Budget GPSDO

Preliminary Information

I was going to wait until this project is finished before making it public. However, the finish line keeps moving so consider this a report on work in progress.

Another GPSDO?

The motivation for this design was to make a really cheap, but still useful, GPS disciplined oscillator (GPSDO). So what is a *useful* GPSDO? It seems this depends a lot on who is answering the question, so it became a matter of deciding what was likely to be achieved with inexpensive parts.

After some calculation, the target became 10MHz ± 0.01Hz.

How Cheap?

A GPS Disciplined Oscillator at a minimum needs 4 components: A power supply, a GPS receiver, an Oscillator, and something between the two to provide discipline. The parts chosen:
Power: A 5V supply that most hobbyists may have already
GPS receiver: NEO-6M with antenna – less than $10 (Australian)
Oscillator: used CETC CTI OSC5A2B02 – less than $4 in quantity
Discipline: PIC16F1455 – less than $3

The PIC16F1455 was chosen because it was available, and I had previous programming experience with the PIC16F628A.

Design philosophy

I looked at this as a software design challenge. The usual hardware approach to synchronising to sources is to use a Phase Locked Loop (PLL) but this may have drawbacks that can be avoided in software.

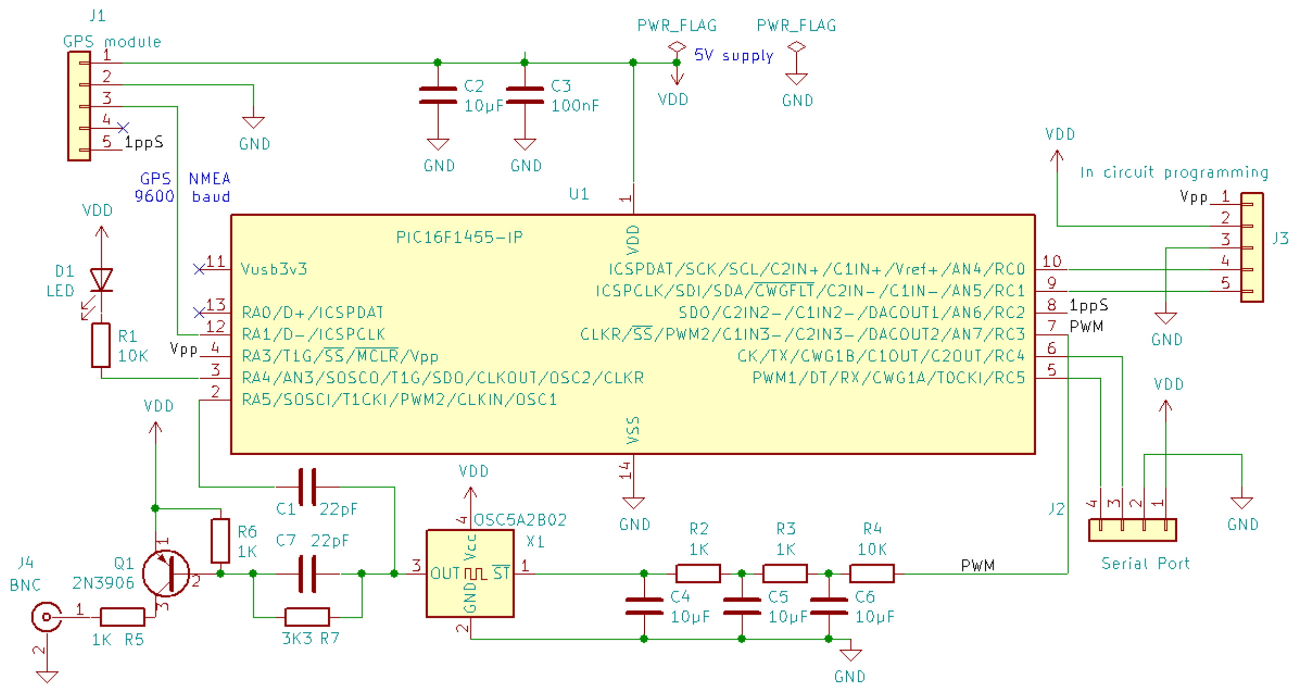
The ideal situation for a PLL is to have two stable signals, and manipulate one to align it with the other. A cheap GPS module isn’t really stable, it has jitter and can wander a bit when changing the satellite constellation. The solution when using a PLL is to have a GPS module that minimises these defects, and an accurate detector for the module output (usually 1 pulse per second – 1ppS) but this comes with an increase in cost.

The solution used here is to accumulate many 1ppS arrivals and work with averages. If there is predictable jitter, the detector does not need to accurately time each pulse. In the case of a cheap GPS, the jitter is caused by quantisation of the 1ppS due to the module clock not being aligned with the GPS signal. This should result in an even distribution of arrival times over the quantisation period. If the detection window is smaller than the jitter (in this case 25nS window and approximately 30nS jitter) then a small number will fall into a window earlier or later than most. If the number in the early window is the same as the number in the late window the statistically most likely average is the middle of the windows.

Errors are determined statistically, using a least squares linear interpolation.

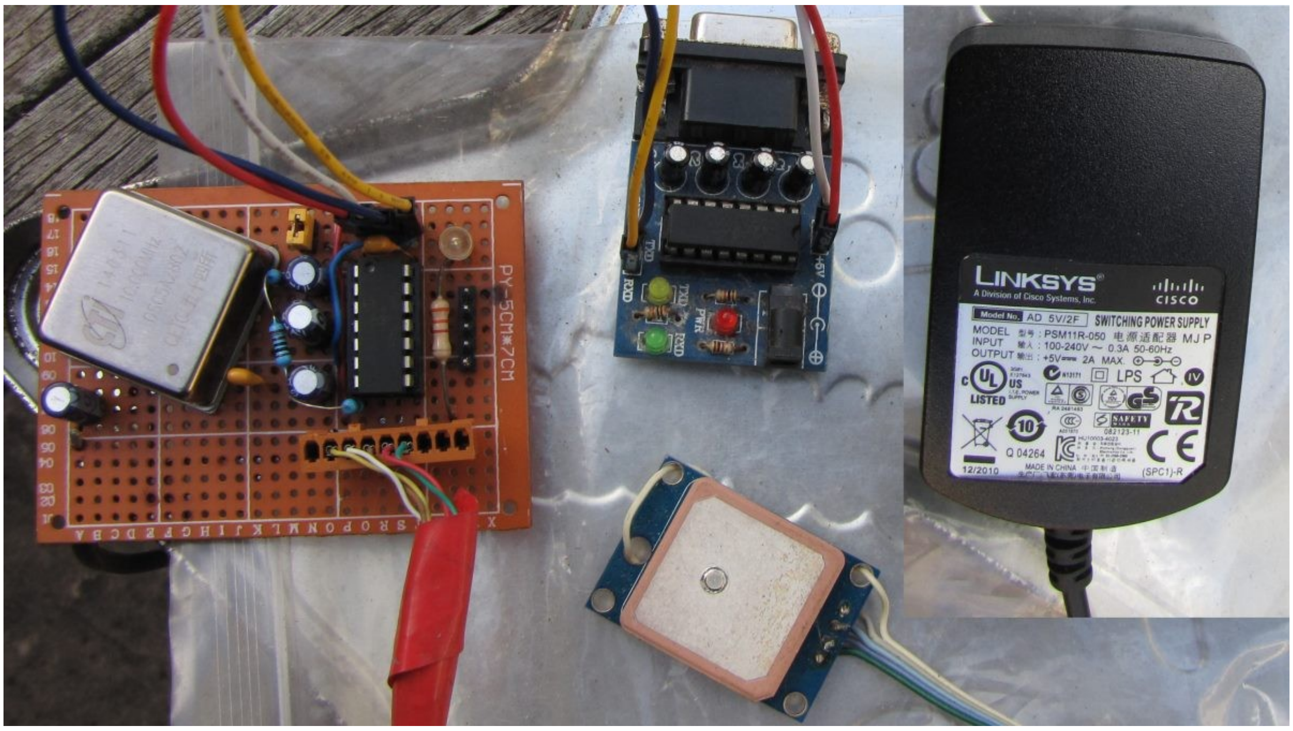
Another innovation is to use a single PWM with a dithered duty cycle and heavy filtering to emulate a precision Digital to Analog Converter (DAC).

The proposed circuit



The BNC output has not yet been implemented so is speculative.

The test rig at the moment looks like this:



The test rig is on the left. It is slightly different to the circuit. The LED resistor is 3K3 but is very bright so I put 10K on the circuit. The yellow 2 pin link was to isolate the processor when it is being programmed. This isn't needed. And there's no BNC output yet.

The board middle top is a TTL to RS232 adaptor that links to an old laptop with a serial port, for logging. I am using the Linksys wall wart to provide power. The GPS unit middle bottom is a few years old, bought as a spare for a clock project. The backdrop is the lid of a cake tin I use as a ground plane. It seems to make a small improvement.

The RS232 serial link is to an old laptop running Linux. Also tested was an FT232 TTL to USB converter linked to a Windows 10 desktop running PuTTY. This worked well but the laptop is more convenient.

How it functions

The processor is clocked by the oscillator so can directly compare the 1ppS to the nominal 10MHz without additional circuitry.

The PIC16F1455 can capture an event to the nearest 25nS by gating a timer. This is used to determine the arrival time of the 1ppS. The cheap GPS modules have considerable jitter in the 1ppS so it is not necessary to capture the arrival time more accurately.

The processor decodes the NMEA messages to determine if the 1ppS is from a valid fix. A problem with some GPS modules is they continue to provide a 1ppS even though they have no valid GPS data. The software can handle a few missed or invalid pulses.

Because the 1ppS has considerable jitter, the processor averages the arrival time of pulses for an extended period. The period is between 5 and 2048 seconds, depending on how well the oscillator is tracking the GPS.

If the oscillator and GPS have diverged by some number of cycles, the control voltage is changed to bring the two into syntony (I think that's the right term) again. The accepted number of cycles of divergence is reduces with longer averaging times, to reflect the increase in reliability from having more data points to average. For a period of 256 seconds, the accepted deviation is less than 2 cycles. This is a pivotal point. If the oscillator runs for 256 seconds and deviates less than 2 cycles, it has bettered the target of 10MHz ± 0.01Hz.

The maximum period the software can handle is 2048 seconds with a deviation of 1 cycle. It was not expected that this would be achieved in practice.

The control voltage is generated from a 10-bit PWM with a lot of filtering. The PWM pulse rate is 40KHz and the duty cycle can be varied in 25nS increments. This is used to emulate a 24-bit DAC by dithering the duty cycle on every pulse. The resultant voltage has increments of less than 1µV.

Calibration

A design goal was to have the system function without the need to use the serial output. The way the system works requires knowing the sensitivity of the oscillator to control voltage (e.g. for the OSC5A2B02 the sensitivity is approximately 0.1V/Hz). When the system is powered up, it looks for the information in non-volatile memory. If it is not there, the processor starts calibration mode to determine it.

First, the processor varies the control voltage from near the maximum specified for the oscillator to near the minimum. The processor will continue to swing the voltage between the limits until the oscillator runs fast at one extreme and slow at the other. If the oscillator can't be controlled, this will continue indefinitely.

Once the oscillator is able to be controlled, the maximum and minimum control volts are slowly converged until the oscillator runs 0.25Hz fast for one value, and 0.25Hz slow with the other. Each value is tested for 256 seconds, the result has to lie within limits to be accepted. The processor runs the tests until it gets 16 successful results. This takes about 3 hours so the oscillator is well warmed up. This assumes a solid GPS signal, it may take longer if the signal is weak and some results are rejected. The difference between the two results allows the sensitivity to be calculated. The average of the two results is used as the initial control volts for subsequent starts. The two values are stored in non-volatile memory and the system restarted to enter normal running.

Note the software should accept positive or negative control slopes, but has only been tested with oscillators of the same type.

The User Interface

The most interesting way to check the operation of the system is to monitor it via the serial port output. There is also a primitive command interface.

However, a design goal was to have the system be useful without using the interface. The LED provides a variety of signals to indicate status.

LED status indications

At startup, the LED should give a single flash to indicate it is powered. A continuous flashing at this point indicates the oscillator is not working and the PIC is running on its internal clock.

The GPS should send serial data as soon as it is powered up. The LED will then indicate satellites in view by flashing every two seconds. One long flash for none, a short flash for 1 to 3 satellites, 2 flashes for 4-7 satellites, and 3 flashes for more.

If serial data is not detected, after 10 seconds the system will either restart, or there will be a one second long flash. The system will restart if there is no calibration data. If there is calibration data the system continues to run using the starting control voltage. If calibration was recent, this is usually within ± 0.1Hz so may be useful.

The satellites in view flash continues every two seconds until the GPS reports it is providing a valid 1ppS. There is a brief pause then the LED flashes on every valid 1ppS received. For the first 5 or 10 minutes this is a double flash. When the system determines the oscillator is within 10MHz ± 0.01Hz this changes to a single flash. It may revert to a double flash on subsequent adjustments, but this is not common.

If there is no calibration data, after the GPS reports a valid GPS signal the system enters calibration mode. The LED flashes to indicate progress through the 16 tests. This is reported as 5 flashes over 5 seconds followed by a 3 second break. The flashes are a binary code, double flash is a 1, single flash is a zero. So if x represents a single flash and o represents no flash, the number 13 (binary 01101) would show as x xx xx x xx o o o over 8 seconds. The counts starts at 16 (xx x x x x o o o) and reduces to 1. Because the system has to converge on the ± 0.25Hz values it can take a long time, but less than an hour, before it changes from 16 to 15.

Serial port logging

This is a typical log from the serial port. Much of the basic information is in hexadecimal and the unit of measurement is 25nS offsets. Many values are signed so if they go negative they start with a 1 bit. If the first four bits are 1 (common) then the displayed letter is F.

```
Simple GPSDO started.
 Ctrl 2.0497282V Sensitivity 0.1051192 Volt/Hz  << from non-volatile memory
Receiving serial data from GPS.
Satellites in view: 09  << the GPS had a fix so only one second of data
GPS data is now valid.
Time Stamp: Date 210221 Time 070700 UTC.  << from the NMEA data
00 01|00  << the first value 00 is the sample collection period in powers of 2
00 02|01  << 00 = 2^0 = 1 – sample every second
00 03|01  << the second value 03 is the sample number starting from 0
00 04|01|01 0C  << the value between | is a 1ppS arrival time offset in 25nS increments
00 05|01|01 0C  << the value after | is the estimated average 1ppS error at this time
00 06|01|01 0C  << the second value after | is the error limit (12*25nS or 3 cycles)
00 07|01|01 0C
00|0000 0000 0100 0100 0100 0100 0100 0100  << averages (see below)
01 04|02 01|01 0B  << two values per sample, the error limit is reduced
01 05|02 02|02 0B
01 06|02 02|02 0B
01 07|02 03|02 0B
01|0000 0100 0100 0100 0180 0200 0200 0280
02 04|02 02 03 02|02 0A  << four values per sample
02 05|03 03 03 03|03 0A
02 06|03 04 04 04|03 0A
```

```
02 07|04 04 04 04|04 0A
02|0080 0100 01C0 0240 0240 0300 03C0 0400
03 04|05 05 04 05 05 06 05 05|05 09  << eight values per sample
03 05|06 05 06 06 06 06 06 07|06 09
03 06|06 06 07 06 07 07 07 07|07 09
03 07|07 08 08 08 08 08 08 08|08 09
03|00C0 0200 02A0 03E0 0500 0600 06A0 07E0
04 04|09 09 08 09 09 09 09 09 09 09 0A 0A 0A 0A 0A 0B|0A 08
<< The last collection the estimated error (0A) exceeded the limit (08) so a correction is applied to
the control voltage >>
 Time 070821 UTC. Ctrl 2.0564135 6.360 ppb
00 01|0A  << start collecting again
00 02|0A
00 03|0A
00 04|0B|0A 0C
00 05|0A|0A 0C 00B3  << the new value estimates how many samples at the current
00 06|0A|0A 0C 0096  << sampling rate until syntony
00 07|0A|09 0C 0096
00|0B00 0A00 0A00 0A00 0B00 0A00 0A00 0A00
01 04|0A 09|09 0B 0035  << sampling duration doubled, so estimate roughly half
01 05|09 09|09 0B 0028
01 06|09 09|08 0B 0028
01 07|09 09|08 0B 002C
01|0A80 0A00 0A80 0A00 0980 0900 0900 0900
02 04|08 08 08 08|07 0A 0013
02 05|08 08 08 08|07 0A 0015
02 06|07 07 07 07|06 0A 0013
02 07|07 07 07 06|06 0A 0014
02|0A40 0A40 0940 0900 0800 0800 0700 06C0
03 04|06 06 06 06 06 06 06 05|05 09 000A
03 05|05 05 04 04 05 05 04 04|04 09 000A
03 06|04 04 03 03 03 03 03 03|02 09 0009
03 07|03 02 02 02 02 02 02 02|01 09 0009
03|0A40 0920 0800 06E0 05E0 0480 0340 0220
04 04|02 01 01 01 01 00 01 01 01 00 00 00 00 FF FF FF|01 08 0005
<< it is calculated that syntony is achieved, so the control voltage is changed to try and
maintain it>>
 Time 070940 UTC. Ctrl 2.0525767 -3.650 ppb
00 01|FF
00 02|FF
00 03|FF
00 04|FF|01 0C
00 05|FF|01 0C
```

The averages line is output when the sampling period changes. Because there can be many readings in each sample, it has an integer part (the first 2 digits) and a fractional part (the second 2 digits).

Usually, only the control voltage changes are of interest. If the log is captured and filtered, it looks like this (recording after running for a day or two):

```
 Time 122416 UTC. Ctrl 2.0495204 0.045 ppb
 Time 154903 UTC. Ctrl 2.0494727 -0.045 ppb
 Time 162310 UTC. Ctrl 2.0495081 0.034 ppb
 Time 190933 UTC. Ctrl 2.0494477 -0.057 ppb
 Time 193924 UTC. Ctrl 2.0494944 0.044 ppb
 Time 231243 UTC. Ctrl 2.0495665 0.069 ppb
 Time 235106 UTC. Ctrl 2.0495210 -0.043 ppb
 Time 022025 UTC. Ctrl 2.0495842 0.060 ppb
 Time 030304 UTC. Ctrl 2.0495420 -0.040 ppb
 Time 072735 UTC. Ctrl 2.0496064 0.061 ppb
 Time 075726 UTC. Ctrl 2.0495530 -0.051 ppb
 Time 095237 UTC. Ctrl 2.0494917 -0.058 ppb
 Time 104348 UTC. Ctrl 2.0495207 0.028 ppb
 Time 135547 UTC. Ctrl 2.0495781 0.055 ppb
 Time 143410 UTC. Ctrl 2.0495326 -0.043 ppb
 Time 160801 UTC. Ctrl 2.0494782 -0.052 ppb
 Time 165456 UTC. Ctrl 2.0495115 0.032 ppb
 Time 181143 UTC. Ctrl 2.0494492 -0.059 ppb
 Time 190254 UTC. Ctrl 2.0494761 0.026 ppb
 Time 232725 UTC. Ctrl 2.0495579 0.078 ppb
 Time 235716 UTC. Ctrl 2.0494993 -0.056 ppb
 Time 055955 UTC. Ctrl 2.0494480 -0.049 ppb
 Time 063402 UTC. Ctrl 2.0494947 0.044 ppb
 Time 084617 UTC. Ctrl 2.0494370 -0.055 ppb
```

Serial Port commands

There are some primitive user commands. They are one character preceded by AT.

AT0 through AT8 control what is logged

AT8 copies the NMEA data stream to the serial output
AT0 stops logging
AT1 logs detail
AT2 logs averages
AT4 logs control voltage changes
It is additive, so AT5 would log detail and changes but not averages. The default is AT7.

ATT logs the date and time

```
Att
Time Stamp: Date 210221 Time 102502 UTC.
```

ATS logs some statistics

```
Ats
Up 11873 secs. Lost Fix:0 No 1ppS:0 Rejected:0
```

ATH logs the last few (up to 24) control volt changes

```
ATh
 Time 070821 UTC. Ctrl 2.0564135 6.360 ppb
 Time 070940 UTC. Ctrl 2.0525767 -3.650 ppb
 Time 071809 UTC. Ctrl 2.0519496 -0.596 ppb
 Time 071928 UTC. Ctrl 2.0495652 -2.268 ppb
 Time 072047 UTC. Ctrl 2.0523559 2.655 ppb
 Time 072710 UTC. Ctrl 2.0514805 -0.833 ppb
 Time 073333 UTC. Ctrl 2.0518782 0.378 ppb
 Time 074828 UTC. Ctrl 2.0516468 -0.220 ppb
 Time 081403 UTC. Ctrl 2.0517570 0.105 ppb
 Time 085226 UTC. Ctrl 2.0516523 -0.100 ppb
```

ATZ clears the statistics
ATR causes a restart
ATC initiates calibration (can’t be stopped once started)

Testing

The system has been tested in a variety of environments with different antenna configurations. Attempts to run the system indoors with the original GPS module and patch antenna gave poor to marginal results. Our house is two storey, has buildings close either side and a garage near the back door. Best results were recorded at a south facing window on the top floor with the antenna on the window sill. This consistently met the design target but in an inconvenient location. There were no other locations that met the target, despite showing usable signal strength on a Samsung tablet in the same position.

The system ran satisfactorily in most outdoors situations with a reasonable view of the sky, but this is even more inconvenient than next to the upstairs window. Looking for improvement, a second GPS module and an external antenna with a 3 meter cable were purchased. Best results were obtained with the antenna on the garage roof and the GPS in the garage. It became clear that changes of ambient temperature were the main cause of deviation, changes were most frequent in the morning and evening. These were not enough to destabilise the oscillator, it still met the target.

At this stage, it was clear there were conflicting requirements. The GPS signal was best when the antenna had a good view of plenty of sky. The system performed best when it was located where there was little temperature variation. In the house these are more than 3 meters apart.

The solution was to transmit the output of the GPS module via several meters of twisted pair to the system. This was achieved using a UA9638/UA9639 dual differential line driver/receiver combination to bring the 1ppS pulse and NMEA data to the GPS input port (no change to the basic circuit). These were powered by a separate 7.5V wall wart with 7805 regulators at each end. The link used 16 meters of Ethernet cable with power to the GPS module/transmitter delivered over two pairs, the signals returning on the other two pairs. With the antenna on the house roof, and the GPS unit cable tied to a rafter, the results are excellent. The first set of 24 logged control voltages (above) are readings from that test for nearly two days, and imply the oscillator was running at 10MHz ± 0.001Hz for that time – an order of magnitude better than the design target and mainly limited by the control logic (which corrects any error in a maximum of 2048 seconds, and if the limit was 1 cycle then this requires a frequency change of at least 0.0005 Hz).

Conclusion

The minimum cost system suffered due to the lack of a decent GPS signal. It could be useful if it is calibrated outdoors then used as a source indoors without the GPS module.

Addressing the two issues of environment and signal strength provided a system that performed better than expected, but costing double the basic system.

I leave it to the reader to decide if it is still a ‘budget’ system.

The future

Build and test the BNC output

A second circuit will be constructed with the added components. The current circuit is fine for further testing, but not laid out well.

Committing the design to PCB

I am experimenting with KiCad. A PCB for the differential line driver/receiver circuits was designed and has been sent for production. The same board can be used at either end, by changing a few wire links. As it is my first attempt at PCB design I’m awaiting the result with interest. It will make wiring the RJ45 connectors much easier.

If this PCB turns out well, and the second circuit with BNC works, then a PCB for the GPSDO will be designed and produced.

Software modification

The control loop has some arithmetic limitations that could be addressed. The loop will not lock on to the OCXO until it is within a little over 1Hz from 10MHz. The starting control voltage in the non-volatile memory is close enough that the OCXO is controllable after a minute or two. Maybe that’s not a bad thing.

Also the loop can average a maximum of 2048 seconds of data. It may be the system could improve with longer averaging times.

Hardware modification

Although it is contrary to the original aim of minimising cost, there is some indication the system could perform well if the control voltage is derived from a precision voltage supply. I have some ideas about this. Also I would like to experiment with a used DOXCO that I have.

Appendix A – software overview

The software was written entirely in MPASM assembler in the MPLAB X development environment. It is a single source file of just over 4000 lines. Assembler was used as there are some tight timing constraints, trying to work with a higher level language may have produced more headaches than it solved. MPLAB has a dual pane feature that makes working with a single source file quite convenient.

Hardware considerations

The PIC16F1455 has some nice features that facilitate a minimal circuit. It has a x4 PLL to boost the externally supplied 10MHz clock to an internal 40MHz clock. Timer 1, a 16-bit timer can be clocked at the 40MHz rate, and can be gated. The usual gate input is a Schmitt Trigger requiring about 4V input. The GPS module is a 3.3V part, so the timer is gated instead by a comparator with a 1.9V threshold. This captures the 1ppS pulse to within 25nS, in a window of about 1.6mS.

The 10-bit PWM can also be clocked at 40MHz. It is reloaded every 250µS to provide a dithered output that after filtering emulates a 24-bit DAC. The dithering system adds 1 bit to the duty cycle as required to interpolate between two 10-bit numbers. The unique bit stream completes more than twice a second and is heavily filtered so the PWM artefacts are reduced to insignificance.

Several ports have interrupt on change capability, and accept TTL level signals. This is used to implement a software serial receiver for the 9600 baud NMEA data stream. The inbuilt UART has a Schmitt Trigger input so would not be compatible with the 3.3V GPS module. Using a soft receiver has the twin benefits of eliminating a level shifter, and releasing the UART to receive user input.

The basic loop

The PWM is clocked by timer 2, which runs at the instruction rate (quarter of the clock rate, 10MHz). Timer 2 is automatically reset every 250 instructions, software counts 40,000 resets to measure a PIC second. Once started, Timer 2 is allowed to run undisturbed. Any alignment of the system to incoming 1ppS signals is achieved by changing the software count so that the 40,000 count is reached just before the next 1ppS. At the 1 second transition, Timer 1 is started to capture the next 1ppS, and information on the last 1ppS is transferred to the main program for processing.

The main program

The main program sets up the peripherals and enables interrupts. It then proceeds a second at a time, completing the task for the current second then looping waiting for an interrupt to flag the next second transition. The tasks for the program are sequential:

- wait for serial input
- log satellites in view and wait for a valid 1ppS signal
- process 1ppS then wait for the next one (either in the calibration routine or the run routine)
- If in the calibration routine, on completion reset the processor (entering run on restarting)

NMEA data

Some deviousness is required to process the NMEA data. The software serial receiver can assemble an incoming character in interrupt routines with very little overhead. However, once a character is received, it has to be processed. If interrupts are disabled for the processing it may cause vital interrupts to be missed. This problem is solved by making a copy of the mainline status then enabling interrupts. The main character processing is to determine message headers (starting with $) and accumulating the checksum for the subsequent message. Each message is assembled in an 80 character buffer as it is received. When a buffer is complete and the checksum is correct, a new buffer is started, and a second copy of the mainline state is taken. The message is then examined for items of interest such as date, time, and a valid fix status. The main program is resumed by reinstating its state and executing a RETURN. By saving state and enabling interrupts the program can handle an interrupt while processing a character while processing a message while running the main program.

The calibration routine

The test points of ±0.25 Hz were chosen as this is a change of 25nS per second in arrival time of the 1ppS. This is conveniently a change of 1 in the captured Timer 1 value. Initially the control voltage is set at the limits, then progressively converges on the desired values. While the values are far from the desired values, the error is picked up in a few seconds. As the values converge, the test times get longer until at convergence they are 256 seconds.

The convergence could be quicker, but since it may only be run once it is not optimised.

The control algorithm

This is the subject of some experimentation. Currently the program sets an anchor value, being the Timer 1 value for a 1ppS arrival just after the loop has been changed. This anchor is not normally changed once it is established, and all 1ppS arrival times are compared with it. Corrections are applied so that the 1ppS arrives at the anchor time. This means the OCXO is locked to the 1ppS over the long term.

The data is collected in samples. For the first period of 8 seconds, each sample is one arrival time offset. After 8 seconds, the first 4 samples of the current period are created by collapsing each pair of the previous period into one sample in the current period, then the next 4 samples are created by accumulating new arrival time offsets. As each sample is completed (has the required number of time offsets averaged) a least squares linear fit is calculated for the samples. The least squares calculation yields a slope – the rate at which the 1ppS offsets are changing. This directly equates to a frequency error. By using the slope, and the average of all the arrival time offsets, it is possible to compute a current deviation. If this is large a correction is made for both the slope and the deviation, such that the deviation is reduced to zero in the same time as has elapsed since the last correction.

In the special case of deviation approaching zero after a deviation correction, only a slope correction is required and the slope should then be zero (oscillator is accurate). This correction quickly damps any tendency for the system to ‘hunt’.

Future changes

The software adequately keeps the system performing better than the target. However, it appears under ideal conditions the hardware is better than the software. So the software needs modifying to cater for averaging periods of over 2048 seconds.