An Improved GPS Disciplined Oscillator

Glen Overby, KC0IYT 11/2024

This is a GPS Disciplined Oscillator for synchronizing a 10MHz signal source with the timing signal from GPS. It uses a Atmel/Microchip SAMD-21 microcontroller to achieve this.

Background

This is a follow-on to my 2017 GPSDO project with an improved PCB and improved software. The software now locks a 10MHz oscillator such that when driving a synthesized oscillator at 5.7GHZ, the signal will be within 500HZ and locks up in less than 30 seconds from when the GPS gets a 3D lock and produces an accurate 1PPS signal.

My 2017 GPSDO project never worked well for me, primarily because the lock time was too long. I purchased a GPSDO made by a Chinese ham, BG7TBL, and was using it on my 10GHz ham radio station in the summer of 2023. I had a number of comments about the amount of spurious garbage on my RF signal at 10GHz, which resulted in my doing comparisons of driving a 10GHz synthesized oscillator with several different oscillators: the GPSDO, and several good quality ovenised oscillators (OCXOs). The difference was easily apparent: the OCXOs were much better. Even my own GPSDO turned the output from a good OCXO into one that generated spurs. I needed something better.

There are other commercial GPSDOs available, probably for less than I spent on this project, but I heard some bad signals on the radio from people who were using them. I decided to make improvements to my own GPSDO (in retrospect, this may have been due to their power supply).

Hardware

My Version 1 GPSDO project used a TI MSP432 microcontroller. There were some compromises, like not being able to read the GPS serial stream directly, due to pins being shared between the microcontroller's peripherals. The development environment, based on Eclipse, ran on Windows and I prefer to use Linux. So for a GPSDO revision, a change in microcontroller was on the table. I had purchased some SparkFun SAMD21 based Arduino boards, which use a 30 MHz ARM chip with a large set of peripherals, for a model rocket project. It's completely overkill for the task at hand. However, the MegaAVRs weren't quite up to the task of counting a 10 MHz clock. The SparkFun SAMD-21 Arduio board costs about $20, and I might make 2 or 3 of them, my time is a bigger factor than the cost of the chip.

I prototyped a new GPSDO using an unused GPSDO v1 PCB with good results, so I proceeded

with the project.

One of the lessons I learned with the version 1 was to have separate voltage regulators for different parts of the circuit. In particular, the GPS needs to not share a regulator with the microcontroller. My V1 GPSDO had a 1HZ noise pulse that was remedied by moving the GPS module to it's own regulator.

Another lesson in power domains was turning on/off LEDs affected the PWM output voltage enough to cause a noticeable sawtooth pattern on the output signal at 10GHz. This problem did not repeat during this project, even though the LEDs are driven by the microcontroller. This may be due, in part, to using the microcontroller to sink current (the ground side of LEDs is connected to the chip) instead of source it. The one exception to this is the red/green LED for the GPS lock.

Another discovery I made while working with the prototype - that might explain problems with v1 - was how much better the resulting oscillator's signal was when driven from the SAMD21's DAC as compared to a PWM signal with filters. The MSP430 only had a PWM, while the SAMD21 has a real DAC (I assume it's based on resistors). Even though the SAMD21 had a DAC on it, I started out using a PWM DAC because I believed I needed more than 10 bits of resolution to control the OCXO. That's only partly correct; the number of bits required depends greatly on what OCXO is being used. The PCB has a footprint for a MCP 4725 12-bit DAC chip, in case I ever need a DAC with more resolution.

The Arduino environment supports using the SAMD-21 USB hardware as a client, making it look like a serial port to the host. This means I only had to use one hardware serial port to listen to the GPS for lock signal (the SAMD-21 has multiple hardware serial ports).

Inputs from the oscillator, GPS, etc. are buffered using a 74HCT125 chip. The HCT technology is safe to use with +5V inputs when the chip is powered by 3.3v.

The serial port connections use a 6-pin header compatible with the headers used on many Arduino boards and many USB to Serial adapters. There is one header to connect to the GPS receiver (with the 1PPS signal taking over the DTR pin). A second header is present to connect a USB-to-Serial adapter to the GPS receiver.

A header is provided for connections to the oscillator. It includes power for the voltage control op-amp and a signal for when the oscillator has reached operating temperature (both of those were inspired by the Isotemp OCXOs).

There are many jumper points on the board:

> JP1 selects the CPU DAC or PWM pins. JP7 connects the output pin of the external DAC, if used (KiCad doesn't have a 3-way jumper). This signal goes to an RC filter then to the first half of op-amp U1.
>
> JP2 selects the input of the second half of op-amp U1. Options are the output of the first half of U1 or the signal selected by JP1/JP7.
>
> JP3 selects which op-amp section goes to the OCXO control.

JP5 selects where the op-amp power comes from: either the 3.3v rail or the Oscillator's control voltage.

JP4, JP5 and JP8 are on the outputs of the voltage regulators. This allows testing the voltage regulator without powering any circuitry, and measuring current draw.

The first half of op-amp U1 is a simple input follower[2] after filter R1/C2. The second half has component footprints (R3, R4, R5, R6, RV2) to make it a summing amplifier[1]. This was intended to provide an offset voltage for use with OCXOs that require a control voltage higher than 3.3v. Alternatively, it can be a non-inverting amplifier[1] with a gain greater than 1 by placing resistors at R5, R6.

The 10MHz clock source is terminated by RV1, a 5K ohm variable resistor. This resistor is used to adjust the voltage level seen by the 74HCT125 input, and is the only tuning the board requires. Adjust the resistor so the 10MHz signal appears on the output of U2, pin 8. I needed an oscilloscope to get this setting right.

Software

I had many ideas for how to improve the GPS lock software. The software is a PID controller (PID controllers are described in detail in various places on the the internet; I won't try to go into them here). The major problem I needed to address was the time it takes to lock the oscillator frequency to the GPS timing signal.

My V1 software always averaged multiple 1PPS clocks before operating on them. This was to remove jitter in the GPS signal (or maybe in the receiver chip). The jitter usually exhibited itself as one clock that was long or short, with the next clock compensating.
I decided that the P controller, which is doing only a rough lock to 1PPS, can operate well even with a little bit of jitter from the GPS since the P controller has a small error band. This version does that, and by operating every second, it locks the oscillator much faster than the old version.

For the I controller, I continue to average several 1PPS clocks but do so using a moving average. This allows running the I controller every second. I discovered that I needed to use double precision floating point to have enough bits for an average of 10,000,000 samples and including a fraction (single precision, 32 bits, used all the bits for the integer portion of the number). It's good that I picked a microcontroller that was a lot of overkill! I've since revisited this decision and am now tracking the error (how far from 10,000,000 the actual count is) and single precision floating point works well.

The software starts as a P controller then switches to an I controller once P is stable "enough". It will switch back to the P controller if an adjustment needs to be too large.

I found that after making adjustments in the I controller, I really needed to flush out the accumulated error. If I didn't, the I controller would swing from too fast to too slow, never (or rarely) stopping in the middle.

Configuring the SAMD21 is quite complicated. I borrowed code from the Arduino forum for

configuring a counter with an external clock and a capture-and-hold. The choice in what counter and sample and hold determined what pins the 10MHz signal and the PPS signal were connected. Similarly, the code for initializing the PWM DAC was borrowed from the Arduino wiring_analog library (which I actually call once during configuration so that I didn't need to duplicate even more code).

Once the board has been assembled an the OCXO connected,

I've used this to control several OCXOs:

- CTi OSC5A2B02
- Isotemp 134-10
- FOX 801

Lessons Learned

There were a few problems encountered along the way:

I configured A0, shared with the DAC output, as a digital output. This caused a problem where the DAC wouldn't swing all the way to 3.3v due to the digital signal pulling it down. I left the initialization in the code, commented out, along with a warning to my future self not to use it.

Whenever the counter for the 10MHz clock wraps at 2**32, it reports 32 extra clocks. I don't know where it gets them. It's not good to interject a glitch every 429 seconds, so I check for outliers and throw away bad samples. Just in case something goes awry, there is a counter: if 10 outliers in a row happen, the code kicks back to the P controller (that actually happened when a test clip came off).

While the PI controllers need to operate based on the 1PPS GPS clock, I needed status LEDs to update even when that clock isn't present (e.g. to update the GPS lock LED to RED!). The loop() function uses the Arduino millis() call to get the time, and updates LEDs about every 2 seconds. The heartbeat update will be noticeably slower when this is happening.

The normal program output, on the USB serial port, only shows status when changes are made to the OCXO control signal. Debugging can be turned on to increase the amount of chatter. All messages from the GPS can be echoed to that serial port as well. Normally only the $GPRMC signal is passed (which can be used to set a computer's clock).

The second board I built didn't include the 10K trim pot on the I found that oscillators weren't behaving themselves. After adding it, things improved significantly. I think the oscillators didn't like the high impedance input and need the ~5k ohms of termination.

I'm using a resistive power splitter on the oscillator output, but I think I need to add a MMIC on each side of the splitter both for isolation and to bring the power level up. I have several of the CTi oscillators, and the ones without the GPSDO are cleaner than the ones on the GPSDO.

The FOX 801 really needs more than a 10 bit DAC. I used those with my V1 GPSDO which is

probably why I started this project thinking that I couldn't use the 10-bit DAC.

I've found that having a GPS with a separate antenna is a lot more useful than having the antenna integrated with the GPSDO box. With an external antenna, I can mount the GPSDO wherever it works best, without requiring that the whole thing having a sky view.

While I've put the GPSDO in a plastic box, I've been keeping the oscillator in a separate metal box.

Future

Some things that could use improving for future versions:

I have a simple RC filter on the control output, along with op-amps to buffer it. I should try using a salen-key low pass filter with one of the op-amps.

I designed the op-amp circuit to allow adding a constant voltage (Isotemp OCXOs use about 8v for the voltage control), but when I tried using that OCXO I actually used a 2x amplification.

Provide some tap points for DC power and ground on the PCB – probably a 2nd 2-pin header right next to the first one. This would have made it easier to build a version that is integrated with the OCXO.

The header for the OCXO should be expanded to include a pin for power TO the OCXO, and a jumper to connect it to +12v power coming onto the board.

Label the GPS connectors better. There are two 6-pin headers (compatible with readily available USB to serial boards used on Arduinos): one connects to the GPS, another to a USB-to-Serial adapter in case I ever wish to program the GPS. It's easy to forget which is which, and which way to connect the cable.

There may be better ways to buffer the input 10MHz signal. A similar project by VE2ZAZ published in QEX[3] used an LTC1485 differential bus transceiver.

References

[1] National Semiconductor AN-31 "Op Amp Circuit Collection", February 1978. Non-Inverting Summing Amplifier

[2] National Semiconductor LMC6482 "CMOS Dual Rail-To-Rail Input and Output Operational Amplifier" data sheet, November 1977, Figure 10.

[3] "A Simplified GPS-Derived Frequency Standard", Bertrand Zauhar VE2ZAZ QEX Sep 2006 pp 14 https://webpubs.arrl.org/pubs_archive/113782