DS3231 Aging Calibration Using GPS PPS

This Arduino sketch sets the Aging register of the DS3231 RTC to the optimum value based on the PPS (pulse per second) output of an inexpensive GPS module. Once set to the optimum Aging value, the RTC should provide very accurate timekeeping - to within a few seconds per year if the ambient temperature and power supply voltage remain stable. The performance of the DS3231's built-in temperature compensation has not been tested, so it is not known whether the optimum Aging setting varies with temperature. The sketch does not require any additional libraries for either the RTC or the GPS.

The default power-up value of the Aging register of both the DS3231SN and the DS3231M appears to be zero. But that may not be the optimum setting, particularly for the M. Permitted values range from -128 to +127, with lower settings producing a faster clock. The optimum value of my single SN example is +5, but my two Ms (one of which is marked as SN) have optimums of -23 and -45.

Method

The RTC can be set to produce a 1Hz square wave on the SQW pin which goes low at the beginning of each second, and back high a half-second later. Many GPS modules include a PPS output which goes high at the beginning of each second. We first select the SQW rising or falling edge that is closest to one-half second off from the PPS rising edge, and then count the number of processor cycles which take place between the SQW and PPS edges. For an Arduino Nano running at 16MHz, the selected SQW edge produces a difference of between 4 and 12 million cycles. If the RTC is running at exactly the same speed as the GPS, that difference will remain constant. If the difference is not constant, we adjust Aging so it will be. In the end, we end up with one setting that is a little fast, with the next value being a little slow, and we pick the one that is closest to perfect.

It does not matter if the Nano's crystal is exactly 16MHz because we only look at whether the time difference is changing. So the crystal doesn't have to be accurate, it just has to be consistent.

Impact of Power Supply Voltage

The voltage of the RTC's power supply affects the speed of its oscillator, and thus the optimum Aging setting. For example, my DS3231SN produces an optimum setting of +5 when powered from the Nano's 5V pin, which is actually about 4.7V following the schottky diode drop. But the optimum is +8 when it is powered from an external 5.1V supply. And when running on its 3V coin cell, the optimum is -2. So the oscillator runs faster at higher voltages. However, my two Ms seem to run slower at higher voltages. This means that in calibrating the optimum Aging setting, you should duplicate as nearly as possible the power supply voltage the RTC will experience in its application. Powering the RTC from a regulated supply, rather than directly from a battery, may improve performance because battery voltage will drop as it discharges over time.

Hardware

The genuine DS3231 comes in two versions. The DS3231SN is the higher-end part with the most accurate timekeeping. It has a built-in crystal oscillator. The DS3231M is the budget version which has a MEMS resonator that isn't as consistent. However, the M can still deliver good timekeeping if the Aging setting is optimized. I suspect that both versions are widely counterfeited.

GPS modules have beome very inexpensive. Many have four pins - power, ground, data in and out. But some have a fifth pin for the pulse-per-second output. I used the GoouuuTech GT-U7 module costing $12.99 at Amazon. Presumably, any module with a PPS output would work.

https://www.amazon.com/gp/product/B07P8YMVNT/

I used an Arduino Nano clone, but the sketch should work on any Arduino with an ATmega328P processor, which would include the Uno and the Pro Mini. The Nano is powered via USB from the computer. Connections to the GPS module are power (5V), ground, and PPS (D8). Connections to the RTC module are VCC, ground, SDA (A4), SCL (A5), and SQW (D2). VCC can be connected to an external source, or to the Arduino's 5V pin.

A third option is to connect the module's VCC pin to D4. The sketch turns on D4 only during I2C activity to power the pullup resistors on SDA and SCL. At all other times the RTC will be running on its backup 3V coin cell. To use this option, you must disconnect any on-module pullup resistor on the SQW pin. You should use this option if you intend to power the RTC only on its coin cell in the field.

Algorithm

In setup the code first determines whether the RTC is an SN or an M, since the two are treated differently, then determines which SQW edge is best.

The selected edge of SQW triggers an interrupt on D2, at which point the ISR starts the 16-bit Timer1 counting from zero. Timer1 is set up with no prescalers or divisors, so it is running at the CPU clock rate. There is an additional byte that is incremented when Timer1 rolls over since the timer can only count to 65,535. So with the extra byte you have in effect a 24-bit timer.

The PPS signal at D8 creates a "capture" event for Timer1. The timer count at that point is captured, and Timer1 is stopped and cleared. We now have the number of cycles which have occurred between the SQW and PPS edges. To help deal with jitter in the system, this process is carried out for 16 consecutive seconds, and the average calculated. The result is stored in the variable "Diff".

After doing an intial baseline calculation, the process is repeated every five minutes. The change in the Diff value ("deltaDiff") from five minutes ago is then used to calculate what change, if any, ("deltaAging") should be made to the Aging value, and any resulting new Aging setting is written to the RTC. These variables are sent to the serial monitor. If the deltaDiff value is shown in [brackets], it just means that it wasn't big enough to cause a change to Aging - so it represents a cumulative deltaDiff since the last Aging change.

You can let this process run for as long as you like. It will get close to the final value on the first 5-minute iteration, but then may wander a bit. It's probably a good idea to let it run for an hour or more just to get any temperature variations to settle out and get a good idea of what the optimum setting is. Included here in a separate file is an example of the output stream for an M over a couple hours.

Other

The serial monitor should be set to produce a line ending - CR, LF or both - with a baud rate of 115200. And you'll probably want to turn on timestamps.

The algorithm does not change the date/time or alarm register contents. Only the Aging register contents are changed. The only exception is when you choose to change the time using "T" below.

While the system is running, you can enter the following commands (either case), followed by [Enter]:

"A" displays the current Aging register setting in the RTC.

"A*n*" sets Aging to the new "*n*" value. Permitted values are -128 to 127 inclusive

"T" stops the Aging algorithm and displays the RTC's time in *mm:ss* format each second. Hit any key to return to the algorithm. This does not reset any time registers.

"T*mmss*" first sets the RTC time to *mm:ss*, then does the "T" second-by-second display.

"T*YYMMDDhhmmss*" first sets the full Date/Time of the RTC as specified, then does the "T" display. "*hh*" must be 24-hour basis. The day of the week is calculated automatically based on Monday being day 1. If you want Sunday to be day 1, change MondayOne to false.

"Q" terminates the session. The original RTC Control and Status register contents are restored.

Note: When "T" is used to set the time, that also sets the beginning of an RTC second. When you hit [Enter], the code waits until the rising edge of PPS to initate the change. So if you are following time.gov, or whatever, to set the RTC, hit [Enter] a half-second before your *ss* value is to occur, and the RTC will be set to the correct time automatically on the next PPS rising edge.