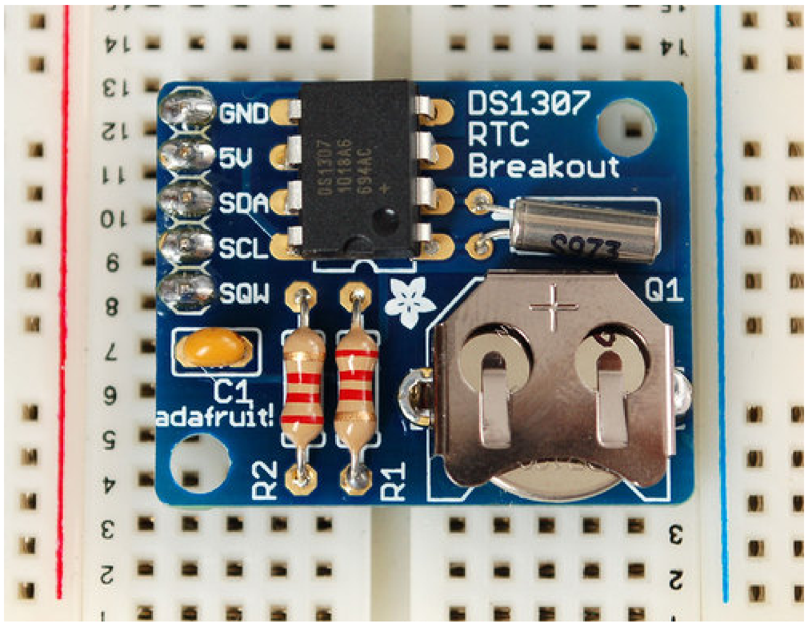
adafruit learning system

Adding a Real Time Clock to BeagleBone Black

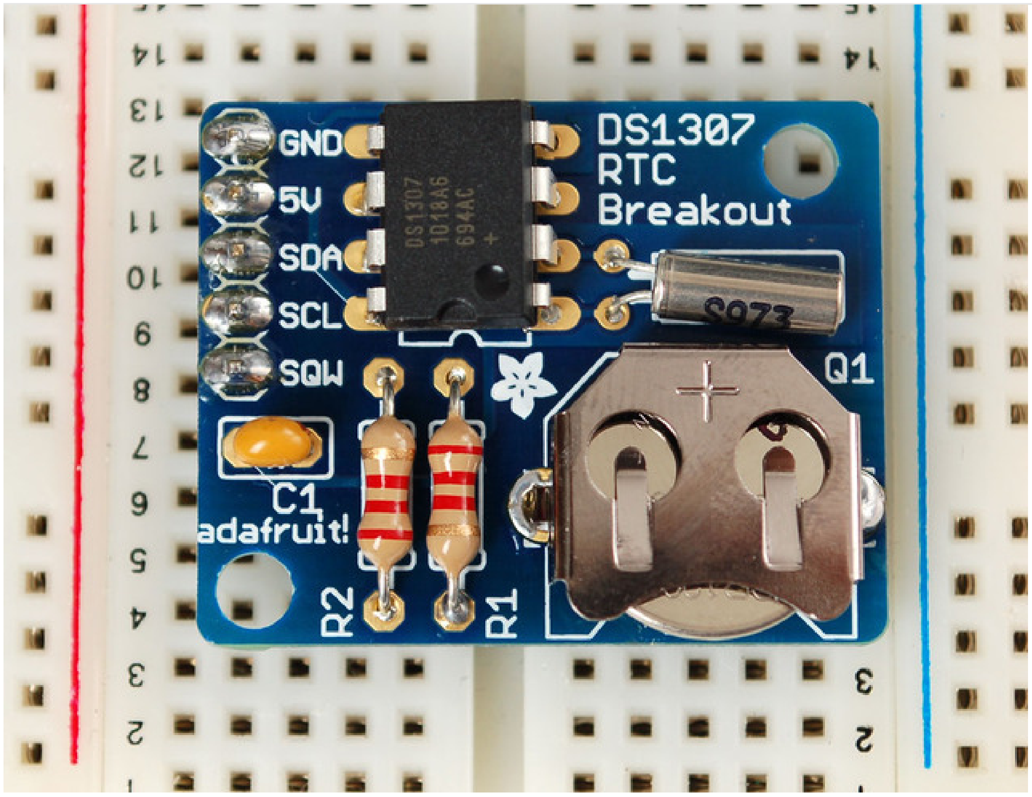
Created by Justin Cooper



Last updated on 2018-08-22 03:36:29 PM UTC

Guide Contents

Overview



This tutorial requires a BeagleBone Black running the Angstrom or Debian distribution. Other distributions may work, but are not tested as part of this tutorial.

The BeagleBone Black (BBB) is similar to the Raspberry Pi in that it is an ultra-low cost computer. This means that some standard computer features had to be cut out in order to reduce costs. For example, your laptop and computer have a little coin-battery-powered 'Real Time Clock' (RTC) module, which keeps time even when the power is off, or the battery removed. To keep costs low and the size small, a battery-backed RTC is not included with the BBB. Instead, the BBB is intended to be connected to the Internet via Ethernet or WiFi, updating the time automatically from the global **ntp** (nework time protocol) servers.

For stand-alone projects with no network connection, you will not be able to keep the time when the power goes out. So in this project we will show you how to add a low cost battery-backed RTC to your BBB to keep time!

Wiring the RTC

To keep costs low, the BeagleBone Black (BBB) does not include a battery-backed Real Time Clock module. Instead, users are expected to have it always connected to WiFi or Ethernet and keep time by checking the network. Since we want to include an external module, we'll have to wire one up. We'll go with the easy-to-use and low-cost DS1307. To make the job really easy, we'll use the the Adafruit DS1307 RTC Breakout Board Kit (http://adafru.it/264)- it comes with all the parts you need, even a coin battery!

The Kit does require a little light soldering. In theory you could use all the parts and build them onto a breadboard, but the coin holder is a little difficult since its not breadboard-friendly, so please go ahead and build the kit (https://adafru.it/clq).

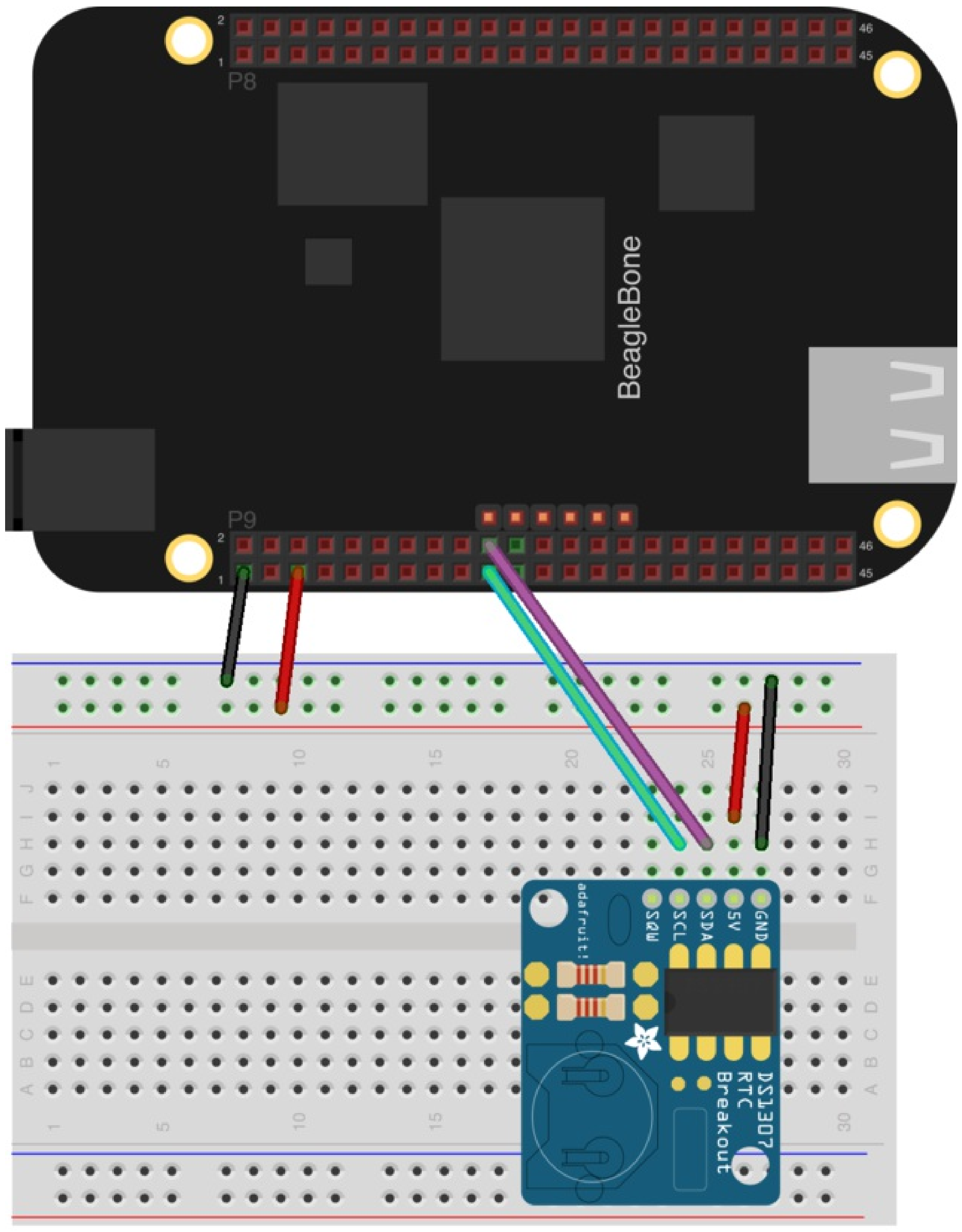
When building the kit, leave out the 2.2KΩ resistors - by leaving them out, we force the RTC to communicate at 3.3V instead of 5V, which is better for the BBB!

The diagrams below show the 2.2KΩ resistors in place - but please remove them either by not soldering them in or clipping them out if you did solder them in! This way, we'll use the BBB's 1.8K pull-up resistors to 3.3V

Wiring is simple:

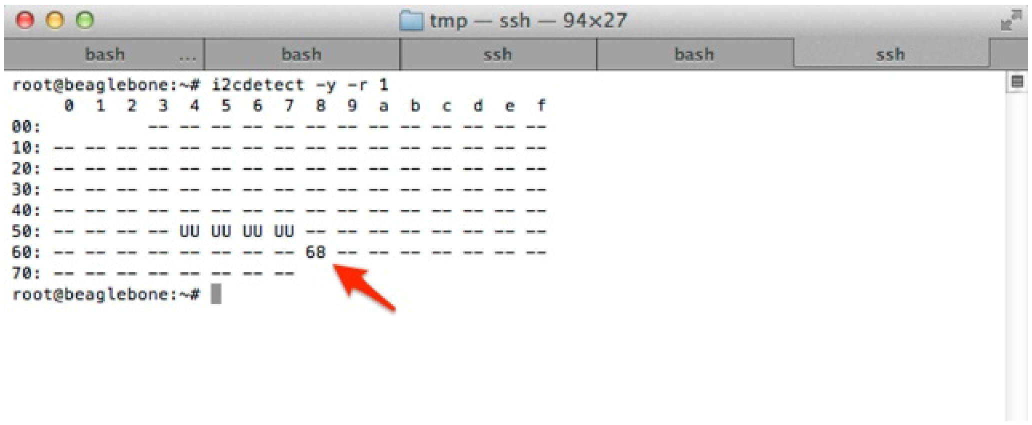
1. Connect **VCC** on the breakout board to the **P9_5** (VCC 5V) or **P9_7** (SYS 5V) pin on the BBB. **NOTE: The P9_5 VCC 5V pin will only be powered if a 5V adapter is plugged in to the barrel jack. If powering over USB use the P9_7 (SYS 5V) pin instead!**
2. Connect **GND** on the breakout board to the **P9_1** (GND) pin on the BBB
3. Connect **SDA** on the breakout board to the **P9_20** pin of the BBB
4. Connect **SCL** on the breakout board to the **P9_19** pin of the BBB

DONT FORGET TO POWER IT FROM 5V - THIS IS FINE AND SAFE FOR THE BBB - JUST MAKE SURE TO REMOVE THE 2.2K PULLUPS AS MENTIONED IN THE LARGE RED BOX UP ABOVE TO KEEP THE I2C PINS AT 3.3V LOGIC LEVELS



Check your wiring and insert the coin cell so that the + is facing up, the RTC requires a coin cell to be installed

Verify your wiring by running **i2cdetect -y -r 1** at the command line, you should see ID **68** show up - that's the address of the DS1307!



```
root@beaglebone:~# i2cdetect -y -r 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- UU UU UU UU -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
root@beaglebone:~#
```

Set RTC Time

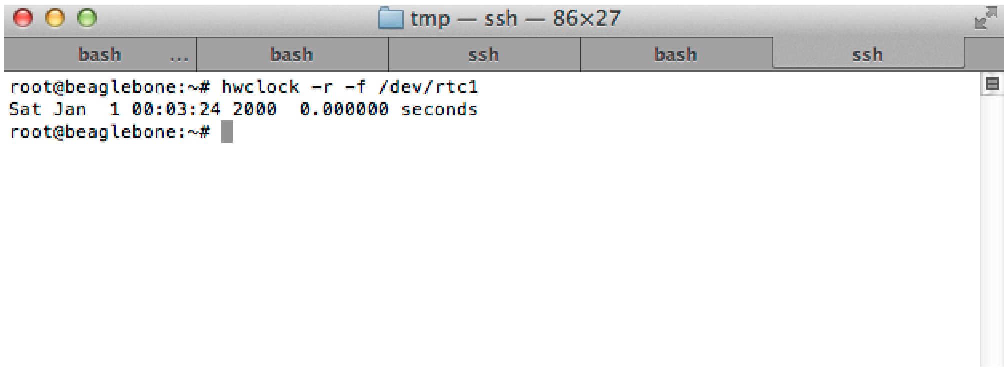
Now that we have the module wired up and verified that you can see the module with i2cdetect, we can set up the module.

Now, execute the following:

```
echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-1/new_device
```

Now you can check the time which will be read from the DS1307 module:

```
hwclock -r -f /dev/rtc1
```



```
root@beaglebone:~# hwclock -r -f /dev/rtc1
Sat Jan  1 00:03:24 2000  0.000000 seconds
root@beaglebone:~# 
```

If this is the first time the module has been used it will report back Jan 1 2000, and you'll need to set the time.

The quickest way to set the time on the BeagleBone Black is to execute the following:

```
/usr/bin/ntpdate -b -s -u pool.ntp.org
```

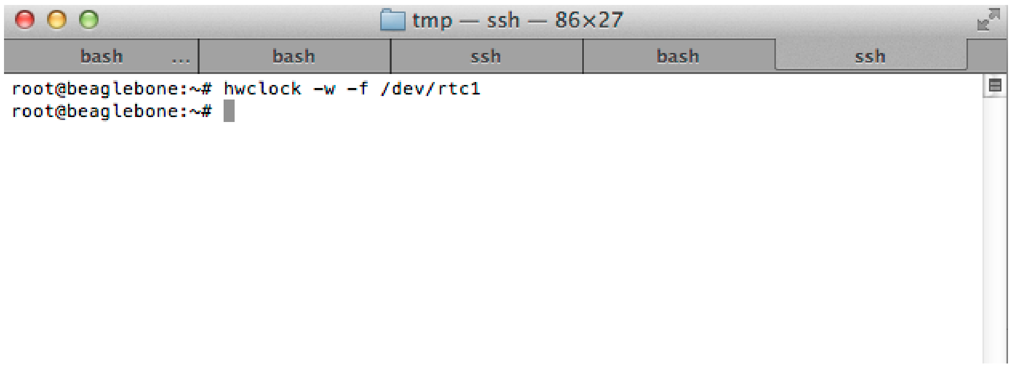
(note on recent Debian OS insatlls you might need to omit the /usr/bin/ part and just run 'ntpdate -b -s -u pool.ntp.org')

To validate the date and time were set correctly, execute:

```
date
#or more comprehensively (the RTC value will be inaccurate, we haven't set it yet):
timedatectl
```

Now that the system time is set correctly, you can execute the following to write the system time to the DS1307:

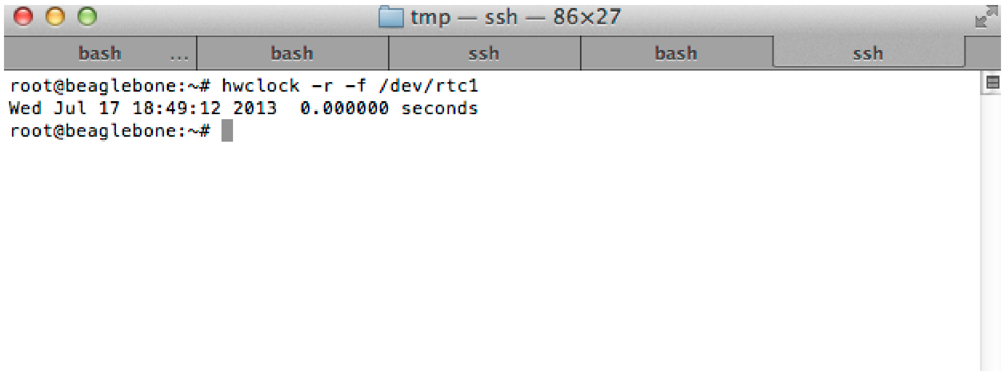
```
hwclock -w -f /dev/rtc1
```



```
root@beaglebone:~# hwclock -w -f /dev/rtc1
root@beaglebone:~# 
```

You can verify it was set correctly by executing the following command to read the date and time from the DS1307 RTC:

```
hwclock -r -f /dev/rtc1
```



```
root@beaglebone:~# hwclock -r -f /dev/rtc1
Wed Jul 17 18:49:12 2013  0.000000 seconds
root@beaglebone:~# 
```

Next, let's create service that will run each time you boot up your BBB. To start, create a directory and script that will be executed:

```
mkdir /usr/share/rtc_ds1307
nano /usr/share/rtc_ds1307/clock_init.sh
```

Now, with the nano text editor open, copy the following into the clock_init.sh script:

```
#!/bin/bash
sleep 15
echo ds1307 0x68 > /sys/class/i2c-adapter/i2c-1/new_device
hwclock -s -f /dev/rtc1
hwclock -w
```

Next, we'll create a service that will get started on boot, and execute the script we just created:

```
nano /lib/systemd/system/rtc-ds1307.service
```

Copy the following contents into that file, and save it:

```
[Unit]
Description=DS1307 RTC Service

[Service]
Type=simple
WorkingDirectory=/usr/share/rtc_ds1307
ExecStart=/bin/bash clock_init.sh
SyslogIdentifier=rtc_ds1307

[Install]
WantedBy=multi-user.target
```

After saving the file, we'll need to actually enable the service so it starts each time as the system boots:

```
systemctl enable rtc-ds1307.service
```

You can always manually start and stop the service as well:

```
systemctl start rtc-ds1307.service
systemctl stop rtc-ds1307.service
```

That's it! Reboot your system and check that it all works:

```
shutdown -r now
```

There are a few things you could do to make this even better. One would be for better error checking, and failure modes in the bash script. You could check to ensure that the ds1307 was actually connected prior to enabling it. This is just a barebones example of how to get it working!

Ubuntu Upstart

If you are using Ubuntu and you'd like to have the RTC taken care of by 'upstart' here's some code to do that, place in **/etc/init/rtc-ds1307.conf**

```
# Ubuntu upstart file at /etc/init/rtc-ds1307.conf
# Enables DS1307 RTC Breakout

pre-start script
mkdir -p /var/log/rtc-ds1307/
end script

start on runlevel [2345]
stop on runlevel [06]

script
exec /usr/share/rtc_ds1307/clock_init.sh >> /var/log/rtc-ds1307/rtc-ds1307.log
end script
```