A GPS Disciplined Oscillator

Glen Overby, KC0IYT
gpoverby@gmail.com

Introduction

When operating on the microwave bands, I've often needed to have oscillators that are both stable and accurate. These don't always come as one package. Combining the N5AC apolLO synthesized oscillator (also sold by Down East Microwave as the A32), with a good ovenized oscillator, has given me good stability when operating on bands up to 10ghz. I desired an oscillator that is accurate as well as stable. One approach for adding stability is to synchronize an oscillator with the timing signal output from a GPS receiver. This paper presents my experience of using a microcontroller to do the synchronization.

This project started out as a derivative of the article by VE2ZAZ in the October 2006 QEX. The author has a web site describing his project at: http://ve2zaz.net/GPS_Std/GPS_Std.htm. During the adventure of assembling the hardware and writing the software, I've arrived at a somewhat different device.

Theory of Operation

This project replaces a hardware phase-locked loop (PLL), which is frequently used for disciplining oscillators to GPS, with a microcontroller and software.

At the center of this project is a microcontroller which counts clock cycles from the 10MHz oscillator over the time period determined by the GPS one-pulse-per-second(PPS) signal, and the compares that count to the expected count of 10 million. The microcontroller calculates the error and adjusts the oscillator's voltage control input to synchronize it with the GPS signal.

I discovered that the simple approach of adjusting every 1PPS signal had a problem: the GPS signal (or the GPS receiver) has some jitter -- the GPS' 1PPS signal is not accurate to nanoseconds on every clock pulse. I saw this the first first time I put this project into operation: the software would see the oscillator clock count being short one second, and long the next second. I experimented with longer sample times, but still saw similar results. In investigating how other GPS disciplined oscillators work, I discovered that commercial oscillators evaluate the GPS signal over several minutes to several hours. I have followed that approach, and the software on this device now operates on several seconds of data, or a minute of data, depending on what mode it is in.

In addition, I found other conditions which need to be handled:

- When an ovenized oscillator is warming up, its frequency drifts enough that it isn't beneficial to try controlling it. These oscillators typically have an output that can be monitored to show when it has warmed up. During this time, it isn't worth trying to discipline the oscillator to GPS, and trying to do so may leave the controller in a confused state.

- The GPS receiver's 1PPS signal may not be generated when the GPS receiver doesn't have lock (or before the first lock), and is likely to be inaccurate when the GPS receiver does not have a 3D lock. During this time, I don't want to change the oscillator's frequency.

Hardware

Shortly before starting this project, I purchased a TI MSP430 LaunchPad board and was looking for a project to use it for. This microcontroller turned out to be an excellent choice because it's peripherals have features that are ideally suited for this project. The microcontroller on this board, the MSP430G2553, has two counters which can count an external clock at the processor's clock speed. The processor can run from an internal clock at up to 16MHz, thus allowing it to count the 10MHz oscillator directly. The counter includes a sample-and-hold register, which is triggered by an external signal. This feature made the chip ideal for this project. Thus, the 10MHz clock is connected to the counter clock input and the GPS 1PPS signal to the counter's sample and hold trigger.

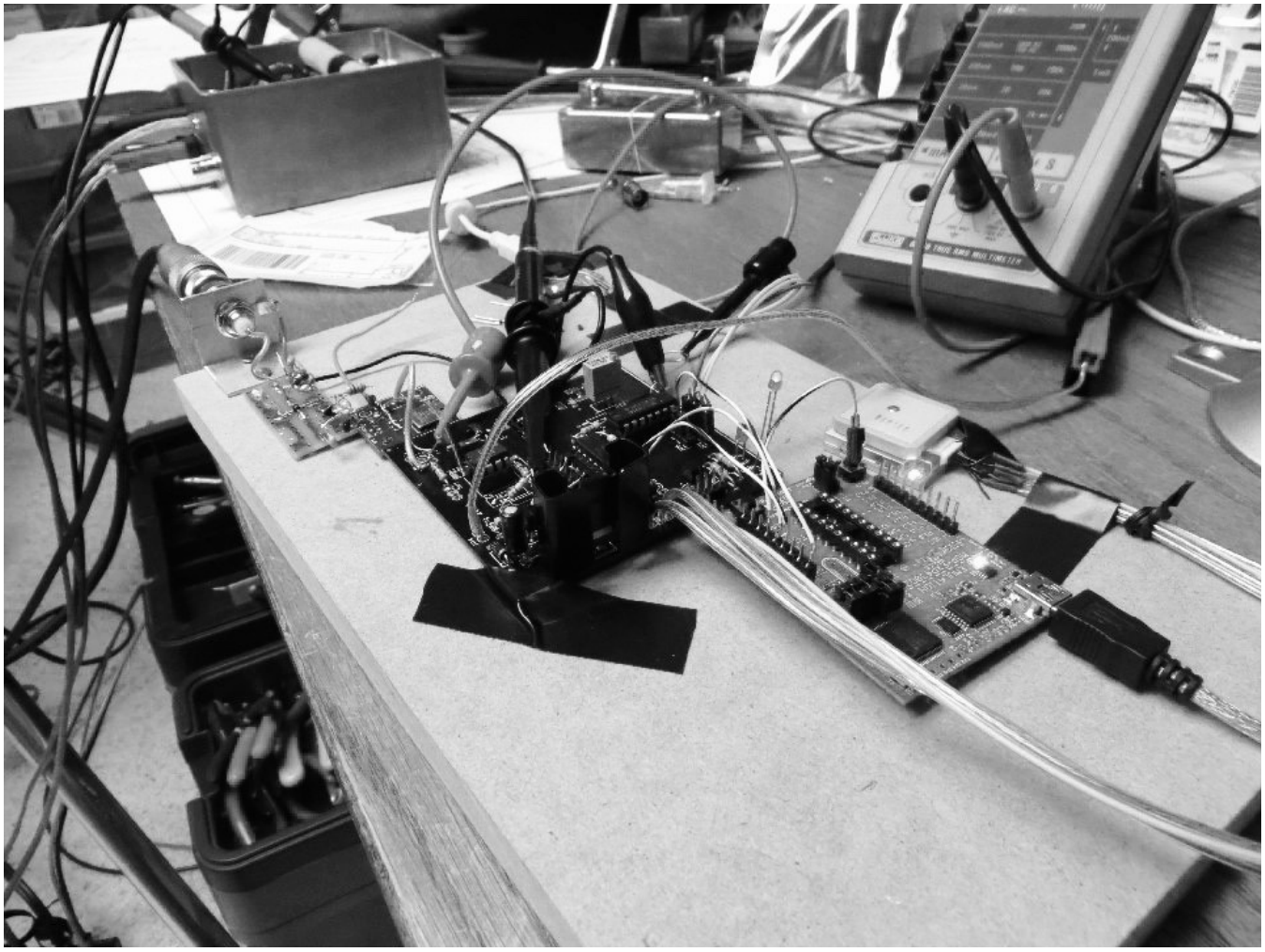
A second counter in the microcontroller is used to control the oscillator's frequency voltage control. The counter is configured to produce a pulse-width modulated (PWM) output signal, which is filtered to produce a voltage. I use this common technique to generate an analog signal from a microcontroller.

The chip's architects left a complication for me: the counter's sample-and hold has two sample inputs, but both of them use the same chip pins as the serial port! So I had to choose between serial input and serial output. I would like to monitor the GPS' serial stream to see when it has 3D lock, and thus the most accurate time, but the ability to debug and monitor the software using the serial output was more important. Instead, I use a separate microcontroller to monitor the GPS. Lock state comes back to this board as a TTL signal representing lock or no lock.

Circuit Board

My first circuit board required several changes, including the addition of a 'dead bug' chip, as seen in the photograph. The schematic and PCB shown here incorporate all of the lessons I learned while working on the first board. I plan to have boards made from this design in the future.

I designed the board using through-hole components to make assembly at home easier, except in a few places where there was no through-hole component available.



Board Inputs and Outputs

The circuit board has several headers for input and output:

- Oscillator Input
- Oscillator buffered output
- Oscillator digital output
- GPS
- GPS Serial access
- GPS Lock Detect
- Microcontroller Serial
- Power

The connections in each of these headers is listed in Appendix 1.

Circuit Description

This description references part numbers on the schematic.

Overview

The main control flows are:

- The 10MHz oscillator signal coming into the board is amplified, filtered and buffered before connecting to the counter input pin on the microcontroller.
- The GPS' 1PPS signal is buffered and connected to the counter's sample-and-hold trigger input.
- The output from a second counter is is filtered and connected to the oscillator's voltage control input.
- Signal inputs for the Oscillator heater and GPS Lock are buffered and connected to signal input pins.

I learned an interesting buffering technique during this project. 74HCT type logic chips are tolerant of 5V input signals when powered at 3.3V, and when powered at 5V will "see" a 3.3V high signal as a "1" input. The MSP430 microcontroller is a 3.3V device, and some of the inputs (such as the GPS) are 5V, and need conversion down to 3.3V. The LEDs are connected to a 5V power supply and are driven from the microcontroller.

The next sections will cover the inputs to the board and the outputs from it.

Inputs

The 10MHz signal enters the board and is amplified by a MAR-5 (or ERA-1) MMIC, U1. This is to isolate the oscillator from the on board circuitry and to amplify it. The output of the MMIC goes through a 5 pole low-pass filter to eliminate high-frequency harmonics.

I put a second MMIC and filter on the board to drive external devices. I always use a buffer on the output of oscillators, as many of them are load sensitive. That is, their frequency varies depending on the output load, requiring some settling time to come back on frequency after connecting or powering on the devices using their signal.

The 10MHz analog signal goes through a variable resistor, which pulls the signal up to a level that the 74HCT125 buffer will detect it as 1s and 0s.
The GPS' 1 Pulse Per second (1PPS) signal enters the board and goes through a 74HCT125 gate for logic level conversion from 5V to 3.3V.

The oscillator clock and 1PPS signals go to one of the microcontroller's counters, which will be used to measure the clock error.

An ovenized (heated) oscillator often has an output to indicate when the heater has heated up to it's operating temperature. The voltage of this output of an Isotemp oscillator is more than 5V, so it drives a 2N2222 transistor (Q2). If a heater signal is not available, the check will need to be disabled in software.

An input to indicate GPS lock, supplied by another microcontroller, goes through another 74HCT125 gate for voltage conversion from 5V to 3.3V.

Outputs

A Pulse-Width-Modulated output, or PWM, is used to provide an analog output signal. This is driven from a second counter in the microcontroller. A pulse-width modulated signal will, after filtering, have a higher voltage the longer the duty cycle of the signal is, or how long the signal is “on” (or “1”) compared to how long it is “off” (or “0”).

Texas Instruments describes this in Application Report SLAA116. The MSP430x2xx User's Guide describes the counter registers being configured.

This PWM signal is passed through a pair of 1hz low-pass filters which smooth out the pulse to a constant voltage (U3).

The microcontroller serial output port connects to a header and is status and diagnostics. I have several of the SparkFun Electronics DEV-09873 - FTDI Basic Breakout boards for connecting to USB ports.

There are three connectors for GPS interfacing:

- One to the GPS, which provides power, 1PPS signal from the GPS, and TTL level serial
- One to a computer to provide access to the GPS serial port
- One to a board that monitors the GPS serial output and returns it's lock state as a TTL signal

Several LEDs provide status output:

- Red LED is heartbeat of the software, which also serves as a power indicator
- Yellow is the status of the coarse oscillator lock
- Green is the status of the fine oscillator lock
- A Red/Green LED from the GPS lock board indicates if there is GPS 3D lock.

Software

I wrote the software in the "C" language, using TI's MPLAB development environment. I created the project as a CCS project and selected the microcontroller on the LauchPad board.

The software is written as a state machine, and operates one of two Proportional, Integral, Differential (PID) controllers. A PID controller is a control loop feedback mechanism used for process control. It can be mechanical, electrical or software. Those who are not familiar with PID controllers, which I was not at the start of this project, Wikipedia had a good introduction to them.

I found it was useful to have two control domains: a coarse mode that will quickly bring the oscillator close to the desired frequency, using short time samples from the GPS, and a fine mode that refines and maintains the frequency using GPS samples from a longer time period.

States

Check for Status

This is a state that looks for problems:

- The oscillator is cold
- No/Poor GPS lock
- No 1PPS signal from the GPS
- No 10MHz clock from the oscillator

This is the state the controller starts out in, and these values are checked constantly when in one of the PID controller states.

Problem states

These states wait for their respective problems to clear. When the problem clears, the state machine goes to the Coarse Lock state.

Coarse Lock

This is a PID controller using only the Proportional part. This is used to quickly synchronize to GPS 1PPS. A Proportional controller looks at the error (how far off of 10MHz the oscillator is) and changes the PWM's duty cycle based on the error. This is to bring the oscillator close to 10MHz as quickly as possible.

This state counts the oscillator clock for 8 seconds before acting to minimize jitter from the GPS 1PPS signal. The error value is calculated by subtracting the actual clock pulse count from what was expected (8,000,000), and is used to adjust the PWM's duty cycle. It also has an Error Band, where if the count is off by a small amount (currently 1 clock count, or 1/8,000,000) no adjustment is made.

After the error is within +/- 1 count of 10MHz for 5 samples (that is, 5 * 8 seconds) it switches to Fine Lock state.

Fine Lock

The fine lock uses a Proportional, Integral controller. The proportional (P) controller operates once every 60 seconds, while the Integral (I) controller looks at data over several minutes. The goal of fine lock mode is to keep adjustments small, preferably being made by the integral controller.

The proportional controller operates like the coarse lock: the error is used to adjust the PWM's duty cycle. However, the adjustment size used is much smaller to enable fine-tuning of the frequency. In addition, the the error value is saved in a circular list for use by the Integral controller.

The integral controller runs on the minute mark, and uses the sum of the errors from the past 10 minutes to determine if it should apply an additional adjustment to the PWM's duty cycle. Currently, adjustment is done only if there are more than 4 errors in 10 minutes.

If, during any one minute time, a large error (greater than 128 clocks) is detected the state goes back to Coarse Lock.

Configuration

Configuring the software requires choosing parameters for the PID controllers. Many books have been written about tuning PID controllers, and it can be an involved process. This is the technique that I came up with by trial and error.

The PID controller software needs to know how much to change the oscillator's frequency voltage control in order to change the oscillator's frequency by 1hz.

I wrote a program, called freq-find, that that runs on the board and uses it as a frequency counter to search for a PWM “pulse on time” value that sets the oscillator to a desired frequency. I search for three frequencies:

- 9,999,999
- 10,000,000
- 10,000,001

The program uses a binary search for the value in the PWM's counter “on time” register. This is a 16-bit value, and it measures the frequency for one minute per bit. The binary search starts with a counter “pulse on time” register value of 0, then sets the most significant bit to a 1. It measures the oscillator frequency, and if the desired frequency is greater than the measured frequency, it keeps the bit as a 1. Otherwise (if the measured frequency is higher than the desired frequency) it resets it to 0. The search then moves to the next most significant bit and does the same thing. After 16 measurements, the oscillator should be running at the desired frequency.

I get three output values which are register values at three frequencies. I calculate the difference between 9,999,999 and 10,000,000 then between 10,000,000 and 10,000,001 to see how much the pulse on time register must change to move the oscillator by 1hz. Guessing too high can cause the PID controller to oscillate around the desired frequency without ever finding the desired frequency, and these numbers usually aren't the same so I select the lower of the two numbers. Depending on the oscillator, using half of this value may work better, with the downside being a longer coarse lock time.

The program has hard-coded values used by two PID controllers:

- P_FACTOR_FAST (for the coarse lock)
- P_FACTOR_SLOW (for the fine lock)
- I_FACTOR_SLOW (for the fine lock)

The value determined above is set in the P_FACTOR_FAST definition. The fine lock's proportional (P) parameter should be much smaller, about 2% of the coarse parameter, and a fine lock Integral (I) parameter of about half of the fine P parameter seems about right.

Experience and Future Work

The first board I built is used to discipline a FOX 801 10MHz oscillator using a much simpler version of the software than is described here. It accomplishes bringing the oscillator close enough to 10MHz, and keeping it there, to use it to drive a marker (I use a DEMI Weak Signal Source). Having an accurate marker available allows me to tune my 10ghz transverter to the marker and measure where the transverter sees the marker, which I can then use to know where to find other stations. I get stability and accuracy in different packages that work well together and minimize how much gear is mounted together with the dish.

The second board I built disciplines an Isotemp 134-10 10MHz oscillator. During the building of this board, the software was redesigned. The finer control of this oscillator showed that the original algorithm would never find a good lock and a better solution was needed. It was during the building of this board that I began to understand the GPS signal jitter better: it really is accurate only when used over a long period of time. Thus the motivation for switching to 1 minute sample times and 10 minute observing of errors.

The most significant design problem I encountered was caused by driving the LEDs from the same voltage regulator as the microcontroller. I found turning on and off the LEDs would change the oscillator's frequency. I believe this was caused by a dip and rise in the 3.3V power rail used by the microcontroller and thus it's PWM output. The oscillator responded to the voltage change by changing it's frequency! I believe this voltage change was a few hundred micro-volts (and not detectable by my DVM), and I saw it only when driving a synthesized oscillator whose output was being watched on the 10ghz band. I solved this problem by driving the LEDs from the +5V regulator, through a 74HCT125 buffer.

A second version of the PCB has been designed, to fix these problems I found:

- I switched to using the Arduino standard pinout for all serial port headers
- Added more voltage regulators to improve noise isolation
- Stopped powering the 3.3V regulator from the output of the 5V regulator, also for improved noise isolation.
- A second MMIC buffer was added to the board, to integrate one that was built dead-bug style on another board.

I purchased a GPS disciplined oscillator designed by GB7TBL and compared the two by driving a weak signal source with one, and a 10ghz transverter with the other. They “locked” a few hertz away from each other and remained stable after locking.

Not discussed here is a project that provides a GPS lock indicator signal to this board. That project uses an Atmel AVR (similar to an Arduino) to watch the serial data stream from the GPS and provide a TTL signal for lock or no lock. It also provides a GPS lock indicator on an LED.

The software, schematic, and PCB designs are available from:
https://github.com/glenoverby/gpsdo

Appendix 1

Serial Ports

The serial ports all use 6-pin headers and the pinout used by Arduino boards. This allows plugging in USB-to-serial adapters made for Arduinos. I have several of the SparkFun Electronics *DEV-09873 - FTDI Basic Breakout* USB-to-serial adapters.

Oscillator

- 10MHz Input
- VFO Tune Output
- VFO Tune Voltage Input

GPS

This connects to a GPS module.

- GPS Serial input and output
- GPS 1PPS input
- +5V power for the GPS

GPS Serial Access

- Connects to the GPS serial port

GPS Lock Detect

This header is for connecting to a second microcontroller that monitors the GPS serial data for lock.

- GPS Serial Output
- Lock Detect Input
- +5V power

Debug

- Microcontroller Serial Output

Misc

- Power (6.5v - 12v)