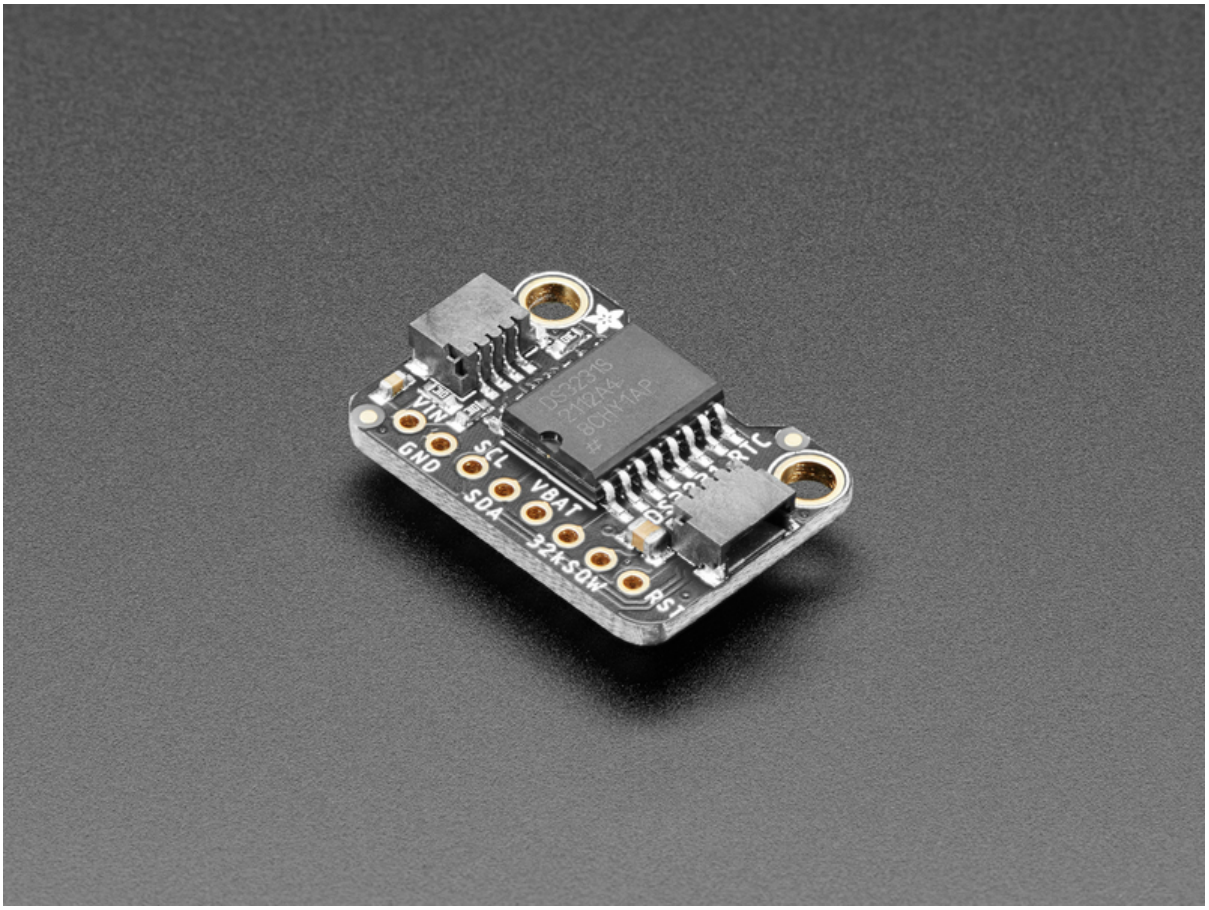
Adafruit DS3231 Precision RTC Breakout

Created by lady ada

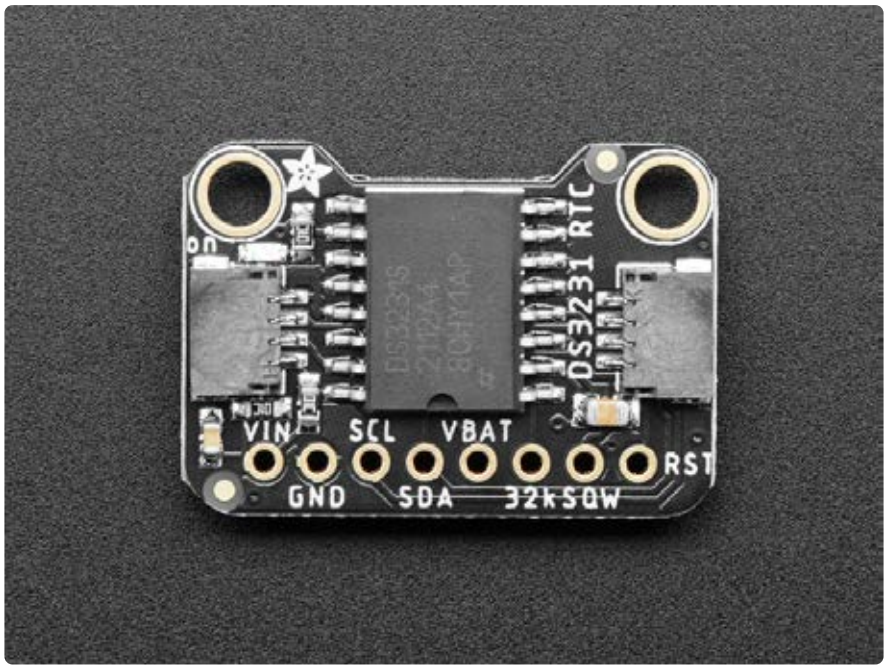


https://learn.adafruit.com/adafruit-ds3231-precision-rtc-breakout

Last updated on 2024-06-03 01:52:11 PM EDT

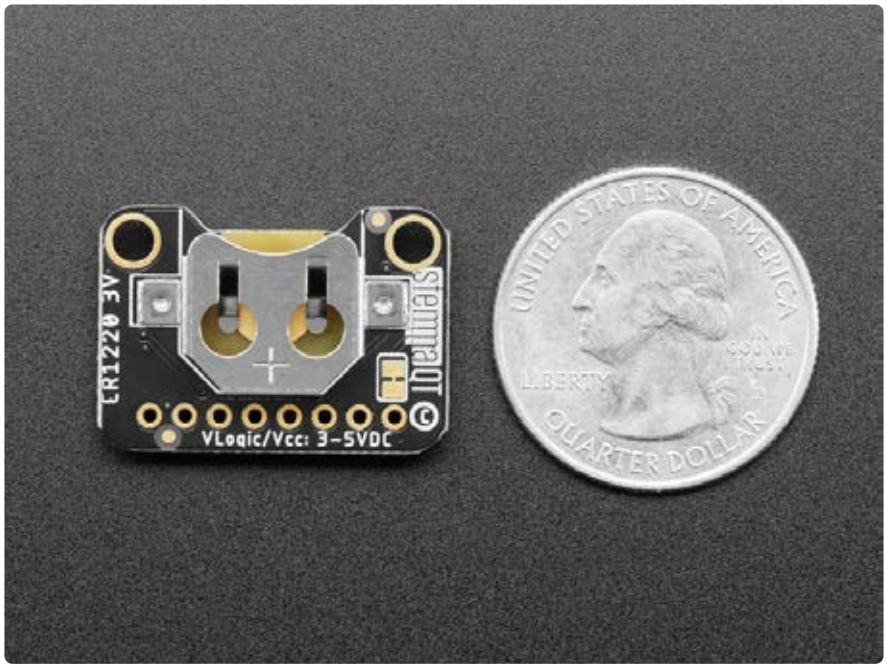
Table of Contents

Overview



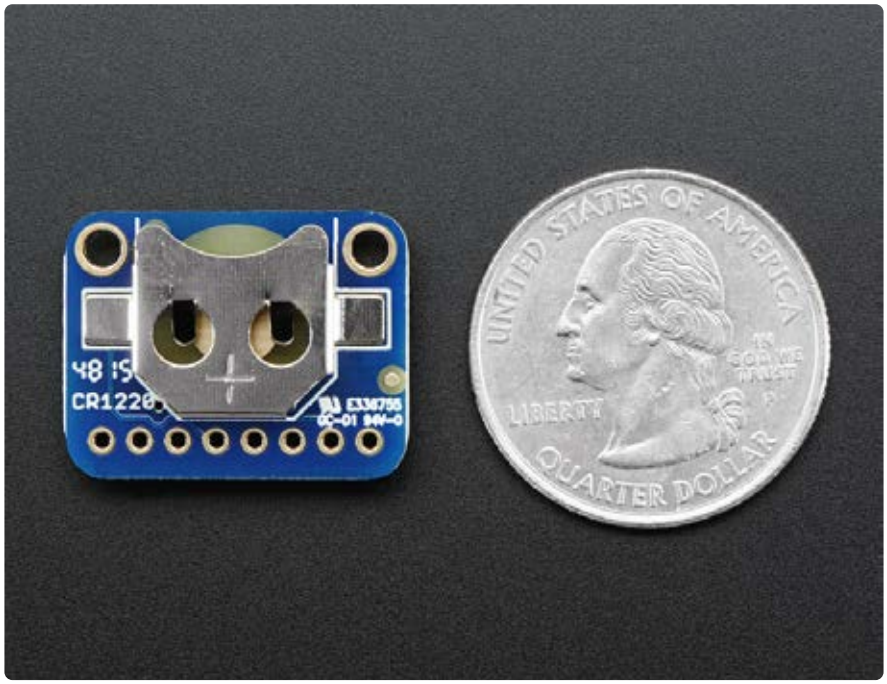
The datasheet for the DS3231 explains that this part is an "Extremely Accurate I^{2}C-Integrated RTC/TCXO/Crystal". And, hey, it does exactly what it says on the tin! ±2ppm accuracy from 0°C to +40°C. This means ±1 minute/year.

This Real Time Clock (RTC) is the most precise you can get in a small, low power package.

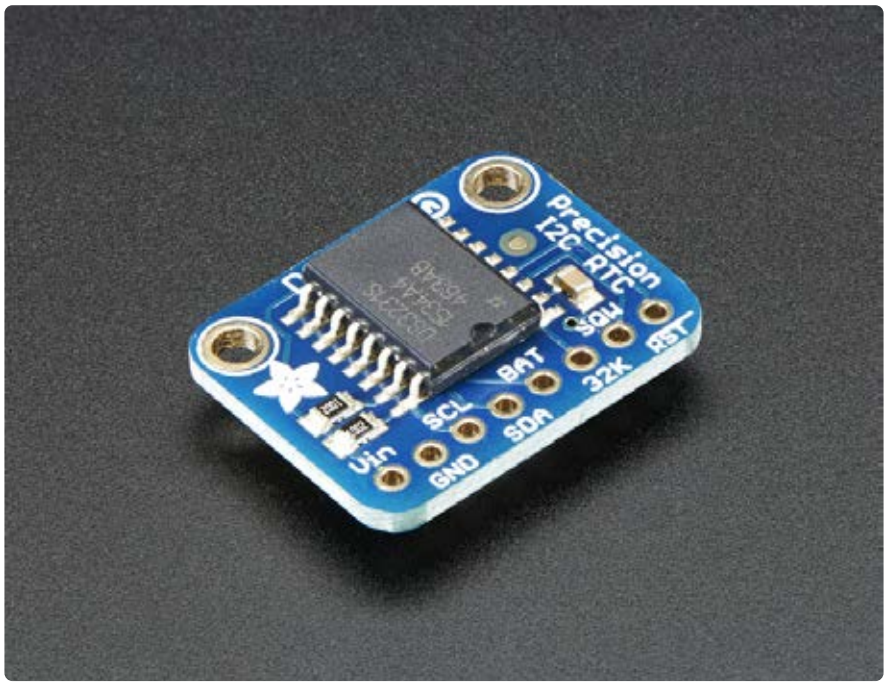


We've had a breakout board version of this RTC for a while (http://adafru.it/3013). We want to make it even easier for folks to use, so now it also comes with STEMMA QT

connectors (http://adafru.it/5188) for plug-and-play simplicity. Be sure you buy the correct version you wish to use.



Most RTC's use an external 32kHz timing crystal that is used to keep time with low current draw. And that's all well and good, but those crystals have slight drift, particularly when the temperature changes (the temperature changes the oscillation frequency very very very slightly but it does add up!) This RTC is in a beefy package because the crystal is inside the chip! And right next to the integrated crystal is a temperature sensor. That sensor compensates for the frequency changes by adding or removing clock ticks so that the timekeeping stays on schedule



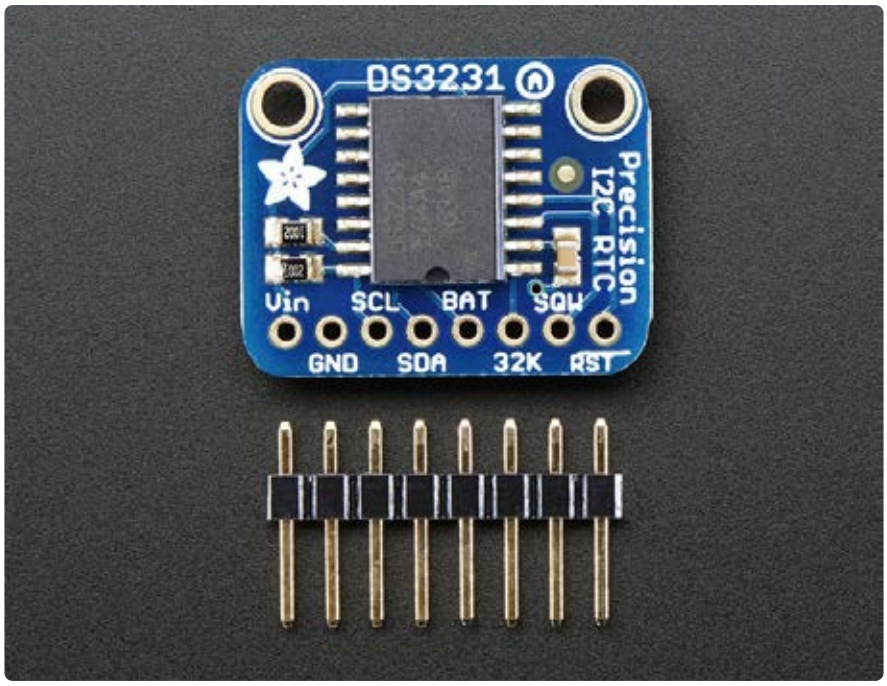
This is the finest RTC you can get, and now we have it in a compact, breadboard-friendly breakout. With a coin cell plugged into the back, you can get years of precision timekeeping, even when main power is lost. Great for datalogging and clocks, or anything where you need to really know the time.

To make life easier so you can focus on your important work, we've taken the sensor and put it onto a breakout PCB along with support circuitry to let you use it with 3.3V (Feather/Raspberry Pi) or 5V (Arduino/ Metro328) logic levels. Additionally, since it speaks I2C you can easily connect it up with two wires (plus power and ground!). We've even included SparkFun qwiic (https://adafru.it/Fpw) compatible STEMMA QT **(https://adafru.it/Ft4)** connectors for the I2C bus so **you don't even need to solder! QT Cable is not included**, but we have a variety in the shop (https://adafru.it/17VE). Just wire up to your favorite micro and you can use our CircuitPython (https://adafru.it/C4w)/Python or Arduino drivers (https://adafru.it/c7r) to easily interface with the DS3231.

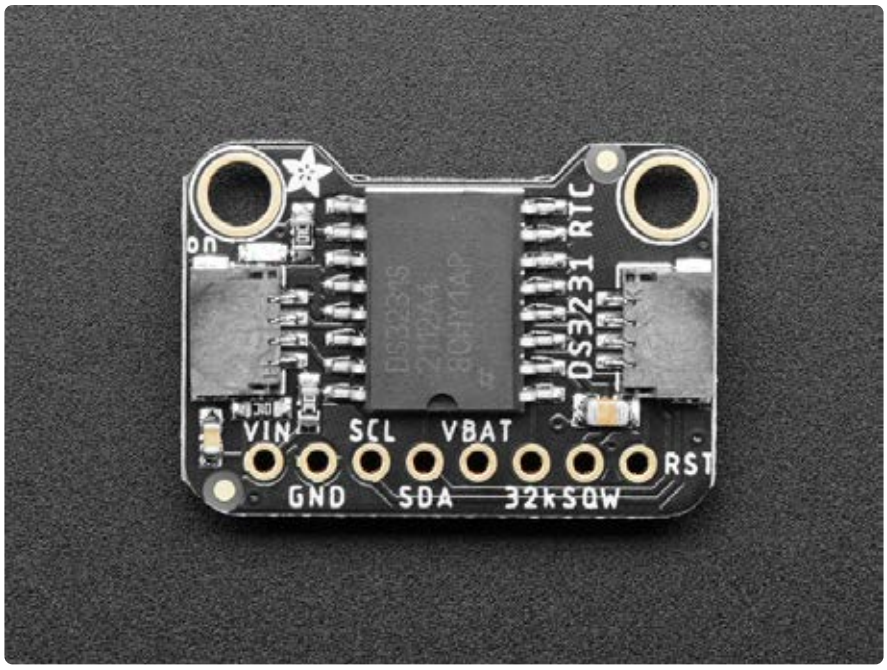
Comes as a fully assembled and tested breakout plus a small piece of header. You can solder header in to plug it into a breadboard, or solder wires directly.

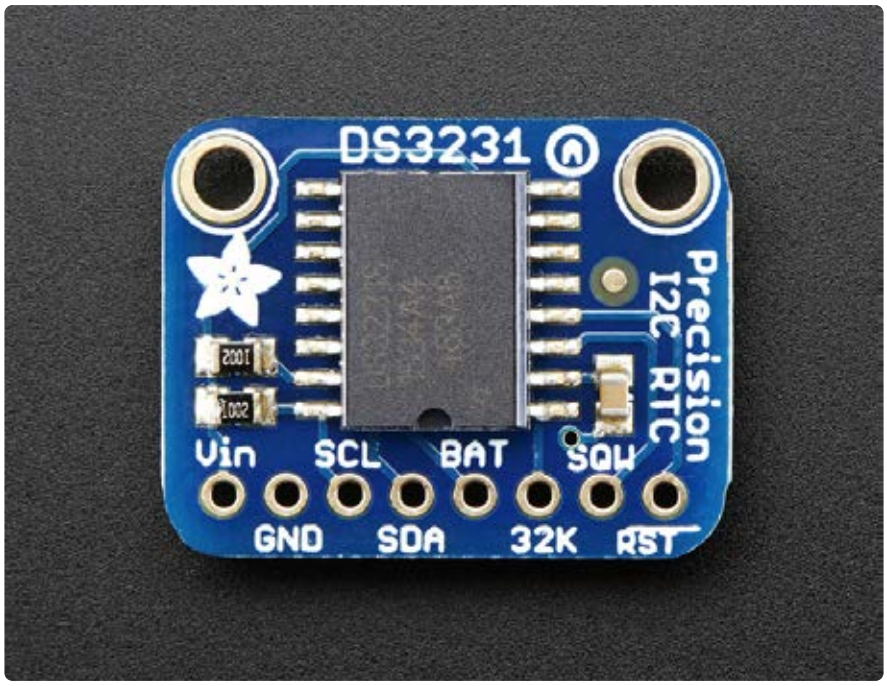
A coin cell is required to use the battery-backup capabilities! We don't include one by default, to make shipping easier for those abroad, but we do stock them so pick one up or use any CR1220 you have handy. (http://adafru.it/380)

There are two versions of this board - the STEMMA QT version shown above (the black PCB), and the original header-only version shown below (the blue PCB). Code works the same on both!


DS3231
Precision
I2C RTC
Vin
SCL
BAT
SQW
GND
SDA
32K
RST

Pinouts


DS3231 RTC
VIN
SCL
VBAT
RST
GND
SDA
32kSQW



Power Pins:

- **Vin** - this is the power pin. Since the RTC can be powered from 2.3V to 5.5V power, you do not need a regulator or level shifter for 3.3V or 5V logic/power. To power the board, give it the same power as the logic level of your microcontroller - e.g. for a 5V micro like Arduino, use 5V
- **GND** - common ground for power and logic

I2C Logic pins:

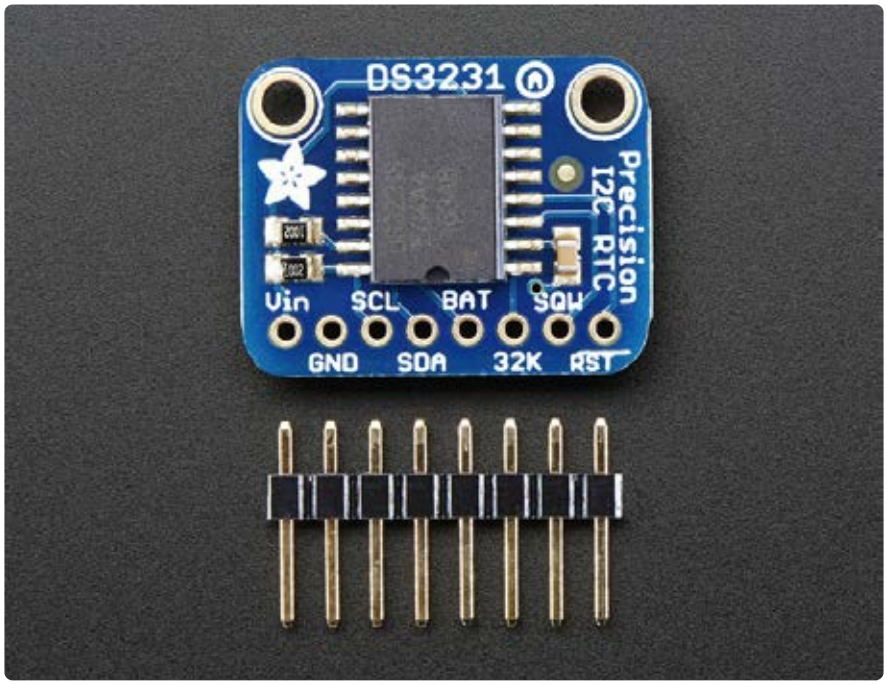
- **SCL** - I2C clock pin, connect to your microcontrollers I2C clock line. This pin has a 10K pullup resistor to Vin
- **SDA** - I2C data pin, connect to your microcontrollers I2C data line. This pin has a 10K pullup resistor to Vin
- **STEMMA QT** (https://adafru.it/Ft4) **- On the STEMMA QT version only!** These connectors allow you to connect to development boards with **STEMMA QT** connectors, or to other things, with various associated accessories (https://adafru.it/Ft6).

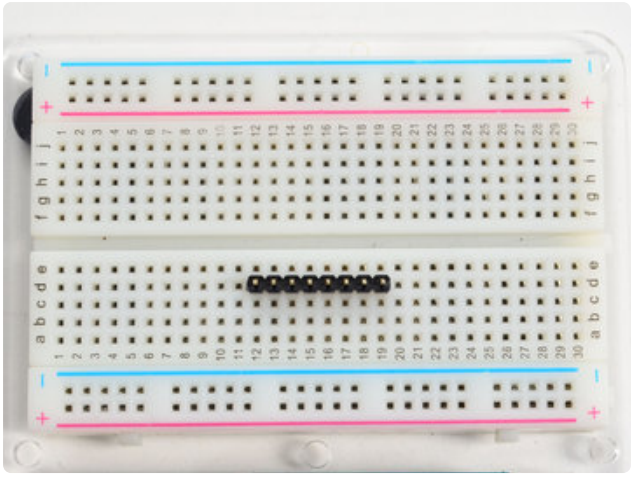
Other Pins:

- **BAT** - this is the same connection as the positive pad of the battery. You can use this if you want to power something else from the coin cell, or provide battery backup from a different separate batery. VBat can be between 2.3V and 5.5V and the DS3231 will switch over when main Vin power is lost

- **32K** - 32KHz oscillator output. Open drain, you need to attach a pullup to read this signal from a microcontroller pin
- **SQW** - optional square wave or interrupt output. Open drain, you need to attach a pullup to read this signal from a microcontroller pin
- **RST** - This one is a little different than most RST pins, rather than being just an input, it is designed to be used to reset an external device or indicate when main power is lost. Open drain, but has an internal 50K pullup. The pullup keeps this pin voltage high as long as Vin is present. When Vin drops and the chip switches to battery backup, the pin goes low.
- **ON LED** - The ON LED will glow on the black STEMMA QT version when power is present. There is an unmarked jumper on the back connected completing the circuit. You can cut the jumper to prevent the LED from turning on if you wish (and resolder to restore the functionality).

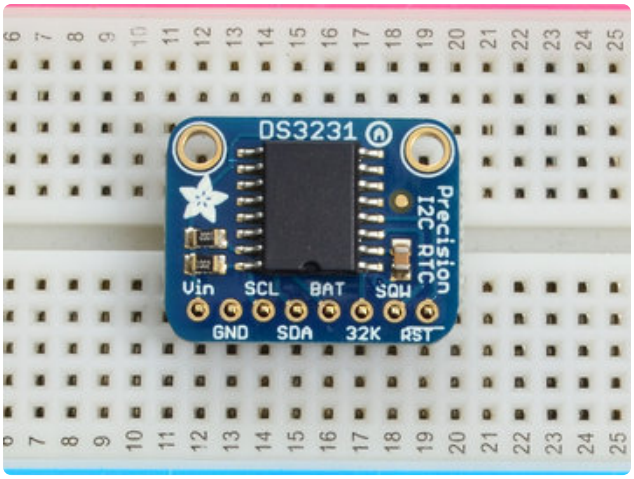
Assembly





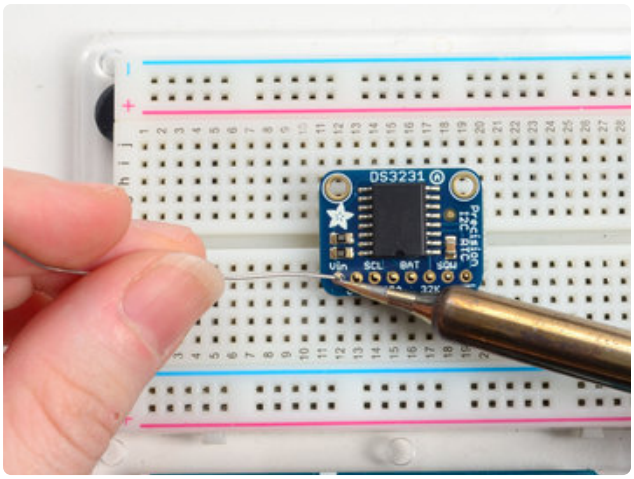
Prepare the header strip:

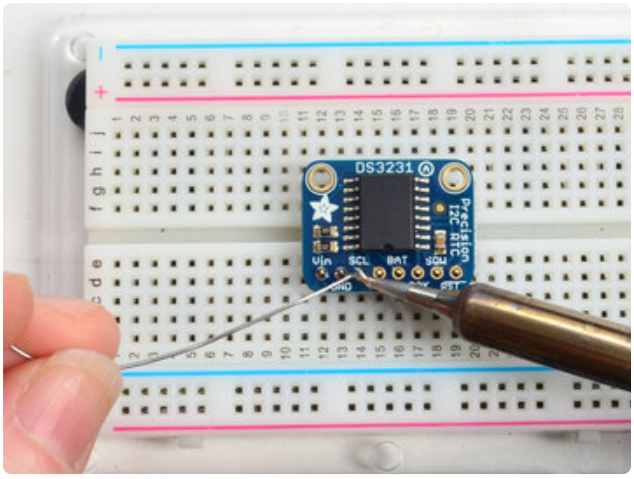
Cut the strip to length if necessary. It will be easier to solder if you insert it into a breadboard - **long pins down**

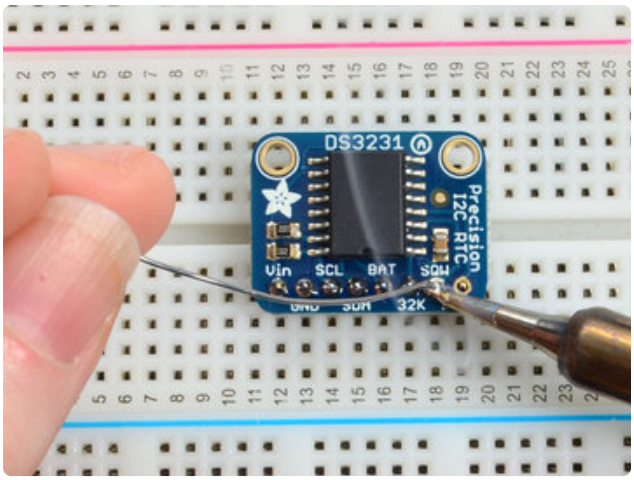


Add the breakout board:

Place the breakout board over the pins so that the short pins poke through the breakout pads



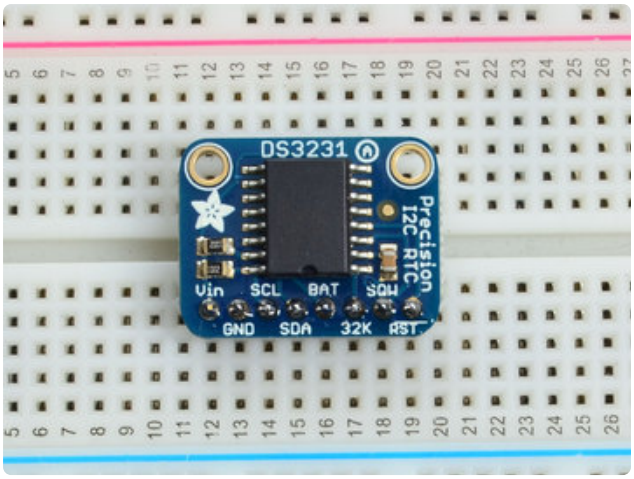




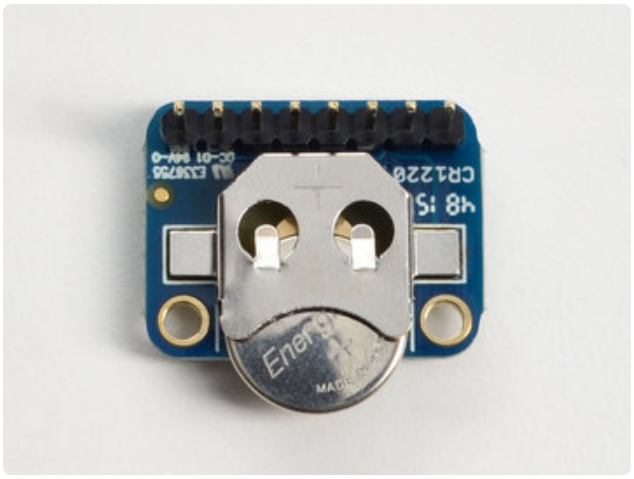
And Solder!

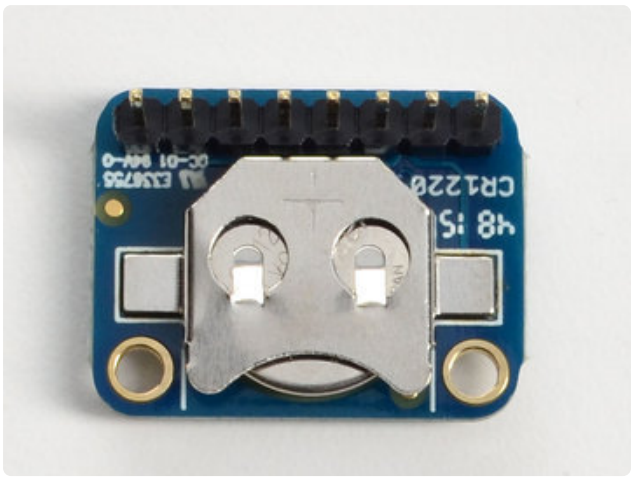
Be sure to solder all pins for reliable electrical contact.

(For tips on soldering, be sure to check out our Guide to Excellent Soldering (https://adafru.it/aTk)).



You're done! Check your solder joints visually and continue onto the next steps

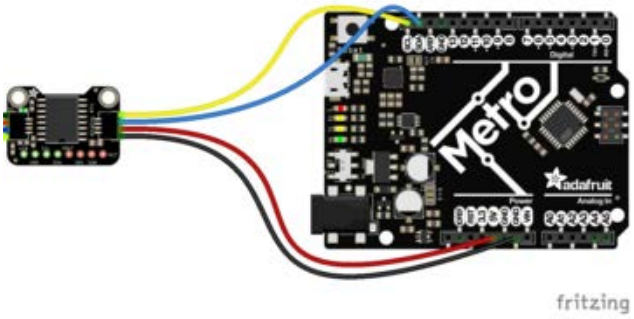


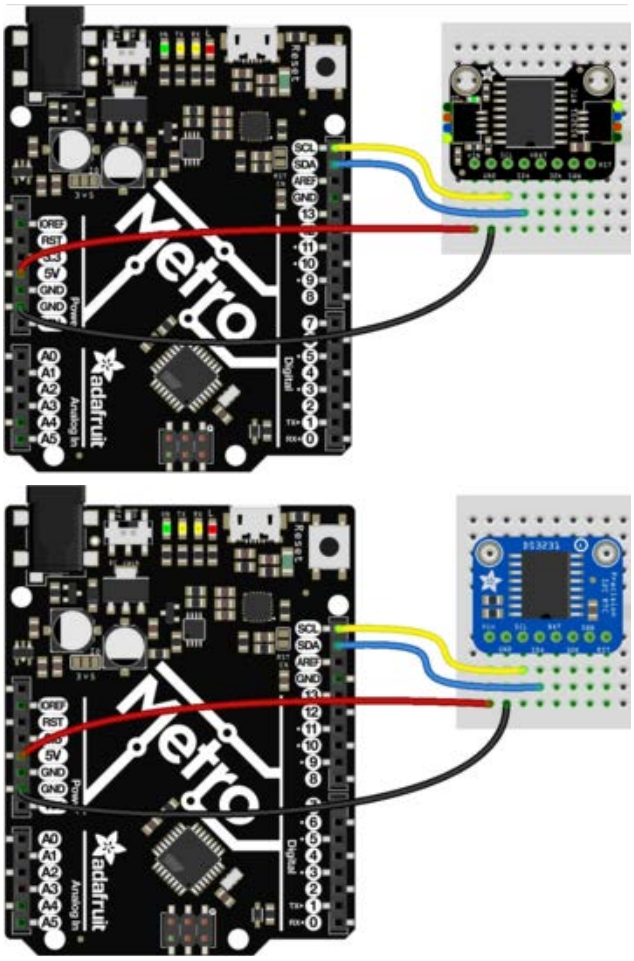


Don't forget that the Real Time Clock requires a battery backup. A CR1220 size battery goes in the back, make sure the + symbol on the battery is visible when you insert!

Arduino Usage

You can easily wire this breakout to any microcontroller, we'll be using an Arduino. For another kind of microcontroller, just make sure it has I2C, then port the code - its pretty simple stuff!





Connect **Vin (red wire)** to the power supply, 3-5V is fine. Use the same voltage that the microcontroller logic is based off of. For most Arduinos, that is 5V
Connect **GND (black wire)** to common power/data ground
Connect the **SCL (yellow wire)** pin to the I2C clock **SCL** pin on your Arduino. On an UNO & '328 based Arduino, this is also known as **A5**, on a Mega it is also known as **digital 21** and on a Leonardo/Micro, **digital 3**
Connect the **SDA (blue wire)** pin to the I2C data **SDA** pin on your Arduino. On an UNO & '328 based Arduino, this is also known as **A4**, on a Mega it is also known as **digital 20** and on a Leonardo/Micro, **digital 2**

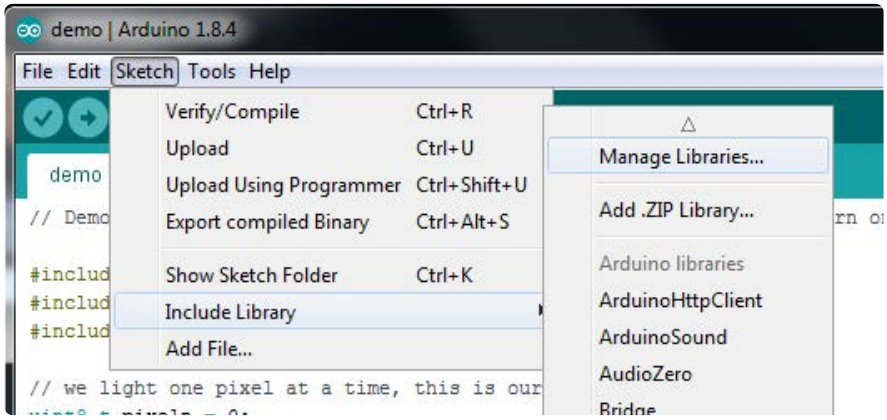
The DS3231 has a default **I2C address of 0x68** and cannot be changed

Download RTCLib

For the RTC library, we'll be using a fork of JeeLab's excellent RTC library RTClib (https://adafru.it/aX2) - a library for getting and setting time from an RTC (originally written by JeeLab, our version is slightly different so please **only use ours** to make sure its compatible!)

To begin reading data, you will need to download Adafruit's RTCLib from the Arduino library manager.

Open up the Arduino library manager:



Search for the **RTCLib** library and install the one by Adafruit

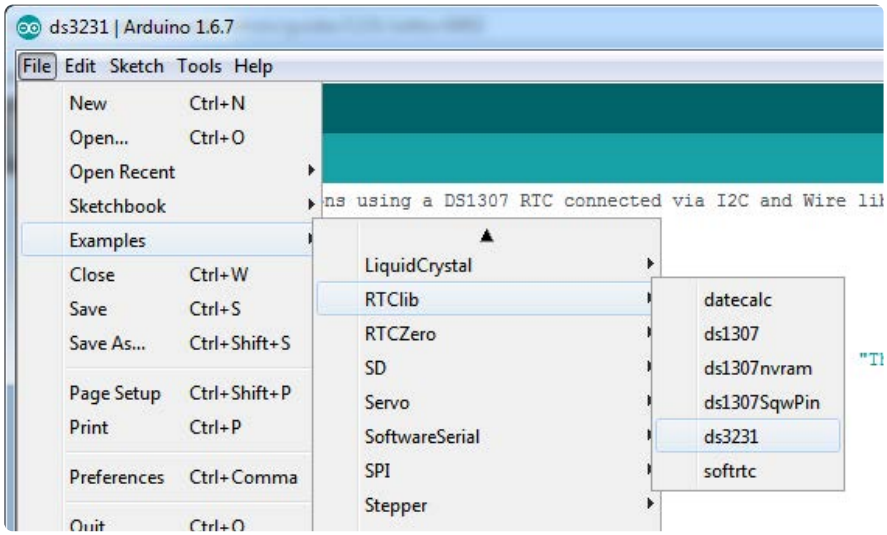
We also have a great tutorial on Arduino library installation at: http://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use (https://adafru.it/aYM)

First RTC Test

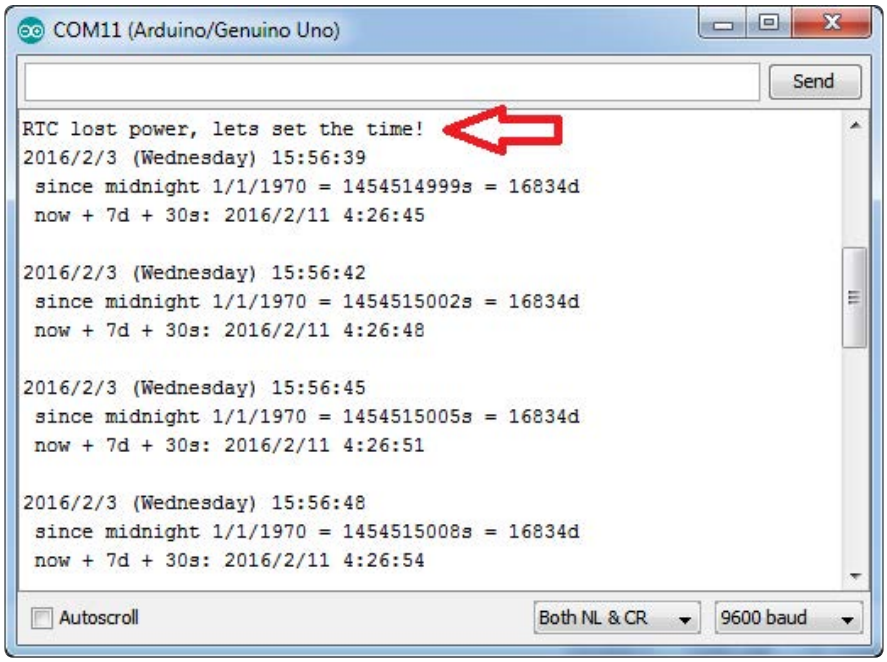
The first thing we'll demonstrate is a test sketch that will read the time from the RTC once per second. We'll also show what happens if you remove the battery and replace it since that causes the RTC to halt. So to start, remove the battery from the holder while the Arduino is not powered or plugged into USB. Wait 3 seconds and then replace the battery. This resets the RTC chip.

Load Demo

Open up **File->Examples->RTClib->ds3231** and upload to your Arduino wired up to the RTC

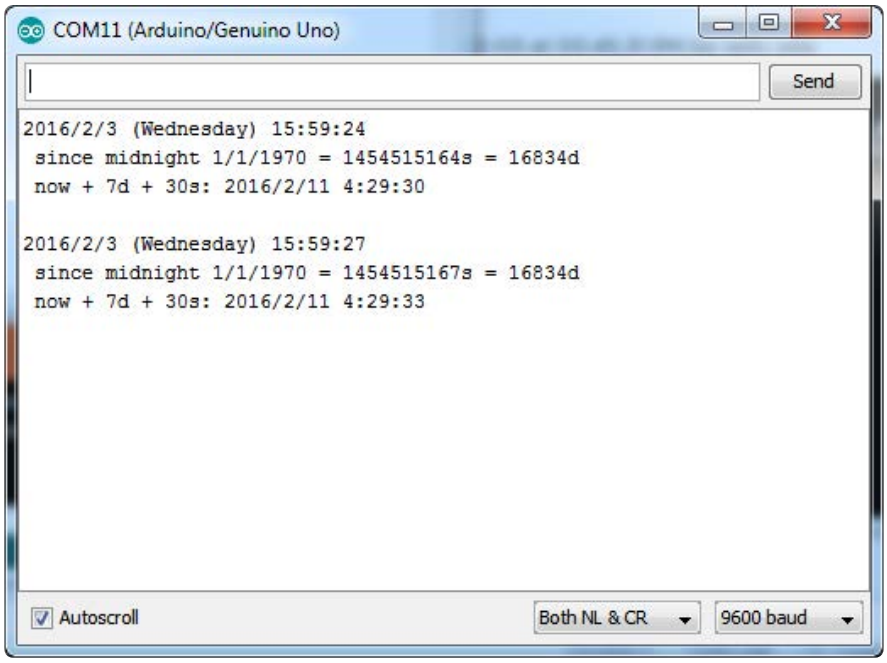


Upload to your Arduino and check the serial console @ 57600 baud (this is a change from the photos). After a few seconds, you'll see the report that the Arduino noticed this is the first time the DS3231 has been powered up, and will set the time based on the Arduino sketch.



Unplug your Arduino plus RTC for a few seconds (or minutes, or hours, or weeks) and plug back in.

Next time you run it you won't get the same "RTC lost power" message, instead it will come immediately and let you know the correct time!



From now on, you wont have to ever set the time again: the battery will last 5 or more years.

Reading the Time

Now that the RTC is merrily ticking away, we'll want to query it for the time. Lets look at the sketch again to see how this is done.

```
void loop () {
    DateTime now = rtc.now();

    Serial.print(now.year(), DEC);
    Serial.print('/');
    Serial.print(now.month(), DEC);
    Serial.print('/');
    Serial.print(now.day(), DEC);
    Serial.print(" (");
    Serial.print(daysOfTheWeek[now.dayOfTheWeek()]);
    Serial.print(") ");
    Serial.print(now.hour(), DEC);
    Serial.print(':');
    Serial.print(now.minute(), DEC);
    Serial.print(':');
    Serial.print(now.second(), DEC);
    Serial.println();
```

There's pretty much only one way to get the time using the RTClib, which is to call **now()**, a function that returns a DateTime object that describes the year, month, day, hour, minute and second when you called **now()**.

There are some RTC libraries that instead have you call something like **RTC.year()** and **RTC.hour()** to get the current year and hour. However, there's one problem where if you happen to ask for the minute right at **3:14:59** just before the next minute rolls over, and then the second right after the minute rolls over (so at **3:15:00**) you'll see the time as **3:14:00** which is a minute off. If you did it the other way around you could get **3:15:59** - so one minute off in the other direction.

Because this is not an especially unlikely occurrence - particularly if you're querying the time pretty often - we take a 'snapshot' of the time from the RTC all at once and then we can pull it apart into **day()** or **second()** as seen above. Its a tiny bit more effort but we think its worth it to avoid mistakes!

We can also get a 'timestamp' out of the DateTime object by calling **unixtime** which counts the number of seconds (not counting leapseconds) since midnight, January 1st 1970

```
    Serial.print(" since midnight 1/1/1970 = ");
    Serial.print(now.unixtime());
    Serial.print("s = ");
    Serial.print(now.unixtime() / 86400L);
    Serial.println("d");
```

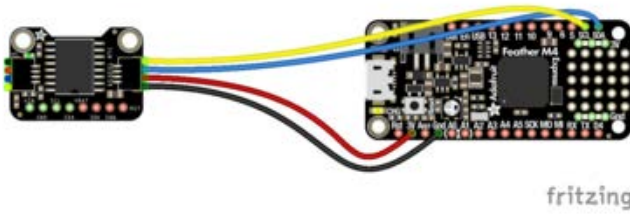
Since there are 60*60*24 = 86400 seconds in a day, we can easily count days since then as well. This might be useful when you want to keep track of how much time has passed since the last query, making some math a lot easier (like checking if its been 5 minutes later, just see if **unixtime()** has increased by 300, you dont have to worry about hour changes).

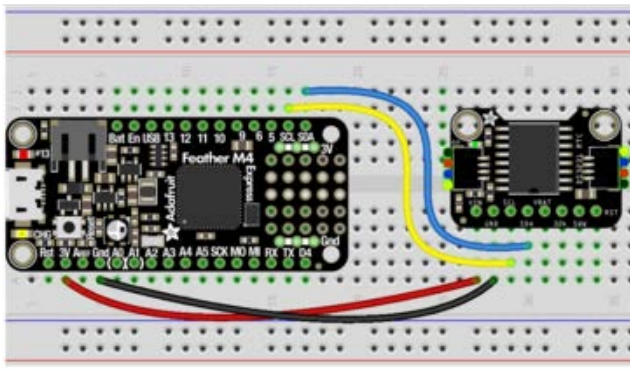
CircuitPython

It's easy to use the DS3231 RTC with CircuitPython too! There's a handy Adafruit CircuitPython DS3231 module (https://adafru.it/C4w) you can load on a board and get started setting and reading the time with Python code!

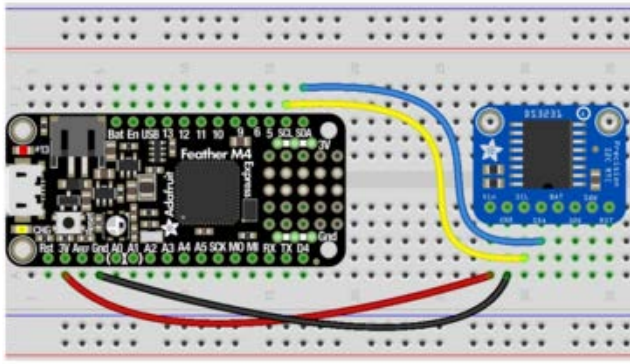
CircuitPython Wiring

First wire up the DS3231 to your board as shown on the previous Arduino page. The DS3231 uses a simple I2C connection with:





Vin (red wire) connected to your board's **3.3V** or **5V** output.
GND (black wire) connected to your board's **ground**.
SCL (yellow wire) connected to your board's **I2C SCL** / clock line.
SDA (blue wire) connected to your board's **I2C SDA** / data line.



CircuitPython Library Installation

You'll also need to install the Adafruit CircuitPython DS3231 (https://adafru.it/C4w) library on your CircuitPython board. **Remember this module is for Adafruit CircuitPython firmware and not MicroPython.org firmware!**

First make sure you are running the latest version of Adafruit CircuitPython (https://adafru.it/tBa) for your board.

Next you'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle (https://adafru.it/zdx). For example the Circuit Playground Express guide has a great page on how to install the library bundle (https://adafru.it/C9M) for both express and non-express boards.

Remember for non-express boards like the Trinket M0, Gemma M0, and Feather/Metro M0 basic you'll need to manually install the necessary libraries from the bundle:

- **adafruit_ds3231.mpy**
- **adafruit_bus_device**
- **adafruit_register**

Before continuing make sure your board's lib folder or root filesystem has the **adafruit_ds3231.mpy, adafruit_bus_device,** and **adafruit_register** files and folders copied over.

CircuitPython Usage

Next connect to the board's serial REPL (https://adafru.it/pMf)so you are at the CircuitPython >>> prompt.

Then import the necessary **board** module to initialize the I2C bus:

```
import board
i2c = board.I2C()
```

Note on some boards like the ESP8266 that don't have a hardware I2C interface you might need to instead import and use the bitbangio module, like:

```
import board
import bitbangio
i2c = bitbangio.I2C(board.SCL, board.SDA)
```

Now import the DS3231 module and create an instance of the DS3231 class using the I2C interface created above:

```
      import adafruit_ds3231
ds3231 = adafruit_ds3231.DS3231(i2c)
```

At this point you're read to read and even set the time of the clock. You just need to interact with the **datetime** property of the DS3231 instance. For example to read it you can run:

```
ds3231.datetime
```

Notice the time is returned as a special Python time structure (https://adafru.it/C4x). This is from the time module in Python and it has properties like:

- **tm_year** - The year of the timestamp
- **tm_mon** - The month of the timestamp
- **tm_mday** - The day of the month of the timestamp
- **tm_hour** - The hour of the timestamp
- **tm_min** - The minute of the timestamp
- **tm_sec** - The second of the timestamp
- **tm_wday** - The day of the week (0 = Monday, 6 = Sunday)
- **tm_yday** - The day within the year (1-366)
- **tm_isdst** - 0 if not in daylight savings, 1 if in savings, and -1 if unknown

Also notice if the time hasn't been set it defaults to a value of January 1st, 2000 (wow Y2K retro!).

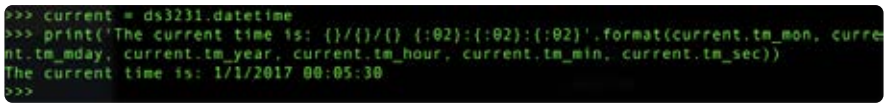
You can write to the **datetime** property to set the time of the clock, for example to set it to January 1st, 2017, at midnight local time you could run:

```
import time
ds3231.datetime = time.struct_time((2017, 1, 1, 0, 0, 0, 6, 1, -1))
```

Notice the parameters to the **struct_time** initializer, you must pass in a tuple of all the values listed above.

Now if you read the **datetime** property you'll see the clock is running from the set time. For example if you want to read and print out just the year, month, day, hour, minute, and second you could run:

```
current = ds3231.datetime
print('The current time is: {}/{}/{} {:02}:{:02}:{:02}'.format(current.tm_mon,
current.tm_mday, current.tm_year, current.tm_hour, current.tm_min, current.tm_sec))
```



That's all there is to using the DS3231 with CircuitPython! Simply import the DS3231 module, create an instance of the class, and interact with its **datetime** property to set and get the time!

Here's a complete example program you can save as **main.py** on your board and see the time and date printed every second to the REPL:

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

# Simple demo of reading and writing the time for the DS3231 real-time clock.
# Change the if False to if True below to set the time, otherwise it will just
# print the current date and time every second.  Notice also comments to adjust
# for working with hardware vs. software I2C.

import time
import board
import adafruit_ds3231

i2c = board.I2C()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a
microcontroller
rtc = adafruit_ds3231.DS3231(i2c)

# Lookup table for names of days (nicer printing).
days = ("Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday",
"Sunday")


# pylint: disable-msg=using-constant-test
if False:  # change to True if you want to set the time!
    #                     year, mon, date, hour, min, sec, wday, yday, isdst
    t = time.struct_time((2017, 10, 29, 15, 14, 15, 0, -1, -1))
    # you must set year, mon, date, hour, min, sec and weekday
    # yearday is not supported, isdst can be set but we don't do anything with it
at this time
    print("Setting time to:", t)  # uncomment for debugging
    rtc.datetime = t
    print()
# pylint: enable-msg=using-constant-test

# Main loop:
while True:
    t = rtc.datetime
    # print(t)     # uncomment for debugging
    print(
        "The date is {} {}/{}/{}".format(
            days[int(t.tm_wday)], t.tm_mday, t.tm_mon, t.tm_year
        )
    )
    print("The time is {}:{:02}:{:02}".format(t.tm_hour, t.tm_min, t.tm_sec))
    time.sleep(1)  # wait a second
```

Python Docs

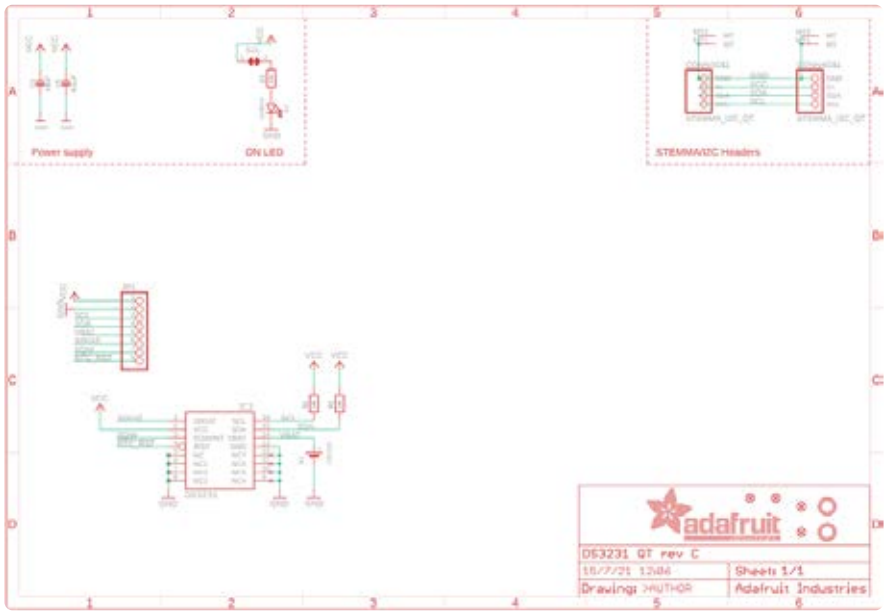
Python Docs (https://adafru.it/C4y)

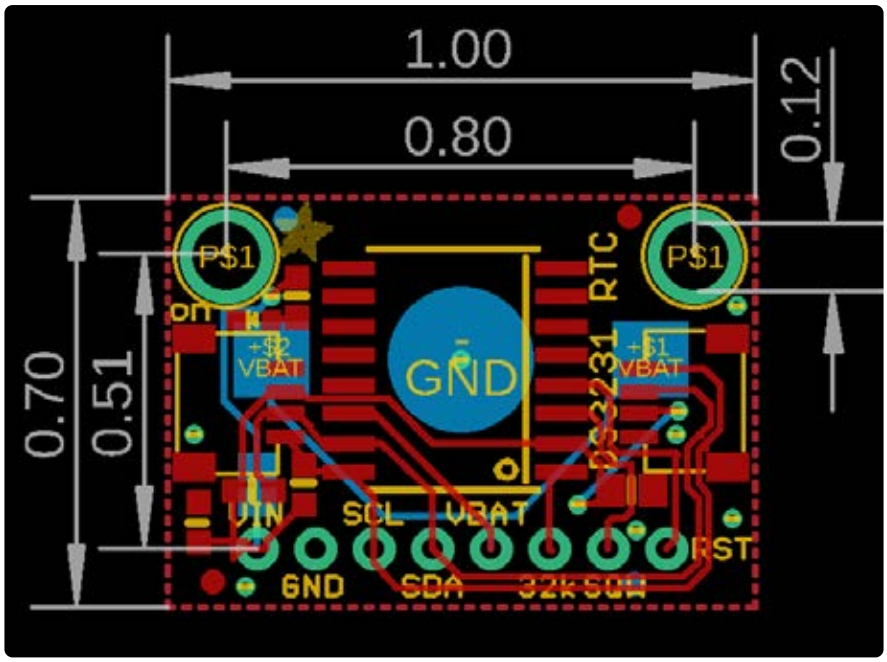
Downloads

Datasheets

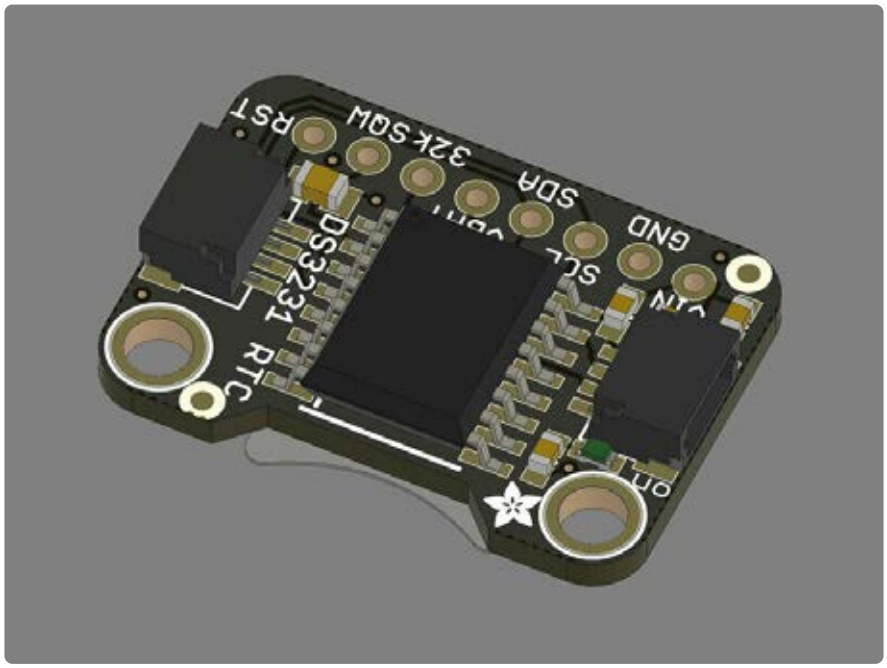
- Analog Devices (formerly Maxim) product page for the DS3231 with datasheets (https://adafru.it/ldy)
- 3D models on GitHub (https://adafru.it/19ee)
- EagleCAD PCB files on GitHub (https://adafru.it/ohE)
- Fritzing object for DS3231 STEMMA QT (https://adafru.it/19ef)
- Fritzing object for DS3231 Breakout (https://adafru.it/19eg)

Schematic and Fab Print for STEMMA QT Version




1.00
0.80
0.12
0.70
0.51
P$1
P$1
RTC
GND
VBAT
VBAT
VIN
SCL
VBAT
RST
GND
SDA

3D Model


RST
SDA
GND
DS3231
RTC

Schematic and Fab Print for Original Version