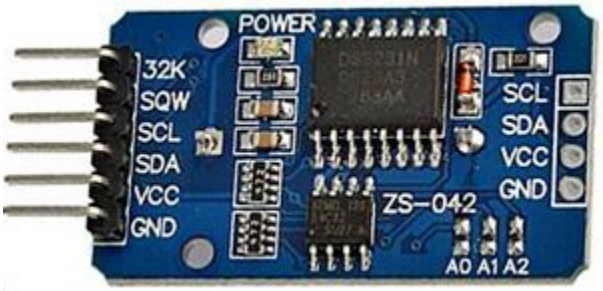
Using a $1 DS3231 Real-time Clock Module with Arduino

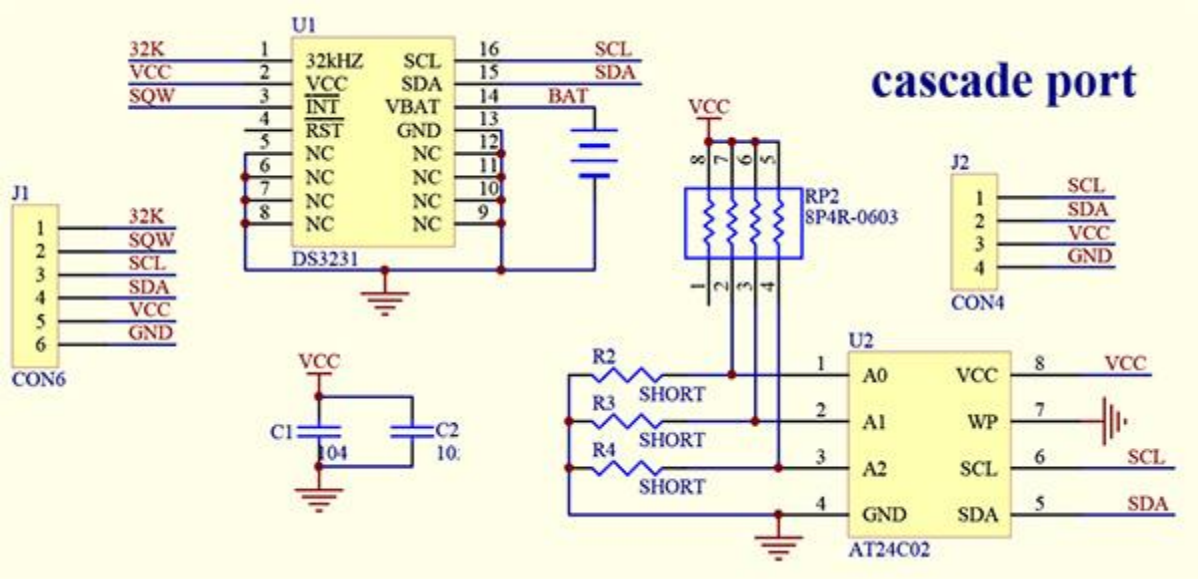


A ±2ppm DS3231N for less than $1? Really? And most of these boards use the industrial SN variant, which is rated for the full -40°C to +85°C temp range. ***NOTE:*** *There is now an M variant of this chip on the market which is more resistant to vibration, but the -M is only temperature compensated to ±5ppm.*

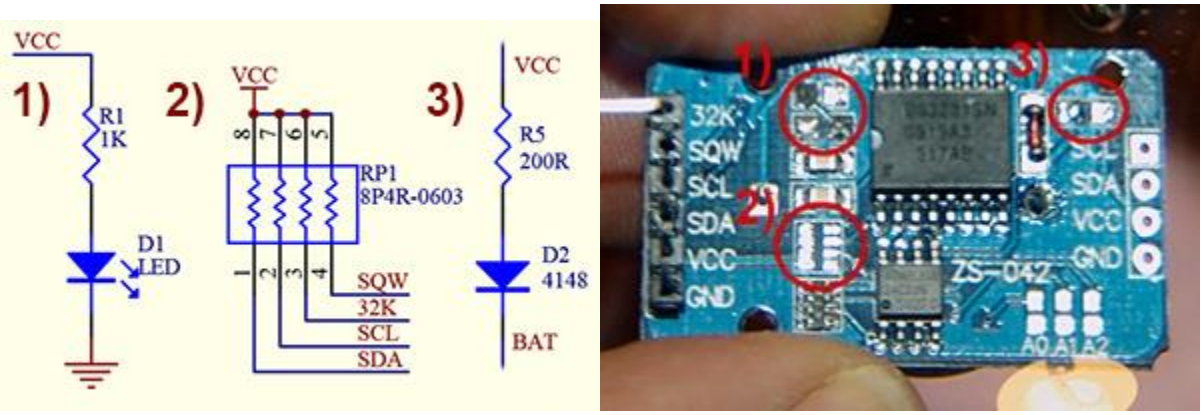
Since **the Cave Pearl is a data logger**, it spends most of the time sleeping to conserve power. So you could say that the most important sensor on the unit is the real-time clock (RTC), who's alarm signal wakes the sleeping processor and begins the cascade of sensor readings. I built the first few 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 this DS3231 module 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 on that circuit diagram that could be removed:



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:

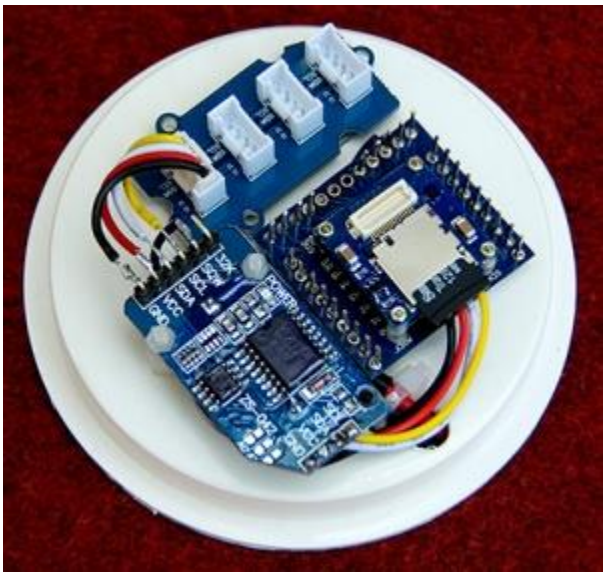
```
digitalWrite(INTERRUPT_PIN, HIGH);  //pull up the interrupt pin
```

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 Pin Powering the RTC

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.

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. Also keep in mind that this only works because the RTC was designed with backup power circuitry. In general, de-powering I2C Slaves is not a good idea because the pullup resistors keep the SDA and SCL lines high. When a regular I2C sensor has no power, you could leak current via SDA and SCL through the I2C device to GND.

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.