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.

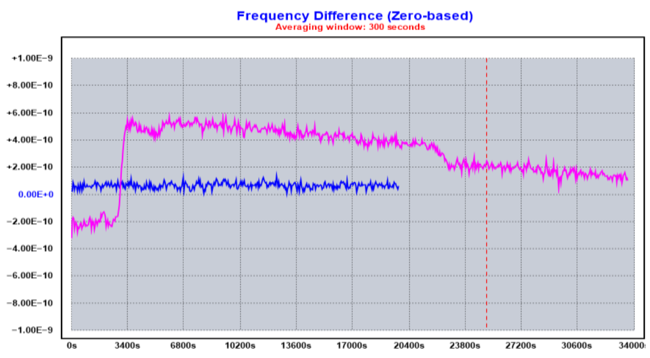
The output on HC4046 pin 15 shall be a pulse 0-1us, once a second. If not check the inputs to the HC4046. Remember that the 1PPS from the GPS module is often just a short pulse in the microsecond or millisecond range. On the ADC0 input it shall be a corresponding ramp.

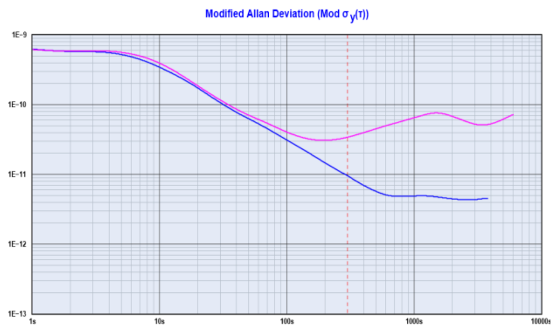
The PWM-DAC outputs can also be checked with an oscilloscope. Frequency is about 488Hz.

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.