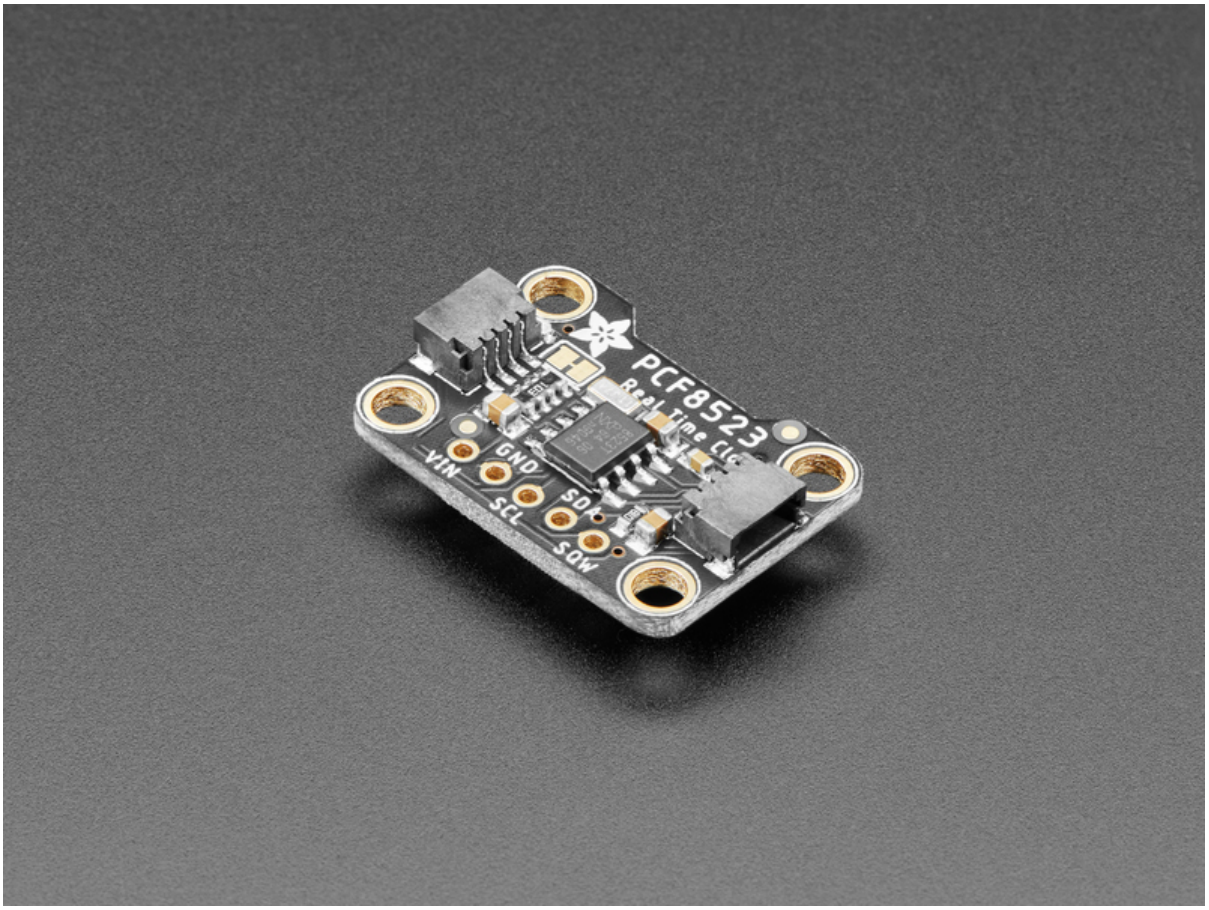
Adafruit PCF8523 Real Time Clock

Created by lady ada



https://learn.adafruit.com/adafruit-pcf8523-real-time-clock

Last updated on 2024-03-08 02:51:45 PM EST

Table of Contents

Overview 3

Pinouts 6

- Power Pins:
- I2C Logic pins:
- STEMMA QT Connectors:
- Other Pins:

Assembly 10

- Prepare the header strip:
- Add the breakout board:
- And Solder!

Real Time Clock 13

- What is a Real Time Clock?
- Battery Backup

RTC with Arduino 15

- Wiring
- Talking to the RTC
- First RTC test
- Setting the time
- Reading the time

RTC with CircuitPython 20

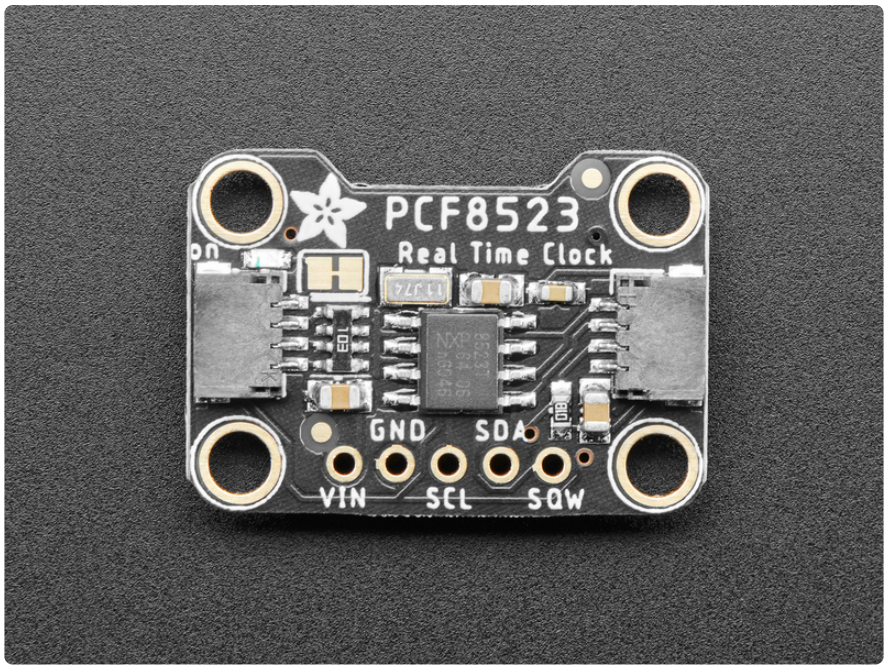
- Wiring
- Adafruit CircuitPython Library Install
- Usage
- Setting the time

Python Docs 24

Downloads 24

- Datasheets and Files
- Schematic and Fab Print for STEMMA QT Version
- Schematic and Fab Print for HVSON Chip Package Version
- Schematic and Fab Print for SOIC-8 Chip Package Version

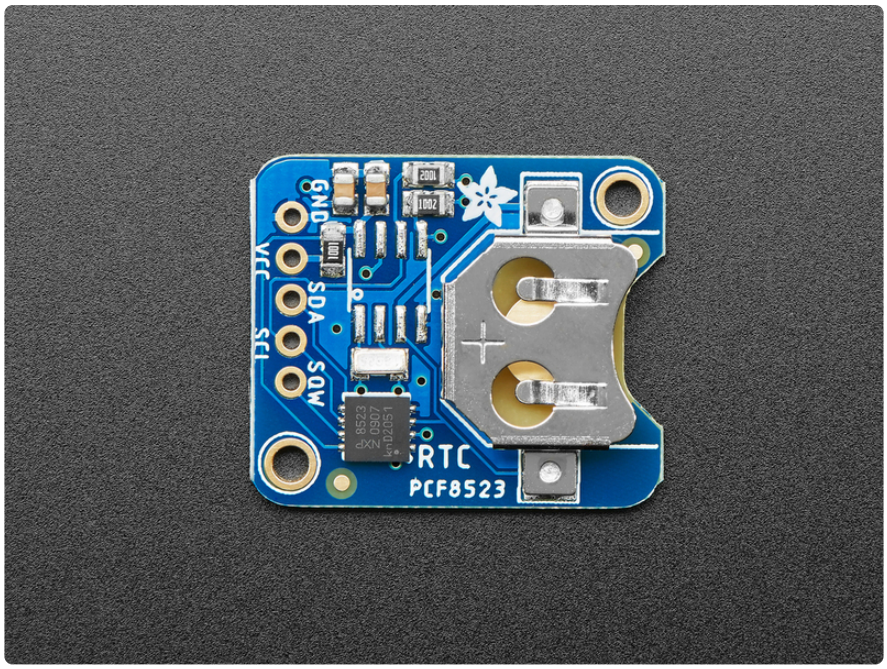
Overview



This is a great battery-backed real time clock (RTC) that allows your microcontroller project to keep track of time even if it is reprogrammed, or if the power is lost. Perfect for datalogging, clock-building, time stamping, timers and alarms, etc. Equipped with **PCF8523** RTC - it can run from 3.3V or 5V power & logic!

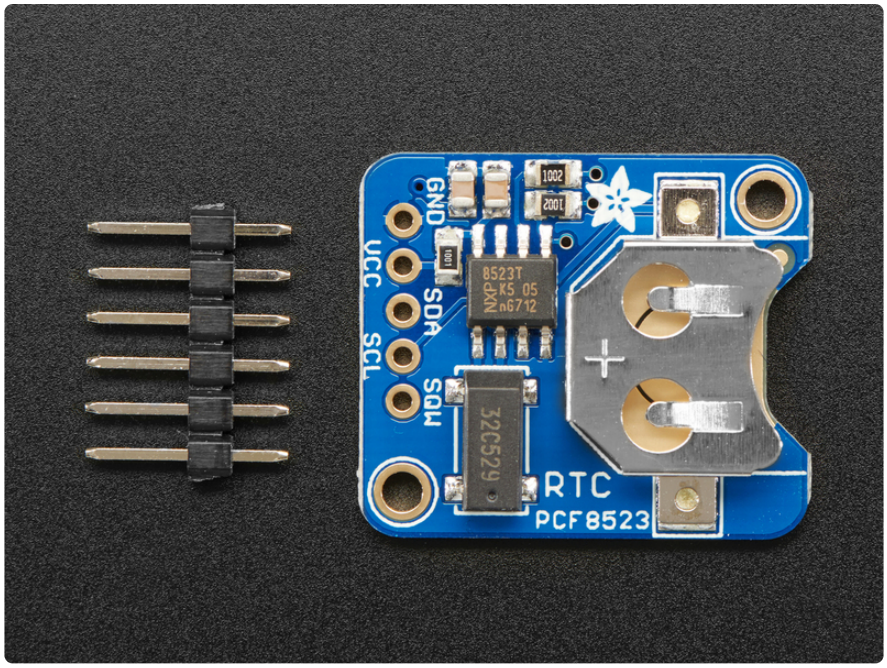
We've had a breakout board version of this RTC for a while (http://adafru.it/3295), but we want to make it even easier for folks to use, so now it comes with STEMMA QT connectors for plug-and-play simplicity.

The blue header-only version of this board may come with the PCF8523 chip in an HVSON or SOIC-8 chip package. Both are identical functionality!



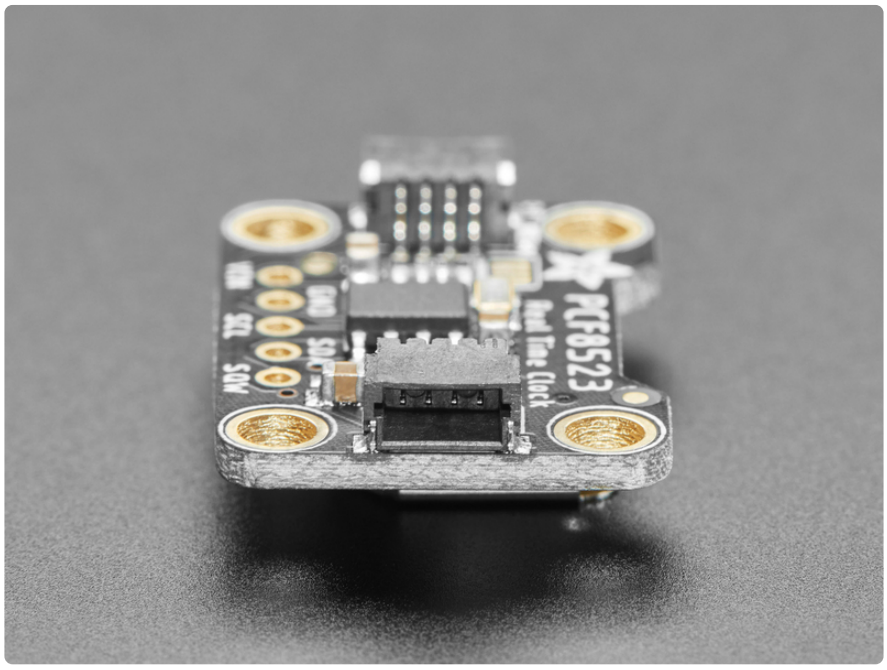
Works great with an Arduino using our RTC library (https://adafru.it/c7r) or with a Raspberry Pi (or similar single board linux computer) (https://adafru.it/C5q)

- PCB & header are included
- Plugs into any breadboard, or you can use wires
- Two mounting holes
- Will keep time for 5 years or more

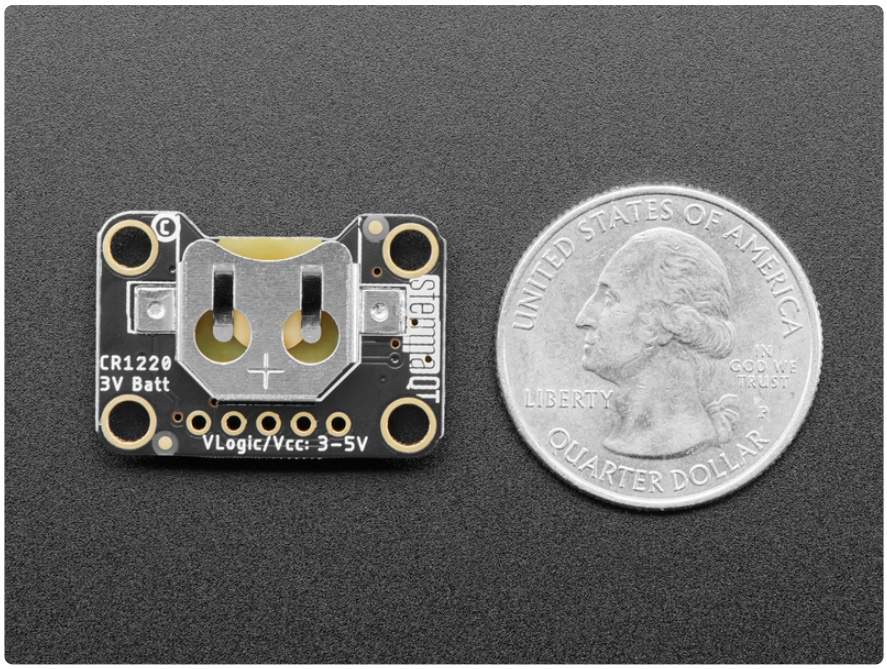


There are two versions of this board - the STEMMA QT version shown below (the black PCB), and the original header-only version shown above (the blue PCB). Code works the same on both!

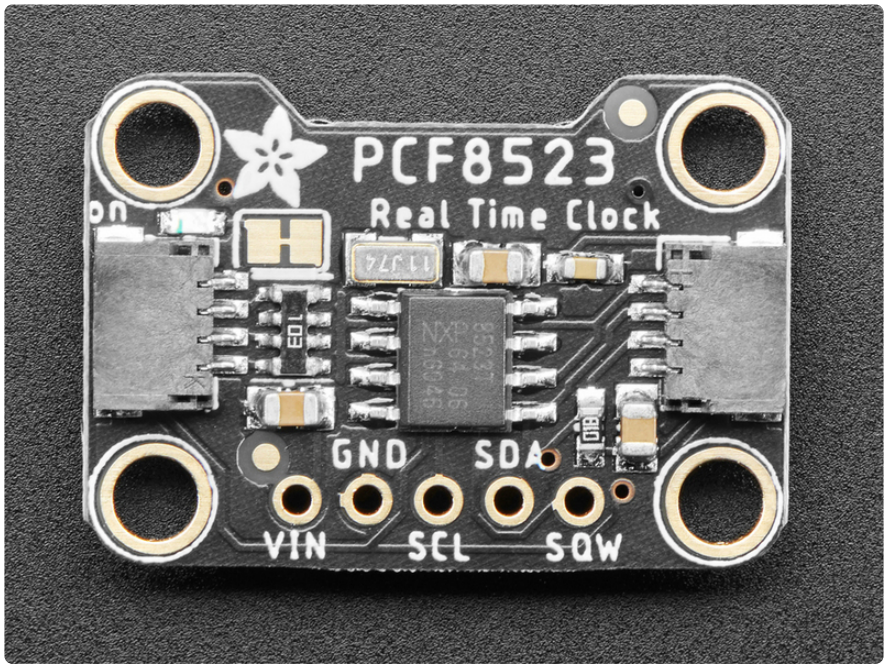
To make life easier so you can focus on your important work, we've taken the sensor and put it onto a breakout PCB along with support circuitry to let you use it with 3.3V (Feather/Raspberry Pi) or 5V (Arduino/ Metro328) logic levels. Additionally, since it speaks I2C you can easily connect it up with 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/Python or Arduino drivers (https://adafru.it/IFR) to easily interface with the PCF8523.

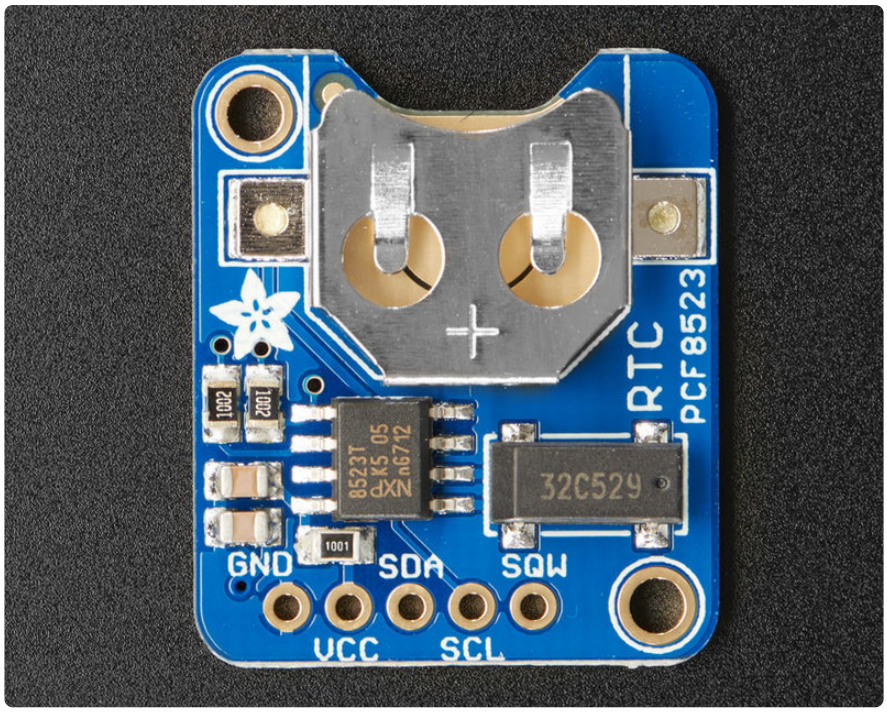


The PCF8523 is simple and inexpensive but not a high precision device. It may lose or gain up to 2 seconds a day. For a high-precision, temperature compensated alternative, please check out the DS3231 precision RTC (http://adafru.it/3013). If you need a DS1307 for compatibility reasons, check out our DS1307 RTC breakout (http://adafru.it/3296)


CR1220
3V Batt
VLogic/Vcc: 3-5V
STEMMA QT
UNITED STATES OF AMERICA
IN GOD WE TRUST
LIBERTY
QUARTER DOLLAR

Pinouts

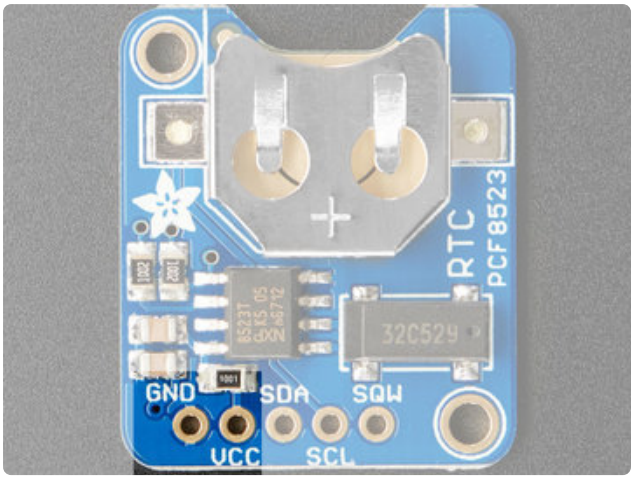

PCF8523
Real Time Clock
GND
SDA
VIN
SCL
SQW

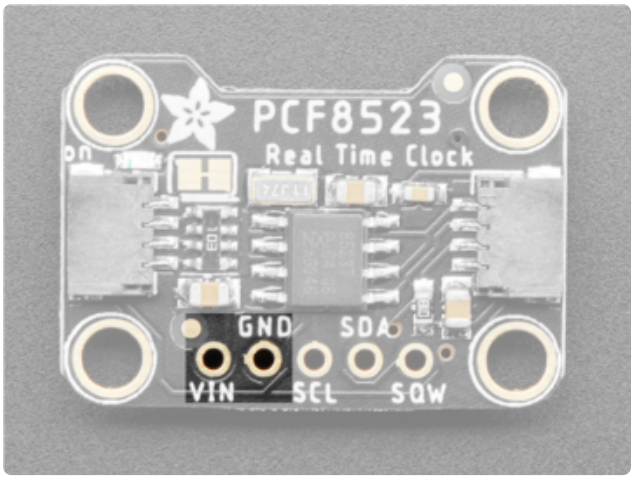


The PCF8523 is a I2C device. That means it uses the two I2C data/clock wires available on most microcontrollers, and can share those pins with other sensors as long as they don't have an address collision.

For future reference, the default I2C address is **0x68**. You cannot change it.

Power Pins:

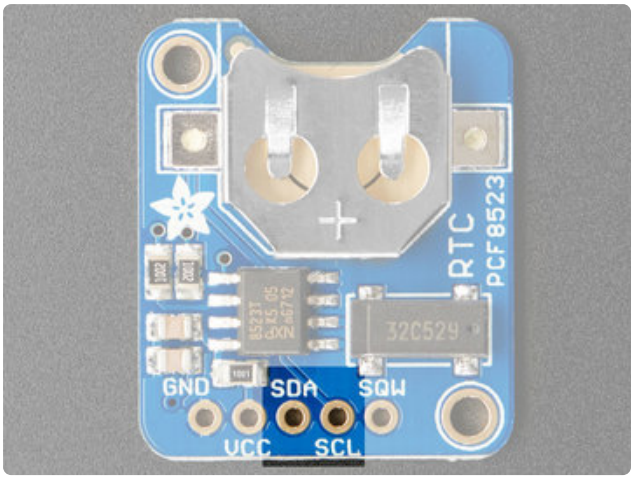


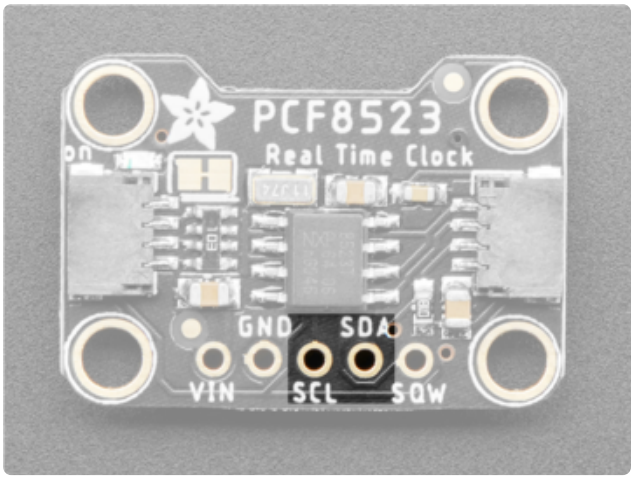


VCC - this is the power pin. This chip can be powered by 3-5VDC so there is now on-board regulator. To power the board, give it the same power as the logic level of your microcontroller - e.g. for a 5V micro like Arduino, use 5V

GND - common ground for power and logic

I2C Logic pins:



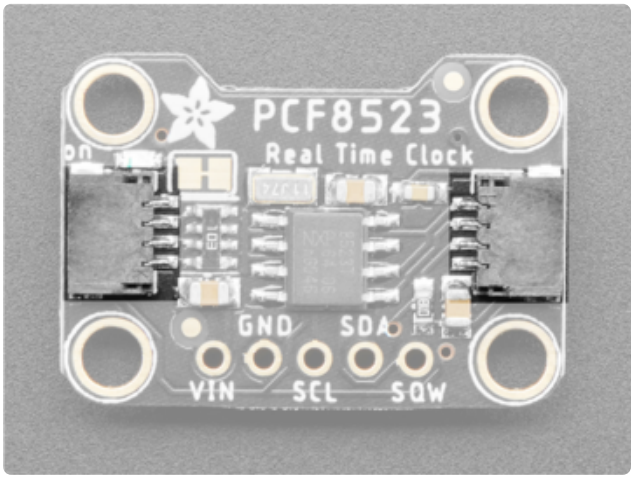


SCL - I2C clock pin, connect to your microcontrollers I2C clock line.

SDA - I2C data pin, connect to your microcontrollers I2C data line.

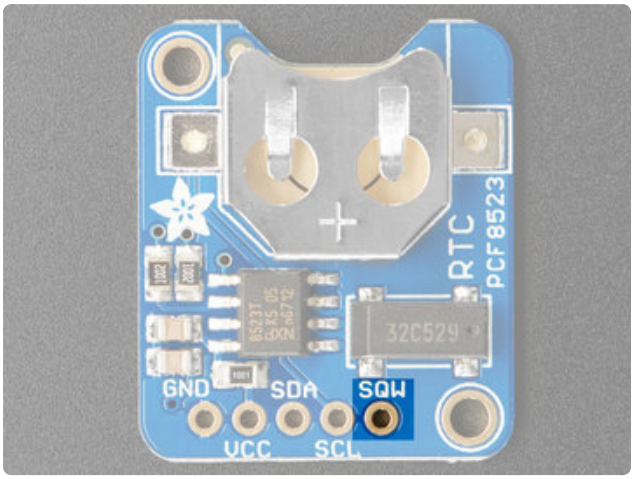
STEMMA QT Connectors:

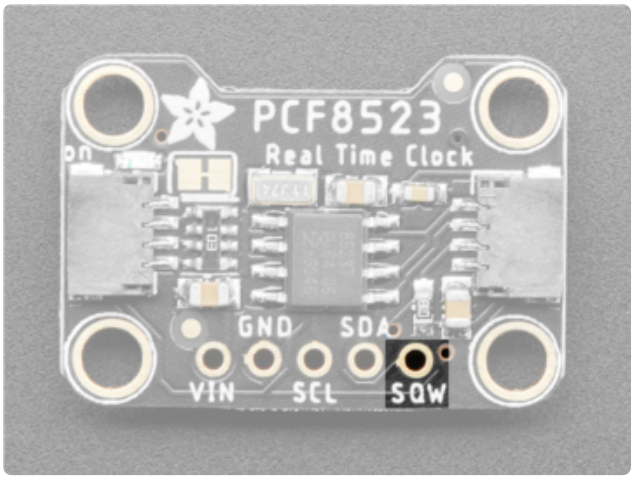
On the STEMMA QT version of the breakout only!



STEMMA QT **(https://adafru.it/Ft4)** - These connectors allow you to connect to dev boards with **STEMMA QT** connectors or to other things with various associated accessories (https://adafru.it/Ft6)

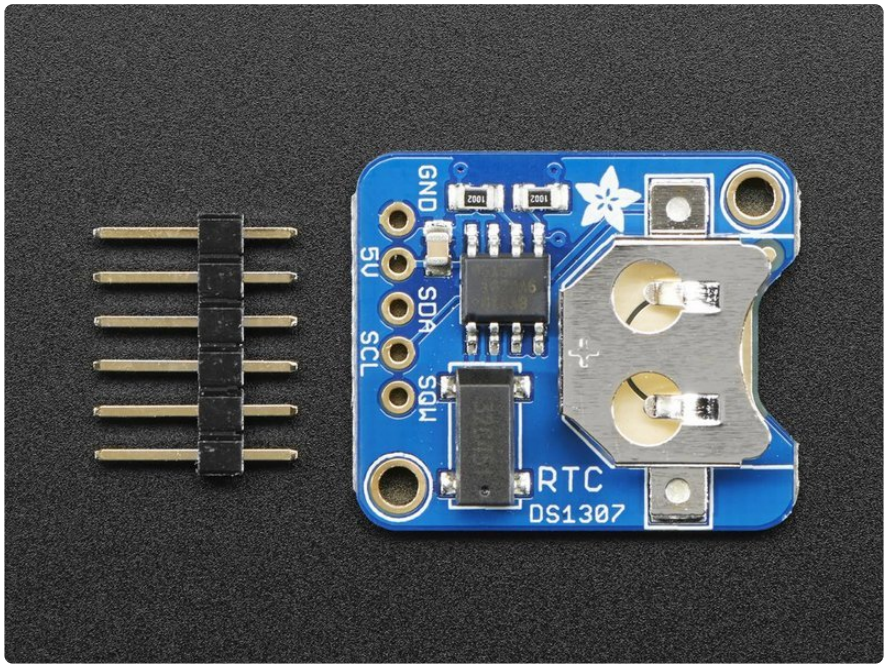
Other Pins:





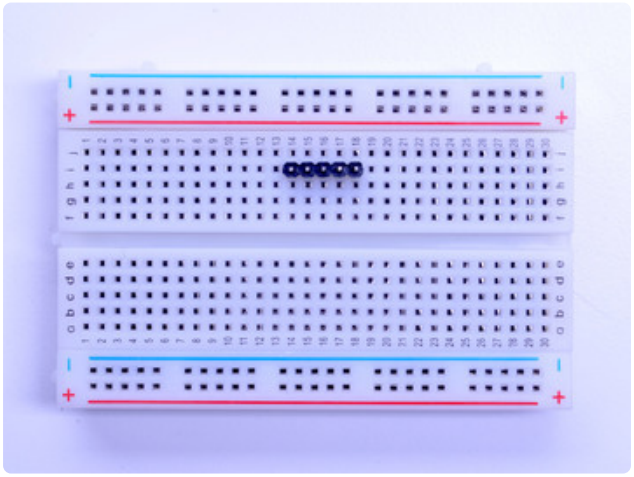
The **SQW** pin is for square-wave output if you enable it

Assembly



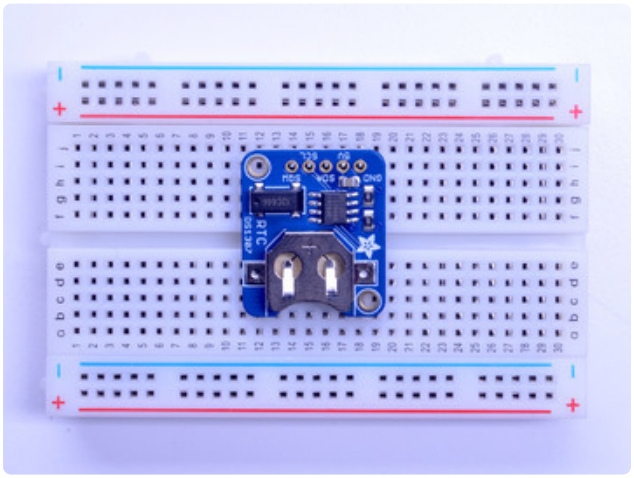
Assembly is really easy, you can use straight or 'right-angle' style headers to attach to the PCB. We'll be using the plain straight headers included

The board comes with all surface-mount components pre-soldered. The included header strip can be soldered on for convenient use on a breadboard or with 0.1" connectors. You can also skip this step and solder on wires.



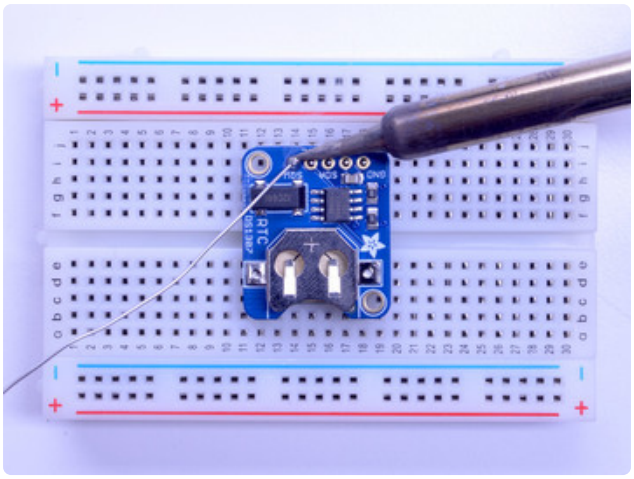
Prepare the header strip:

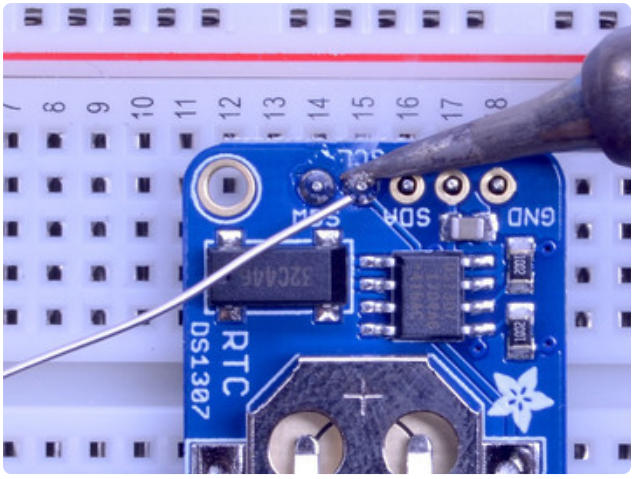
Cut the strip to length if necessary. It will be easier to solder if you insert it into a breadboard - **long pins down**

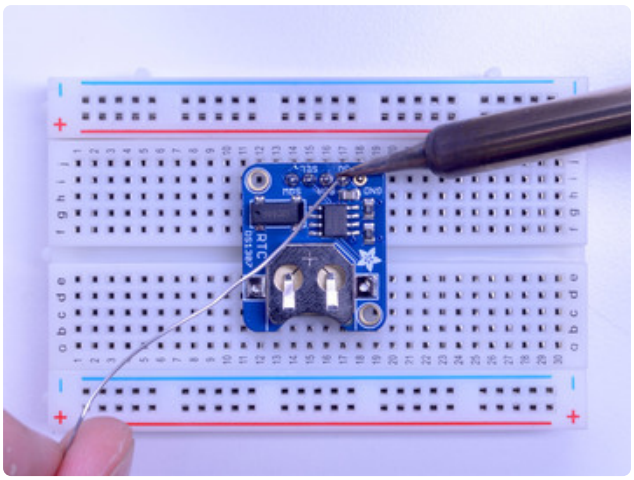


Add the breakout board:

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



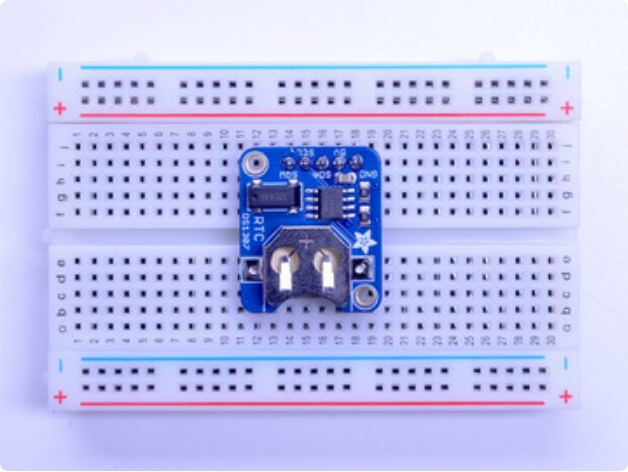




And Solder!

Be sure to solder all 5 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.

Real Time Clock

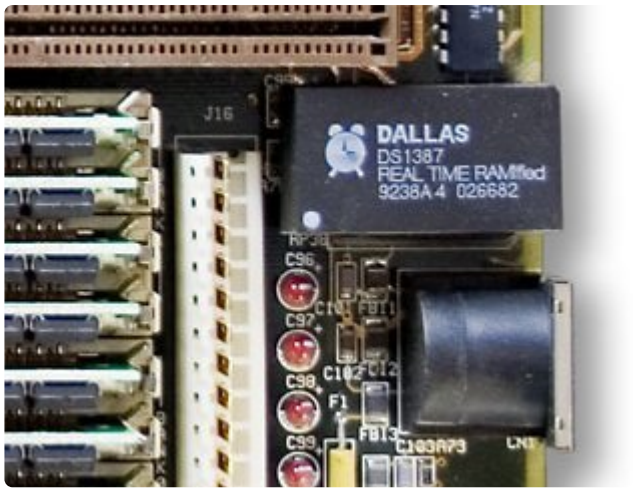
What is a Real Time Clock?

When logging data, it's often really really useful to have timestamps! That way you can take data one minute apart (by checking the clock) or noting at what time of day the data was logged.

The Arduino IDE does have a built-in timekeeper called **millis()** (CircuitPython has **time**) and theres' also timers built into the chip that can keep track of longer time periods like minutes or days. So why would you want to have a separate RTC chip? Well, the biggest reason is that **millis()/time** only keeps track of time since the board was last powered - that means that when the power is turned on, the millisecond timer is set back to 0. The board doesn't know its 'Tuesday' or 'March 8th' all it can tell is 'Its been 14,000 milliseconds since I was last turned on'.

OK so what if you wanted to set the time? You'd have to program in the date and time and you could have it count from that point on. But if it lost power, you'd have to reset the time. Much like very cheap alarm clocks: every time they lose power they blink **12:00**

While this sort of basic timekeeping is OK for some projects, a data-logger will need to have **consistent timekeeping that doesnt reset when the power goes out or is reprogrammed**. Thus, we include a separate RTC! The RTC chip is a specialized chip that just keeps track of time. It can count leap-years and knows how many days are in a month, but it doesn't take care of Daylight Savings Time (because it changes from place to place)



This image shows a computer motherboard with a Real Time Clock called the DS1387 (https://adafru.it/aX0). Theres a lithium battery in there which is why it's so big.

The RTC we'll be using is the PCF8523 (https://adafru.it/reb)

Battery Backup

As long as it has a coin cell to run it, the RTC will merrily tick along for a long time, even when the Feather loses power, or is reprogrammed.

Use any CR1220 3V lithium metal coin cell battery:

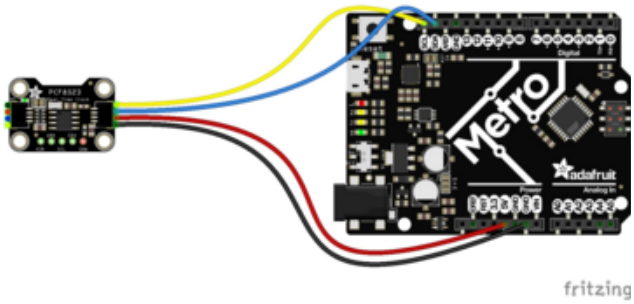
CR1220 12mm Diameter - 3V Lithium Coin Cell Battery
These are the highest quality & capacity batteries, the same as shipped with the iCufflinks, iNecklace, Datalogging and GPS Shields, GPS HAT, etc. One battery per order...
https://www.adafruit.com/product/380

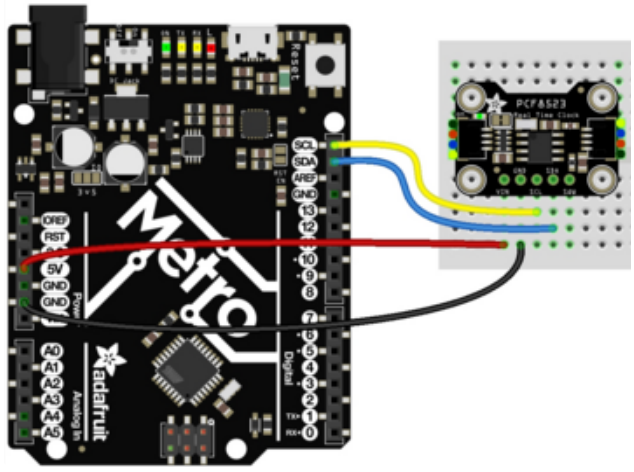
You MUST have a coin cell installed for the RTC to work, if there is no coin cell, it will act strangely and possibly hang the Arduino when you try to use it, so ALWAYS make SURE there's a battery installed, even if it's a dead battery.

RTC with Arduino

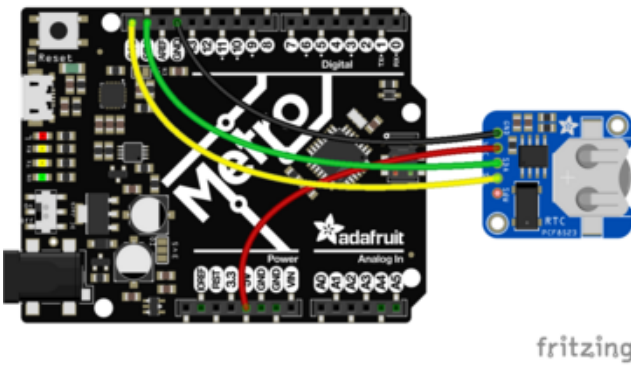
Wiring

Wiring it up is easy, connect it up as shown below.





GND to **GND (black wire on STEMMA QT version)** on your board
VCC (red wire on STEMMA QT version) to the logic level power of your board (on classic Arduinos & Metros use 5V, on 3.3V devices use 3.3V)
SDA to the **SDA (blue wire on STEMMA QT version)** i2c data pin
SCL to the **SCL (yellow wire on STEMMA QT version)** i2c clock pin
There are internal 10K pull-ups on the PCF8523 on SDA and SCL to the VCC voltage

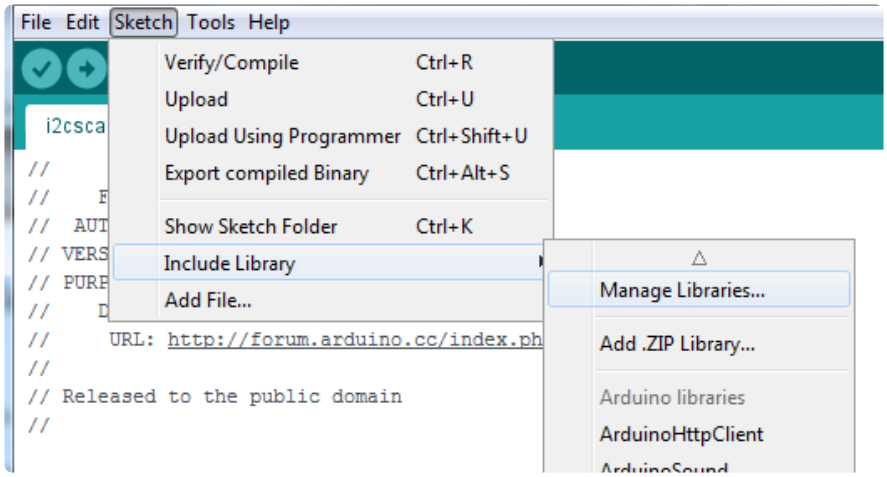


pcfmetro Fritzing

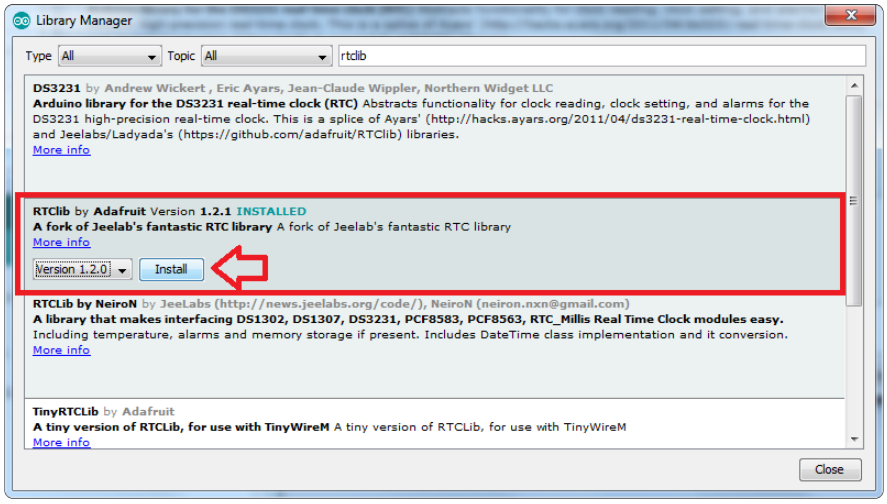
Talking to the RTC

The RTC is an i2c device, which means it uses 2 wires to to communicate. These two wires are used to set the time and retrieve it.

For the RTC library, we'll be using a fork of JeeLab's excellent RTC library, which is available on GitHub (https://adafru.it/c7r). You can do that by visiting the github repo and manually downloading or, easier go to the **Arduino Library Manager**



Type in **RTClib** - and find the one that is by **Adafruit** and click **Install**



There are a few different 'forks' of RTClib, make sure you are using the ADAFRUIT one!

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

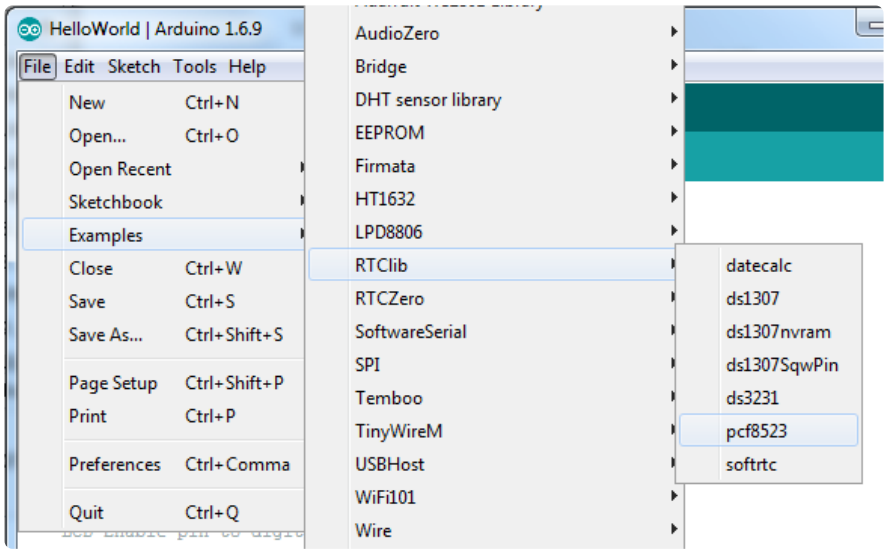
Once done, restart the IDE

First RTC test

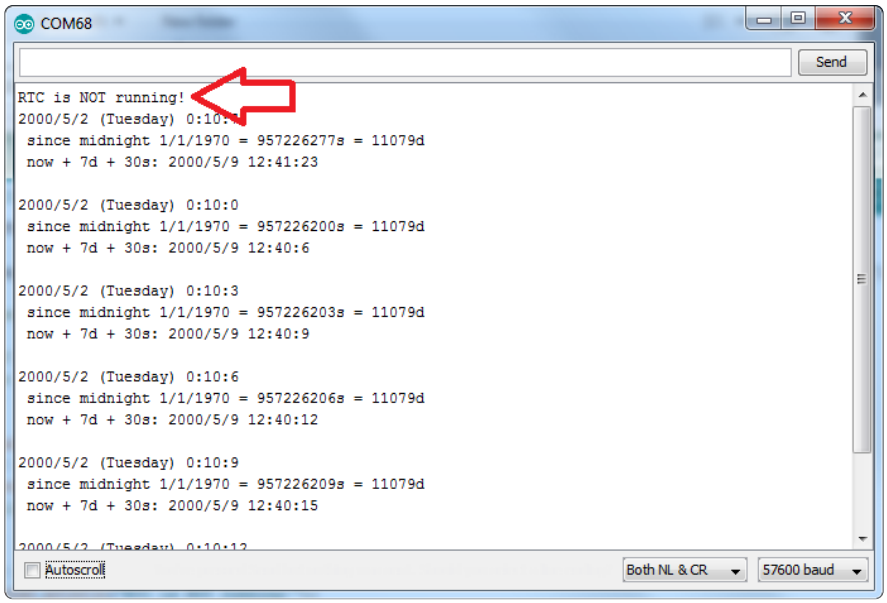
The first thing we'll demonstrate is a test sketch that will read the time from the RTC once a second. We'll also show what happens if you remove the battery and replace it since that causes the RTC to halt. So to start, remove the battery from the holder while the Feather is not powered or plugged into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

Open up **Examples->RTClib->pcf8523**

Upload it to your board with the PCF8523 breakout board or FeatherWing connected



Now open up the Serial Console and make sure the baud rate is set correctly at **57600 baud** you should see the following:



Whenever the RTC chip loses all power (including the backup battery) it will reset to an earlier date and report the time as 0:0:0 or similar. Whenever you set the time, this will kickstart the clock ticking.

So, basically, the upshot here is that you should never ever remove the battery once you've set the time. You shouldn't have to and the battery holder is very snug so unless the board is crushed, the battery won't 'fall out'

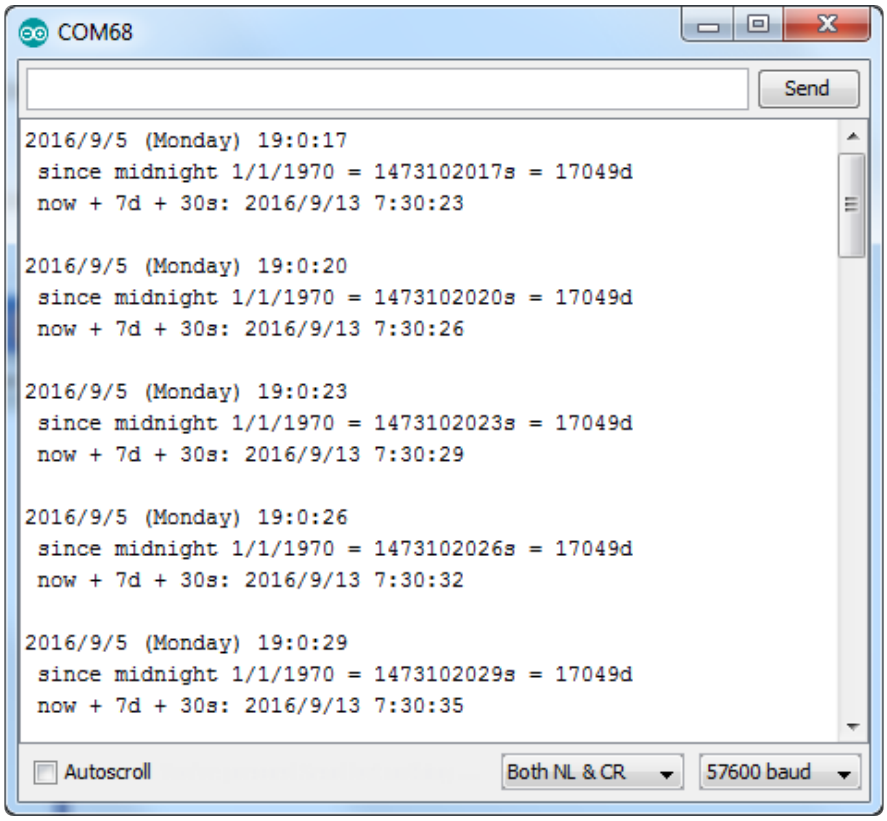
Setting the time

With the same sketch loaded, uncomment the line that starts with **RTC.adjust** like so:

```
if (! rtc.initialized()) {
  Serial.println("RTC is NOT running!");
  // following line sets the RTC to the date &amp;amp; time this sketch was compiled
  rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
```

This line is very cute, what it does is take the Date and Time according the computer you're using (right when you compile the code) and uses that to program the RTC. If your computer time is not set right you should fix that first. Then you must press the **Upload** button to compile and then immediately upload. If you compile and then upload later, the clock will be off by that amount of time.

Then open up the Serial monitor window to show that the time has been set



From now on, you won't 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. Let's 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 occurance - 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. It's 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

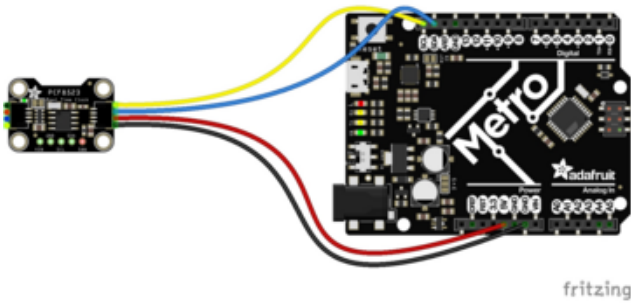
```
    Serial.print(" since 2000 = ");
    Serial.print(now.unixtime());
    Serial.print("s = ");
    Serial.print(now.unixtime() / 86400L);
    Serial.println("d");
```

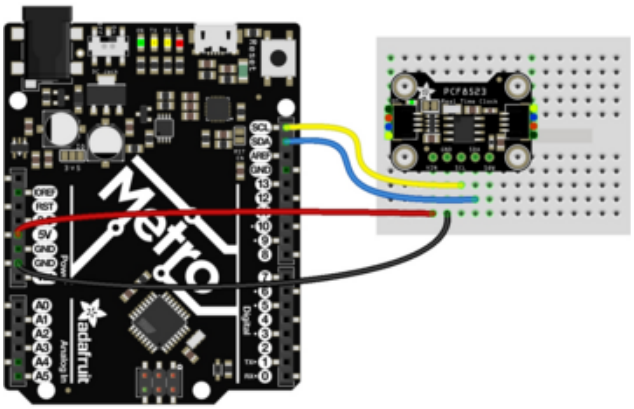
Since there are 60*60*24 = 86400 seconds in a day, we can easily count days since then as well. This might be useful when you want to keep track of how much time has passed since the last query, making some math a lot easier (like checking if it's been 5 minutes later, just see if **unixtime()** has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

RTC with CircuitPython

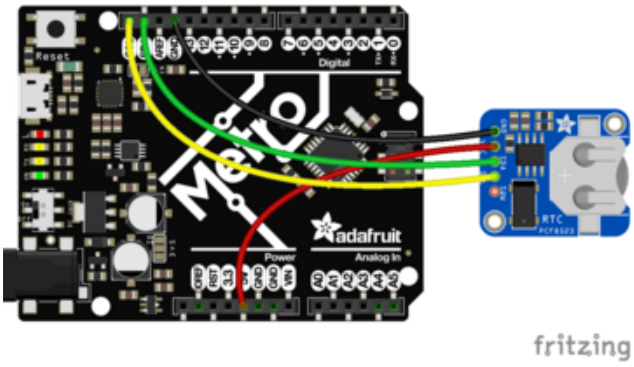
Wiring

Wiring it up is easy, connect it up as shown below.





GND to **GND** on your board
VCC to the logic level power of your board - every CircuitPython board uses 3.3V
SDA to the **SDA** i2c data pin
SCL to the **SCL** i2c clock pin
There are internal 10K pull-ups on the PCF8523 on SDA and SCL to the VCC voltage



Adafruit CircuitPython Library Install

To use the RTC sensor with your Adafruit CircuitPython (https://adafru.it/AIP) board you'll need to install the Adafruit_CircuitPython_PCF8523 (https://adafru.it/Bvh) module on your board.

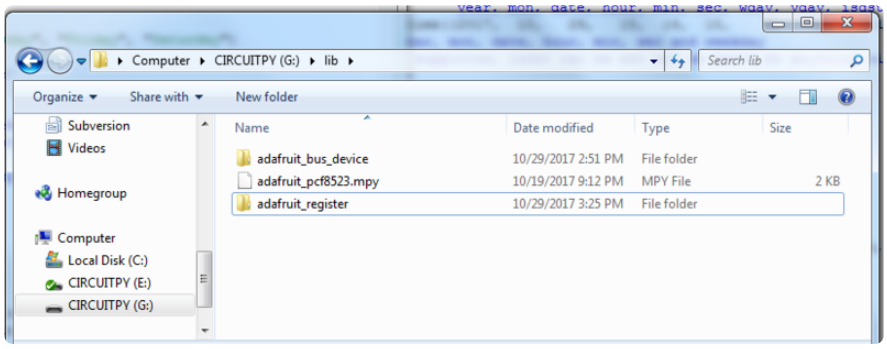
First make sure you are running the latest version of Adafruit CircuitPython (https://adafru.it/Amd) for your board.

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). Our introduction guide has a great page on how to install the library bundle (https://adafru.it/ABU) for both express and non-express boards.

Remember for non-express boards like the, you'll need to manually install the necessary libraries from the bundle:

- **adafruit_bus_device** folder
- **adafruit_register** folder
- **adafruit_pcf8523.mpy**

Before continuing make sure your board's **lib** folder or root filesystem has the **adafruit_pcf8523.mpy** module, the **adafruit_register** folder, and the **adafruit_bus_device** folder copied over.



Usage

To demonstrate the usage of the PCF8523 module you can connect to your board's serial REPL to see the output while saving our example sketch to **code.py**

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

Then save this script to **code.py** (back up or remove whatever was there before)

```
import busio
import adafruit_pcf8523
import time
import board

myI2C = busio.I2C(board.SCL, board.SDA)
rtc = adafruit_pcf8523.PCF8523(myI2C)
```

```
days = ("Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
"Saturday")

if False:   # change to True if you want to write 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()

while True:
    t = rtc.datetime
    #print(t)     # uncomment for debugging

    print("The date is %s %d/%d/%d" % (days[t.tm_wday], t.tm_mday, t.tm_mon,
t.tm_year))
    print("The time is %d:%02d:%02d" % (t.tm_hour, t.tm_min, t.tm_sec))

    time.sleep(1) # wait a second
```

Setting the time

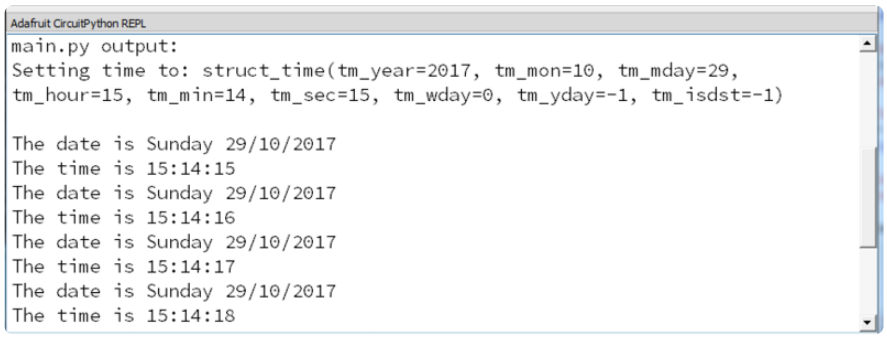
The first time you run the program, you'll need to **set the time**

find these lines:

```
if False:   # change to True if you want to write 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
```

Change the **False** to **True** in the first line, and update the `time.struct_time` to have the current time starting from `year` to `weekday`. The last two entries can stay at -1

Re-run the sketch by saving and you'll see this out of the REPL:



Note the part where the program says it is **Setting time to:**

Now you can go back and change the if **True** to if **False** and save, so you don't re-set the RTC again.

The script will now output the time and date

```
main.py output:
The date is Sunday 29/10/2017
The time is 15:14:56
The date is Sunday 29/10/2017
The time is 15:14:57
The date is Sunday 29/10/2017
The time is 15:14:58
```

Python Docs

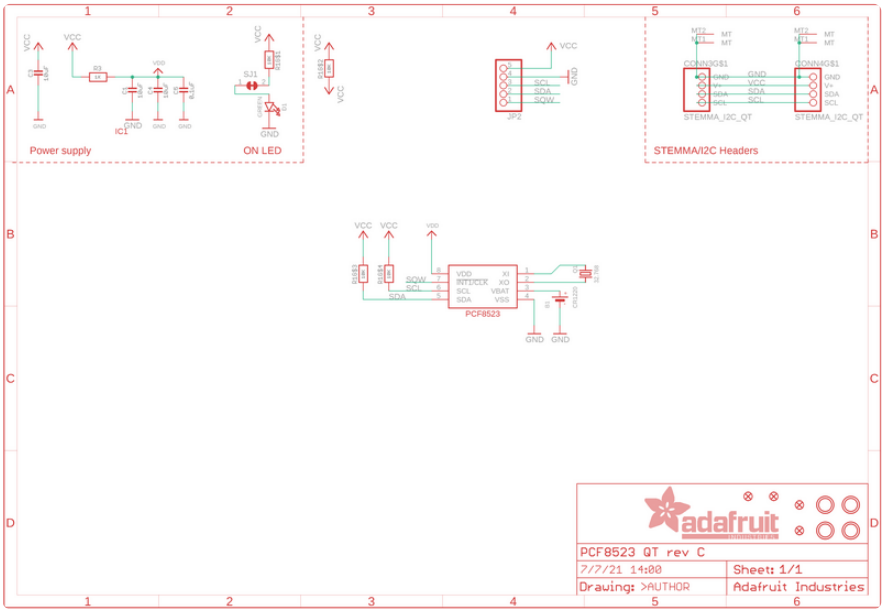
Python Docs (https://adafru.it/C5r)

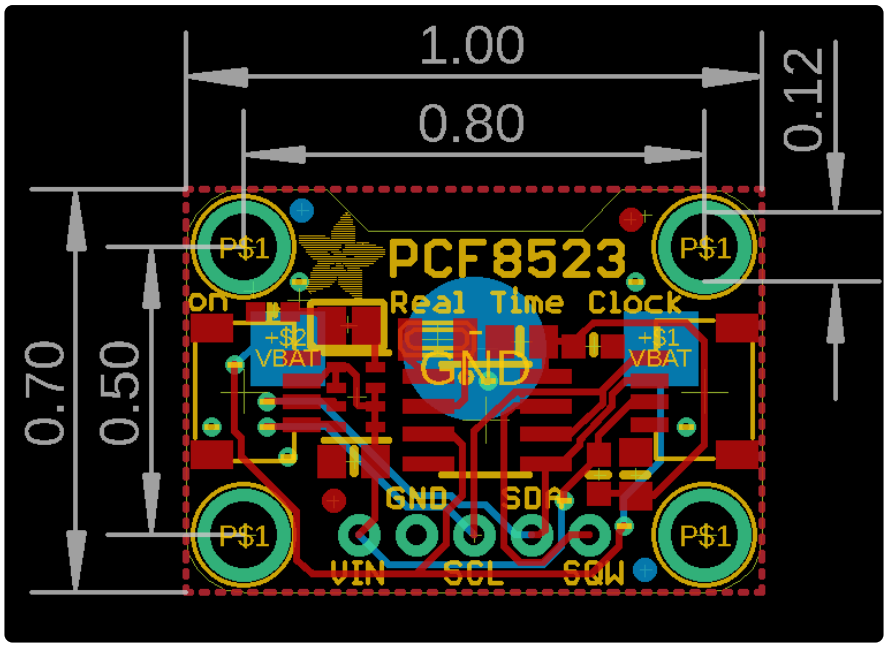
Downloads

Datasheets and Files

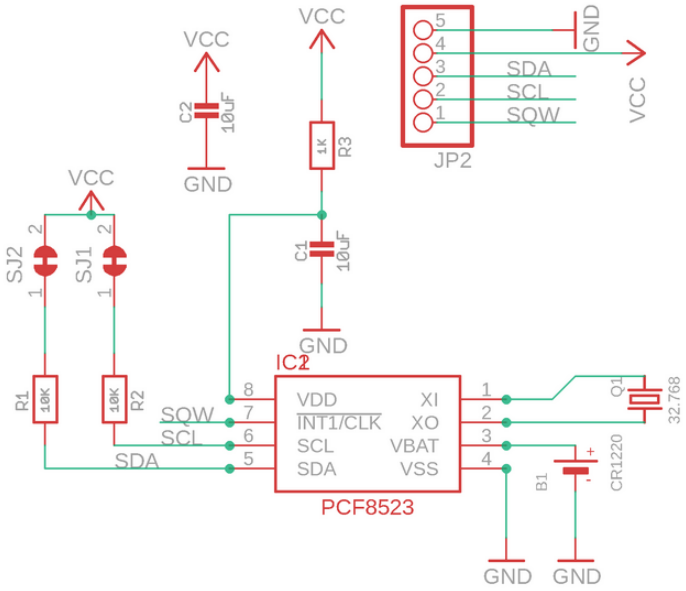
- EagleCAD PCB files on GitHub (https://adafru.it/C5s)
- Fritzing object in Adafruit Fritzing library (https://adafru.it/c7M)
- PCF8523 product page (https://adafru.it/reb)

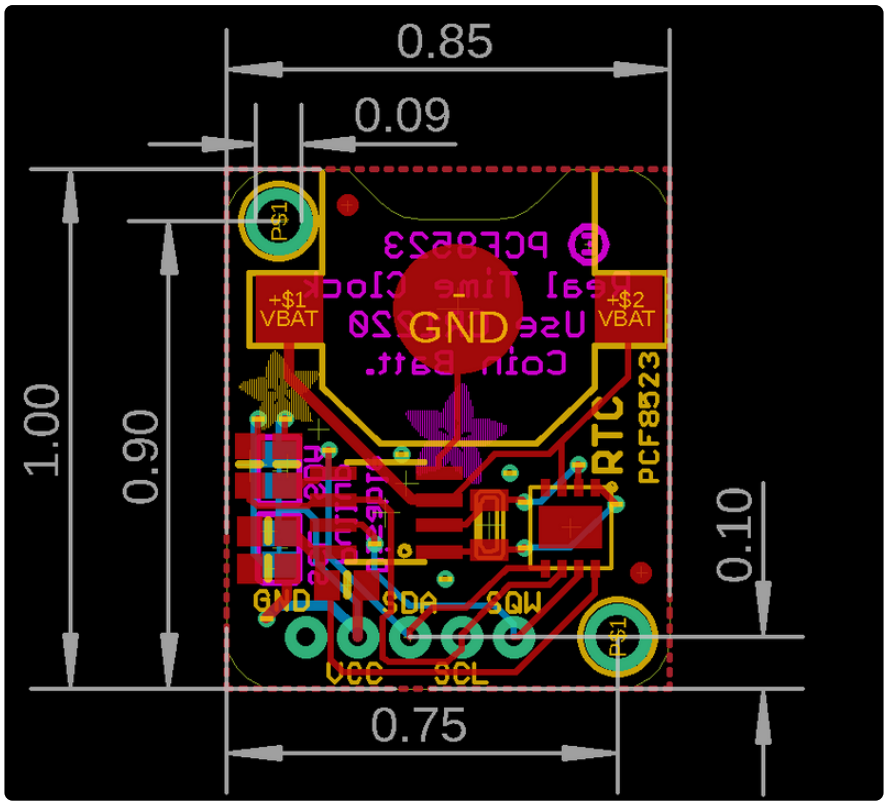
Schematic and Fab Print for STEMMA QT Version



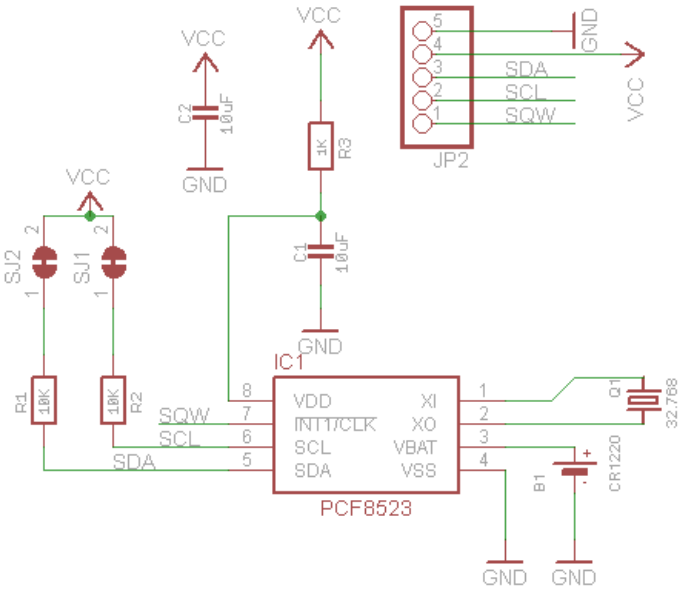


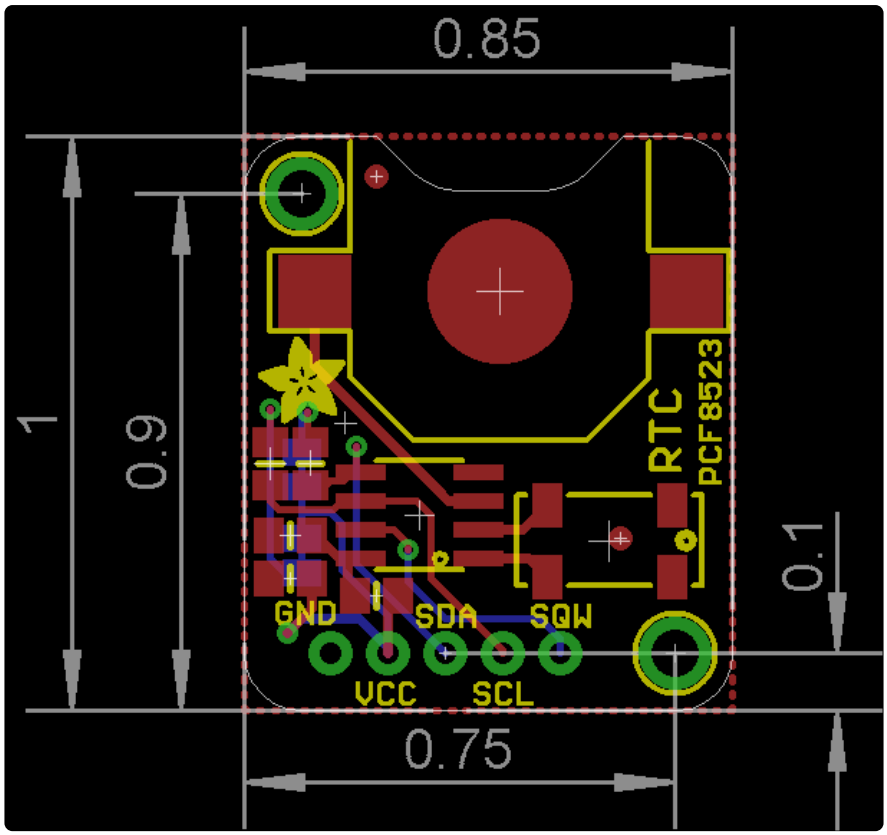
Schematic and Fab Print for HVSON Chip Package Version





Schematic and Fab Print for SOIC-8 Chip Package Version




0.85
1
0.9
0.1
0.75
RTC
PCF8523
GND
SDA
SQW
VCC
SCL