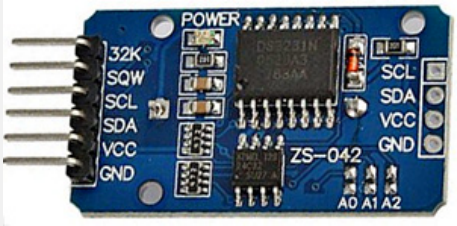
Using a $1 DS3231 RTC & AT24C32 EEprom from eBay

Posted on May 21, 2014

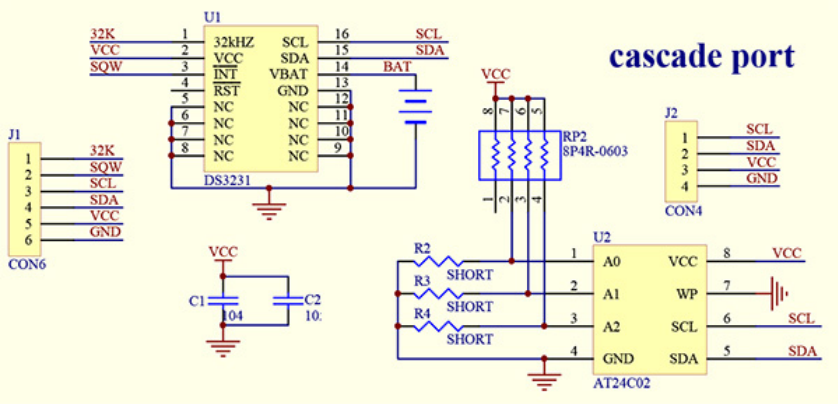


A DS3231N for less than $1? Really? And most of these boards use the industrial SN versions (rated to -40°C)

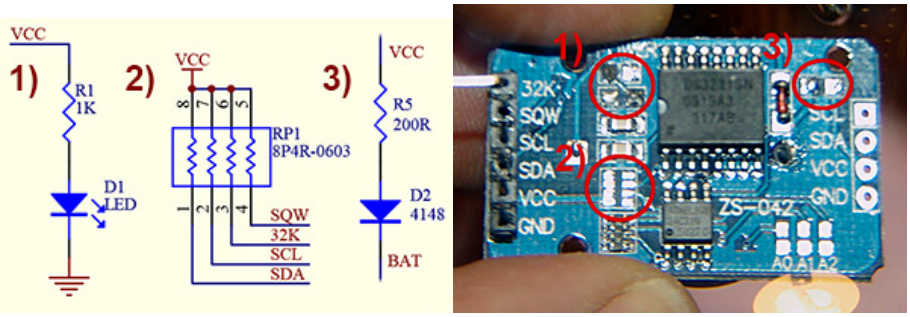
Since the Cave Pearl is a data logger, it spends most of the time sleeping to conserve power. So you could say that the most important sensor on the unit is the Real Time Clock, who's alarm signal wakes the sleeping processor and begins the cascade of sensor readings. I built the first few beta units with the DS3231 Chronodot from Macetech (about $18 each), but I kept on stumbling across cheap RTC modules on eBay, Amazon, etc. and I eventually bought a couple to try them out. While I waited for them to ship on the proverbial slow boat, I did some digging, because these modules were (almost) selling for less than the chip itself if I bought them directly from trusted sources like Digikey, Mouser, etc.

So perhaps they are counterfeit chips, which are simply pin & code compatible? I also found rumors about "ghost" shifts, where legitimate manufacturer plants/equipment are used off the clock to produce extra parts. Or legitimate production runs which test out defective (if 10% of a run's chips are bad, they often scrap the entire run) but someone intercepts the chips before they can be destroyed, and they resurface on the grey market. But even with all these possibilities in mind, I still have to make the Pearls as inexpensive as possible if they are going to be deployed in large numbers, and having an I2C eeprom on the board for the same money, made the temptation too great to resist.
(and you get temp to 0.25°, although only to ±3°C accuracy)

When the RTC's arrived they had an LIR2032 rechargeable battery underneath the board, and a LED power indicator above. I had a feeling that neither of these were going to be friendly to my power budget so I went hunting for the schematics to see what I could do to improve the situation. I quickly found an Instructables post which described how to remove the battery charging circuit from a very similar DS1307 module, and then I found the datasheets and schematic for my DS3231 board over at at a site in Europe. Most of the parts were pretty straight forward:



But thanks to the tutorial by msuzuki777, I immediately zeroed in on a few parts that could be removed easily:



The power indicator (1) was pretty pointless, so that was the first thing to go. I already had pullups on the I2C lines, so they were not needed here, but they were in a combined 4 resistor block, which meant that to get rid of the pullups on SCL and SDA, I also had to remove the pullup on the alarm line. This had me a little concerned, as that alarm line is vital to the whole design. Without that resistor on SQW, I am relying on the weak internal processor pullups keep the alarm line high with:

```
digitalWrite(INTERRUPT_PIN, HIGH);  //pull up the interrupt pin
```

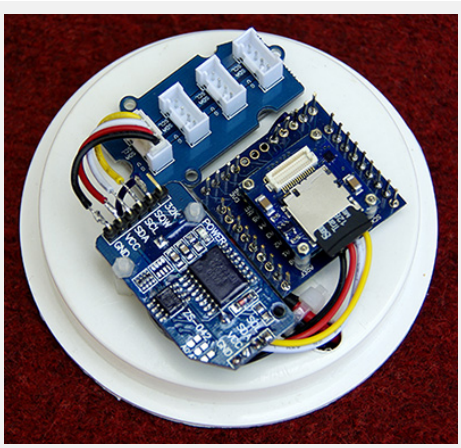
Fortunately the pin stays high in all sleep modes and *so far* everything has been working with this setup (so fingers are crossed….again… 😉)

Then I looked at the 200Ω resistor & 1N4148 diode (3) that are supposed to provide a trickle charge to the rechargeable battery, though the folks at BU suggest this is a bad idea. The LiR2032 that these modules ship with is 3.6v, and while capacity varies depending on where you buy them, most provide 35mah to 45mah capacity. Looking at the power draw from the DS3231, a fully charged battery would keep the unit backed up for at least 200 days (in a perfect world, with no self discharge, etc) But, it requires a 4.2v charging voltage for maximum charge, so vcc would have to be above 4.3-ish volts. I don't anticipate my 3x AA power supply staying in that territory for the duration of a long deployment (especially if I end up powering the units from cheap AA's) so there really was no compelling reason to keep the charging system in place. Once I de-soldered the resistor, I popped in a CR2032 (3v 240mAh) as a replacement which should backup the clock for several years of operation.

Libraries for that RTC?
I am using the Date, Time and Alarm functions in the library from Mr Alvin's github,

which is based largely on Jean-Claude Wippler's (aka JeeLab) excellent RTC library.
And it's worth noting the clear alarm interrupt issue over in the Arduino playground.

Then we come to the AT24C32N (2.7 to 5.5v) memory chip that is also on this breakout board. Another of those 4 resistor bricks is lifting pins 1,2 and 3 to Vcc, so according to the eeprom datasheet this unit is being set to 0×57 on the I2C bus. There are pads there to ground out these lines if you need to reassign the address to something else. Although I have already determined that eeprom is not the power savior I hoped it might be (all that eeprom reading & writing uses about 1/3 the power of simply writing the data to the SD card in the first place) it's presence lets me be really lazy on the coding and just pop *any* numbers or characters that I want into a PSTRING'd buffer which then gets sent to a standard eeprom page writing routine. This flexibility allows me to swap sensors with dramatically different output, while retaining essentially the same code to handle the eeprom loading and the transfer of data back out to the SD card. If you want more information about that you can head over my earlier post on buffering sensor data to an I2C eeprom for the gory details.



The May 2014 build of the data logging platform, which used a hacked Tinyduino light sensor board to regulate & pull up the I2C bus. SQW is soldered to interrupt pin 2. **Later in 2014 I switched to Pro Mini style boards with 3.3 v regulators, so I left that four resistor block in place (2 in the schematic above) to provide I2C and SQW pullup.**

To top it all off, the cascade ports on the far side of the module let me "just barely" fit the rtc, the I2C hub (with corners sanded off), and main TinyDuino stack onto the platform in the middle of my housing. I am lifting the voltage regulated I2C bus traces from the TinyDuino light sensor board, so I am also hunting around for an off the shelf vreg & level shifter combination to replace that hack (because that bit of soldering is a pita). But overall, I am very happy with this build, as all the central data logging functions have come together into a nice securely mounted package, that should withstand significant knocking about during the deployment dives. Of course there is plenty of field work testing still to be done, so time will tell (sorry, couldn't resist…) if these cheap RTC's will cause more trouble than they are worth.

Addendum: 2014-05-21
It just occurred to me that sooner or later Tinycircuits will be releasing an RTC board, and that will give me a chance to directly compare these cheap boards to a "trusted" clock signal provided that their chip does not want the same bus address. Or if their clock wants the same I2C bus address as this eBay RTC, I could use a DS3234 on the SPI bus. I will post an update when I can run that test to spot clock drift, alarm errors, etc. Several sites have mentioned that *real* DS3231's drift about 2 seconds per month, while the cheaper ds1307's drift 7-10 seconds per day. If you have the right equipment, you can make the chip even more accurate by adjusting the aging offset register.

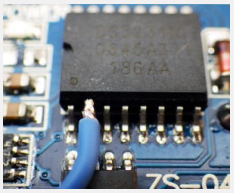
Addendum: 2014-05-21
I just realized something else odd about my setup here. The I2c bus is held at 3.3 volts by the regulator on the tiny light sensor shield, but I am pulling up the SQW via the tinyduino cpu, which is following the voltage on the battery pack because the tiny CPU is unregulated. So the pull-up voltage on the alarm line is out of sync with the voltage seen by the rest of the DS3231 chip….hmmmm.
(2014-10-28 : data sheet says its Ok to pull the line all the way up to 5v, even on Vbatt)

Addendum: 2014-07-01

I created a very inexpensive 3-component data logger with this RTC, a Pro Mini mcu board, and a cheap sd card adapter. And you can see a post about the latest version of that logger concept here which has added a power shutdown feature. In those Pro Mini based loggers I do not remove the I2C pullup resistor bank as shown earlier in this post (2 in the photo above), as the removal is only needed if you already have pullups in place, as I did when using the hacked Tinyduino light sensor board to drive the RTC. I have built *many* loggers now, and some of them have come close to 400,000 alarms & eeprom write cycles, so these cheap RTCs are proving to be pretty durable.

Addendum: 2014-10-28

I have noticed that when I power this module from Vcc at 3.3v, it draws around 89 µA. But according to the datasheet,the RTC should only draw ~2 µA on average when powered from Vbat. (1µA baseline plus about 500µA for 100ms every 64 seconds when the crystal is doing temperature compensation) Nick Gammon found an elegant way to power a DS1307 by connecting Vcc to one of the Arduino pins , driven high in output mode when the system is active. (look about half way down the page) When the Arduino pin is low, the clock reverts to battery power, and goes into the low current timekeeping mode. But according to the datasheet, Bit 6 (Battery-Backed Square-Wave Enable) of control register 0Eh, can be set to 1 to force the wake-up alarms to occur when running the RTC from the back up battery alone. [note: This bit is disabled (logic 0) when power is first applied] So you can still use the RTC to wake the Arduino, even if you have de-powered it by bringing the pin low. I have tested this and it seems to work fine on my RTC modules, reducing the sleep current by about 70 µA. (or ~ 600 mAh per year = almost 1/4 of a AA) Tests are underway now to see if this is stable as a direct jumper to the pin without using a current limiter, which might give me a problem with inrush current unless I also add a resistor as N.G. did.



Wedge a tweezer tip behind the pin and "gently" lever it away from the board as you apply an iron to the pad. Then solder your pin-power jumper directly onto that raised leg. At this point *the chip's Vcc pin is no longer connected to the Vcc line on the breakout board*, so you can leave power on the board's Vcc line to pullup SDA,SCL,SQW and supply power to any I2C devices / sensors you have connected to the cascade port.

And since my loggers are going in caves where the temperature does not change very quickly, I am bumping the temp conversion time from 64 to 512 seconds as per Application note 3644, in theory reducing the battery drain to < 1 µA. It's a little unclear from that datasheet if this only really works on the DS3234 (?) but if it does this puts the battery discharge on par with electrolyte evaporation if Maxim's coin cell lifespan estimates are to be believed.

And finally, doing this means that you are relying on the Cr2032 to power the clock for a substantial amount of time, so you need to make sure you are not using fake coin cell batteries. Name brand packaging is no guarantee of good batteries either! In learning this little lesson I discovered that you can not simply read a CR2032 coin cell with your volt meter to determine if it is healthy, as the *no-load voltage stays above 3v even when the cells are nearly dead*. As per the Energiser datasheet, I read the cell with a 400 Ω resistor pulse load (for 2 seconds). If that gives me >3v I call the cell good. If you are stuck out in the field without a meter, check if the battery bounces well.

I do wonder if its worth putting a 100uF multilayer ceramic capacitor on the coin cell to buffer the impact of the alarm events. But I don't know how much I would then loose to capacitor leakage. Abracon seems to

think its a good idea in their application note, claiming 11 µA leakage for a 100µF MLCC. But that is more than 3x the current draw of the DS3231 in timekeeping mode.

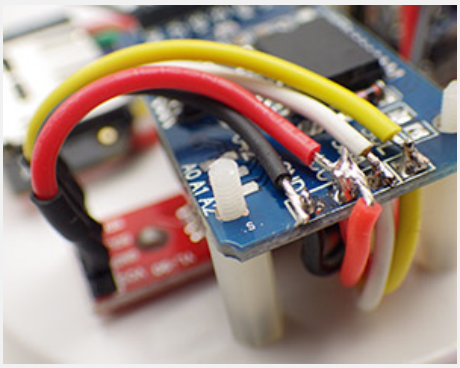
NOTE: If you try powering the entire breakout board from a digital pin, you are essentially turning the onboard SDA & SCL resistors into pulldown resistors and these fight against the Atmels internal pullup resistors on the 328 that get enabled by default in the two wire library. For details on how to fix that problem, check out this post on the Arduino playground: DS3231 drawing 200+µA through SDA/SCL Also note that I had to go all the way to *(x86)\Arduino\hardware\arduino\avr\libraries\wire\utility* to find the twi library on my machine, but if you power your DS3231 by lifting the pin from the board like I do, the library edit does not change the sleep current.

Addendum: 2014-11-04

This 32k AT24C256 is pin for pin compatible with the 4K AT24C32 on this RTC module. For only $1, it's really tempting me to do one more little modification to the RTC breakout, although on reflection I think it might be quite handy to have two easily accessed eeproms in the system, using the smaller one for persistent storage of calibration & configuration info, and the other much larger one for sensor data. Keeping the 4K eeprom will limit the I2C bus speed to 100 kHz, while the larger AT24256 opens up the possibility of raising the I2C bus speed to 400 kHz.

Addendum: 2014-11-05

Testing confirms that the AT24C256 is a drop in replacement. The code I was already using to write data to the eeprom on the RTC breakout worked fine provided I changed the I2c address to 0x50 (while the 4k eeprom on the rtc breakout is 0x57 because its address lines are pulled up). In my case, the larger eeprom allows me to buffer 512 of my two-page write cycles before having to transfer the data out to the SD card. And after some testing, I have confirmed that both eeproms go into standby mode at 1 µA when they are not being accessed. The only challenge is that this many buffered readings represents several days worth of data…so I will need to come up with some kind of procedure for shutting down the loggers without losing information. One solution would be to add a function that flushes the entire eeprom to the SD card during setup. That way simply hitting the reset button would make sure that any residual data in the buffer gets saved before I disconnect the batteries.



I simply let the wires I am already using to tap the I2C lines on the cascade port poke through, giving me solder points for the eeprom. Don't forget to remove the 10K's on the little eeprom board or you could be left with too much pull-up on the bus. 15mm M2 standoffs give enough room to comfortably tuck the EEprom board under the RTC breakout.

In some of my older loggers that were put together ages ago, there is not enough space to easily do this jumpering right onto the RTC breakout, so I came up with some "in-line" eeprom upgrades that I could just drop in without changing any other wiring on the build.

Of course, you could do this with *any* I2C device.

Addendum: 2014-12-02

I have recently run into another RTC related issue, which is how to *set* the RTC's more accurately. Now that I have multiple data loggers on the go, the led pips show me that the loggers are as much as 4 seconds* different from each other, and that gets even more pronounced when I use different computers to upload sketches. Retrolefty has proposed one method for "syncing by hand" at the playground. I will post some results when I find out if the Whac-A-Mole method reduces my inter-unit time offsets.

One solution would be a sketch that uses an Ethernet shield and connects to an internet time server. Then you could get the offsets down to average round-trip time (using the nearest NTP server) plus the serial communication. But I do not have an Ethernet shield, so that is a non-starter. Some use a GPS for an accurate time signature, or a dedicated time signal receiver. But my PC is already synced, so buying hardware just to reproduce information that is already available seems like overkill. A more logical approach would be to have two programs, one running in the PC environment then second running inside Arduino. Then both programs could communicate (PC -> sends time stamp via serial line -> Arduino reads value from serial line & sets clock to match). I have not found a set like this yet.

In addition, I would like to have all my loggers running UTC, but that's easily address by just setting my machine to UTC before setting the clock. UTC avoids all the problems with 'local' daylight savings time, etc.

* *It looks like I might have been causing that problem by opening the serial window to to check that the clock was updated properly. Makes me wonder why the RTC set sketch was written with serial output in the first place?*

Addendum 2014-12-04

Someone at the allaboutcircuits.com forum has done *accuracy verification testing* on these cheap RTC boards and found the chip to be well within the DS3231's "official" spec:

http://forum.allaboutcircuits.com/threads/seen-the-new-high-accuracy-rtc-for-1-99.99839/

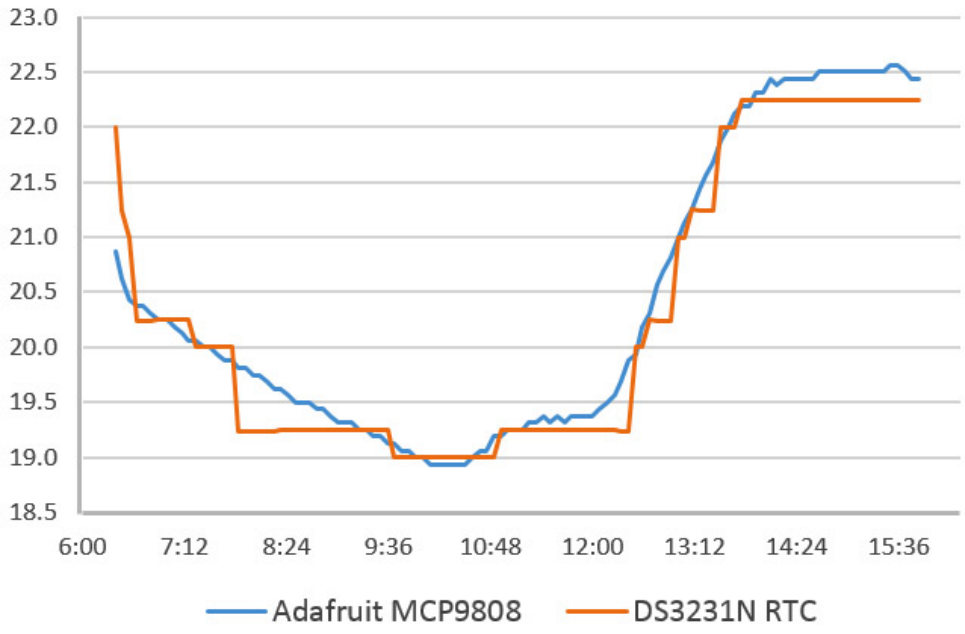
This is good to know, although of course one source/batch doesn't confirm them all when you are dealing with cheep eBay knock-offs. For a different approach to testing, jremington over at the playground notes: "By comparing the rising edge of the RTC 1Hz square wave output to that of the 1 Hz PPS output of a GPS unit with a good satellite lock, you can determine within a few hours how much the RTC is drifting. "

See http://www.romanblack.com/onesec/High_Acc_Timing.htm
(no need to use a PIC, though…)

Addendum 2014-12-06

I have been noodling around with new sensor combinations for my next set of builds, and I thought I would post a quick overnight comparison of the DS3231 temperature register (rated at ±3°C) to data from the Adafruit MCP9808 (± 0.25°C).

Degree Celsius vs Time: *(5 min samples)*



You can see that the DS3231 has a much lower bit depth, but I was pleasantly surprised by how closely they tracked each other. If the datasheet claims are to be believed, the 9808 should be dead-on in this temperature range. This gives me more faith in the data from that humble RTC, which currently records ambient temperature in my drip sensors.

Addendum Update: Although I did not catch it when I posted this graph, I was converting the LSB portion of the temperature register with:

```
TEMP_degC = ((((short)MSB << 8) | (short)LSB) >> 6) / 4.0;
```

from Coding Badly at the Arduino forum. There should have been no "bumps" on that graph smaller than 0.25°C. But what I was actually getting a mix of xx.00, xx.25, xx.244 and xx.238 in my data. No half degrees, and no temps that read xx.75 You can see those temps are missing, as "steps" in that graph.

So I tried this code to fix that with this code from the Arduino forum:

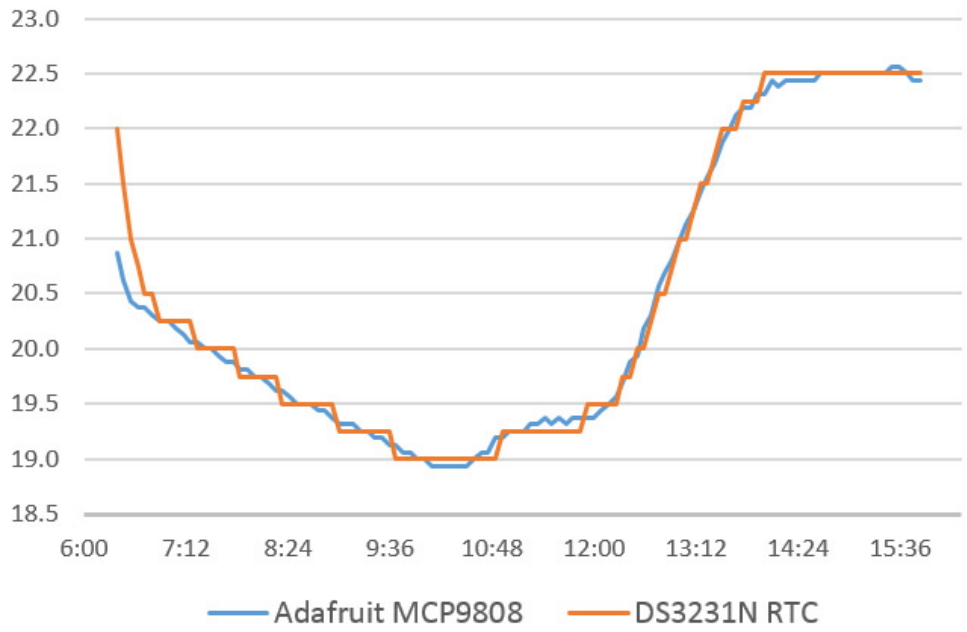
```
Wire.beginTransmission(DS3231_ADDRESS);
Wire.write(0x11); //location of Temp register MSB, LSB at 0x12
Wire.endTransmission();

Wire.requestFrom(DS3231_ADDRESS, 2);
bytebuffer1 = Wire.read(); // Here's the MSB which is an int
bytebuffer2 = Wire.read();  bytebuffer2 = bytebuffer2 >> 6;
// the upper 2 bits of the LSB represent quarter degrees 00=.00 01=.25 10=.50 11=.75

TEMP_degC = float(bytebuffer1);

switch(bytebuffer2){
case 0:
TEMP_degC = TEMP_degC + 0.00;
break;
case 1 :
TEMP_degC = TEMP_degC + 0.25;
break;
case 2:
TEMP_degC = TEMP_degC + 0.50;
break;
case 3:
```

```
TEMP_degC = TEMP_degC + 0.75;
break;
}
// see http://forum.arduino.cc/index.php?topic=262986.15 for temps below zero with no floats
```

But I got the same result with that code too which is very puzzling to me?? Where are the .238 fractional temperatures coming from? Why do I never see xx.5 or xx.75 temperatures?

Addendum Update Update:

So it turns out that both examples of code above work fine, but the way I was converting the fractional part of the decimal (so I could print them as integers: printing real #'s takes too much ram) was incorrect. All my other temperature sensors provide at least three digits of information after the decimal place so I had been using fracTemp=(TEMP_degC – wholeTemp) * 1000; to extract the fractional data. But this did not work for the RTC fractional data. Changing it to fracTemp=(TEMP_degC*100) – (wholeTemp*100); converts the decimal part of the RTC temperature into integer values normally. Thanks to Stack Exchange for showing me that you need determine exactly how many decimal points you want to turn into the integer before you do a conversion like this, or the calculation yields weird results. In my case the error changed xx.05 into xx.0244, and xx.75 into xx.238. Fortunately that error was correctable, so here is what that graph should have looked like:



Addendum 2014-12-20

Recent fieldwork gave me a chance to check clock drift on six loggers using these RTC boards. All of these RTCs were set at the end of August and over the course of about 4 months, all of them lost between 30-40 seconds. That puts these cheap units over the minute per year territory that I see claimed for "real" DS3231 breakouts like the Chronodot, but not by enough to make me worry too much as this was a pretty crude test. These modules are still way better than any DS1307 alternatives.

Addendum 2015-01-11

I have been putting together some smaller underwater sensors using two inch pvc pipe, and the tight curved profile of the housing forced me to flip the RTC over. As soon as I did this, I realized that I should have been mounting the RTC this way *all along*, as it makes it easy to replace the coin cell without undoing the nuts on the standoff bolts. And if I am going to be pin-powering the RTC, I will probably need to change those coin cells regularly. It also lets me use shorter 12mm standoffs and still tuck everything under the RTC.

Obvious, I know, but it still took me ages to realize it…

Addendum 2015-01-22

Steve Hicks over at envirodiy.org has posted on how to convert the DS3231's epoch time (ie: #number of seconds since January 1, 2000) into unix time and here they convert unix time into Excel standard dates with [=CELL/(60*60*24)+"1/1/1970" note: you have to have the RTC set to UTC for this to work]. Using epoch time lets you store or compare times as a single 32-bit number (another conversion method here) rather that dealing with 6 numbers and details like the number of days in each month and leap years. A very handy tip. You can view a datalogger script example using *long epoch = now.unixtime();* over on Github.

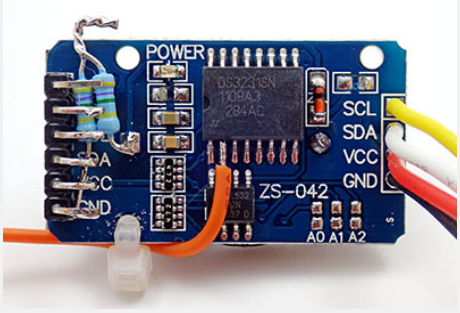
P.S. The RTC library I'm currently using gives me unix time with:

DateTime now = RTC.now();
followed by:
Serial.print("Seconds since midnight 1/1/1970 = ");
Serial.print(now.unixtime());

But if yours does not, it is fairly easy to calculate an epoch time integer from the standard YY MM DD MM SS format, *if you have your clocks set to UTC.*

Addendum 2015-03-11

I have decided to pin power all of my next generation of loggers, including the long chains of DS18b20 temperature sensors I have been working on. But I still don't know exactly how much impact generating the interrupts will have on the coin cell over time, so I have added a voltage divider connected to the backup coin cell on RTC board, with the center drawn off to an analog input pin on the Arduino. I am hoping these 4.7 MΩ resistors will add only 0.35µA draw to the ground line and perhaps double that when the ADC input capacitor is being charged for a reading. The readings wobble a bit without a capacitor to stabilize them, but I was afraid that leakage on an MLCC would be larger than the RTC's sleep current so I left it out. I read the pin three times with a 1ms delay, throwing away the first reading and averaging the next two, and that gets me a reading pretty close to what I see on an external volt meter. But CR2032's are lithium batteries, so I might need to put some kind of load on the



The coin cell monitoring divider tucks nicely under the header pins, which I fold back in my builds for the small diameter housings. The two resistors take the voltage to 1/2 actual, as some of my coin cells read higher than my system voltage of 3.3v when they are new and have no loads on them.

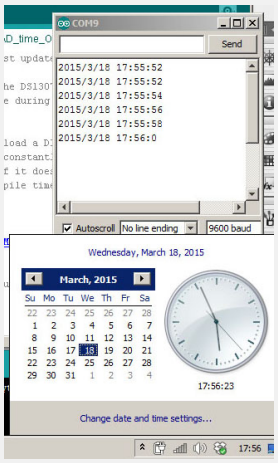
coin cell to actually read it's capacity. I was thinking I could do this by forcing a temperature conversion while the pin power is removed. (setting bit5 of reg 0Eh which draws 575 µA for 125-200 ms) This approach would waste some energy and create time delays, so I will do my first few test runs without the "load" to see if I can interpret the reading from that voltage divider without it.

Addendum 2015-03-13

There is another question about pin powering these RTC's that is niggling at the back of my mind: What happens when I have Battery-Backed Square-Wave Enable set, so the RTC generates alarms when it is powered only by the backup battery, and I disconnect power to the main Arduino before the next alarm? Presumably the alarm still gets generated, but nothing can respond and reset it. My hope is that the open-drain SQW pin, which should only sink current, does not somehow create a circuit through the Arduino that bleeds away power in this situation. Especially now that I have the voltage divider in place…?

Addendum 2015-04-01

I just returned from another fieldwork trip, and I had the chance to do proper RTC drift checks on twelve data loggers that were deployed in Dec. 2014. After three months of operation they all had offsets between -24 to -30 seconds, and the remarkable consistency across these units made me suspect that I was looking at something other than random errors. I reset the clocks with the little netbook I had on hand, and re-checked the clocks. Sure enough every one of them was reading the current time -24 seconds. I checked the time on the six new new loggers that I had prepared before the trip and every one of them was exactly nine seconds slow (my computer back home is much faster than the netbook I take into the field). When I reset those new loggers with the netbook every one of them became 24 seconds slow. So it looks like the time lag caused by the compile & upload of the RTC set sketch was responsible for the majority of the offsets I reported back in December, and that these DS3234SN RTC's actually have a drift somewhere between 0-5 seconds over a three month deployment. This is well within the manufacturers spec. And now that I know the compile time is the limiting factor, at least I can be certain that the units all have the same negative time offset before each deployment.



To do the drift check, I screen grab the terminal window of a sketch that outputs the current RTC time *with* the windows system clock. Both are running UTC, and I make sure the computers clock was sync'd via web time servers.

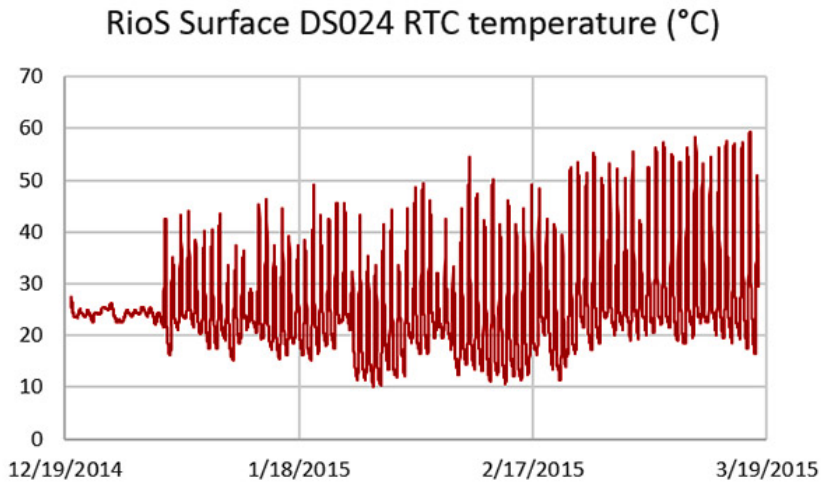
NOTE: With further testing I have found that if you simply hit the verify button before you hit the upload button, the resulting RTC time offset is reduced to 1/2 (or more for slower systems). On my home based system this reduced lag caused by the compile & upload from ~20 seconds to about 9 seconds. I expect to see even more of a difference on my really slow netbook. In theory you can improve things even more by removing the verify option as you upload. The RTC time setting sketch is the only place where I would risk errors to get to a faster upload time, since I immediately have to replace the RTC "setting" sketch with a "read time only" sketch to confirm it worked anyway.

Update 2016-10-14: I've been referring here to the setTime sketch that used to be provided with MrAlvin's library. This sets the RTC to the compile time with the command RTC.adjust(DateTime(__DATE__, __TIME__)); His new version has a method of setting the time using the

serial monitor, which removes the compile time lag time problem. I've gotten used to using setTime & getTime , so I still keep a copy of those older utilities on my GitHub. Paul Stoffregens DS1307 library uses the same compile time method to set the DS3231, but you have to install his Time library as well.

Addendum 2015-04-05

Just digging into the recent data set, and noticed that one of the drip sensors we left out on the surface (to act as a rain gauge) got baked as the local climate went into the dry season:



This is the record from the RTC, and I am surprised the batteries did not pop with the loggers internal temp hitting 60°C. The good news is that even after subjecting the RTC to this ordeal, the drift for this unit was the same as the units that were left down in the caves. This logger went back out for another stint in the tropical sun, as I am a firm believer in testing my builds to their limits.

Addendum 2015-04-07

That last deployment saw several loggers run successfully with pin powered RTC's so I though I should post the little code snippet I use to do that. I have the de-powering embedded inside the function that puts my loggers to sleep

In setup: *(note brackets missing around includes!)*

```
#include LowPower.h   // https://github.com/rocketscream/Low-Power
#include RTClib.h       // https://github.com/MrAlvin/RTClib
#define RTCPOWER_PIN   7 // this is the pin I have jumpered to the RTC's vcc leg
```

So after the setting the next alarm time in the main program loop

```
RTC.setA1Time(Alarmday, Alarmhour, Alarmminute, Alarmsecond, 0b00001000, false, false, false);
//The variables ALRM1_SET bits and ALRM2_SET are 0b1000 and 0b111 respectively.
RTC.turnOnAlarm(1);
```

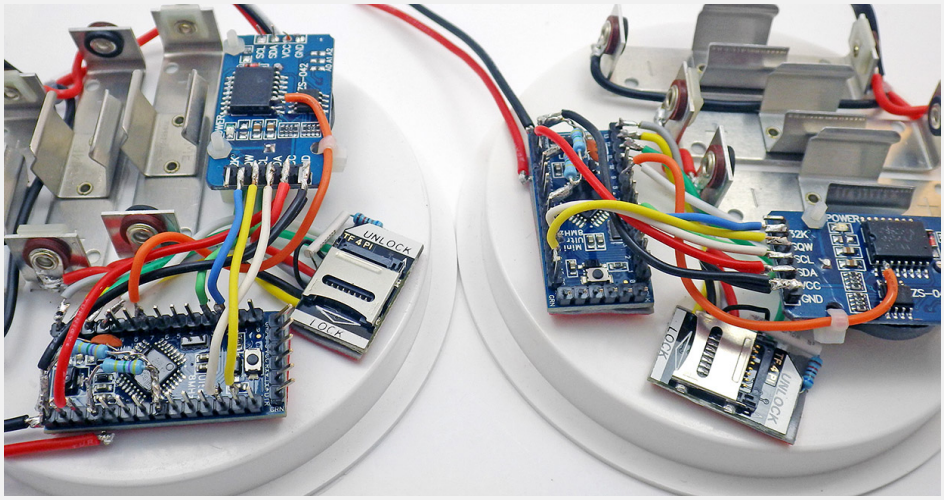
I use this function to de-power the RTC and the data logger

```
void sleepNwait4RTC()
{
//
#ifdef RTCPOWER_PIN       //if using pin power on RTC, now depower it:
pinMode (RTCPOWER_PIN, INPUT);
digitalWrite(RTCPOWER_PIN, LOW);
// driving pin LOW FORCES to the RTC to draw power from the coin cell during sleep
#endif
//
noInterrupts ();      // make sure we don't get interrupted before we sleep
attachInterrupt(0,clockTrigger, LOW);
interrupts ();          // interrupts allowed now, next instruction WILL be executed
LowPower.powerDown(SLEEP_FOREVER, ADC_OFF, BOD_OFF);
detachInterrupt(0);  //HERE AFTER WAKING UP
//
#ifdef RTCPOWER_PIN
digitalWrite(RTCPOWER_PIN, HIGH);     // about to generate I2C traffic
pinMode (RTCPOWER_PIN, OUTPUT);    // so provide power to the RTC
#endif
//
}
```

and clocktrigger is the ISR that updates a variable checked in the main loop

```
void clockTrigger() {
clockInterrupt = true;
}
```

So there you have it. After 3 months of reliable operation, and no coin cells killed off in the process, I am calling this good code. BTW this is how I currently connect the RTC boards to the Arduino:



For more information on how I assemble these logger platforms, visit the How to Build an Arduino Data Logger page.

Addendum 2015-06-10

After finding Rob Tillarts multispeed I2C bus scanner, I was happy to notice that all my I2C devices showed up on the higher speed scans. So I have started pushing the bus to faster 400 khz speeds with TBWR=2 on the 8Mhz boards, and TBWR=12 on the 16Mhz boards right after Wire.begin(); The DS3231 is rated for it. The larger AT24C256 eeprom that I have been adding to my loggers is also rated to that speed, but even the smaller AT24c32 on the RTC board seems to work ok at the higher I2C bus speeds, though it is only rated to 100kHz. Since I had been using the I2C communication delays as part of my led pips, I could immediately see shortened operating time (my pips became too short to see) . I have some doubts about whether a humble 8Mhz Arduino can run the I2C bus that fast. Without a scope, or some way to determine the capacitance on the lines, there's no way to know if I'm actually reaching 400khz with the 4.7kΩ pullups on that RTC breakout. But with quite a few run tests going well thus far, I think add the TBWR settings to my standard code build to shorten mcu up time.

Addendum 2015-06-17

Since these boards are always covered with flux, I picked up a cheap ($15) ultrasonic cleaner and used it on a batch of 12 of these boards with 90% Isopropyl alcohol. After the cleaning I put the used fluid in a jar, and this batch of goo settled out.

I have done a few more tests using a 2x 4.7MΩ divider to monitor the coin cell. The divider definitely works but as expected it also bleeds 0.32µA from the coin cell when the Arduino is powered & sleeping. If I remove power from the whole Arduino, the current drain from the battery through the divider rises to almost double that at 0.56µA. Pin Powering the RTC during uptime and letting it go into timekeeping mode (3 µA) while the Arduino sleeps (with the coincell divider in place) appears to be causing a 5-7mV drop per day on the CR2032 coin cell. With the 2300 mV minimum for the DS3232's vBatt, that probably means the coin cells will only provide about 4-5 months of pin powering before the little cells need to be replaced. This is a somewhat irritating as I thought I would get more time than that from the 240 mAh coin cells. I am suspecting there are other drains occurring somewhere.

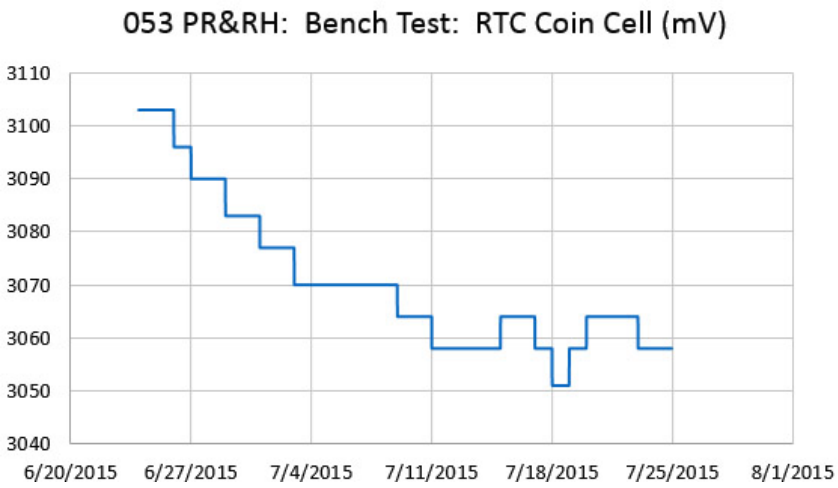
One trick I can try is to set the coincell reading analog pin to INPUT_PULLUP with a pinmode setting while the Arduino sleeps. This would raise the middle of the voltage divider to 3.3v – above the coincell. This will also send 0.7µA from the analog pin through the grounded leg of the divider. When I tried this I found that it also pushes about 0.03µA *back towards the coin cell's positive battery terminal* where I have the divider connected. I don't know if that power is flowing into the lithium coin cell (which is probably bad for CR2032's – but perhaps it would be ok with an LIR2032?) or into the RTC somehow (?) So this strategy would shut down the divider power leakage from the coin cell and hand it off to the much larger main batteries. This is much lower than the 89µA that the RTC draws if you power it via the Vcc line, but it seems a bit dodgey to flip flop between analog and digital modes on that pin all the time.

I will have to do more tests before I trust that this is not hurting the RTC or the Arduino. Having the lithium coin cells catch fire when their voltage got low would not make my day either. And if I was going to have a small constant drain from one of the pins I might as well just replace the coin cell backup battery with a capacitor – which could be kept charged by the pin that I am currently using to check the backup battery voltage. That way I'd never have to worry about changing the RTC batteries once I got the system rolling… hmmmm…I wonder what the capacitor leakage current would be?

P.S. In my tests to date, the faster 400khz I2c bus settings settings still seem to be working OK.

Addendum 2015-07-23

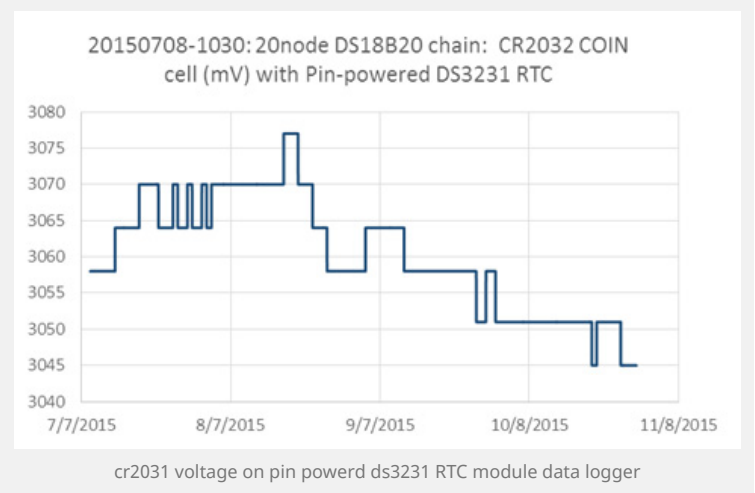
Looks like my earlier concern about the new divider creating and excessive drain the RTC backup battery were unfounded. Several of my bench test loggers saw an initial drop off but all of them seem to have leveled out around a nominal 3.05 v.



Most of the new batch have this 2 x 4.7 MΩ divider in place and I am now confident that it will be Ok to deploy those units, which likely will not be retrieved till the end of the year. Btw there is a fantastic page over at ganssle.com testing the behavior of CR2032 batteries at low currents. Granssle's article on Issues in Using Ultra-Low Power MCUs is worth reading. Hackaday's post on TI processors shows how far the art in low power operation goes. Ignoring self discharge, a CR2032 should be able to last about 4 years if the average draw stays below 5 µA, and the divider is adding ~ 0.7 µA to the RTC's base load. Actually the DS3231 datasheet specifies the average battery current is less than 3.0 mico-amps, so a typical 200 mAh CR2032 should be able to supply that for about seven years.

Addendum 2015-10-30

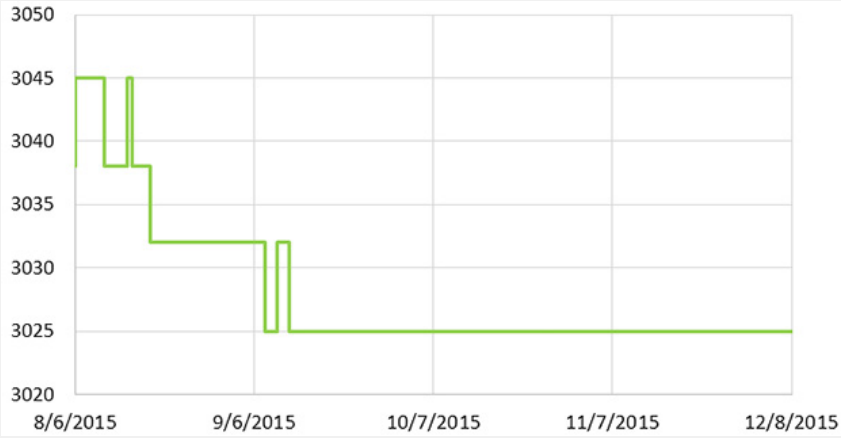
Just a quick update on that coin cell reading. I continued that test (on a logger with 20 DS18B20 temp sensors) and the coin cell voltage rose after the break then fell again to about 3040mv:



cr2031 voltage on pin powerd ds3231 RTC module data logger

So at least I am not seeing a catastrophic fail when trying to read the coin cell, but I am still left with the question of whether this reading actually means anything, given that the loading on these lithium cells is a mere 3µA when the RTC is in timekeeping mode. (If straight ADC reads don't work, I might try the LED/resistor method so that the coincell is loaded during the readings) I still might be shortening the lifespan of my loggers below my one year target with the pin powering technique if the coin cells can't go the distance. At 150-200 mAh /cell, there should be no problem…but I have made the mistake of counting those chickens before. And I still might need that 1µF cap across the lower resistor, which in theory will cost me ~1nA in leakage.

Note: Data from the batch of loggers deployed in Aug 2015 displays a similar pattern to my bench test results:

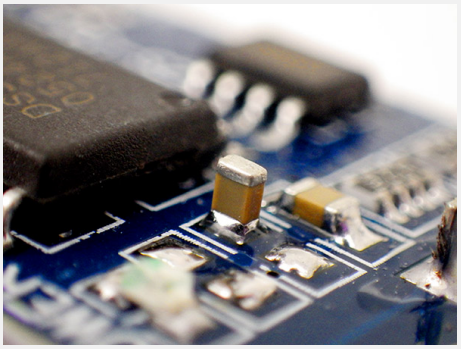


RTC coin cell (mV) record from a real world drip sensor deployment

All the loggers using 2×4.7MΩ dividers to track the coin cell leveled out somewhere between 3020 & 3040 mV, and I attribute the differences there to ADC & resistor offsets. So I will adopt this as a standard part of my new builds.

Addendum 2016-01-08

I prep these RTC's in runs of 10 to 20 pieces, as this makes the best use of the isopropyl alcohol in the ultrasonic bath. While I was de-soldering resistors on the latest batch (to disable that useless charging circuit) I realized that the first part of the UNO based Datalogger Tutorial (that I put together to help some teacher friends of mine bring Arduinos into the classroom) gives you great platform for testing a bunch of these RTC's quickly. You can just pop them onto the breadboard in quick succession before you invest any time cleaning them up. You don't even need to put the battery in! And the code that I posted to Github for that logger is about the simplest example you are likely to find of how to use this RTC breakout to wake a sleeping Arduino.



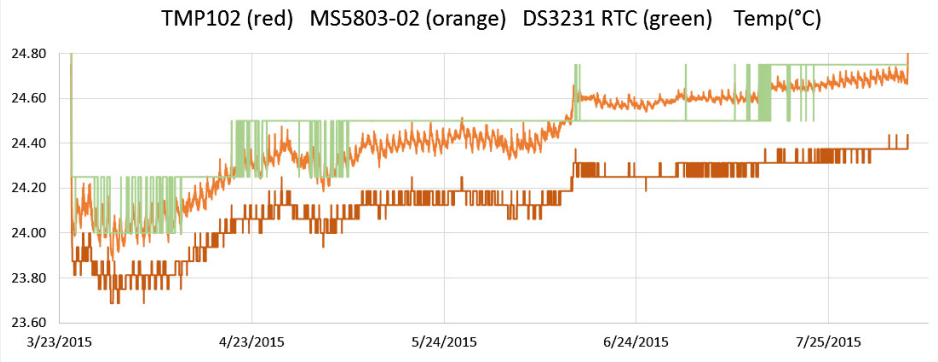
The most common manufacturing defect I see is IC pads being bridged by bad reflow, but you also see tombstone errors like this on $1 eBay boards…

Addendum 2016-01-16

Just stumbled across a post at Arduino Stackexchange on combining day-month-year data into strings, and using that as *a date-stamp file name*. This could be handy for "threshold/event" based loggers, as opposed to the more typical take a sample every X minutes approach. I think this method is limited by fat16 to generating a max of 512 entries in the root directory.

Addendum 2016-01-21

Data from a cave deployment of one of our multi-sensor units:



The MS5803 is a 24bit pressure sensor which has a metal ring in contact with the air, while the TMP is 12-bit sensor embedded under 3-4mm of epoxy, and the DS3231 RTC is inside the housing body

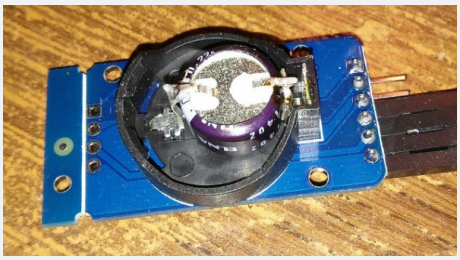
so I am impressed again with the temp accuracy of those humble RTCs. This is also a really great illustration of what gain when you add more bits to your ADC.

Addendum 2016-02-13

Over at raspberry-pi-geek.com they did some benchmarks with four I2C RTC's: the DS1307, the PCF8563, the DS3231, and the MCP79400. Their graphs show the DS3231 as the overall winner, suggesting that this is is a result of the temperature compensation. It will be interesting to see if they do the tests over again with temp variation to see how this affects the RTCs accuracy.

Addendum 2016-02-26

Just stumbled across a playground forum thread where user Tominakasi tried replacing the backup battery with a capacitor. He reached 24 hours of operation with a 0.22F capacitor, but I would need significantly more time than that in a logger application. If I play with some datasheet numbers over at Maxim's Super Capacitor Calculator, it looks like it might be feasible with a one farad cap. But folk's over at Sparkfun, seem to think that leakage current would be a serious problem. Since I am already tracking the coin cell voltage with a resistor divider on the top of these boards, I think I will pickup a 5v 1F super cap and try an experiment to find out how long it *actually* takes for it fall to the



One of Tominaksi's photos of his retrofit. There actually are capacitors built to coin cell shape/size specs for this purpose over at Mouser, but they are not much larger capacity than the 0.22F he used.

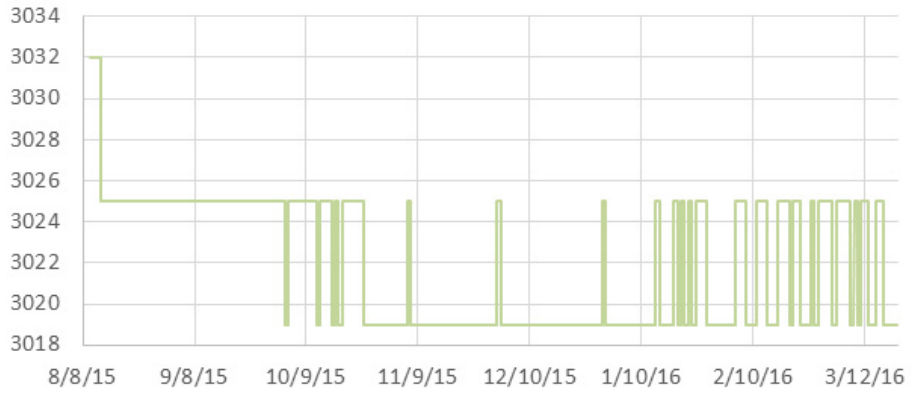
RTC's 2.3v minimum. It would not take much to connect one end of that cap to a separate digital pin and then top it up when necessary because the Arduino will keep a pin driven high even while sleeping. Probably not worth doing for a regular logger, but if I was building something completely potted in epoxy… hmmmm…

Addendum 2016-03-04

There must be a million great clock projects out there, but I stumbled across a couple recently that looked like they would be fun to build. The idea embedded in the Laser Cut Clock by [Buckeyeguy89] really has legs, and I think it could go on to some very interesting higher levels of complexity. And I am sure that I am not alone in drooling over the Ferrofluid Clock by Zelf Koelma, to the point of wondering how I could drive batch of small electromagnets with an Arduino…

Addendum 2016-04-07

Another coin cell curve (mV) from a longer deployment:



This was from a 2×4.7 MΩ set, so I am more confident that we will go a year even with the added drain from the divider. I have since switched over to using 10 meg Ω resistors, but there is some question of whether the ADC sample & hold caps can will get enough current to read that properly. I've been dating the coin cells with a marker to help keep a tab on their lifespan.

Should have been doing this from the start…

Addendum 2016-04-21

Just had to post a link to the Arduino Sport Watch Instructable by Alexis Ospitia which combines this DS3231 board with a pro-mini and one of the ubiquitous Nokia 5110 LCDs.

Addendum 2016-05-21

There are a host of changes coming as the ATmega 328p becomes the *328pb.* But relevant to this thread, the full-swing oscillator driver circuitry is being removed. While this won't affect a low&slow application like

data loggers, it might spur a few more people to look into the 32K output from these boards. If that's your thing, there are some interesting frequency measurement guidelines over at sitime.com

Addendum 2016-05-18

Just had to add a shout out here to Luke Millers Tide Clock as a great use for the DS3231 breakout, and a great addition to his Open Wave Height logger project. There are only a handful of us focusing on underwater logging, and his work on the MS5803 was a great contribution.

Addendum 2016-06-13

I've been trying out some $2 coin cell testers from eBay, and so far they seem to be working ok. There's a big logger service trip coming up, and this will come in handy.

Addendum 2016-07-02

Just found out about a time-related function in Excel that is really useful to know about if you are trying to isolate subsets from your time series data:

> *…use a helper column and then filter on the results of the helper column. Assuming your date/time record starts at A2, and column B is available, then enter the time interval you wish to filter on in cell B1 (e.g 10 or 30 etc). Then enter the following formula in cell B2 and copy it down the rest of the column:*
>
> ***'=MOD(MINUTE(A2),b1)=0***
>
> *This will provide a TRUE/FALSE value if the time conforms to the interval value contained in cell B1. Then filter all the records based on the TRUE values in column B.*

I tend to run my bench tests pretty fast to give new builds a real workout, but then I end up with far more data than I need for calibration / normalization. This little trick works a charm to bring that back to more typical 15 minute sample intervals.

And while we are on the topic of time in Excel, it's worth mentioning that the program sometimes refuses to convert time stamps from commercial loggers into its native number format. In that case you end up having to extract and convert each piece of text data with a *=DATE(year, month, day)* and *=TIME(hours, minutes, seconds).* As an example, converting some weather station times tamps that looked like this **01.01.2009 02:00** ended up needing this beast:

=DATE(MID(B2,7,4),MID(B2,4,2), LEFT(B2,2)) + TIME(MID(B2,12,2), RIGHT(B2,2),0)

If you google around you can find good guides explaining how to do that, but it's still a pain the backside to deal with

Addendum 2016-08-03

I was showing a friend how to set the time this RTC recently, when we made the surprising discovery that you can not easily set the system clock on an Apple to UTC. You can select London, England, however England uses daylight savings time and as a result uses GMT (UTC+0) during the winter and British Summer Time (UTC+1) during the summer. (aka selecting London as the city to base my timezone does not provide UTC year round, only in the winter). A bit of hunting revealed that there are other cities in the GMT timezone that do not use daylight savings time such as Ouagadougou, Burkina Faso, and there are other fixes out there if you are willing to go under the hood. But its just hard to believe that Apple made it so hard to set a computer to the global time standard…weird.

Addendum 2016-09-09

With long startup latencies & initialization issues in the SDfat library, I haven't pursued approaches that remove power from my loggers between samples. But I've been reading about what might be the most elegant approach to the complete shutdown method for data logging applications: Using the RTC alarm (which outputs low) to control a P-channel Mosfet (AO3401) on the high side of the main battery supply. When the INT/SQW alarm goes low, it turns the mosfet on and powers everything including the main mcu board which would then goes to work taking samples and storing data. Then the final step after a sample is taken would be to re-program the time for the next RTC alarm, and then write zeros to the alarm flag registers (A1F and/or A2F) which would then release the INT line on the gate of the mosfet. (you would need a pullup resistor on the gate to make sure the pFet turned off properly). Geir Andersen discusses this over at LetsMakeRobots, and I think it's the method that he used on the Deadbug shield. Even more interesting were hints that this approach was used with an ESP8266 to build a mains dimmer switch. Food for though, as I can see how stabilizing the Mosfet control line might be a little bit tricky, and in my case, the main battery voltage is higher than the RTC's 5.5v maximum, so I would have to use a lower voltage battery as the main power supply.

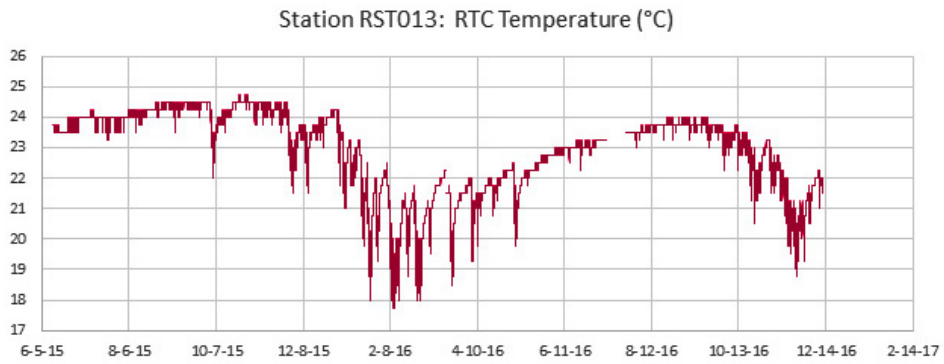
Addendum 2016-12-31

I tweak the code on most of my loggers between deployments so often only the more unusual sensor combinations get run for long periods of time without a clock reset . The last round of fieldwork had me updating several of those old dogs, most of whom had been running for more than a year, and this let me do some RTC drift checks. There were two end members for the set; with one unit losing 1.1 seconds per month, and another gaining 1.4 seconds per month. Most of the other loggers drifted forward by ~ 1/2 second per month. So my two worst-case units have drift rates about twice as large as the 0.026 seconds per day they saw at SwitchDoc Labs, but most of my units are in good agreement with their benchmarks. HeyPete.com is doing detailed testing, and usually sees +/- 0.5ppm (~16 seconds a year of drift) – which is less than the +/- 2ppm spec.

All of these loggers were in a very stable thermal environment (ie. a cave) at around 24°C. Depowering the RTC does not seem to increase the drift, (in fact David Pilling found that timekeeping on these boards actually improves when you disable the lithium coin cell charging circuit) and the coin cells look like they will last well past two years with this level of drain, but it's still uncertain exactly when they will quit due to flat lithium discharge curve.

And while we have plenty of high-precision sensors for temperature data, the RTC registers continue provide a convenient 'inside the housing' record:

(The log shown above is from a very dynamic site with several entrances to provide airflow, but most of the other temp. records hover near the bit toggling point all year.)



While there is a lot of lag in the RTC temperature reading due to the thermal mass of the housing, these logs still provide a good sanity check when my *other* sensors are starting to fail.

Addendum 2017-01-18

Hackaday released a post on the quartz crystal resonators which provide the heartbeat for RTC modules and Kerry Wong demonstrates how to adjust the aging offset register with a HP 5350B in high resolution mode. Adafruit has produced a DS3231 module, if you want something more compact than these cheap eBay units, without the EEprom.

Addendum 2017-02-15

I just noticed that the RTClib from Adafruit supports the use of this RTC with an ESP8266, which will come in handy in future. And there is another library out that makes use of the eeprom on these boards for circular buffer logging. Given the limitations of the ESP, a combination of those two could prove very useful…

Addendum 2017-02-22

After looking at the old logger code I have posted on the projects Github, Mark Behbehani emailed more elegant way to update the next alarm time using modulo, rather a cascade of if statements:

The calling code:

```
pinMode (RTCPOWER_PIN, OUTPUT); // RTC vcc connected to this pin
digitalWrite(RTCPOWER_PIN, HIGH);
delay(15);
DateTime now = RTC.now();
SetNextAlarmTime(now); RTC.turnOnAlarm(1);
delay(5);  //give the RTC a few ms to finish operations
pinMode (RTCPOWER_PIN, INPUT);
digitalWrite(RTCPOWER_PIN, LOW);
```

```
// (Current minutes + Sample time) % 60  will give min for next alarm
// then utilize mask bits to ignore Hours and days, set seconds to 00
// Bit 7 on (AM3, AM4) 0x0C 0x0D to 1 and only min sec match
// i2c_writeRegBits(DS3231_ADDRESS,DS3231_ALARM1_HOUR,1,Bit7_MASK);
// i2c_writeRegBits(DS3231_ADDRESS,DS3231_ALARM1_DAY,1,Bit7_MASK);
// Using the existing libraries you can call
// rtc.getA1Time(byte A1Day, byte A1Hour, byte A1Minute, byte A1Second,
// byte AlarmBits, bool A1Dy, bool A1h12, bool A1PM
// Pull in Day,Hour,Min,Sec
// For sec (or min) interval (Sec+intval)%60 for Hours sest (H+intval)%24
// Using AlarmBits X|A2M4|A2M3|A2M2|A1M4|A1M3|A1M2|A1M1 to set mask to ignore
// Update only variable of interest for secintval Sec, for min interval Min,s=00

void  SetNextAlarmTime(DateTime now) {      // this replaces my cascade code
RTC.getA1Time(Alarmday, Alarmhour, Alarmminute, Alarmsecond, AlarmBits, ADy, Ah12, APM);
if (SampleIntSeconds > 0){   //then our alarm is in (SampleInterval) seconds
Alarmsecond = (now.second() + SampleIntSeconds) %60;
// gives seconds from 0-59 sec e.g. 50s+15 = 65s  65 %60=5s
AlarmBits = 0b00001110;  // set to ignore any match other than seconds
RTC.setA1Time(Alarmday, Alarmhour, Alarmminute, Alarmsecond, AlarmBits, 0, 0, 0);
}
else {     //means seconds is set to zero and use SampleIntervalMinutes
Alarmsecond = 0;  //force matching on even min
Alarmminute = (now.minute()+ SampleIntervalMinutes) % 60; // gives from 0-59
AlarmBits = 0b00001100;  // set to ignore days, hours but match min, sec
RTC.setA1Time(Alarmday, Alarmhour, Alarmminute, Alarmsecond, AlarmBits, 0, 0, 0);
}
```

That’s the first time I’ve seen modulo being used, and I think it’s quite elegant.

Addendum 2017-04-06

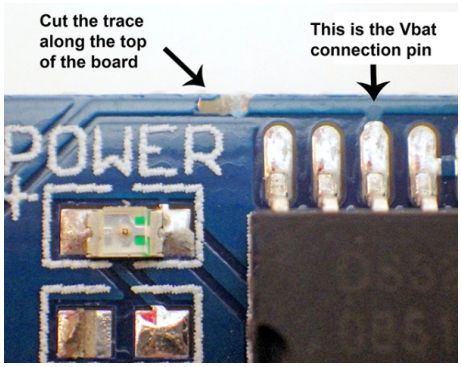
Following on that modulo comment, I came across a post using it to encode dates with only 7 alpha characters, as opposed to the standard 10 digits you would see with the ascii version of a Unixtime date. Of course, if you take samples at whole minute intervals, you can use Unixtime/(sample interval*60) with no data loss. If you take samples every 15 min, then you are dividing the unix time by 900; reaching the same 7 character size without any complicated algorithm.

Addendum 2017-04-11

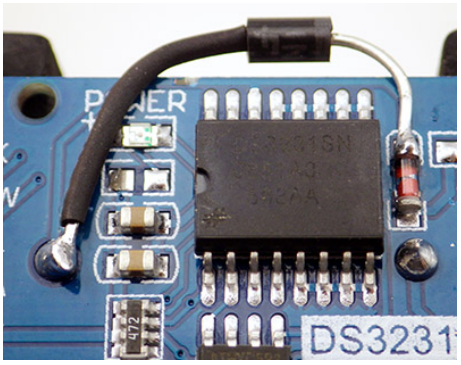
I just noticed that Energisers CR2031 coin cell datasheet lists something interesting: **Max Reverse Charge: 1 µA** With the number of people warning about non-rechargeable cells exploding if you put them in a trickle charge circuit, I've simply been removing the charge circuit resistor. But with their 20-30 ohms of internal series resistance, I am now wondering if the relatively low 3.3v Vcc on my promini's means that the voltage drop on the resistor & 1N4148 diode combination would give me enough wiggle room to keep the coin cell below that rev charge spec, while still supplying the 0.3µA (max) timekeeping current to the RTC from the main AA batteries when that supply is available.

…Thinking about it a bit more, I guess what I am really after is a simple modification that provides a Diode-OR behavior to switch between the coin cell & the 3.3v rail on the chip's Vbat line. If I cut the trace from the positive terminal of the coin cell and jumper a 1n5817 across to the common pad on the existing charger circuit, I think we would have the best of both worlds. There would be some drop across the 200Ω resistor & 4148 diode, so the 3.3v rail would deliver less than that to Vbat, and this would drag the coincell/shottky combination down, but once they equalize that the drain on the coincell should go to zero. Perhaps, I should add a little cap in there to smooth the voltage transitions?

Addendum 2017-04-15



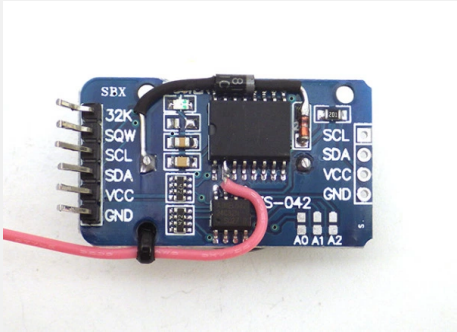
I tested the 1n5817 Diode-OR idea: with 3.289v on the Vcc line from the Promini's board regulator, the DVM sees a voltage of only 3.028 on the Vbat line, so the drop across the diode/resistor pair was 0.261v, which is pretty low for a 4148 because of the extremely small current flowing through it. My primary concern was that leakage through the Shottky would exceed the reverse current spec on the coin cell. So I put a very dead CR2032 in the RTC module (which read at 2.8v unloaded, so around 75% discharged) and that showed a steady 0.69µA of reverse leakage going backwards through the 1n5817 into the coin cell when the charger circuit side was powered. When I disconnected the main logger's voltage regulator, the current through that diode changed direction as it was supposed to, and increased to 0.84µA, which is less than the typical timekeeping current for these RTC's, so the coin cell can't be loosing much power to the main Vcc line by *backwards leakage* through the charger circuit. You could also clearly see the periodic current spike from the temp register updates when they occurred. After several power/depower cycles like this the RTC did not loose its internal time even with this crummy backup battery. Then I switched to a slightly less dead coin (at 3.08v unloaded which is still low) and the reverse leakage fell down to only 0.19µA. So a really low voltage coin cell will see some power flowing back into it, but both were below that 1µA reverse current spec.



Switching to a brand new coin cell (read unloaded at 3.265v) and there is no reverse leakage when the loggers Vcc is powering the charge circuit, but a small forward current from the battery to the main pad of 0.01µA. The coin cell is now applying a higher voltage to the common pad than the 3.028 it would receive through the 200Ω resistor/1N4148 diode combination. So I think that a new coin cell will eventually be pulled down to match the charger pad voltage, but since the normal discharge plateau for Cr2032's is at

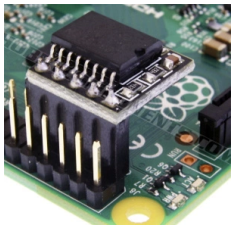
2.9v, and the Vcc supplied pad stays just above 3.0v, the coin-cell should never really have the opportunity to discharge if the logger is powered by the main AA battery.

WooT!



Note that this whole idea assumes that you are providing pin-power to the RTC during I2C communications!

Addendum 2017-08-01



I've been noticing more DS3231 breakout boards on the market as this chip is also a go-to chip for the raspberry pi , but for some reason many do not have the alarm line broken out. This is a mystery to me as I don't understand why you would leave that functionality out of a design?

This entry was posted in Developing other sensors and tagged AT24C32, DS3231. Bookmark the permalink.

45 Responses to *Using a $1 DS3231 RTC & AT24C32 EEprom from eBay*

Alex *says:*
November 16, 2014 at 5:43 am

hi, thanks for the article. A few questions for you. When the DS3231 module is in timekeeping mode , does INT line still work (assuming it is pulled high elsewhere and not Vcc as described in the spec sheet)
We are trying to see if it is possible to use the interrupt line to turn on microcontroller (arduino) while ds3231 is in timekeeping mode.

thanks

Reply

edmallon *says:*
November 16, 2014 at 7:18 pm

Yes. I still use the interrupt from timekeeping mode to wake my sleeping Arduino which then drives a pin high to apply power to the DS3231's VCC input only when the MCU is awake. Before the logger sleeps again, I pull that power pin low to force the RTC back into timekeeping mode. The caveat here is that I have not yet done any longer run tests to see if generating those interrupts chews through the little coin cell, or causes some other system weirdness. I will post more results on that as they come in.

Reply

Alex *says:*
November 18, 2014 at 6:13 am

great, thanks. Have you seen this other design where a p mosfet is used in conjunction to the low interrupt line to power on/off arduino : http://electronics.stackexchange.com/questions/73250/turn-on-arduino-with-rtc-alarm (the first answer, he uses an ds1305, but same functionality)

Reply

edmallon *says:*
November 20, 2014 at 12:39 am

I think a few people have taken that approach:
https://www.tindie.com/products/Dead_Bug_Prototypes/extreme-low-power-data-logging-shield-for-arduino/

But my loggers have "always on" sensors which are also providing interrupts, in addition to the RTC's alarm, and I need to do processing depending on "how" the unit is being woken up. So I have gone the route of using a Rocket Scream Ultra board with a low quiescent current MCP1700 voltage regulator. With an SD card, and an ADXL345 running, pin powering the RTC is getting me down to around 0.18 mA, which gives me a decent run time on AA batteries. Fatlib16 brought his Pro Mini down to 0.09 mA (with a sleeping SD card) by externally powering from the same regulator:
http://forum.arduino.cc/index.php?topic=269004.msg1940524#msg1940524

And the SD card is responsible for at least 0.06mA of that. So the question is: if an off the shelf Arduino can sleep around 0.03mA do you still need to cut the power?

Reply

Alex *says:*
November 24, 2014 at 5:33 am

Great thanks. I got my own DS3231 module from eBay. I measured the current draw with the interrupt line low . Result came in at about 60 uA with interrupt on on battery.

thanks

Reply

edmallon *says:*
November 24, 2014 at 8:03 pm

Hmmm, so we *are* putting a drain on the coin cell that is significantly greater than the 3uA timekeeping current by making it generate those interrupts without power on Vcc. But assuming a CR2032 has about 200 mAh available, we are probably still within the safe zone for a couple of years of operation as long as there is not too much delay responding to that interrupt. What value pull-up resistor did you have on the interrupt line?

I have also been trying to find out if I2C communications with other devices on the bus would draw more power from the RTC battery. (due to a slow "recovery" time in the DS3231 chip?) The datasheet seems to indicate that the switch-over to Vcc power is virtually instantaneous, but I did find this:

http://www.forward.com.au/pfod/ArduinoProgramming/I2C_ClearBus/index.html

"The I2C interface is accessible whenever either VCC or VBAT is at a valid level. If a micro-controller connected to the DS3231 resets because of a loss of VCC or other event, it is possible that the micro-controller and DS3231 I2C communications could become unsynchronized, e.g., the micro-controller resets while reading data from the DS3231. When the micro-controller resets, the DS3231 I 2 C interface may be placed into a known state by toggling SCL until SDA is observed to be at a high level. At that point the micro-controller should pull SDA low while SCL is high, generating a START condition."

I still have to think about it for a while before I understand if this has implications for my loggers, but I have not seen any weirdness in the tests so far. As I have wakeup events where I need to communicate with other I2C devices, but *not* the RTC, and have been wondering if it is safe to leave the clock de-powered during those events. Guess I will just have to try it and see.

Reply

Antoan *says:*
December 1, 2014 at 7:35 am

Really interesting stuff 🙂 I want use this board for my project too. Did you compare these cheap DS3231 to a "real" one? You wrote that you will but I did't find it in article. (Addendum: 2014-05-21)

Reply

edmallon *says:*
December 1, 2014 at 5:41 pm

A drift test against a "real" DS3234 is still far down on my project to-do list. And unless these cheap RTC's are quite bad, the drift test itself will take months to perform. I would probably need to set up 4 or 5 of them to get reliable data, so think that I will have to do this test with a whole "generation" of loggers that actually go into the field for a few months. I would love to hear about the results if someone else tries this experiment before I get around to it. I know that there are people out there have found much faster ways to verify clock accuracy.

(I should add that my field data to date doesn't bail me out on this one because I had an routine in my logger scripts that automatically updated the RTC's (to by comparing the clock to the compile time). Great for making sure the RTC's are up to date, but bad if what you are trying to do is observe drift, because it wipes out any accumulated errors every time you reload the scripts. I have removed this auto-update from setup for the latest generation of loggers…so my field data should give me at least a rough sense of the drift mid-late next year.)

Reply

nonokunono *says:*
December 28, 2014 at 6:06 am

hi, I can confirm what you said: "I have noticed that when I power this module from Vcc at 3.3v, it draws around 89 µA" . I am also powering the module using an Arduino output pin. But I've found if the Vcc to the ds3231 module is off, external interrupt on the SQW/INT pin no longer works. Can you verify this ?
I have set the appropriate control registers to enable interrupt, alarm 1, 2 and the interrupt bit.
Everything works very well except for if the Vcc is turned off the interrupt stops working.
thanks

Reply

edmallon *says:*
December 29, 2014 at 9:44 pm

My units continue to send an SQW/INT interrupt even when the RTC is powered by the little backup battery. That I how I wake up the Arduino, which then drives the pin attached to Vcc high to power the RTC before any I2C communications. In fact one of my main concerns is that powering these interrupt alarms might drain the coin cell dry before I reach the end of a deployment.

If your units are not sending the alarms, I would hunt around in the data sheet to see if there is some other register setting that needs to be changed. Also, test more than one unit. About 10-15% of the cheap breakout boards I get from eBay are duds (more for complex sensors). You might also try replacing the coin cell battery.

Reply

Jim Remington *says:*
January 25, 2015 at 2:51 am

When power to the module is off, SQW/INT still works. However, the output is open collector and for you to "see" the signal, you will need a pullup from the SQW/INT to Vcc of the input device. For an Arduino, simply turn on the internal pullups. Let us know if that was the problem.

Very useful blog article!

Reply

edmallon *says:*
January 25, 2015 at 5:01 pm

If you do not remove the I2C resistor block from the breakout, you have a 4.7k pullup on the SQW line that is soldered on rtc board. However I had forgotten that I was also applying the arduino internal pullups in the code, because my early loggers used a tiny duino stack and I had lifted the lines from their I2C light sensor board. This required me to remove that resistor block or have too much I2C pullup (thus also removing the SQW pullup resistor from the rtc, because its in the same block). So I had belt&suspenders there, because I had forgotten about the cruft in my old scripts. Does not seem to be hurting the functionality as those alarms are still working. But does it hurt to have a physical pullup, and an internal Arduino pullup on SQW at the same time?

Reply

Jim Remington *says:*
January 26, 2015 at 12:45 am

The pullup in the resistor block is connected to Vcc of the DS3231 module, so if the module power is off, it is not effective. The SQW/INT is in that case still open drain. So, you must have a pullup to the Arduino input if you want to read the SQW/INT line when the module power is off.

The Arduino pullups are very weak (50-100K) and are very unlikely to cause a problem if paralleled with other pullups.

Reply

Jim Remington *says:*
February 3, 2015 at 9:34 pm

UPDATE and correction! I did some experimentation with the eBay DS3231 modules regarding pullups on the SQW output in the "battery backed square wave mode", and the results are really surprising. You need a low value pullup to 5V (3.9K or so) in order to see the square wave output on SQW. It does NOT work with 50K pullups — you do see pulses but they are about 1V and thus below logic level. Could the output transistors be leaky? I'll try this with some genuine DS3231 modules from JeeLabs at the next opportunity (they are both logging data at the moment).

Reply

Jim Remington *says:*
February 3, 2015 at 10:42 pm

Problem solved! I forgot about the 4.7K pullup resistor pack on the module, from Vcc to SQW, etc. That was loading down the external pullup resistor.

Reply

edmallon *says:*
February 4, 2015 at 11:40 pm

Because I have other I2c sensors that must remain powered hanging off of the RTC breakouts cascade port, I am pin powering the RTC's Vcc line directly on the leg of the chip itself, which I lift off of the breakout board. This gets me away from a conflict with the 4.7k pullup on the board.

It sounds like you were powering the VCC line on the breakout, which is connected via a 4.7k pullup resistor to SQW, and ALSO applying the internal pullup resistors on the interrupt pin, which is also connected to SQW. So in effect you had: internal pulldown – PowerPin – 4.7k – SQW – InteruptPin – internal pullup. When you pulled the power pin low to de-power the RTC, the two pins were fighting each other leaving SQW at an intermediate logic level.

I think the key to making it work with that configuration is to put the power pin into a high impedance state before you drive it low to force the RTC to draw power from the coin cell:
pinMode (RTCPOWER_PIN, INPUT);
digitalWrite(RTCPOWER_PIN, LOW);

Input mode is equivalent to a series resistor of 100 Megohms in front of the power pin, so this might (?) give your interrupt pin a chance to win the battle, and successfully pull SQW high so that the RTC alarm function could work even if you used the Vcc line on the board, rather than the chip leg as I have.

When you re-powered the pin, you would have to then use two steps:
digitalWrite(RTCPOWER_PIN, HIGH);
pinMode (RTCPOWER_PIN, OUTPUT);

At which point both pins are pulling SQW to Vcc, so they don't conflict with each other there.
(One other thing to mention is that I am using a 3.3v regulated board, rather than a 5v arduino)

Reply

Peter *says:*
May 21, 2015 at 2:31 am

Hi Edward,

Very interesting project you have made. Thanks for all information. I've a question about the RTC module with eeprom. I've also powered the module from the MCU pin but the alarm does not get fired when battery powered on the SQW pin. Bit 6 (Battery-Backed Square-Wave Enable) of control register 0Eh is set to 1 to force the wakeup alarms to occur when running the RTC from the back up battery alone. The SQW pin is pulled high. I've tried internal MCU pull-up and a external pull resistor of 2KOhms but it none of them fires. When the RTC module is powered on VCC it does fire the SQW interrupt low. Any idea what the problem could be?

Thanks,
Peter

Reply

edmallon *says:*
May 21, 2015 at 11:17 am

The first thing I would do is read back the value from that register and send that readout to the serial window to make sure the BBSW bit actually got set. I often run into situations where I think I have a register configured, but find out that the communications did not go as I expected. If the bit is actually set then next I would try another RTC board. These things only cost a couple of bucks on eBay, and I reject about 15% of them as defective in one way or another – generally from bad solder joints. I have a few breadboard loggers with "known good" Chronodot RTCs, and this helps me triage cheap eBay duds, because I can be confident that my code is working. In fact I now use this strategy for almost all the stuff I get from eBay. 2K is a pretty strong pullup, so I am wondering why you had to add that when there is already a pull-up resistor on the boards alarm line (its part of the 4.7K – 4 resistor block that also pulls up the I2C lines)

Reply

Peter *says:*
May 21, 2015 at 2:09 pm

I did add the pull up resistor because when the VCC power pin goes down there's no pull up voltage anymore. Maybe I'm wrong, but it seems the 4.7K resistor block is connected to the VCC and isn't powered during battery powered operation. I also verify the BBSQW bit. The BBSQW, INTCN and A1IE are set correctly of 0xE register.

It's a good point to try another RTC modules, the cheap Ebay hardware isn't that good.

Reply

edmallon *says:*
May 21, 2015 at 7:28 pm

Yes, you are quite right: if you de-power the VCC line on the whole RTC board you loose your SQW pull-up, although I did manage to get the Tinyduino loggers to receive the alarm OK by simply by enabling the internal Arduino pullup on the interrupt pin I had soldered to SQW (after I had physically removed the four resistor block from the board). But those loggers never de-powered the Vcc line, so the alarms were not being generated while the RTC was running on a battery.
On my current loggers I have I2C sensors connected to the cascade port of the RTC that need to be powered all the time, so pulling Vcc down on the whole breakout board was not an option. That's why I now solder a jumper directly onto the Vcc leg of the RTC chip – which I carefully lift away from the pad on the breakout board. (see the photos in the post) This lets me leave power on the *board's* Vcc line, keeping the I2C and SQW pullup working. I think that is a more reliable approach with these RTCs. The soldering is a bit tricky though.

Reply

scrungy_doolittle *says:*
May 23, 2015 at 10:07 pm

So I have a real quandary. I am using this module. AS it comes from ebay. I can set the clock, and read the clock. I have an LCD display running, as well as monitoring it with the serial port.
I have installed CR2032 batteries in these. When I disconnect the arduino from the computer of course it kills the

power to the rtc. When I plug it back in, the code runs, and the time it reports was the programmed time when I programmed it.
So say I programmed it Thursday. Saturday night I unplug the arduino from the PC and plug it back in. It comes up with thursday as the current date, and the original time. That is, when the power is removed from the arduino, killing the 5v supply to the clock board, the clock stops timing, though it DOES remember the original time. How do I fix this, so that I can remove the clock, and reattach it later, or have the power be turned off on the arduino, and still have the clock in counting mode.
Is this a flaw in the 3231, or a screwup with some bit that is not being set correctly?

The whole point of this is that if power is unterrupted to the device I want to control, I will still have an accurate clock when the power comes back on.
This board is un-modified. I also have one that I have removed the power led from.

Reply

edmallon *says:*
May 24, 2015 at 5:15 pm

This is just a guess, but did you leave the Arduino programmed with the clock setting sketch? If you did it will automatically reset the RTC to the compile time for that sketch every single time you power up your Arduino. In fact it will do it again as soon as you open the serial window as well. I had to separate the clock setting sketch that I found into *two separate programs* because of the "serial window relaunch" problem on the Arduino platform, which would reset the clock just because I opened a serial window to check if the time was actually updated. Now my clock setting sketch only does two things 1) set the RTC and 2) blink an LED – it has no serial output capability. After I see the led blinking, I then load the RTC *reading* sketch which has no ability to set the clock; it just reads and sends the time to the serial window. This also forces me to removed the RTC setting sketch from the Arduinos memory so that I don't forget that I left there and accidentally reset the clock the next time I apply power to the Arduino.

Reply

John Wells *says:*
December 1, 2015 at 12:36 pm

Thanks for the extremely! informative info. Super good point about de-powering i2c devices when not needed.

Reply

edmallon *says:*
December 12, 2015 at 9:51 am

Keep in mind that the DS3231 RTC is a special case, as it is designed to gracefully switch over to the backup battery when it is depowered. Most other I2C devices would require a full initialization of they were depowered, and then powered up again, which can waste a great deal of power/processor time. It is usually better to put those devices into sleep modes (by toggling a register bit), or even better search for sensors that automatically go into uA sleep modes as soon as the bus traffic stops. For example the MS5803 pressure sensors draw almost nothing when the loggers are sleeping, and I don't have to do anything to them for that behavior.

Reply

Tom *says:*
January 10, 2016 at 5:15 pm

Interesting project. I found your page when I was looking for Information on this particular module. The module I received is exactly like the one you have pictured but it came with a non-rechargeable Lithium battery (CR2032) so I was glad to see the simple mod needed to remove the trickle charge. I wouldn't have even known that it was supposed to be a rechargeable battery.

Reply

edmallon *says:*
January 10, 2016 at 5:50 pm

Yes, that's happened to me a couple of times too. There is a reason the phrase *Caveat Emptor* shows up so often on eBay's policy docs, but generally I attribute errors like that to simple ignorance on the part of the sellers, rather than anything deliberate.

Reply

Pingback: *Using a $2 DS3231 RTC & AT24C32 EEprom from eBay - Electronics-Lab*

G8UJS *says:*
February 8, 2016 at 9:27 pm

Great project, I used the DS3231 with a 16F877A project written in MPLAB ASM, I produced some code to set the timer chips using a 60Khz time signal so they were correct to within a second. After about three months just using the attached battery the clocks were tested and all were within 5 seconds some within 2seconds. In the UK there was not a vast difference in temp. The DS3231's time was monitored on an lcd and the variables of time, date etc either inc or dec so the time set very accurately. One additional benefit was to install a small relay switching the serial data lines over to another blank chip allowing an easy cloning of the chips from a master.

Reply

dylanh5 *says:*
February 9, 2016 at 11:13 pm

Wow great information on DS3231's and a great project! Personally I use the DS3232RTC library (works with DS3231) as you can set the RTC in the serial monitor instead of relying on the compile time, after reading your 'making of story' I thought I'd post about it here:
https://github.com/JChristensen/DS3232RTC
You've even inspired me to start my own blog once I get a bit of free time!

Reply

edmallon *says:*
February 10, 2016 at 12:29 am

I will have to look into that SetSerial utility, as I usually end up with a 9-10 second offset from my current compile time method. I am sure I could get that down to 1-2 seconds with his method. Usually, I see these RTCs slowly creep forward over a deployment (so they seem to run a bit fast), so a more accurate set point would let me get a better handle on that. I have also heard that some people see different amounts of creep depending on the voltage they feed the board (which takes up to 5 volts) while mine are pin powered at the 3.3v coming from the regulators. And David Pilling saw some changes to the 1 Hz output after changes to the battery voltages.

Reply

Tomas *says:*
February 27, 2016 at 3:05 pm

I am doing just another temperature logger for fun. I use chinese ZS-042 RTC module and arduino nano, later mini. Rather than download sketch from internet, I code it on my own, as opportunity to dive little bit more in arduino coding/developing. While I was searching some information about RTC, I was several times pointed to this article, so I guess, it must be one of your entry point to your blog :). Did not look much around at first, while coding something, later during next visits, I realized, that your blog is full of very valuable information, ideas and stories. I think, it will take me while to read them backward 🙂 Thanks for sharing!

Reply

edmallon *says:*
February 28, 2016 at 10:41 am

Glad to help. The wide variety of manufacturers making these cheap boards means that there is quite a bit of variation in soldering quality and in the other low level details. I like the boards with the ZS-042 markings because the holes for the cascade port have larger contact pads than those on some of the board variants where those contacts barely even exist. That's important if you are hanging other I2C sensors off of that port like I am. It's a bit scary to think about how much time I have invested in these things if I am starting to get picky about the vias…

Reply

Sunny *says:*
February 28, 2016 at 7:28 am

Hi Edward. Great article on the rtc s'. Have you got the PCF8563 up and running? I'm looking for a way to wakeup the arduino from its slumber using the PCF8563. Some help would be appreciated.

Reply

edmallon *says:*
February 28, 2016 at 10:46 am

I considered the PCF8563 at the start of the project, but then I figured out how to pin-power the DS3231 during cpu up-time, and the low power advantage of the 8563 disappeared. So I am afraid I can't help other than to say using the alarm to drive an interrupt which wakes the Arduino should work the same as with the DS3231. For a bare bones version of the code that does this, you can look at the script I posted for the UNO based data logger.

==========

Hi Ed! Good to see the recorders are still going strong.

I wan't able to post a reply on the $2 DS3231 page post, so I'm writing an issue I faced with your barebones rtc code. Please cut-paste this reply on your rtc page. Everything works just as promised with the ds3231 and I've been running my logger for the past few days now. Except for one issue in code- everytime the minute rollover occurs, the interrupt signal gets triggered, although its not yet wake-up time. This is kinda annoying because I switch ON the logger at 11.56am with SampleMinutes of 30. But, the darn thing causes an interrupt 12am. Did you face this issue? How did you go about solving it….
Sincerely,
Sunny.

———– reply ———–

Hi Sunny,

I am assuming you are using the basic data logger code posted at:

http://github.com/EKMallon/UNO-Breadboard-Datalogger/blob/master/_20160110_UnoBasedDataLogger_v1/_20160110_UnoBasedDataLogger_v1.ino

The only place in the code where the alarm interval is set is in the define statement right at the beginning:

#define SampleIntervalMinutes 1

Changing that from the default 1 to some other number should reprogram the alarm time. For that simple version, you only have the option of 1-60 minutes, though the library supports seconds as well. If you change that define, and the alarm time is not changing, then there could be problem with the line that actually sets the alarm at:

RTC.setAlarm1Simple(Alarmhour, Alarmminute);RTC.turnOnAlarm(1);

What I would do is run the code with the logger tethered via a USB cable, to check the serial text output to make sure it agrees with what you think should be happening. Very occasionally you will run into conflicts where a library assumes one I2C address for your RTC (usually 0x68), and the particular board has the chip set to another bus address. Running an I2C bus scanner utility will help you out there – I like the ones by Rob Tillart, which you can find at the arduino playground.

The trigger at the 24 hour rollover is probably just a bug in my code. There are so many things that I check ‘once per day’ at the midnight rollover, (like main battery voltage, RTC coincell voltage, etc) that I have never tried to turn that alarm off…

ALSO: note that there are TWO programmable alarms on the DS3231, so you might need to add some code in the setup to explicity turn off that second alarm. Generally these are off by default, but again your RTC might have it turned on by accident….

Reply

Tom G *says:*

February 20, 2017 at 2:40 am

This lengthy & thorough blog was very helpful as I experimented with the DS3231 module. I wrote an application to set the rtc by sampling the computer’s clock and it waits for the minutes to rollover, and then it copy’s the computer time into the module at that instant.
What I didn’t realize is that my computers clock is about 10 seconds too fast every day.
I kept adjusting the aging offset register in the rtc, attempting to make it as accurate as possible.

Soon I realized that in order to validate the rtc’s accuracy, I have to update the computer’s clock from the time.nist.gov website just before checking the rtc’s time. And it’s also important to chose the same nist address every time the check is done. There are other addresses like time.windows.com that are different by seconds.
So far I’ve been able to trim it so that it’s accurate to within one second every three days.

Reply

edmallon *says:*

February 20, 2017 at 2:22 pm

That pretty much reflects the challenges I see with accuracy for any sensor: past a certain point, I rapidly find myself needing to upgrade all the other parts of the system, before I can improve the actual sensor itself. WRT time, we see the same issues with the professional data loggers as well, so we don't worry too much about ± a few seconds on the initial setting. I've done quite a few drift tests, and we rarely see more than about 30 seconds in a year, and most come in about 1/2 that. Since you've taken the setting accuracy to the next level, I'd love to hear what kind of drift you see on that RTC in 6 months, given that the offset registers are adjusted properly.

Reply

Dan Drown *says:*
February 27, 2017 at 10:56 pm

You have a very interesting project, thank you for sharing it. You might be interested in my test of the DS3231 RTC. I set it up to output 1Hz on the squarewave pin and compared it to a GPS module's 1Hz. I adjusted the DS3231's frequency by -0.1ppm and let it run for a week. After a week, it was just under 10ms off (0.016ppm). Without the frequency adjustment, it would have been around 70ms off. That's roughly half a second every year for the adjusted and 1 second every three months for the unadjusted. There's more info and some graphs on my website link.

I also measured a PCF2129, but the DS3231 was better.

I hope this is useful information

Reply

edmallon *says:*
February 28, 2017 at 9:51 am

I'm happy to see more confirmation that these cheap DS3231 boards are good, though you never know how much the quality varies from one vendor to the next. I passed the link on to a few other academics, who I know would be interested in your GPS testing.

Reply

Pete Stephenson *says:*
July 28, 2017 at 8:04 pm

I've taken one of the DS3231 modules I purchased from eBay apart and taken several die shots. Maxim says the markings on the package and die correspond with a legitimate lot made in 2011, so that's encouraging. I have no idea where the Chinese eBay vendors got them, as I don't suspect they've had them sitting around on the shelf for years gathering dust. Maybe they've been taken off of older systems as part of the scrapping process, but there's no evidence of having been desoldered on any of the chips I have. Weird.

Anyway, I have several photos at https://blog.heypete.com/2017/07/29/a-look-inside-the-ds3231-real-time-clock/ and will have some updated photos in the next few weeks once I'm able to get the camera on the lab microscope to play nice with my computer. I thought you might be interested.

Reply

edmallon *says:*
July 30, 2017 at 5:04 pm

That's some impressive work! We just returned from another big fieldwork trip and I finally had my first serious failure on one of these boards after putting more than 150 of them into service on various logger builds. It ran fine for about 6 months, and then simply stopped advancing. All other functions on the chip are fine as far as I can tell wrt coms, temp register, etc, but the clock does not advance or generate any output on the 32K line. As I've never used any of the "official" maxim chips, I don't know if that's a good or a bad failure rate…

Reply

Pete Stephenson *says:*
July 31, 2017 at 5:37 am

Thanks! It was pretty straightforward, with stuff you'd have around the house (excepting, perhaps, a really nice Zeiss microscope that I have access to at work).

As for the oscillator stopping, that's very strange. What is the status of the EOSC bit (bit 7 in the 0x0E register)? The oscillator is halted if it is 1 and the chip is running from Vbat, but the oscillator should start automatically whenever the chip is running from external power. If the bit is set to 1, what happens if you set it to 0?

You're the only individual I know who's used so many of these boards, and so you're the reference when it comes to longevity of the chips. If you've only had one failure in all this time, that sounds pretty good to me.

Reply

edmallon *says:*
July 31, 2017 at 11:06 am

I'm sad to report that I did a field repair on the bad RTC, as it was faster to just solder another one into place rather than do further testing. So I think it got binned. Sensor self-triggering loops, & the resulting AA low battery leaks are still the #1 reason for loggers to go down. Compared to those problems these RTCs are solid as a rock.

Reply

Pingback: *A look inside the DS3231 real-time clock – HeyPete.com Blog*

Stergios Zgouletas *says:*
July 31, 2017 at 7:10 am

Thanks for your detailed work. But I'm confused
https://edwardmallon.files.wordpress.com/2014/05/rtc-mods.jpg?w=300&h=203

1) Need to remove the charging system or need only to solder the Schottky ?
2) Removing the LED instead of resistor is the same? (1)
3) Removing the 4 resistor block, does it solved the POWER_PIN way? (known issue, if is powered from batteries the arduino never wakes up)

Reply

edmallon *says:*
July 31, 2017 at 10:59 am

The whole point of the Shottky approach is that the charging circuit is still in place and **this can only be done on 3.3v systems**. You need to make sure you cut the trace from the top of the coin cell to the charging circuit so that the coin cell is protected from the charger circuit voltage by that shottky. And you must also provide pin-power the RTC during I2C communications. HERE is a photo of the complete conversion. Keep in mind that I have not fully tested this idea yet, so you do this at your own risk. If you are going are pin powering the leg of the RTC, then leave the 4 resistor block in place to provide I2C & alarm pullup.

These modifications to the RTC board are ADD-ons that are not required at all for the data loggers to work. I strongly suggest building a few loggers without them, to make sure all your other details are sorted, before tackling these changes.

Reply

Arduino based underwater sensors