

The KE0FF 10MHz GPS Disciplined Oscillator

By Joseph Haas, ke0ff

8/18/2013

joeh-at-rollanet-dot-org

In 1985, I built a frequency counter kit as part of a national electronics competition. The design was based on an article in the now defunct *73* magazine and was centered on an LS7030 frequency counter chip. While the project cut some corners for cost and simplicity, the result was a surprisingly useful piece of test equipment. Until recently, it was my only frequency counter, and I still regard it as a useful addition to my test equipment suite.

One of the corner-cuts was that the 10MHz reference oscillator was very simple. The design in the *73* magazine article went to great lengths to stabilize the 10MHz reference oscillator, but the contest project offered no temperature leveling or compensation. Recently, I decided to improve the accuracy of the instrument, so I added a temperature compensated oscillator that I had happened across at a good price.

While this worked well, I was intrigued at the prospect of having a reference slaved to GPS. The promise of these instruments is that they closely approach atomic standards, but require much less overhead and cost. Most importantly, a GPS slaved oscillator can be (somewhat) self-monitored so that the accuracy can be established and reported in real time. There are limits to the accuracy, of course, but as I discovered, 100 ppb or better accuracy can be achieved with relative ease.

Essentially, the effort involves constructing a PLL circuit that locks a local VCO to the timing output of a GPS receiver. Of course, the choice of GPS is not a trivial one. Some of the first amateur designs for a disciplined oscillator were based on the Rockwell/Conextant Jupiter GPS receiver. This receiver features a 10 KHz reference output. While most GPS receivers provide a 1Hz reference output, locking a local VCO to a 10 KHz reference is simpler, and can be achieved with a handful of CMOS logic devices and a decent quality VCO.

A 1Hz reference requires more logic to lock a 10MHz reference, and has a much longer loop filter time constant to get a first-order lock. This is generally not a significant issue except that at long time constants, the discharge path for the loop filter starts to dominate. This includes stray circuit resistances and the input resistance to the VCO. At integration rates of 1 second, the degree of discharge in the loop filter can approach 1 ppm for a 3.3V VCO input.

The difference between 10 KHz and 1 Hz references represents a trade-off scenario. The choice between the two boils down to the following:

- Overall accuracy
- Circuit complexity
- Availability of components

While the 10KHz reference offers simplicity, the timing accuracy is generally not as good as with some 1 Hz options. In the case of the Jupiter GPS, it is specified to maintain between 300 μ s and 1 ms of timing accuracy (while the timing version of these receivers offers better accuracy, they can be more difficult to find). This equates to a frequency error of about ± 3 KHz for a 10 MHz timebase, or about 300 ppm (assuming a PPS accuracy of 300 μ s). On the other hand, the Trimble Resolution-T GPS provides a ± 20 ns timing accuracy for the PPS output which results in an accuracy of about ± 0.2 Hz for

a 10 MHz timebase, or about 20 ppb. The Resolution-T also offers a quantization error value that is updated for each PPS pulse. This error value can be used to refine the phase error detector result to obtain accuracies that approach the thermal limit of the VCO stability. In truth, the PPS accuracy limitations of non-timing GPS receivers can be mitigated by averaging the phase error results over several minutes. This has the result of filtering the phase jitter induced by the PPS accuracy allowing a better time-base lock than might be obtained otherwise.

Finally, while the 1Hz PLL is more complicated to implement, this can be mitigated by using a medium scale CPLD to contain the majority of the PLL logic. The addition of a microcontroller also simplifies the hardware design. While the microcontroller adds software to the mix, the complexity is still low due to the slow process time – this allows the use of low-cost microcontrollers to accomplish the task. For the Resolution-T, the microcontroller is required anyway since the fine timing alignment needed to improve on the 500 ppb base-accuracy level is only available via serial connection to the GPS.

What is all this “Parts per whatever stuff, anyhow?”

For those not familiar with the ppm (parts-per-million) and ppb (parts-per-billion) terms, this short discussion might help. Percent is essentially parts-per-hundred and is employed quite often in everyday life. You take a quantity, divide it by the maximum quantity for a given situation, and multiply by 100 to get percent. ppm and ppb work the same way, only for ppm, multiply by 1,000,000 (10^6) and for ppb, multiply by 1,000,000,000 (10^9). To convert from ppm or ppb, the process is simply reversed. For a 10 MHz oscillator, 20 ppb relates to frequency error by the following: $10 \text{ MHz} * 20 \text{ ppb} / 10^9 = 0.2 \text{ Hz}$. Generally, “parts-per” values are listed as numbers greater than 1, although this isn’t a hard rule. Thus, a 0.2 ppm value is the same as 200 ppb, but the ppb form is preferred since it relates a root value greater than 1.

A Basic PLL

The PLL needed for this project is not terribly complicated and follows the basic PLL architecture for digital divide-by-N PLLs. Other than the fact that it features a fixed N division ratio, it has no other special requirements. As such, a commercial PLL chip could be employed to accomplish the task. However, I decided to use a DAC to drive the VCO which meant that I would need to digitize the output of the phase comparator so that it could be integrated and converted to DAC settings. I decided to use a CPLD to create the N divider and phase comparator if only to have an excuse to create a CPLD design. I had Lattice tools already and had toyed with some Mach4000 parts in the not too distant past, so there wasn’t much learning curve involved.

In the middle of the CPLD design, I decided that I wanted to measure the phase error digitally using the CPLD rather than trying to digitize the analog output of the phase comparator with a pulse-width or A/D measurement. The CPLD had sufficient logic available, so it made sense. There wasn’t enough logic to allow serial output of the phase error, so that had to be accomplished in parallel form. The CPLD was getting pin-challenged, so the error value was broken up into two halves to reduce the pin count needed. The result is the schematics of Figures A1 through A3. Figure A1 shows the N divider and a phase-frequency comparator. Figure A2 is the phase error accumulator (PEA). Figure A3 is the UART routing switch logic.

The phase-frequency comparator is only partially used. While “dead-code” is generally not a good idea for software or hardware, I left it in place since it might come in handy at some point and having it there already would mean I could keep the CPLD programming tools at bay.

The reset circuit features a synchronizer so that the N divider is readily synchronized to the reference input from the GPS to start a phase measurement. This significantly reduces lock times when the system is in acquisition mode and allows phase error measurements to remain within the ± 511 range limit of the PEA counter.

The PEA is simply a gated counter that measures the pulse width of the phase comparator (EOR) output. A sign detection circuit produces a logic “1” or “0” depending on which rising edge occurs first, the N divider or the GPS reference. The counter is 9 bits wide and can be clocked by either the VCO or by the microcontroller SYSCLK. By choosing SYSCLK to be a higher frequency than the VCO, this allows for greater phase error resolution when the system is stabilized. With the VCO as the PEA clock source, each PEA word gives ± 255 ppm of error range, with about 500ppb/lb resolution. With an 80 MHz clock source, the range decreases to about ± 64 ppb of error range, with about 12.5 ppb/lb resolution.

The PLL thus constructed consists of the fixed-N divider ($N = 10E6$) and PEA counter. The microcontroller reads the PEA when RDY goes low and uses the signed error value to determine how to adjust the VCO control voltage to drive the VCO towards the GPS reference. The current architecture takes a PEA reading, then resets the PEA counter. This requires another PPS cycle to synchronize the PEA to the PPS signal and results in a new PEA value every 2 seconds. However, another approach would be to allow the PEA to run for several seconds to get better resolution. Since each PEA reading will have 1 lb of uncertainty, adding up multiple readings that are close to zero adds up the uncertainty which greatly dilutes the average value that results. If the per-PPS PEA is less than about 10, the register can accommodate up to 50 seconds of run time.

Taking the resulting PEA divided by the run time results in a single LSB of uncertainty whereas taking 25 readings over the same time period results in 25 lbs of uncertainty. Currently, the PEA is sampled 10 times over a 20 second period. It was relatively easy to modify the code to allow the PEA to run for 20 seconds, then take a single PEA reading and dividing by 20 to get an average PEA value with a greater level of certainty.

The VCO

There are a number of VCOs available that could be used for this project. I chose a FOX VCO based mostly on cost and availability. The jitter performance was reasonable, but it is not temperature compensated. I was originally unconcerned with temperature drift thinking that the device will be operated in a lab environment with reasonable temperature consistency and the PLL would compensate for any drift that might crop up. However, after considering this further, a long loop filter time constant could prevent the PLL from keeping up with short term drift. I had already constructed the VCO components, but decided that I could oven-ize them relatively easily with a PTC thermistor-heater and some insulating material. I located a 40°C device and constructed a heat-spreader and clip to hold it in place. I also decided to add an I2C temperature sensor to the VCO so that I could have the microcontroller monitor the system to determine when thermal equilibrium had been reached.

Figure A4 illustrates the overall schematic of the main clock circuit card. The VCO consists of a precision voltage reference, a 12-bit i2c DAC, and a VCXO. The FOX device features a 50ppm/V V_c input that is centered at 1.65V ($V_{cmax} = 3.3V$). While the V_c input is not linear over the entire 0-3.3V swing, it is rather linear over a $\pm 1V$ swing centered at 1.65V. For the DAC side, 1 LSb is about 806 μ V which equates to about 40 ppb/lb.

The Enclosure

I used a Bud chassis and a prototype PWB that I had previously designed for the SiLabs C8051F360 microcontroller. I wanted to have a distribution output panel and decided to provide individual control switches for each output. This would allow the outputs to be disabled when not used rather than terminating them with terminators. I fabricated a module for the switches and distribution amplifiers and another module for the front-panel status LEDs and COM port selection switch. The chassis fit the form-factor of my contest-frequency counter rather well, so I simply configured it to attach to the top of the counter. This was the prime destination for the reference anyway, and would allow me to piggy-back on the power infrastructure of the counter, saving some power supply overhead. Figures A5 and A6 illustrate the chassis configuration.

The Diversion

After much assembly work, I finally had the major pieces ready for integration testing. As a simple and quick test, I simply coupled the counter's reference oscillator (the newly minted GPS reference) back to the counter input. It dutifully displayed 10MHz. However, after a few minutes, the reading began to INCREASE! The amount of offset was considerable, reaching as much as 14 MHz after 15 minutes. Not only was this behavior disturbing, it was also very perplexing since the counter should have read 10MHz no matter what the reference oscillator frequency might be. Even if it was unstable, it should always read exactly 10MHz when connected in this fashion.

I observed the reference and noted that it wasn't particularly remarkable. In the process of probing the circuit, I also discovered that capacitive loading on the input to the frequency counter prescaler would correct the issue. Furthermore, freeze spray on the prescaler would temporarily correct the problem. I wondered why I had never seen this problem before, but then realized that I had never performed this test before.

I was at a loss as to how to proceed. The prescaler, an Intersil 11C90, went out of production many years ago with no clear replacements to choose from. If this wasn't a defective prescaler (and I was not yet convinced that it was defective), that wouldn't fix the problem anyway. Still, I was able to locate some 11C90 devices and purchased 4 of them just in case. They were located in China, so it would take a few days to get here. In the meantime, I decided to work on the Frequency counter.

I had already started to modify the counter to add a TCXO and replace the power switch so making changes was not an esoteric problem. After leaving the counter on for a few hours, I was struck by how warm it became. Of course, this mostly due to the linear power supplies and the transformer from the original design. I had already decided to use a small switcher for the GPSlave but now decided that I would also replace the original transformer-rectifier-linear supply for the counter with a 120VAC to 12VDC switcher and a 5V buck-switcher. I also wanted to add a pre-amplifier to the low-Z input

(hoping that this would both improve the sensitivity and cure the wonky count issue) and some other improvements. The overall list soon grew to the following:

- Upgrade the power supply – the GPSlave power should always be on, the counter supply is controlled by the front-panel power switch.
- Replace the fixed power cord with a Euro-DIN power connection – Hard-wired power cords are a real pet-peeve of mine.
- Configure the internal connections so that the different panel sections can be completely separated for re-work or repair.
- Replace the front-panel BNC jacks with rear-mount bulkhead connectors. This allows the circuit cards to be separated from the front-panel without de-soldering the connectors. This also greatly improves the match and shielding for VHF/UHF signals.
- Convert the reference output jack to rear-mount bulkhead crimp connector (better shielding).
- Clean the 30 year-old rosin-flux residue from the circuit cards.
- Add an auxiliary rear-panel bulkhead connector for the 10MHz TCXO that will be otherwise unused.
- Add an MMIC pre-amp to the low-Z counter input.

After completing the pre-amp, the original frequency error was no longer observed. A check of the sensitivity yielded a figure of about 55mV (about -12 dBm). While not screaming, 55mV is a useable figure of merit and further efforts to improve it in the near-future are unlikely. This allows the GPSlave effort to resume with the next step being to replace the 12 bit DAC with a 16 bit device.

I had originally decided that 12 bits would provide sufficient resolution, but the early integration tests showed that a 12 bit DAC would require the system to dither by a considerable amount. The added resolution of the 16 bit DAC would reduce the impact of the dithering and would be easier to accomplish now before the insulation cube was constructed for the VCO/DAC circuit.

Once this is complete, the last hardware bits are the distribution panel, RS-232 connection cable, and painting the outside of the enclosure flat-black to help reduce thermal transfer out of the enclosure.

The First Full-Up testing

The first tests with the assembly in its final configuration were promising. I was able to plot the VCO temperature vs. time with and without the 40C PTC heater. I also soldered a type K thermocouple to one of the edge metallization terminals of the VCO to get a comparison of the VCO case temperature against the temperature sensor that is attached to the top of the VCO.

The VCO temperature sensor is an I2C type in an SOT package and was attached to the VCO with a best-compromise approach. I fabricated a small die of 0.024" thick FR-4 and carved a circuit pattern to accept the SOT-23-6 package of the sensor. By removing the minimum copper from the die, I hoped to minimize the thermal resistance between the bottom of the FR-4 and the sensor die. The FR-4 die was then epoxied to the top of the VCO and the power and I2C connections were attached.

Comparisons of instantaneous measurements of the VCO case and top sensor showed a difference of about 1°C with the heater off, and about 2.5°C with the heater on. This could be due to a calibration

difference or actual temperature gradient. A test with the VCO power removed would be required to establish the calibration differential. It is likely that the gradient dominates, however.

The first overnight plot of VCO temperature vs. time shown in Figure 1 illustrates a cyclic variation of about $\pm 0.3^{\circ}\text{C}$ with a period of about 25 minutes. I was perplexed by this phenomenon, but couldn't think of any likely suspects. The second overnight test, Figure 2, exhibited the same cyclic variation, but during this test, I was manually taking data from the K-thermocouple (since currently I don't have a way to automate this instrument) and was able to correlate the likely source of the variation. I noticed that my HVAC system (currently in cool mode) was triggering a slight reduction in the temperature readings. I noted the period of the HVAC system activation and was able to closely correlate the period of the two systems.

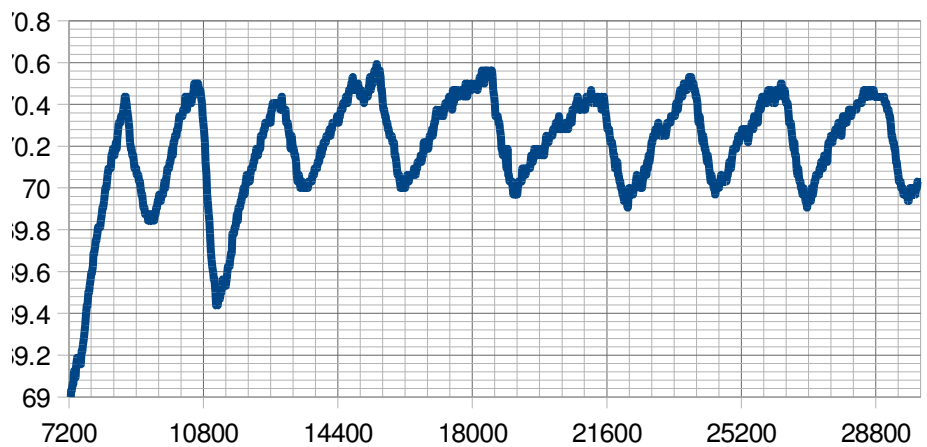


Figure 1. VCO Temperature vs. time (heater = ON).

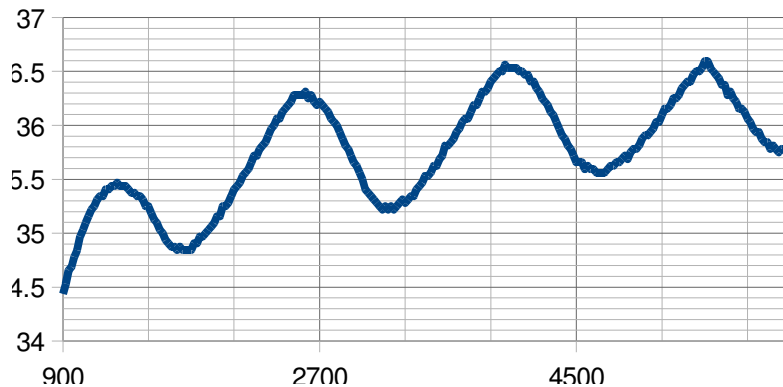


Figure 2. VCO Temperature vs. time (heater = OFF).

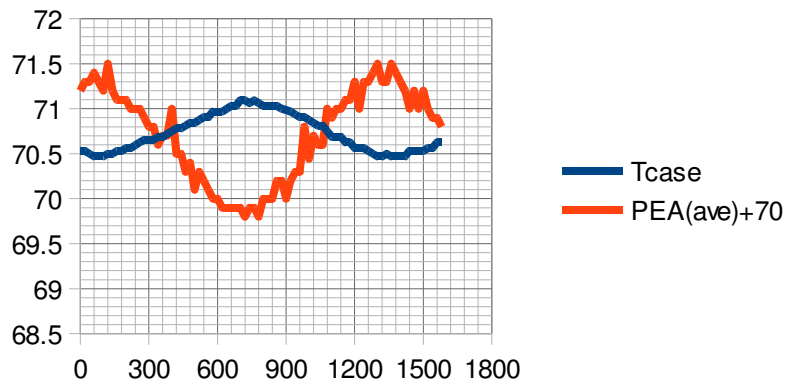


Figure 3. Case Temperature and phase error vs. time (heater = ON).

Figure 3 illustrates the interaction of the VCO output with the GPS 1PPS (a fixed value of 70 was added to the PEA reading to allow the two plots to share the same axis). As the plot shows, the VCO frequency variation vs. the PPS output shows a change of about 1.5 PEA units (about 150 ppb) of variation over the temperature excursion provided by the HVAC operation. The external temperature variation was measured at about 3°C (max to min) which indicates that the approximate tempco of the system is in the 50 ppb/°C range. There is about a 30 to 60 second lag between the VCO sensor reading and the PEA effect, which will limit the effectiveness of any temperature compensation efforts when the external temperature is changing quickly. Also, the system will need to be characterized at around 15°C and 35°C to get better numbers for the thermodynamic behavior at these temperatures. It is clear from these measurements that any effort to reach the 20 ppb base-accuracy limit of the PPS output will require either improvements to the VCO system insulation, a comprehensive temperature compensation algorithm, or a combination of the two.

Initial Software Configuration

The initial software effort was geared towards getting basic systems in place (UART, timers, I2C, etc...) and tested. This produced a very simple command line interface that allowed the DAC to be adjusted and the system parameters (PEA, DAC, and system variables) to be displayed periodically. While this allowed the control loop to be manually maintained, it did not provide for any automatic adjustments. With the basic systems in place, it was time to consider how to architect the control loop.

I am a big fan of state machines. While I concede that I probably over-use them, they allow multi-thread control with relatively simple coding structures. With this approach in mind, I envisioned two or three state machines to drive the primary control loop, as well as the temperature compensation loop and the GPS serial communications loop.

The UART control provides for 3 distinct UART configurations, $\mu\text{C} \leftrightarrow \text{PC}$, $\text{GPS} \leftrightarrow \text{PC}$, and $\mu\text{C} \leftrightarrow \text{GPS}$. These three UART modes are cycled by pressing the pushbutton on the front panel. After considering the mechanics of gathering data from the GPS and sending log data to the PC, I decided to add a fourth mode (called "auto") that would handle the situation where the system could access both

devices automatically. After much consternation and gnashing of teeth, I was able to get the UART switch to behave.

It was at this point that I began to re-think the modes. I wasn't convinced that I would implement the quantization error correction algorithm, but that also implies that I wasn't giving it up either. In order to implement the algorithm, the system would have to continuously capture GPS data to be sure that the quantization error value would be captured for each PEA value captured. Since the log output to the PC was considered an important function, a direct conflict is the result: The system needs to constantly receive GPS data, and periodically send log data to a PC. Fortunately, there are two aspects that mitigate this conflict: 1) The two resources are opposite in direction of data flow. 2) The CPLD handles the UART switching function and can be easily modified to include a state where the μC -GPS connection and the μC -PC connections are simultaneously established. The primary drawback to this is that it will no longer be possible to enter user commands while in the "auto" mode. This is mitigated by the fact that the front panel switch will still allow the user to switch to μC -PC mode to issue user commands.

The last requirement for the mode switching is to establish a 5 s delay for the GPS-PC mode. This allows the modes to be cycled without having to encounter bursts of GPS data as the mode passes through the GPS-PC state.

Advanced Software Configuration

Once the basic resources were established, the more advanced loop structures could be addressed. My first effort was to establish a simple tracking loop for the VCO control voltage. By averaging PEA values and applying a simple proportional feedback equation, it was possible to get better than 1 ppm phase-lock within a few minutes of GPS acquisition. A two-level approach was used to apply the average over a shorter period when the phase error was over 5 ppm out (100 s was used) and a longer time average, 600 s, when the phase error was better than about 3 ppm. This worked well, but was plagued by the temperature-dependence of the VCO circuit. If the phase lock is able to compensate for fast temperature changes, then it is running on a time constant that is too fast to smooth any GPS discontinuities.

I captured some long baseline data which allowed me to relate the DAC setting (as driven by the proportional loop) to VCO temperature. I was able to plot a temperature coefficient line vs. temperature. This showed a non-linear relationship of the slope of temperature vs. DAC. I also logged with the PTC heater turned off and noticed that the crystal T/C curve must peak somewhere between 38°C and 68°C. In the 70°C range, the changes in DAC setting oppose small changes in temperature – if temperature rises, the DAC setting falls. However, in the 38°C range, the DAC follows the changes in temperature. This is interesting in that it suggests that any open-loop compensation implementation will only work over a narrow temperature range.

I have waffled over the compensation question a number of times. On the one hand, it looks like one should be able to apply a compensation scheme that will at least minimize temperature effects and be reasonably simple to implement. However, on the other hand, these compensation schemes are open-loop and thus are bound to fail in the face of aging effects. "Why not use PID algorithms?" Unfortunately, PID systems are closed loop by definition and this control problem is not closed loop IN

TERMS OF TEMPERATURE. Thus, without a way to measure the frequency of the VCO and use that as a process variable into the PID temperature loop, one can not use PID techniques to improve the temperature coefficient of the system. Using the GPS PPS as a frequency reference could allow one to implement a PID algorithm, but this limits one to the short term stability limit of the PPS signal, which will vary based on the constellation configuration and GPS transient events.

I have located a TCVCXO that appears to be a better VCO candidate. The cost is only \$20 and would likely fit into the space occupied by the existing VCO. This new device in concert with the PTC heater (doubling the heaters by adding one on top of the VCO would also improve thermal stability) would greatly improve the temperature performance of the VCO sub-system making software compensation unnecessary. This would allow the software to focus on the long-time-constant averaging of the GPS PPS signal.



Figure 4. (Nearly) Finished GPS Disciplined Timebase and VICA contest Frequency Counter.

References

Confessions of a Counter Evolutionary (part 1), D.N. Ellis, WA2FPT, 73, August, 1982

Confessions of a Counter Evolutionary (part 2), D.N. Ellis, WA2FPT, 73, September, 1982

A GPS-Based Frequency Standard, Brooks Shera, W5OJM, *QST*, July 1998

GPS Disciplined 10 MHz Oscillator, <http://www.jrmiller.demon.co.uk/projects/freqstd/frqstd.htm>, James Miller, G3RUH (web site observed on 8/2/2013)

Appendix A: Figures

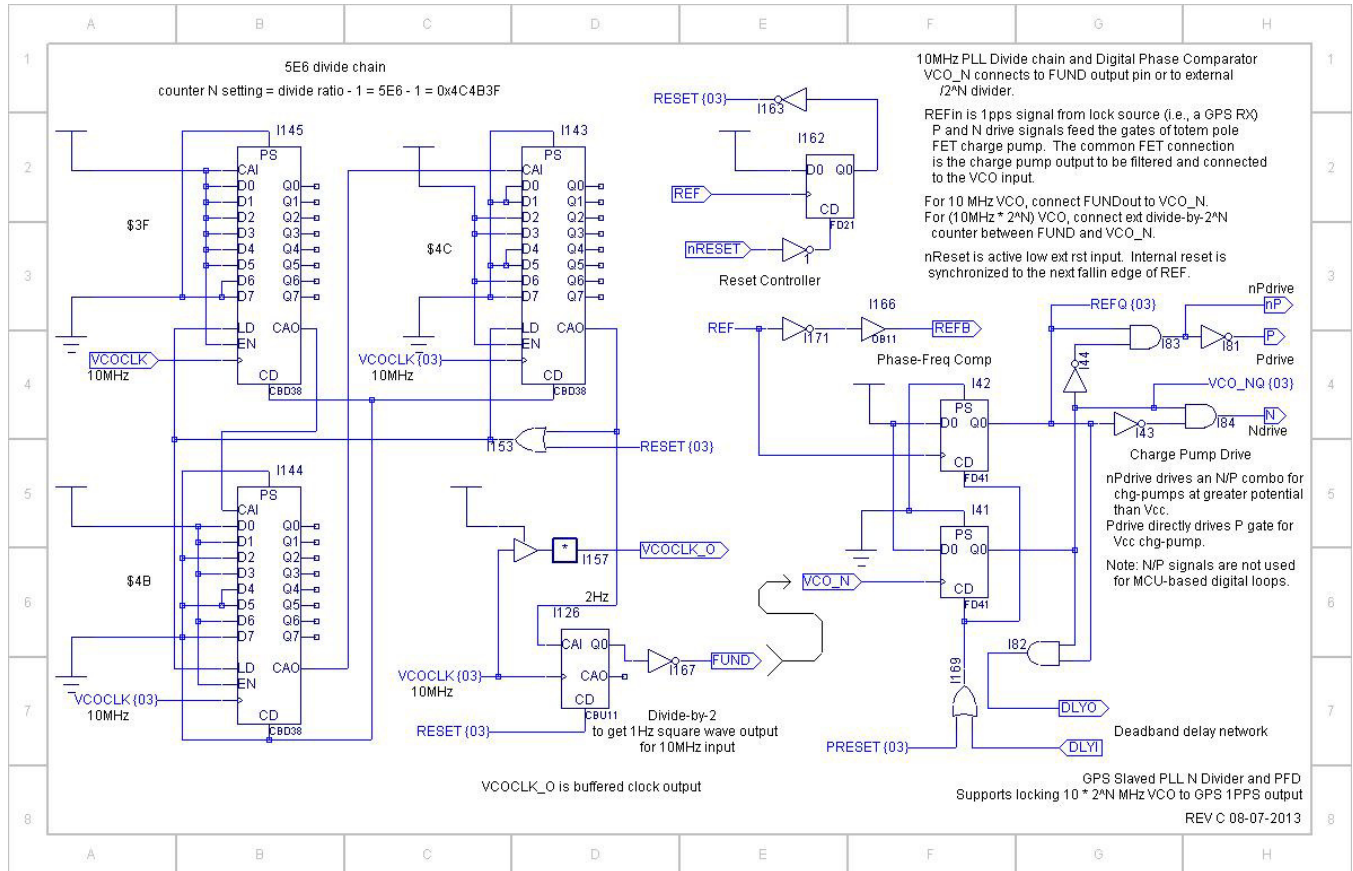


Figure A1. CPLD N-divider, reset circuit, and frequency detector

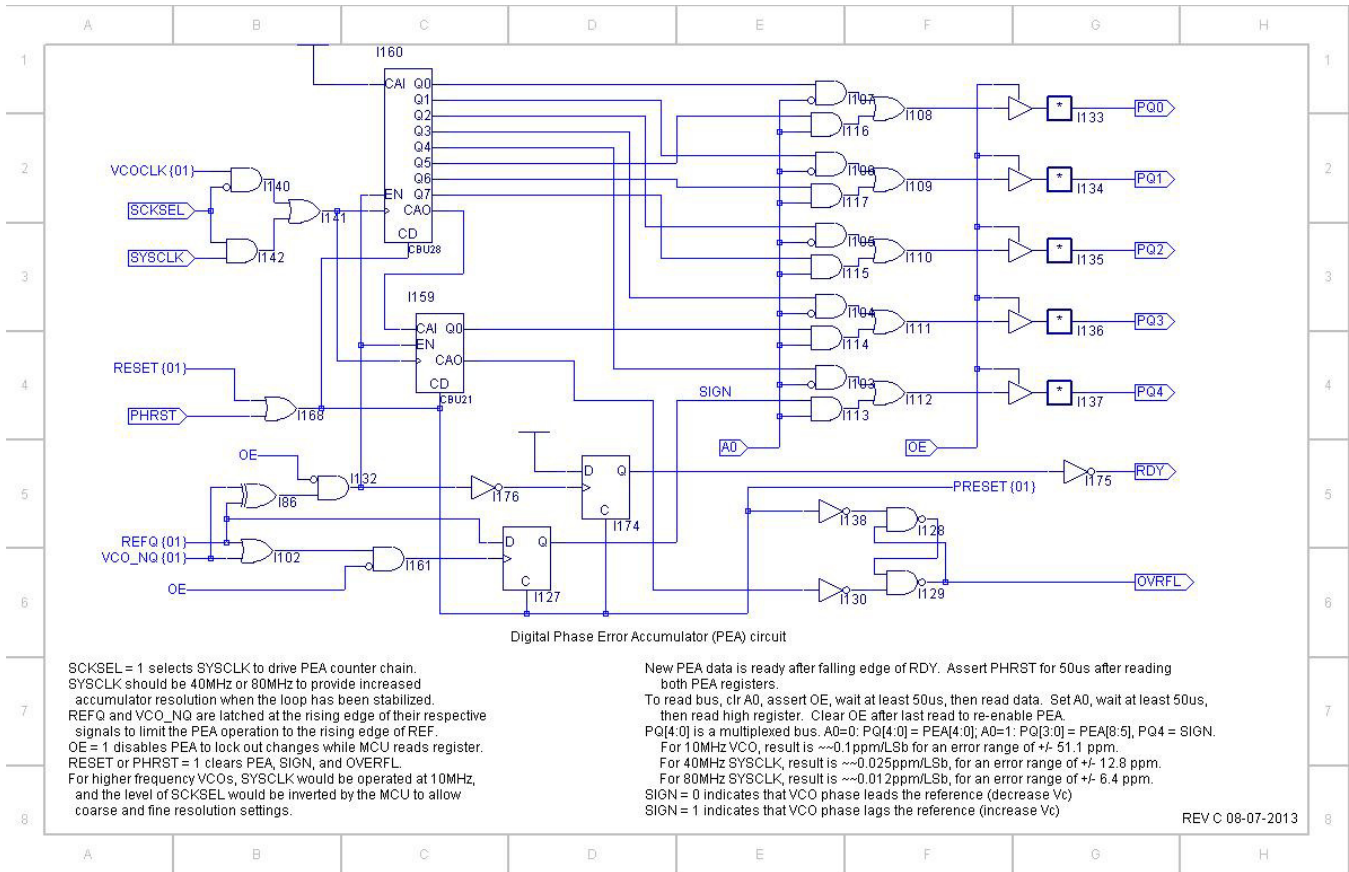


Figure A2. CPLD Phase Error Accumulator (PEA) and processor interface.

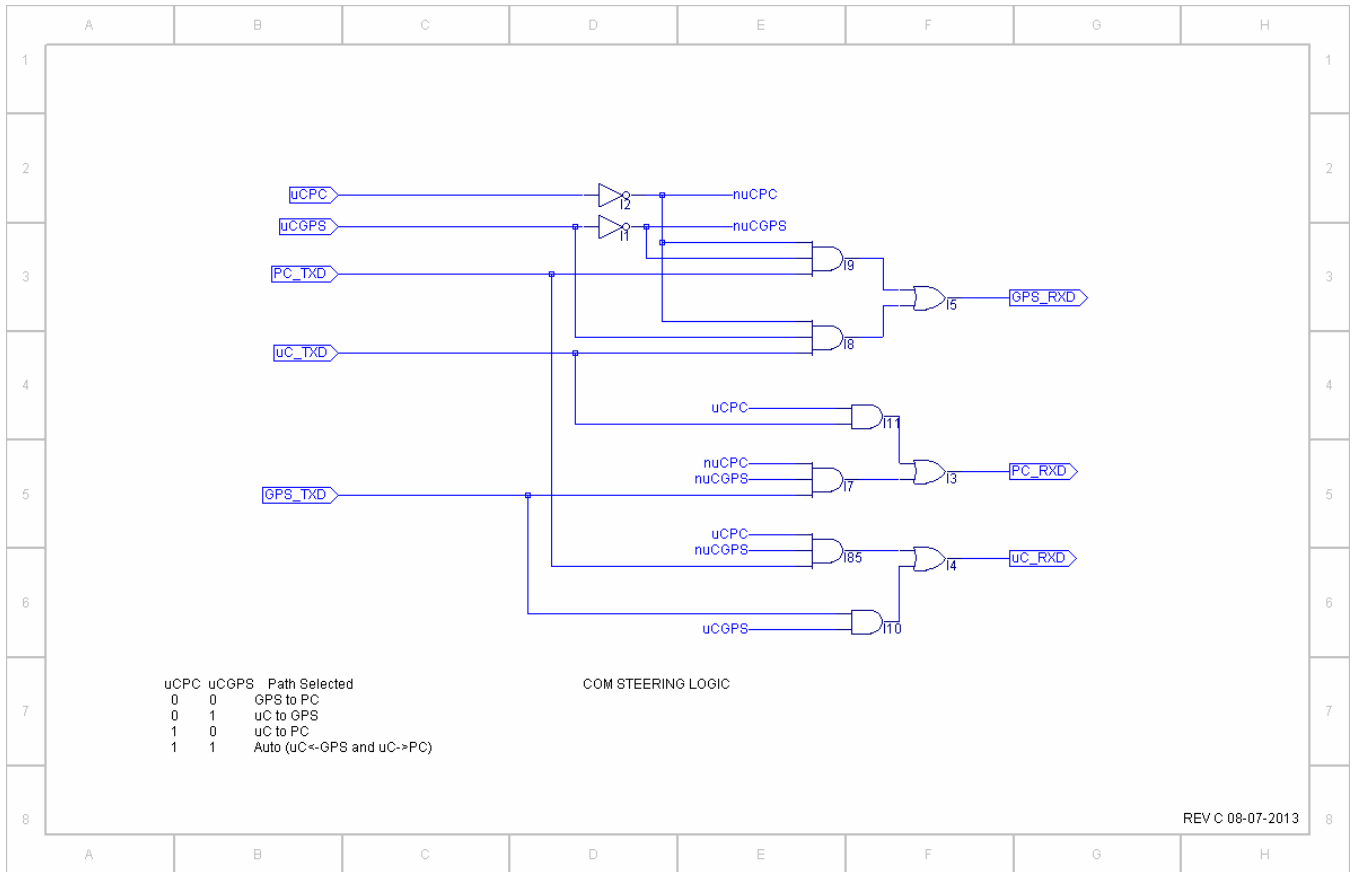
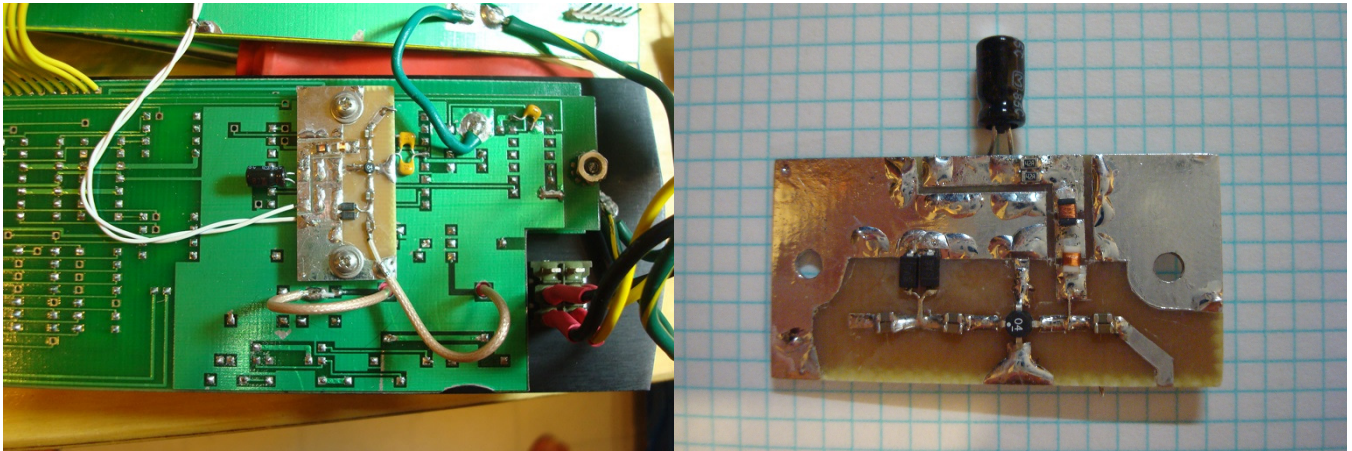
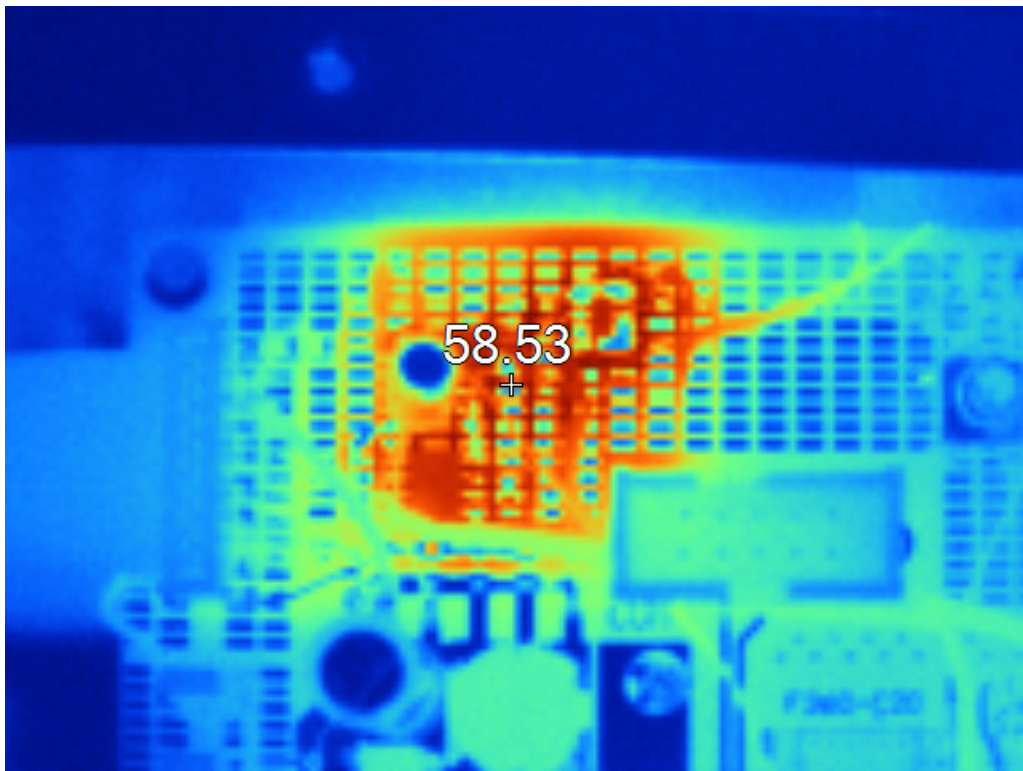


Figure A3. CPLD UART data switch logic.

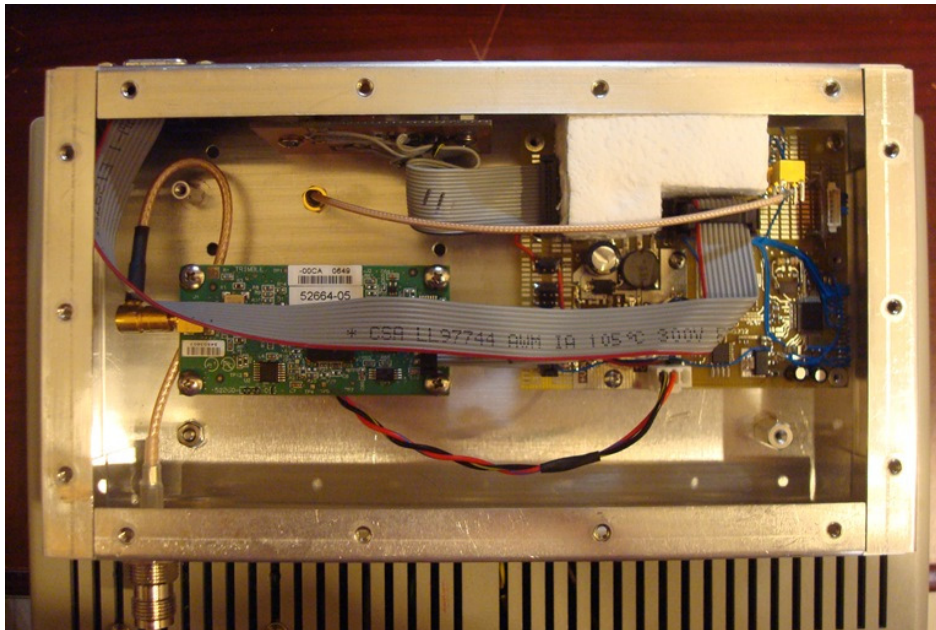
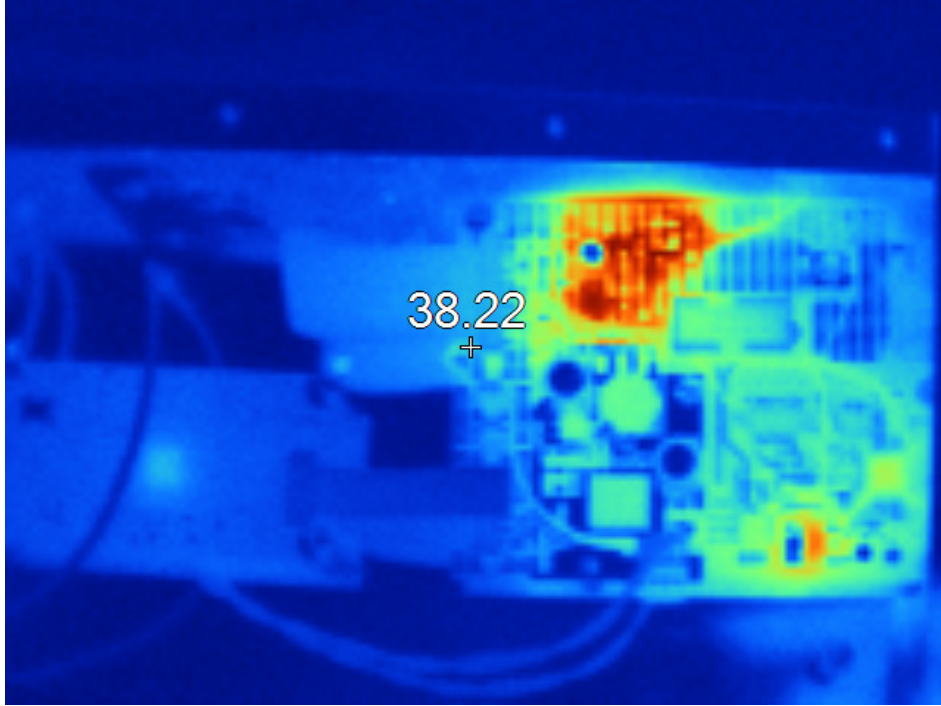
Photo Gallery



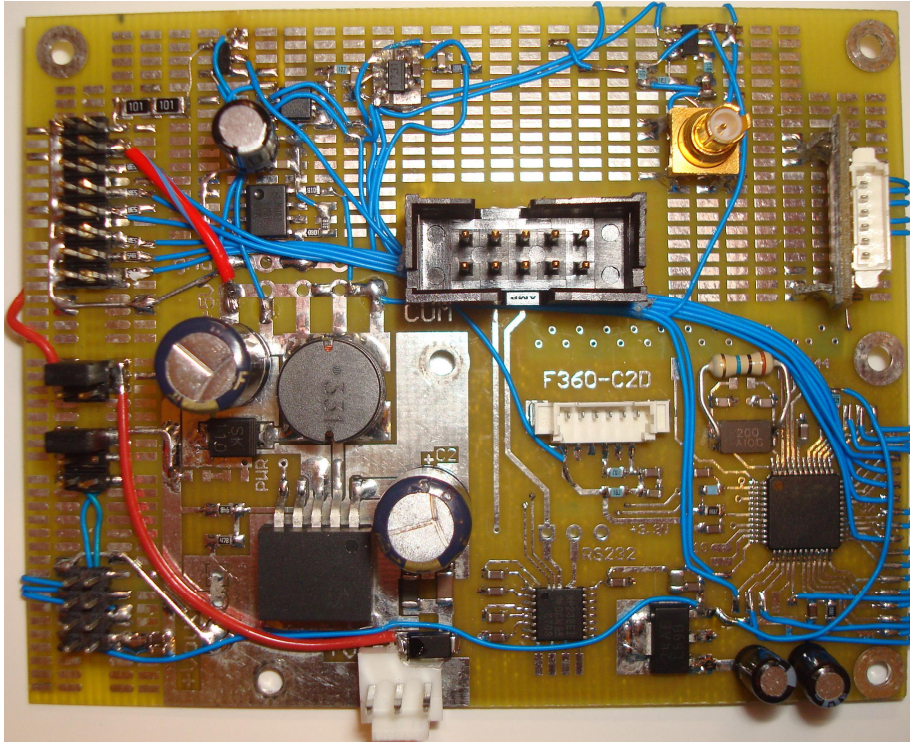
Front-end amplifier for Frequency Counter



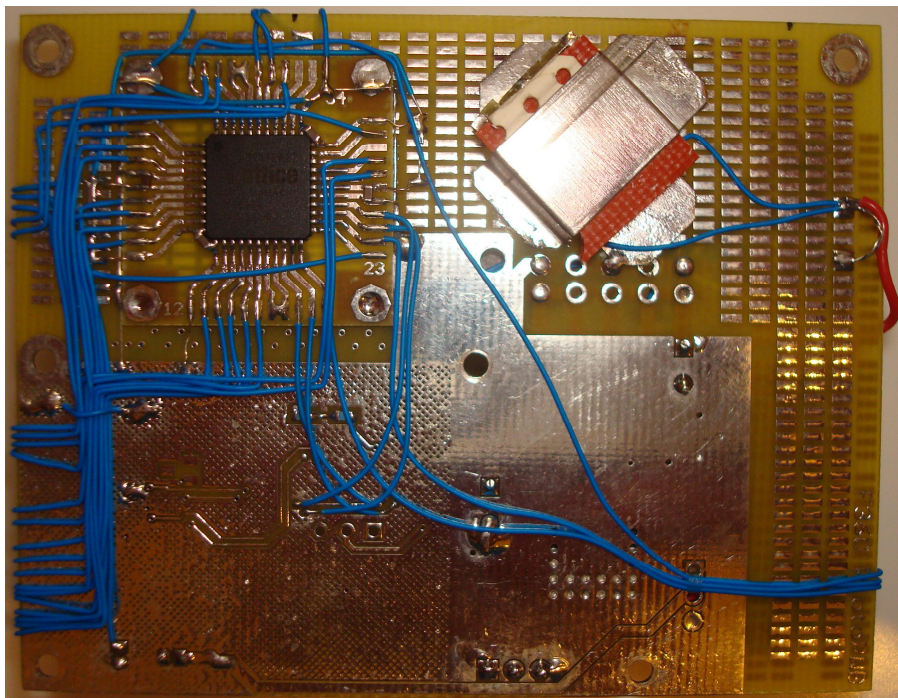
Thermal image of DAC/VCO with insulator removed



Thermal and visible image of GPS slave assembly
(In thermal image, note reflection of power cable, and others, from bottom of chassis)
(Insulator, serial cable, and 10MHz out cable all removed for thermal image)



Controller CCA top view



Controller CCA bottom view (CPLD and PTC heater shown)