What is Lars Arduino based GPSDO Controller with 1ns resolution TIC?

By Lars Walenius, Aug 11, 2017

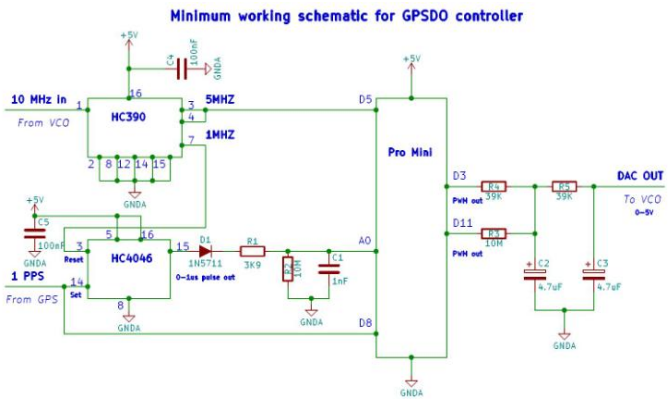
This is a flexible GPSDO (GPS Disciplined Oscillator) controller mainly for the experimenter and will work as a complete GPSDO with a GPS receiver that can output 1PPS (pulse per second) and a 10MHz VCO (voltage controlled oscillator).

I have built several complete GPSDO prototypes to test both hardware and software but probably it still has bugs.

As it has a TIC (Time Interval Counter) with a resolution of 1ns (nanosecond) it is also useful as a cheap and simple frequency counter for 10MHz oscillators.

I started the design many years ago, around 2010 I think, just because I was lazy and didn´t want to tweak my 10MHz house standard. The design is very inspired from other designs and mostly from Brooke Shera's excellent design in QST 1998. The only thing I am proud of is the very simple 1ns TIC.

The software is tailored to my needs so many will say it is far too complex but it contains the functions I want. Of course, I have a long list with other functions I also want but that are another story. I have many times thought of using another processor than the ATmega 328P but think it has it charm to use this simple processor and the Arduino platform. I also should say I have worked too long as an analog and RF engineer so software is not my best discipline and another reason to stay with the Arduino.



The hardware is simple. It is an Arduino controller with just a few HCMOS IC´s and discrete components. The Arduino and HCMOS circuit measures the time difference between the 1PPS and 10MHZ and outputs a voltage to the 10MHZ oscillator, so basically a PLL (Phase Locked Loop). As the 1PPS isn´t so reliable the software takes care of some conditions such as missing or wrong pulses to some extent. I guess this might be a reason it is called a disciplined oscillator instead of PLL.

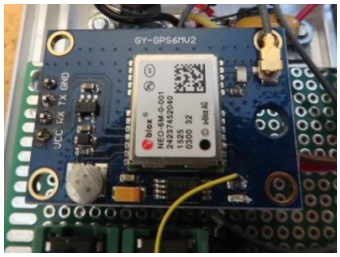
The resolution of the time difference measured is about 1ns. The range is 10ms (milliseconds) as both an analog hardware measurement with a range of 1000 ns and a timer measurement are combined.

The main loop is a normal PI-loop (Proportional + Integral) in software. As the 1PPS has a lot of jitter, normally in the range 10-50ns p-p, the time measured between the 1PPS and 10MHz is filtered with a low-pass filter in the software.

The hardware may seem simple but the performance is still mostly dependent of the GPS receiver and 10MHZ oscillator. So, first I recommend selecting a GPS receiver and oscillator depending on your requirement on stability.

GPS receiver

Almost all receivers with 1PPS out will work. The PPS pulse shall be positive and enough to drive the inputs. So far 3.3-5V pulses from the receivers have worked for me. I have tested both normal position only receivers and timing receivers. Timing receivers gives the chance to have better performance but if you have no receiver I would recommend to start with a uBlox NEO-6M or 7M that can be had, mounted on a board, for less than 10USD at eBay.



I recommend the blue boards that I have tested several without problems. With the red boards, I have just had problems. On the blue boards the 1PPS is taken from the resistor before the flashing LED on the uBlox pin side.

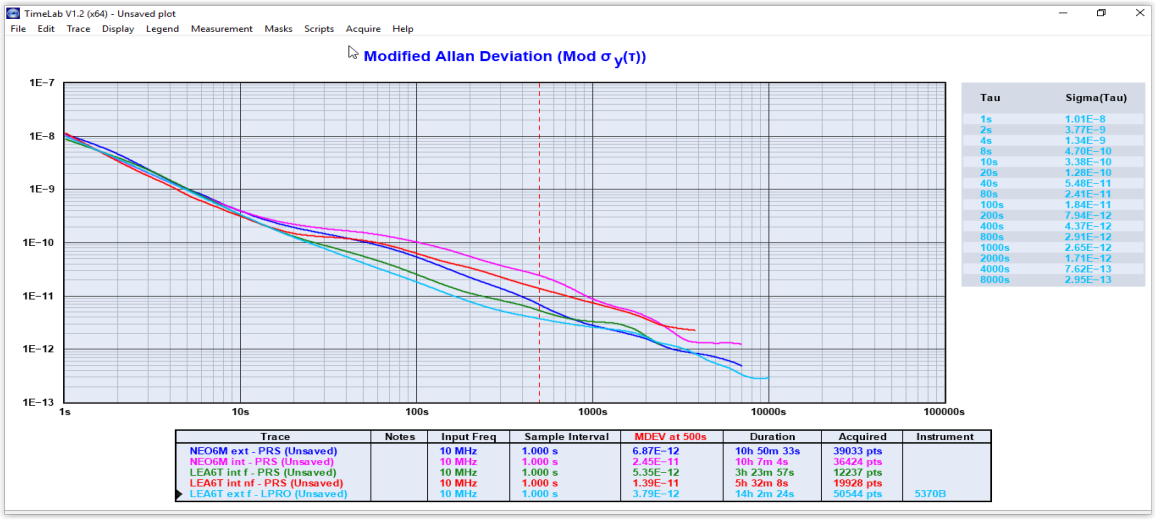
My recommendation before you build the controller is to test a GPS receiver.

Modern GPS receivers like the uBlox often work inside a building but having an antenna outside is better even if just near the window. I have used both cheap patch antennas (around 3-5USD on eBay) to more expensive antennas with good results.

Using a program like the manufacturers own (eg uBlox) or other programs like Lady Heather that can show you how many satellites you have and the signal strength is very useful. Seeing at least 4 satellites is preferred even if timing receivers might work down to one satellite. With outdoor antennas, I often see 50dB Signal to Noise Ratio but in-house on the upper floor I have only 30dB but the signal is still ok to lock the GPSDO. With S/N ratios of 20dB it is a large risk that the signal is too bad. I also have a lock on the Altitude charts available in the u-center, if you have deviations above 50 meters you may have problems with the timing also. With a poor in-door antenna position I loosed lock of the GPSDO even with seven satellites used but the S/N ratio averaged only 20db and with an altitude peak of about 80 meters wrong the GPSDO loosed lock as the time deviation was more than 100ns for more than 16 seconds (PPS lock filter in software is 16 seconds).

The GPSDO controller works best if the PPS signal only is available when it is valid. Check if the PPS stops if you have a really bad signal or the antenna is disconnected. Some modules have settings to disable the PPS output. Check for your particular receiver.

Some examples of what you can get with NEO6M and LEA6T GPS modules:



In this EEVblog thread you can see some more of my results with different receivers and internal or external antennas: https://www.eevblog.com/forum/projects/ocxo-stable-reference-and-control-voltages/50/

10MHz oscillator

The Arduino GPSDO controller described can handle voltage controlled 10MHz oscillators with a range of 1ppb (part per billion or 1E-9)) to 6.5ppm (part per million or 1E-6) with a DAC output voltage of 0-5 Volt and DAC output impedance of 78kohm.

If your oscillator doesn't have a range of 0-5V it should be quite simple to add a resistor network and/or an op-amp to change the range. If your VCXO (voltage controlled crystal oscillator) for example have a range of 30ppm for 0-3.3V a simple resistive divider with a couple of resistors or a trim pot will be useful to restrict the range to 6ppm.

Note: Oscillators with inverse control characteristics, that is: lower frequency with higher control voltage, needs hardware inverting op-amp or similar or changed software.

I have tested the controller with both VCXO, VCTCXO (voltage controlled temperature compensated XO), OCXO (Oven Controlled XO) and rubidium oscillators. Some of my experiences: The cheap VCXO for a few USD from e.g. Digikey and Mouser works but has just slightly better performance than the 10MHz you can have from the NEO-7M, that is about 7-8 digits for short times (seconds). The cheap VCTCXO I bought from Digikey are not better as they sometimes make jumps in the region of 50ppb (5E-8). For experiments, they work. The better VCTCXO´s as DOT050V from Digikey works well and I have got ADEV´s below 1E-10 all the way from 1s. See figure 8 in appendix. Most of my prototypes have had OCXO´s from eBay. As they can be had for about 10-50USD they are a bargain if they work. Some have reported just problems but that is not my experience. I have had one that just works a couple of minutes. Some have had quite frequent small jumps in the 1E-10 range but not really a problem. Some are excellent with ADEVs in the low E-12 or even high E-13 for some Tau´s (time scale for ADEV´s).

I recommend buffering the 10MHz oscillator. Not so much for the Arduino controller but for your outputs that you will use with different loads. Otherwise you will have jumps every time you connect or disconnect a load. I have used 74AC04 and often even 74HC04 even if they are slightly worse but if you just need ADEV's (see description below) in the upper E-12 range the HC is ok. If you want 50ohm outputs, I have paralleled three inverters each with 120 or 150ohm series resistors. I have also added an external LCL tee-filter to get a sine output if needed. I have got about 11-13dBm sine output with an L of 10uH and C about 50pF (trimmed for maximum output).

Remember that the loop that controls the oscillator mainly changes with very low frequency so the noise above 1Hz is mainly due to the oscillator (and 1Hz from the PPS with overtones if bad grounding). If you have requirements on the noise for example for instruments or RF work the oscillator is most important.

What is ADEV?

One way to learn more is to check Wikipedia for Allan Variance https://en.wikipedia.org/wiki/Allan_variance . This gives a very good description.

Otherwise a very simplified way of describing ADEV (Allan Deviation) is to say it is the frequency stability (not absolute accuracy) described as the standard deviation of difference of two nearby frequency measurements with a gate time of Tau in seconds. So, if you have an ADEV of 1E-10 at Tau 10 seconds and you have a perfect frequency counter, the readings will fluctuate several parts in E-10 (as the ADEV is just standard deviation not peak to peak) that is in the 10^{th} digit but this is not the accuracy.

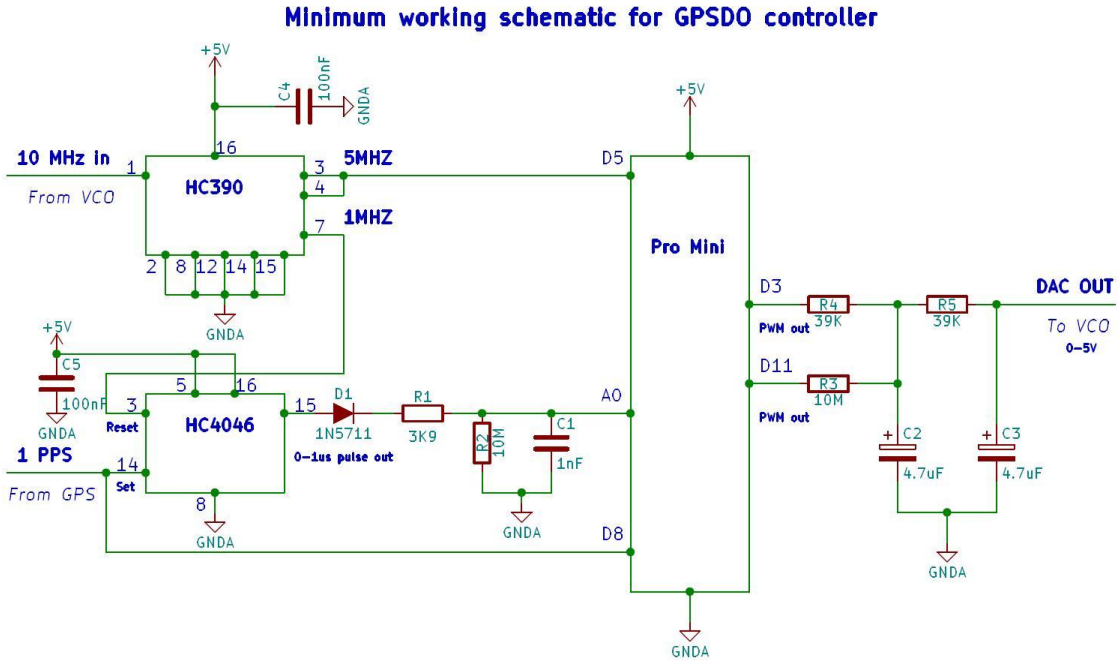
For a GPSDO that is locked to a "perfect" reference the absolute accuracy may be as good if the loop is tight so the oscillator doesn´t deviate for longer times (that is for larger Taus). A free running oscillator drifts away with time and so will have worse ADEV at larger Taus.

The nice thing is that the ADEV for a GPS is only bad at low Taus but gets better and better for longer Taus so by combining these two in the PI-loop gives the best of two worlds. At some Tau, the ADEV for the GPS and oscillator are about the same (line crosses). At this point the loop should change from the oscillator to the GPS, of course it isn´t a switch-over but a gradual change between the GPS and oscillator. This is selected by setting the time constant for the PI-loop. As the error signal is filtered it is sometimes better to use the cross over point of MDEV.

In the appendix you can see some examples of ADEV etc.

The software can be set through the serial port for time constants from 4 to 32000 seconds with command "t"
Typical values for an XO might be 5-30 seconds, for a good TCXO 30-100s, OCXO 100-1000s and a rubidium 2000-10000s.
With the XO you can get ADEVs in the E-8 -9 range, for a good TCXO E-10 to -11 and an OCXO E-11 to -12 and a rubidium oscillator (at Taus greater than 1000secs) in the E-13 range. At Taus of a day all GPSDO's correctly built have ADEVs in the E-13 and some even high E-14 ranges.

Hardware description



The “fine” TIC with a range of 1us consists of the 10MHz divided down to 1MHz fed into an HC4046 reset pin and the 1PPS fed into the HC4046 set pin. The 1PPS sets the HC4046 output high and starts to charge the 1nF capacitor through the diode and 3.9kohm resistor. Within 1 us the stop pulse from the 10MHz divided down to 1 MHz set the HC4046 output low and charging stops. As the diode is in series the low level on the HC4046 will not discharge the capacitor. The leakage current on the ATmega 328P is quite low so the main discharge will be through the 10Mohm. As the 1PPS gives an interrupt to the processor the ADC will measure the voltage almost immediately after the charge is ended. The ADC start time is not synchronous with the 1PPS but the internal 16MHz so a timing error of several us gives a ripple on the reading if the discharge is too fast. I used 1Mohm before and that gave a ripple of several ns. The stability of the TIC at around the midpoint of 500ns is good. I have measured around 0.3ns/°C. Most of it comes from the about 2mV/°C change on the diode I guess. The linearity of the TIC is of course not perfect as it is more or less an RC circuit so it is exponential. The good thing with the ATmega328p is that is has a selectable ADC range so I have chosen the 1.1V range. That makes the nonlinearity quite small. Some other DIY GPSDO that has adapted my TIC like Nick Sayers and the Elector GPSDO has tried with a FET as current source as they have the full ADC range. So far I have seen no evidence that this is better, it also adds a component with much more variation than a simple resistor. Screenshots from the Elektor GPSDO ramp still has much worse linearity than using the 1.1v range of the 328P. My software also has the possibility to linearize the slope. I have got the INL down to around 1ns by using that. If you use the controller only as a TIC I have managed to get ADEVs below 1E-9 at 1s with the serial line directly feeding the excellent Timelab software. See examples in appendix.

10MHz divided by 2 is fed to timer1 that make the TIC range much wider in the software. The timer1 need to be fed with less than half of the processor clock frequency so that is the reason for the division. The resolution of 200ns still is enough to “overlap” the 1000ns resolution of the fine TIC so the software can combine them.

The 16 bit DAC is implemented as a PWM-DAC. The ATmega328p has two 8-bit PWM´s that are used to get the 16 bits. The two 8-bit outputs are combined with two resistors with a ratio of about 256. With 1% resistors, it is possible to get at least 13-14bits of monotonicity that is the most important parameter. The stability is mainly dependent on the 5V regulator. Often the 7805 types have around 100ppm/C or more that will limit the performance. So far I have used different types of 7805 with good results despite that. The noise is not the problem as the output is heavily filtered. One way to make this controller better is to change to one or two monotonic DACs for example MAX5217 with a more stable reference like the MAX6070. Changing the software for that should probably not be very difficult but I have not tried. For me the PWM-DAC has worked fine so far. For those skeptical that a 16 bit DAC is enough I recommend to use the GPSDO simulator found on leapsecond.com to simulate different resolutions. This simulator also shows that 1ns is more than enough for most GPSDO's. I have one Isotemp 131-100 OCXO based GPSDO with a "poor" resolution of 1.2E-11 (about 1ppm VCO range) that have ADEVs in the E-12 range. See figure 8 in appendix.

The hardware and software can use ADC2 to measure temperature and also for temperature compensation. The software has a setting to read °C for an LM35 or 10k NTC with beta 3950 + pull-up 39k, 47k or 68k. The value is output every second on the serial line. As the NTC is cheapest I have used that most of the time. Adding a capacitor (1nf or larger) in parallel with the ADC inputs are recommended. But easiest is to have an LM35 connected to ADC2.

Using ADC1 to read temperature or something else is optional. The input range is about 0-1.1v. I have often used an NTC + pull-up connected near the processor and the ADC2 sensor near the OCXO. The software outputs the value from ADC1 every second to the serial port.

ADC3 also has a range of 0-1.1V and can be read by a command ("f3") on the serial line. I have only used it to read the lamp voltage on my rubidium based GPSDO.

A LED + resistor connected to D13 will give an indication that the loop is locked. After five time constants with a slightly filtered (tc=16s) TIC value within 100ns it will light. If the DAC value, in locked state, is near the ends (<3000 or >62535) it will flash. If you have "no PPS" it will give a brief double flash. I added this to give a visual indication if the antenna is disconnected. It will of course also indicate at all other conditions that give no PPS.

Power supply:

In the beginning I powered my prototypes from lab power supplies without extra regulators and that worked quite well. But when I mounted them in boxes I added regulators.

I have used MC7805 type regulators for the Arduino pro mini and HCMOS power. I have also used the same regulator for the GPS receiver. If you use one regulator for both be careful to take the +5V directly from the regulator in two different paths as the GPS receivers often have very fluctuating current that might modulate the Arduino PWM-DAC.

For OCXO's with 5V I have used a 7805 both for the OCXO and CMOS buffer stage (not the same as for the Pro Mini and GPS). For 12V OCXO's I have used 7812 style regulators and added an extra 7805 for the CMOS buffer stage powered from the 7812.

For the power input I have used a 5.5/2.1mm jack and added a 1N5819 in series for reverse power protection. Most of the time I have used cheap 15V stabilized switched mains adapters even if I don´t know if they have given a lot of noise. If you are using the GPSDO to drive radio equipment or instruments that needs low phase noise it might be better to use a linear supply.

Building the hardware - selecting components

For the processor I from the beginning used an Arduino UNO board, I have also used ATmega328P IC with separate 16MHz xtal and UNO boot loader. But what I prefer are the Arduino ProMini 5V 16MHz boards for under 2USD found on eBay. So far I haven´t had any problems with them and have used both types with crystals and ceramic resonators. They require that you have a USB-TTL converter with DTR (to be able to program) but I still find them best.

The HC4046 and HC390 might also be HCT without problems.

I have selected the diode due to the low capacitance but am not sure if that is a real problem. Using a schottky is also preferable due to the low voltage drop even if a 1N4148 probably will work (I have tested one). 1N5711 and 1N6263 are types I have tested several of. BAS70 could probably be a cheaper alternative. If the capacitance in the diode is too high, it will discharge the 1nF a little when the HC4046 output goes low.

The 1nF capacitor shall be NPO ceramic or some other temperature stable type. I have mostly used SMD0805 types. For the 4.7uF I have used tantalum drops with 25 or 35V ratings as I have hoped they have lower leakage current. The leakage current by itself is not a problem if it is stable but if it varies with temperature it will change the output voltage due to the voltage drop over the 39+39k resistors.

Having decoupling capacitors is necessary. Use 10-100nF at the IC's +5v.

The resistors I use have been 1%. No special types. I have found that SMD0805 types fit well on the bottom side of prototype boards with 0.1 inch spacing.

Just for experimenting it isn´t necessary to have a metal box but for a lab standard I think it is mandatory to have a well shielded box and put some effort to have good grounding e.g. ground planes or good connection to the box at relevant points. My first GPSDO was an Arduino UNO with a prototype shield and together with the 10MHz it gave a lot of disturbance to other parts in my lab.

Good grounding is not only necessary because of the 10MHz but also on the very sensitive voltage control signal. I have the last 4.7uF (sometimes I have used 10uF) close to the OCXO for example. The GPS modules often have very fluctuating supply current so having good control of the ground currents is necessary if you don´t want a lot of 1Hz modulation on the 10MHz output. Just a blinking LED might give too much modulation if not properly handled. As the 5V to the controller is also used for the PWM DAC it is important that fluctuating supply current don´t give ripple on the 5V (might be another reason to change to a DAC with separate Vref).

Software description

See .ino file

Most of the software is driven by the PPS interrupt. The PPS interrupt function is very short and does mainly three things. First it reads the ADC0 to get the TIC value (nanoseconds). Second it reads the timer1 value. The timer1 value is already captured by hardware. The third is to set a PPS read flag.

Most of the main loop function waits for the PPS read flag. Before it gets that it checks the timer1 and counts overflows that happen every 10ms. If the count is over 130 (1.3sec) it serial prints "No PPS". The main loop also checks if anything is written on the serial line and have a function to blink the locked LED if the DAC value is near minimum or maximum. With the PPS flag set (by the interrupt) it goes to the calculate function.

The calculate function first combines the TIC value (time error in nanosecond) and timer1 value. After that it low pass filters the value if the loop is in the locked state and goes to the PI-loop.

The low pass filter time constant is set by the preFilterDiv that can be configured as 2, 3 or 4. Default it is set to 2.
The filter time constant is*: (PI-loop) time constant / preFilterDiv*.
With a time constant of 100 the filter time constant will be 50. Maximum filter time constant is 1024s (this is mainly because I wanted to avoid overflows in internal calculations).

The low pass is implemented as:
Filtered time error = Filtered time error + (time error - Filtered time error) / filter time constant

The proportional term changes the output (in ppb or ns/s):
Filtered time error (in nanoseconds) divided with the time constant (in seconds).

The integral term adds this every second to the output (in ppb or ns/s):
Filtered time error (ns) / time constant (s) / time constant (s) / damping (unit less)

Damping is default 3. This gives a quite fast response with just a little overshoot. The damping can be configured between 0.5 and 10. 10 give a slow response but no ringing. 0.5 gives a lot of overshoot with a lot of ringing.

The PI-loop uses a combination of long and float variables to give about 15 digits of precision. The Arduino float is only about 6 digits and the long about 9 digits so the combination was one way to get enough resolution to handle the large span of time constants.

A large part of the calculation function is storage of 300 and 10800 second averages of time error, DAC value and temperature (ADC2). I have selected 300seconds a little arbitrary. It is a compromise between the maximum storage time (now 12 hours) that I of course wish to have longer and having as short averaging time as possible. The 10800second (3 hours) is selected to give eight points per day. Four points could have been enough to see daily variations but I choose eight points. This gives 18 days of storage. I like the 10800secs storage as that gives the possibility to see both long term and temperature drifts.

The printDataTo Serial function also is very long. It prints a lot of information. See example from startup in figure 1 in Appendix. If you just wish some data it is possible to change with the "i" command. The short version only shows the first five rows: time, ns, dac, temp and status.

Use "f1" for help on commands. See example in figure 2 in Appendix. Using "f1" gives a lot of information from the code for the getCommand function. By the way a special thank to Jim Harman for this function and other small comments.

The time error prints without decimals but for better accuracy it is possible to have one decimal. As the time error is linearized, truncation may give worse results if used as a Time Interval Counter together with Timelab without the extra decimal.

A special function for linearization of the temperature sensors selects different conversions for different sensors on ADC2. This is selected with the "j" command. The default "j0" is raw ADC values (0-1023). Right now "j1" selects LM35, "j2" 10k NTC with beta 3950 + 68k pull-up, "j3" 10k NTC with beta 3950 + 47k pull-up, "j4" 10k NTC with beta 3950 + 39k pull-up and "j5" 22k NTC with beta 3950 + 120k pull-up. "j9"selects LM35 with Fahrenheit readout. "j8" is a special case if the Aref is low as it assumes a Aref of 1070mv instead of 1100mV for an LM35.

Installing the software

Download the Arduino IDE for your computer from https://www.arduino.cc/en/Main/Software

Open the GPSDO sketch (program). Power up and connect the processor to the computer. Set the Arduino IDE to the correct processor and serial port.

Download the program to the processor

Startup and setting the configuration

Before testing the function of the controller, it is good to test that the 1PPS and 10MHz is ok. A 10MHz sine output from the oscillator preferably is converted to 5VCMOS with just an HC04 or AC04 inverter with a 10k resistor between in and output.

If power up works, connect the serial line and use a serial monitor (e.g. Arduinos own) to see if the program runs.

The serial monitor in the software is right now using 9600 bauds standard setup.

If the processor and program is ok the serial monitor first will print "Arduino GPSDO with 1ns TIC by Lars Walenius". Next it will print revision and a header. If the PPS is not connected or the GPS receiver not outputting the PPS yet the output will be "No PPS" otherwise it will start to output a long string of values.

It is possible to test the help functions by sending "f1" to the processor, a long text should return with almost all commands. Sending "f2" will give some variables (See figure 3 in Appendix), " f3"

reads the ADC3 and “f4” reads part of the EEPROM (See figure 4 in Appendix). As the EEPROM is new it will mostly read 65535. Later you will save parameters to the EEPROM with the “s1” or “s2” commands. Sending “e22” will set all the EEPROM to zero. Now is a good time to send “e22”. After that you can check it with “f4” that now shall read all zeros.

The next step is to connect the 1PPS and 10MHz and Voltage Control to the oscillator if not yet done.

Before the PPS comes on from the GPS receiver it shall show “No PPS”. As soon as you get PPS it shall show a long string and it is time to learn the different fields of the serial monitor. The first field is just seconds since start. The second field is the TIC value in nanoseconds if the 10 MHz is missing it will be 1023 (as the stop pulse to the HC4046 is missing) but be shown as “missing 10MHz?” The third field is the DAC value between 0 and 65535. The first five minutes it will show “warmup” in the fifth field and the DAC value shown in the third field with an empty EEPROM is 32768.

Test to send different DAC values by sending “h” + “value” +” enter”. Value shall be 1 to 65535. If you send “h0” it will hold the old DAC value.

If you send “h1” the output, if not connected to the VCO, will be close to zero and with “h65535” it shall be close to 5V measured after the low pass filter. So temporarily disconnect the DAC output to the oscillator. With just a 3 ½ digit DMM it is possible to see if the low and high 8bit PWMs are working. Set “h256” and measure the value it shall be about 20mV. Set “h511” it shall be about 40mV. Set “h524” and the value shall be about 1mV higher.

With the VCO connected again it is time to test the VCO range. Set “h1” and check the field diff_ns. It will show the frequency deviation roughly in ppb (1E-9) but with the opposite sign. As the PPS from the GPS has a lot of jitter the diff_ns will have that also as it is only the difference between the new and old TIC value. The diff_ns will be even worse as the fine TIC “overflows”. Later with linearization the extra step at “overflow” can be minimized. By averaging it is possible to see the minimum oscillator value. Another way is to see the frequency deviation is to use the TIC value in the second field and calculate the frequency.

Do the same for “h65535” to find the maximum frequency. The minimum shall be below zero and the maximum above. Calculate the difference between minimum and maximum. For a VCXO or VCTCXO it might be 10000-50000 ppb. For most of the OCXO’s from eBay, that often have SC-cut crystals, it is normally 500-5000ppb (AT-cut OCXO can have higher values). Now or later you can restrict the VCO range by using resistor networks and/or trim potentiometers to restrict the VCO range. Setting “h32768<enter>” is a good check also. If the VCO range is above 6500ppb you should restrict the range.

The VCO range is needed to set the gain and the gain is needed to get a working PI-loop.

The gain is calculated as *gain = 65536 / VCO range* (in ppb). A range by e.g. a VCTCXO of 6500ppb (6.5ppm) gives a gain of 10 and with a rubidium with a range of 1ppb the gain will be 65535. For a good OCXO restricting the range to say 130ppb, giving a gain of 500, gives a DAC resolution of 2E-12 per bit that has worked very well for me. Setting an OCXO to 130ppb range gives less margin for long-term drift but most OCXO’s with SC-cut crystals drifts less even over a life time (otherwise you could change the resistor network/ trimpot). As you also noted the gain is the inverse of the DAC LSB so the

LSB =1/gain ppb that is useful to convert the dac value to frequency for example when using the Timelab to read the dac value.

Load the gain value by entering “g<value><enter>”. Save it to EEPROM by “s1”.

Now it is time to close the loop. Enter “r”. This will set run mode. The default time constant is 32 seconds. If everything is correct the PI-loop will manage to get the TIC value stable to within 100ns in say 10 time constants (about 5 minutes) and if the ns (TIC) value is within 100ns for more than 5 time constants the status field will say “Locked” instead of “Nolock”. Yes!!! It is working!!! Time for experimenting and fine tuning.

Trouble shooting

To troubleshoot the hardware an oscilloscope and DMM are useful. As the most important signals are the 1PPS and 10MHZ an oscilloscope are useful to follow the signals.

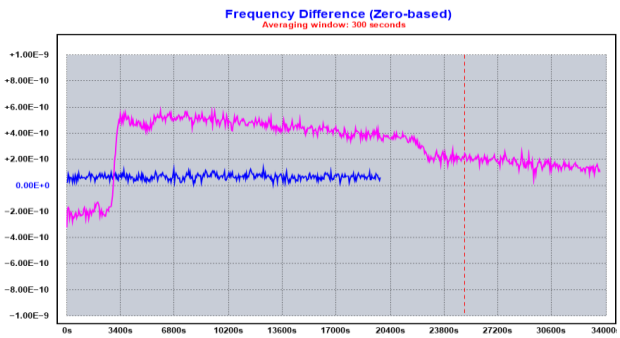
The output on HC4046 pin 15 shall be a pulse 0-1us, once a second. If not check the inputs to the HC4046. Remember that the 1PPS from the GPS module is often just a short pulse in the microsecond or millisecond range. On the ADC0 input it shall be a corresponding ramp.

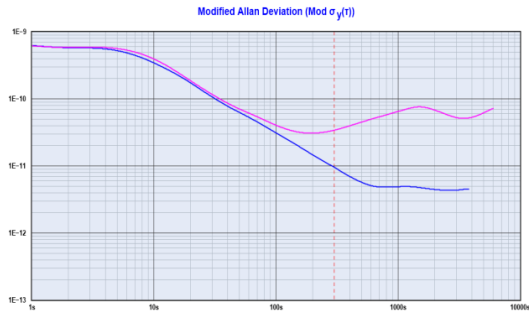
The PWM-DAC outputs can also be checked with an oscilloscope. Frequency is about 488Hz.

Experiments and tweaking

Finding optimum time constant and setting linearization parameters:

After getting lock the next question is to find the optimum time constant. One way is to set the GPSDO in hold mode (“h0”). Now the GPS and oscillator can be measured by the internal TIC. With Timelab it is possible to see the time, frequency, ADEV, MDEV etc. in real time. I prefer to use MDEV to find the optimal time constant. The reason is two. First it gives lower time constants than ADEV (ADEV for me seems to get too long TC). The second is that the MDEV is filtering the signal, as the GPSDO value also is low pass filtered. Always check that the frequency response shows no “jumps” as that can upset the MDEV curve. Below are examples of response with and without jumps.





Of course if your oscillator has jumps it may be a reason to lower the TC. In the example above maybe a TC of 150s may be reasonable with the jump and 500s without the jump. A large jump may also make the GPSDO to lose lock. A rule of thumb I have is that a jump of 1ppb and a TC of 100s may give a time excursion up to 100ns. As the lock is lost at 100ns time offset, that is on the edge. If you have a TC of 1000s a jump of 0.1ppb could be enough to lose lock. This is also a reason that I really dislike the digitally compensated VCTCXO's that often have jumps of about 50ppb due to temperature compensation. With 50ppb jumps you can lose lock with just a TC of 2s!

One important thing if you use the internal TIC to get ADEV etc. is to set the linearization parameters. The most important is to set right span of the TIC. That is linearization min and max. The serial monitor shows the linearized value in row two. Further on you find a column "filtx10" that shows the filtered ADC value. If the GPSDO isn´t locked the filter is off so the value is the raw ADC values times ten, so just take away the last zero. If the TIC offset is set to the default 500 "filtx10" will be close to 5000 at lock. This is also true for the 300 and 10800s logged values.

One way to find the min and max easy is to have the GPSDO locked and change the TIC offset (note the TIC offset is subtracted away in column 2). With a TIC offset of 1000 the GPSDO will go between min and max if the max ADC is less than 1000. The filtx10 values will get you the min and max. Use the command "l" (small L) to set these and store them with "s1". Remember to set the TIC offset back to 500 before saving. The "x2" value (non-linearity compensation) is difficult to get with just the noisy PPS. The best way to get it has for me been to use the PICDIV PD26 from Tom van Baak (see leapsecond.com). With the internal 10Mhz connected through the PICDIV PD26 to the PPS it will give you steps of 400ns but as the range of the TIC is 1000us the readings will be 0, 400, 800, 200, 600, 0, 400 etc. So you get 5 calibrations points. The "diff_ns" readings shall be 400+-1ns if the linearization is correct. I have managed to get ADEVs of less than 1E9/Tau in this way with Timelab (remember to turn on the extra decimal with "i"). See figure 9-13 in the appendix.

The second method to get a good TC is to use the GPSDO in locked mode with minimum TC of 4 s. This don´t require the linearization. Now look at the DAC value with the Timelab program. See figure 5-6 in the appendix how to setup Timelab. By setting the Timelab to frequency difference mode and adjusting the scale to the DAC gain (1E-9/gain e.g. gain 500 gives scale 2E-12) you can see how the DAC is adjusted. This will also give you an MDEV curve. Below about 10s the result is too low as the loop will filter the DAC. At Tau 1s you probably will see slightly below 1E-9 with a PPS from the uBlox series 6 or 7 (NEO 6-7, LEA6T etc.). Here it is the same comment about jumps as above. See pictures above. In the picture you see the downward slope from the PPS jitter. At some point the slope changes and the oscillator drift and noise is seen. At this "intersect" a reasonable TC is found. If the oscillator has a lot of drift the slope start to go upwards to early (remember that the I-term compensates for linear drift). One recommendation is to use the "subtract linear drift" in time lab. See figure 7 in appendix.

A third possibility to find a TC is to set a long TC and see if you lose lock. Lower the TC until you get reliable lock and say maximum +-50ns excursions. But it is not my preferred method.

The second method probably is quickest.

With method one or two you will also get a feeling for the noise floor of the oscillator. Even if the only reasonable correct point is the intersect point that contains both the PPS and oscillator jitter, a

good guess is that the oscillator set the ADEV-MDEV for the complete GPSDO at Taus below the intersect point and will be no worse than a decade higher for Taus down to 1s. A guess for the long Tau is that it will be around 1E-12 at 10000secs and 1E-13 at a day even with a simple NEO-6M receiver.

Setting Damping:

Damping is default set to 3. If you want to experiment, it might be set to between 0.5 and 10.

0.5 will give a lot of ringing and 10 a slower response.

As a note I think the Thunderbolt GPSDO damping of 1.2 corresponds to what I call 3, 0.5 is my 0.5, 0.7 is my 1 and 2.2 is my 10.

TIC offset:

If you change this the 10 MHz will shift the corresponding nanoseconds relative to the input PPS. This may be useful sometimes, but don´t change too much. +-50ns is probably ok.

Checking step response:

By sending a new DAC value it is possible to see the step response. Check the time and DAC value and make a graph.

Setting readout of temperature:

See the software info above for different options with LM35 or NTC´s to get readout in Celsius or Fahrenheit (for LM35).

See long-term drift and temperature drift with the three hour log:

I like the three hour log as it gives the possibility to quite easy see the drifts. The DAC value can be transferred to frequency just by dividing with the gain factor. This gives relative frequency in ppb (1E-9).

Making temperature compensation:

I have only used this for rubidium oscillators. For OCXO´s it has not worked so well and I have not figured out if hysteresis, jumps or other factors have been a problem. Should probably do some more tests some time… For LPRO rubidiums it works quite well. Remember that the temperature compensation factor uses the raw data from ADC2. Also check the formula in the software, especially the *100 factor. (Formula: for my rubidium e.g. the log shows 60 dacsteps (9E-13) for 20 tempsteps (1°C) multiply by 100 gives minus 300 as factor, minus due to inverse to correct). I set the tempref to the ADC2 raw reading at the average lab temperature.

Setting Warm up time

The default warm up time is 300s, that is five minutes, this work well for most OCXO and rubidium oscillators. If you have an oscillator without oven, for example a TCXO, a shorter time could be set down to 3s by the command “w”. Save by ”s1”.

Adapt to different oscillator frequencies:

If you have an oscillator on other frequencies e.g. 5, 20, 50 or 100MHz you can change the division of the HC390 (for 50 and 100MHz change to AC390). Even other frequencies as 12.8 or 13MHz should be possible to discipline if the software timer1 value 49999 is changed and the TIC hardware RC time constant is adjusted. I have not tested this yet but it shall be possible for frequencies between 6.5 and 13MHz to get all frequencies in steps of 200Hz.

Using as a Time interval Counter:

The most important is to find the correct linearization. See above in the instruction for time constant. The PICDIV PD26 is also very useful if you want to compare two 10MHz oscillators. Of course one might be a GPSDO, I have used my Rb GPSDO for this connected to the 10 MHz and the Device under Test (DUT) connected to the PICDIV input and PICDIV out to the PPS in. Here the serial monitor column “diff_ns” gives a quick feeling for the frequency offset. Connected to Timelab it makes an excellent logging frequency counter for 10MHz OCXO´s.

Links

Wikipedia GPSDO - https://en.wikipedia.org/wiki/GPS_disciplined_oscillator

Shera's GPSDO - http://www.rt66.com/~shera/index_fs.htm
http://www.rt66.com/~shera/QST_GPS.pdf

Wikipedia PID - https://sv.wikipedia.org/wiki/PID-regulator

Poul Hennings simple description of a PI loop - http://phk.freebsd.dk/time/20141018.html

Pages 14-18 shows control loop of 1PPS locking of PRS10 (similar to the Arduino):
http://www.thinksrs.com/downloads/PDFs/Manuals/PRS10m.pdf

On pages 26-31 you find ADEV and selection of time constants for a modern GPSDO FS740 with different oscillators and similar control loop as in the Arduino:
http://www.thinksrs.com/downloads/PDFs/Manuals/FS740m.pdf

Arduino - https://www.arduino.cc/

ADEV - https://en.wikipedia.org/wiki/Allan_variance

GPSDO simulator - http://www.leapsecond.com/pages/gpsdo-sim/

Timelab - http://www.ke5fx.com/timelab/readme.htm

Time nuts forum - https://www.febo.com/mailman/listinfo/time-nuts

uBlox u-center program - https://www.u-blox.com/en/product/u-center-windows

Lady heather program - http://www.ke5fx.com/heather/readme.htm

PICDIV - http://www.leapsecond.com/pic/picdiv.htm

GPS receiver M12 ADEV-MDEV charts - http://www.leapsecond.com/pages/m12-adev/

GPS receiver M12 sawtooth - http://www.leapsecond.com/pages/m12/sawtooth.htm

GPS receiver uBlox ADEV-MDEV charts - https://www.eevblog.com/forum/projects/ocxo-stable-reference-and-control-voltages/50/

Test of four GPSDO's - http://www.leapsecond.com/pages/gpsdo/

GPSDO HP Z3801 OCXO variations - http://leapsecond.com/pages/z3801a-osc/

FLL versus PLL See ke5fx for example - http://www.ke5fx.com/gpscomp.htm

Appendix:

Arduino GPSDO with 1ns TIC by Lars Walenius					Rev. 3.0 170801 ID:1311										
Type f1 <enter> to get help+info															
time	ns	dac	temp	status	diff_ns	filtX10	tc	filt	timer1	temp1					
0	-390	34221	28.3	WarmUp	62	830	32	1	13838	28.1	Five minute averages: TIC+DAC+temp				
1	-384	34221	28.4	WarmUp	5	890	32	1	25021	28.1	Now acquiring value: 0				
2	-382	34221	28.3	WarmUp	3	920	32	1	25031	28.1					
3	-378	34221	28.3	WarmUp	4	960	32	1	25042	28.1	0	0	0	0	
4	-377	34221	28.3	WarmUp	1	970	32	1	25042	28.1	1	0	0	0	
....															
292	418	34221	28.9	WarmUp	-2	8980	32	1	25046	28.7	1143	4982	34222	33.6	Locked
293	433	34221	29	WarmUp	15	9110	32	1	25046	28.7					
294	431	34221	29	WarmUp	-2	9090	32	1	25046	28.7	TimeConst = 32 sec				
295	426	34221	28.9	WarmUp	-4	9050	32	1	25046	28.7	Prefilter = 1 sec				
296	441	34221	29	WarmUp	15	9180	32	1	25046	28.7	Damping = 3.00 Gain = 80				
297	434	34221	29	WarmUp	-7	9120	32	1	25046	28.7	Type f1<enter> to get help+info				
298	446	34221	28.9	WarmUp	12	9230	32	1	25046	28.7	Rev. 3.0 170801 ID:1311				
299	437	34221	29	WarmUp	-9	9150	32	1	25046	28.7					
300	450	34221	28.9	WarmUp	12	9260	32	1	25046	28.8	Five minute averages: TIC+DAC+temp				
301	441	35276	29	NoLock	-9	9180	32	1	25018	28.8	Now acquiring value: 1				
302	440	35285	29	NoLock	-1	9170	32	1	25018	28.8					
303	439	35293	28.9	NoLock	-1	9160	32	1	25018	28.8	0	5165	34221	28.6	
304	432	35289	29	NoLock	-7	9100	32	1	25018	28.7	1	0	0	0	
305	407	35244	28.9	NoLock	-25	8880	32	1	25018	28.7	2	0	0	0	
....															
523	-21	34418	29.4	NoLock	10	4790	32	1	25016	29.2	1074	4993	34216	33.3	Locked
524	-31	34392	29.4	NoLock	-10	4690	32	1	25016	29.2	1075	4995	34216	31.7	Locked
525	-23	34411	29.4	NoLock	8	4770	32	1	25015	29.2	1076	4995	34216	33.1	Locked
526	-16	34428	29.4	NoLock	7	4840	32	1	25016	29.2	1077	4993	34216	33.8	Locked
527	-30	34393	29.4	NoLock	-14	4700	32	1	25015	29.2	1078	4992	34216	33.2	Locked
528	-24	34393	29.4	Locked	6	4703	32	16	25015	29.2	1079	4994	34216	33.6	Locked
529	-19	34394	29.3	Locked	5	4710	32	16	25016	29.2	1080	4993	34216	33.4	Locked

Figure 1. Example of serial data pasted to Excel

```
Info and help - To get values for gain etc type f2 <enter>, f3 <enter> reads ADC3 and f4 <enter> EEPROM
Arduino GPSDO with 1ns TIC by Lars Walenius     Rev. 3.0 170801  ID:1311

Typing a<value><enter> will set a new damping between between 0.50 and 10.00 set 50 to 1000
Typing b<value><enter> will set a new tempRef between 1 and 1023
Typing c<value><enter> will set a new tempCoeff set between 0 and 10000. Adding 10000 gives negative tc
Typing d<value><enter> will set a new dacValue between 1 and 65535
Typing e<value><enter> will erase the 3 hour storage in EEPROM if value 1 and all EEPROM if 22 (33 sets all EEPROM to FF)
Typing g<value><enter> will set a new gain between 10 and 65535
  gain = (65536/settable VCOrange in ppb) (eg. 100ppb DACrange gives gain=655)
Typing h<value><enter> will set hold mode and the entered dacValue if not h0 that uses the old
Typing i<value><enter> with value 1 will toggle ns decimal point else will toggle amount of information
Typing j<value><enter> Set temp sensor type 0=raw 1=LM35 2=10kNTC+68k 3=10kNTC+47k (second digit=adc1 eg 3x)
Typing l<enter> will set TIC linearization parameters min max square
  values 1-500 sets min to 0.1-50, values 800-1023 sets max, values 1024-1200 sets square to 0.024-0.200
Typing n<value><enter> will set ID number 0-65535 that is displayed
Typing o<value><enter> will set a new TIC_Offset between 200 and 1020 ns
Typing p<value><enter> will set a new prefilter div between 2 and 4
Typing r<enter> will set run mode
Typing s<value><enter> will save gain etc to EEPROM if value 1 and dacvalue if 2
Typing t<value><enter> will set a new time constant between 4 and 32000 seconds
Typing w<value><enter> will set a new warmup time between 2 and 1000 seconds

time    ns      dac     temp    status  diff_ns filtX10 tc      filt    timer1  temp1
3642    -4      34413   32.3    Locked  3       4972    32      16      25016   32.7    39      0       0       0.0
3642    0       34413   32.4    Locked  4       4974    32      16      25016   32.7    40      0       0       0.0
```

Figure 2. f1 info and help

```
Gain            80      Damping         3.00    TimeConst       32      FilterDiv       2       TIC_Offset      500
TempRef         280     TempCoeff       0       TICmin          12.0    TICmax          1012    Square comp     0.100
Warm up time    300     LockPPScounter  3277    MissedPPS       0       TimeSinceMissedPPS  3645
ID_Number       1311    Restarts        19      Total hours     450
```

Figure 3. f2 serial prints some variables

```
EEPROM content:
restarts = 19
totalTime3h = 150
temperature_Sensor_Type =  53
ID_Number = 1311
TICmin = 120
TICmax = 1012
x2 = 100
TIC_Offset = 500
filterDiv = 2
warmUpTime = 300
damping = 300
tempRef = 280
tempCoeff = 0
dacValueOut = 34221
gain = 80
timeConst = 32
k = 6
```

Figure 4. f4 serial prints EEPROM contents

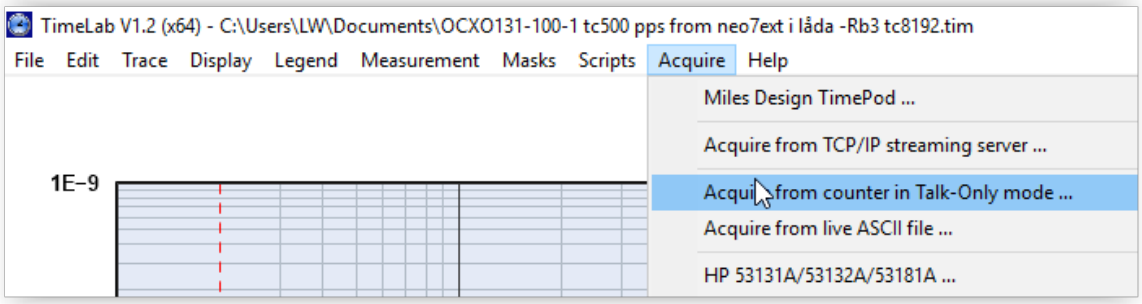


Figure 5. Acquire from Arduino serial port

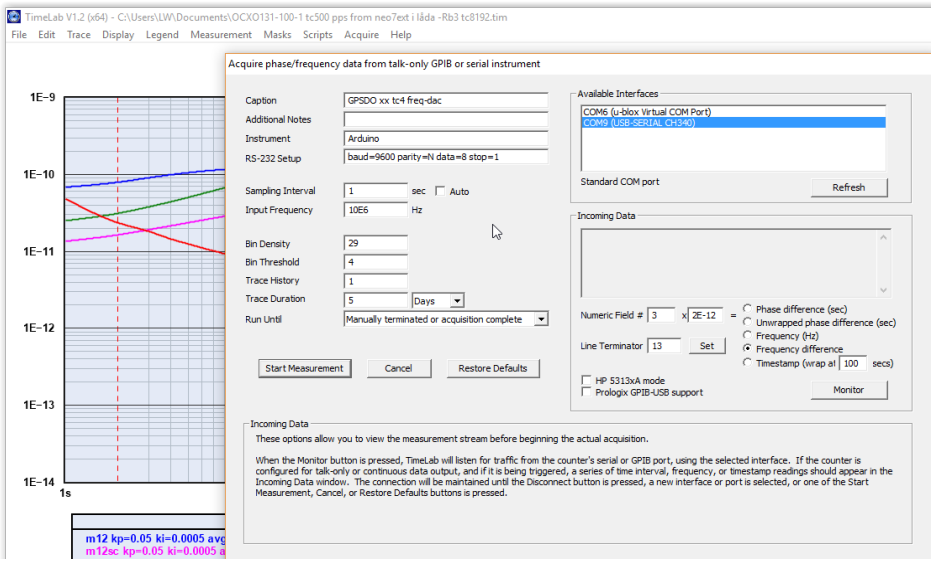


Figure 6. Acquire DAC value from Arduino serial port

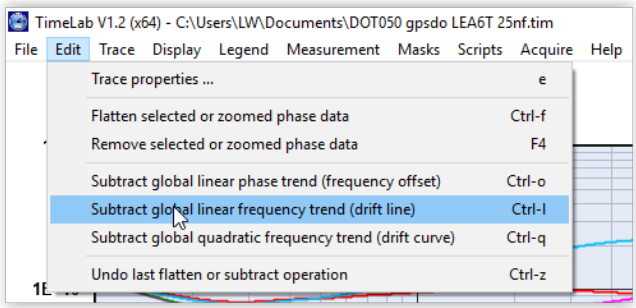


Figure 7. Subtract linear drift in Timelab

Figure 8. ADEV´s of two of my GPSDO´s. DOT050 is a VCTCXO, here tested with different GPS modes and time constants.

Figure 9. OCXO in holdmode measured with Arduino TIC (OCXO offset about 10ppb)

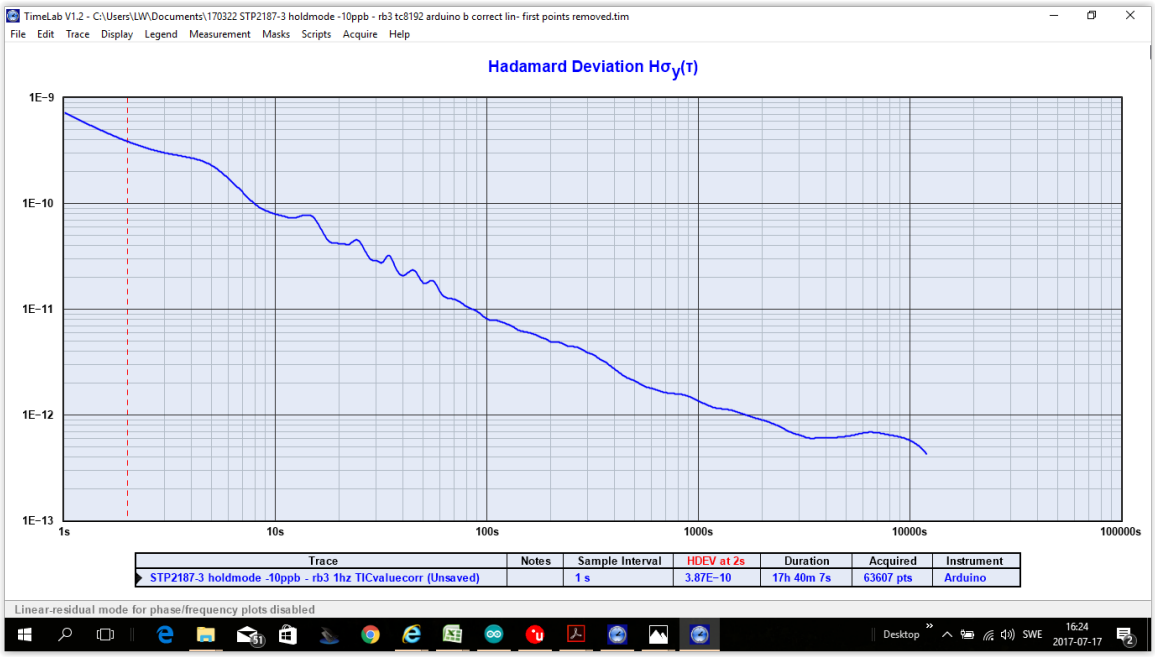


Figure 10. OCXO in holdmode measured with Arduino TIC

Figure 11. OCXO in holdmode measured with Arduino TIC

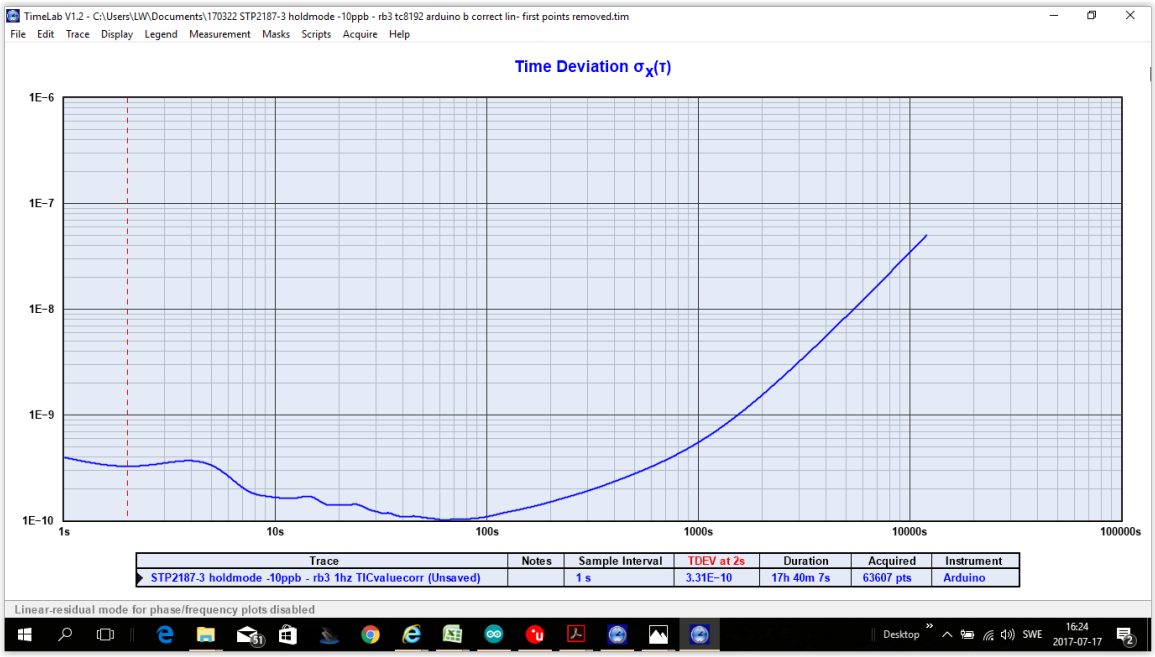


Figure 12. OCXO in holdmode measured with Arduino TIC

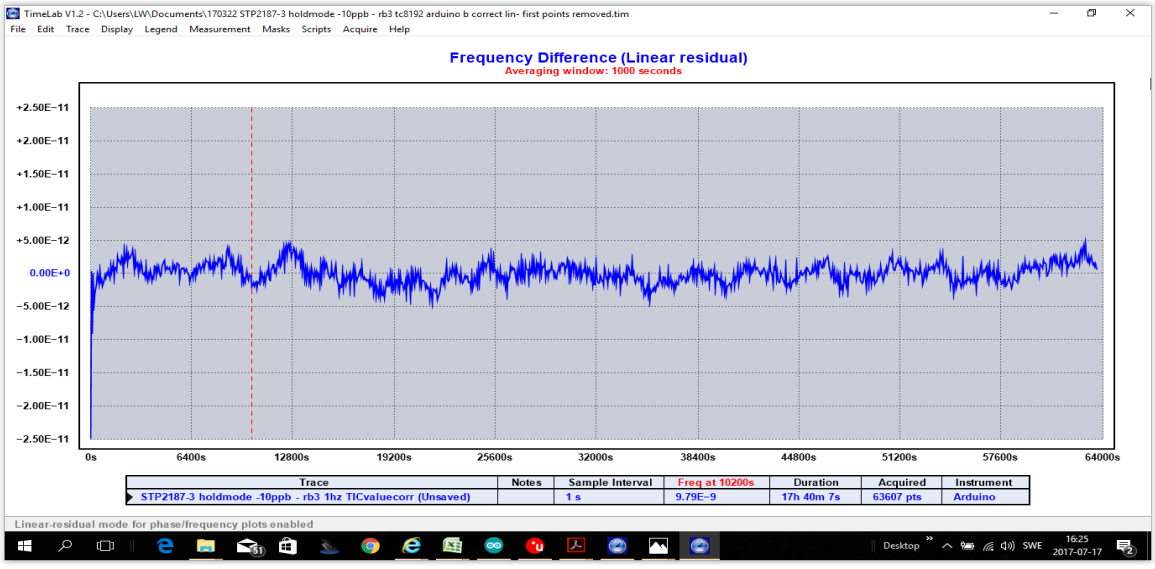


Figure 13. OCXO in holdmode measured with Arduino TIC (OCXO offset about 10ppb)