Budget GPSDO software

Introduction

A GPSDO (GPS disciplined oscillator) uses data from a GPS receiver to control an oscillator so that the oscillator generates a known frequency to a high degree of accuracy. The Budget GPSDO is designed to use the 1 pulse per second (1pps) from a low cost GPS module to discipline a cheap 10MHz Oven Controlled Crystal Oscillator (OCXO) to an accuracy of better than ±0.01Hz.

A GPS receiver in the long term – days or weeks – provides a very accurate time signal. But in the short term – second to second – there can be variations due to receiver design, signal reception and other factors. Many GPSDO implementations use specialised GPS receivers designed to minimise these variations, and allow good control with relatively straightforward control strategies. The Budget GPSDO was designed to use low cost GPS modules with larger variations in time signal.

The processor used in the GPSDO is the PIC16F1455. It is programmed in PIC assembler for speed and compactness. Much of the GPSDO functions rely on features found in this processor, it is unknown if similar functions are available in other processors.

Compilation Environment

The program was built within the development environment MPLAB X v5.35. This was the last MPLAB X version to support the assembler mpasm. It can be obtained from Microchip archives. A separate document details setting this up.

Overview

To overcome short term variations, the algorithm used by the Budget GPSDO is statistical in nature. The data from many GPS 1pps signals are averaged, and oscillator control is varied at intervals of many minutes rather than continuously.

The software records phase errors for a period, then applies a correction to remove the errors. Phase errors are detected by noting the difference in time between the arrival of the 1pps from the GPS module and the expected arrival time. A correction is applied by adjusting the control voltage applied to the OCXO to remove any phase error. This is similar to an analog phase locked loop (PLL) except that a PLL usually corrects continuously. In this case the corrections are made periodically. This has advantages and disadvantages.

An advantage is that second to second deviations have no effect on the control voltage if they average out in the measurement period. An analog PLL will always follow a deviation to some extent, and must be tuned to have minimum disturbance while still be sensitive enough to react to genuine discrepancies caused by OCXO variations. This can lead to long delays when the GPSDO is first turned on, or the OCXO frequency output following every small change in the GPS signal.

A second advantage is the process is self monitoring. To calculate an adjustment, the size of the error must be known. So it is possible to put a figure on how faithfully the OCXO is following the GPS. Assuming the GPS is accurate in the longer term, the error is absolute – it is a good measure of inaccuracy.

A disadvantage is the phase error can grow to be more than that delivered by an oscillator disciplined by an analog PLL before being corrected. If the user is interested in accurate frequency for comparison with test instruments, or as the basis for a down converter for GHz frequencies, this is not an issue. For calibrating high accuracy equipment such as rubidium oscillators, the user may need longer comparison times to achieve a result.

To calculate accurate corrections, the sensitivity of the OCXO to control voltage changes needs to be known to within a few percent. So before the GPSDO can be put into service, this is determined in a calibration session. The process is automated, if power is applied and the calibration has not been done, the unit starts self calibration. Calibration data is written to non volatile memory so the process is only required once.

Units of Calculation

There are two variables of interest – difference between actual and expected arrival time of 1pps; and sensitivity of the OCXO to control voltage changes.

For ease of calculation, the units used by the program are not traditional units.

The 1pps arrival time is measured in 25ns units. A value of 3 would indicate there is a difference of 75 to 100ns between expected and actual arrival time of 1pps. It is a signed value.

The control voltage is generated by a dithered pulse width modulator. The number used as a basis for this is 24 bits. The most significant byte can take a value of 0 to 249 for the full range of the control voltage (0 to 5V for the current OCXO). For logging, this value is converted to a voltage but within the program all values are in terms of the 24 bit value.

Combining these two units, the sensitivity of the OCXO is calculated as change in 24 bit value to achieve a change of one 25ns unit per second in frequency. It is quite large – the OSC5A2B02 OCXO used in this project has a nominal sensitivity of 0.1V/Hz – a change of 0.1V causes a change in frequency of one Hz. The equivalent internal unit is ((0.1/4) * (16,384,000/5) = 81,920. This is derived as follows:
25ns is a quarter cycle of 10MHz. So the voltage change for 1Hz change is four times greater than that needed to get 0.25Hz change.
The full range of the 24-bit number is 16,384,000. Divide by 5 to get the change per Volt.

There are places in the program where the sensitivity may be multiplied by a 16-bit number. The program uses 32-bit arithmetic, so using the raw value of the sensitivity could result in an overflow. To avoid this, the sensitivity is converted to a form of floating point with a signed 13 bit value and a power of 2 exponent. Wherever the multiply is used, the program requires that it be followed by division by a power of 2. The exponent is used to change the number of shifts, resulting in an integer answer.

Timer 2 interrupt

Timer 2 controls all the timing in the program. The microprocessor uses the OCXO output as its clock source. It has an internal x4 PLL so the processor clock is 40MHz, each instruction takes four clock cycles so the instruction rate is 10MHz. Timer 2 is set up and started at power up, and is not altered after that. It rolls over every 250 instruction cycles and if interrupts are enabled, causes an interrupt. Timer 2 also is the timer for the PWM, it generates one pulse per roll over.

The interrupt routine has several functions. It dithers each pulse generated by the PWM. The 24 bit control voltage value is split into most significant 10 bits and least significant 14 bits. Each interrupt the 14 bits are added to a 14 bit accumulator. If the accumulator does not overflow, the most significant 10 bits are loaded into the PWM register. If there is an overflow, one is added when the 10 bits are loaded, increasing the pulse length by 25ns. The PWM output is heavily filtered to provide the control voltage. The dithering allows the voltage to be changed in steps of less than 1µV.

The interrupt routine also maintains a counter, 40,000 interrupts measure one (nominal) second. At the end of the second, selected values are copied for mainline use, a flag is set to indicate to the mainline that the second is complete. Usually Timer 1 is started to detect the next 1pps (see detecting the 1pps).

The routine also times the display on the LED. The display control is a 3 byte register that is shifted approximately every 50ms. The LED is switched on or off depending on the value of the bit shifted out.

Detecting the 1pps

The 1pps from the GPS module is received on a pin connected to one of the microprocessor comparators. The other input to the comparator is 1.9V derived from an internal source. The output of the comparator is used to gate Timer 1 (if Timer 1 is running, it is stopped by the comparator when its output changes). Ideally, Timer 1 would be gated directly by the 1pps. However, this requires a 5V CMOS signal level. Many GPS modules have 3.3V outputs, which may not be detected. Using the comparator overcomes this problem.

During startup, the arrival time of the 1pps is unknown. To ensure the Timer 2 interrupt starts Timer 1 at the right time to be running when the next 1pps arrives, a comparator interrupt is enabled. When the comparator detects the 1pps, the interrupt routine changes the count of 40,000 interrupts so it reaches zero about 800µs before the next 1pps. Timer 1 is clocked at the processor clock rate (40MHz) and is 16 bits. It counts up to overflow in about 1.6ms, so the next 1pps should arrive about midway through the count and stop the counter.

When the Timer 2 interrupt starts Timer 1, the interrupt may be delayed by another interrupt. To ensure Timer 1 reaches the same count at the same time each second, it is not started from zero but a count derived from Timer 2.

A Timer 1 overflow interrupt is enabled to detect if a 1pps is missed. This does not happen often but ensures that there is a result every second: either a 1pps is timed, or the 1pps was not detected.

Other interrupts

The processor has only one interrupt vector. When interrupted, the program has to interrogate each possible interrupt source until the cause is located, then act on it. The Timer 2 interrupt is critical and the most frequently occurring, so is interrogated first. It occurs every 250 instruction times, and can be up to 50 instructions. It follows that interrupts should not be disabled (either by another interrupt or by program) for long, ideally less than 200 instruction times. The program is written with this in mind,

Soft serial port

The processor UART is dedicated to the user interface, so to receive serial data from the GPS module, a software receiver is implemented. The software UART uses both Interrupt on Change (IOC) on the selected input pin and Timer 0 (TMR0). These are the interrupts interrogated after the Timer 2 interrupt as they are somewhat time critical.

Initially, the first byte of the character assembly area (RXassem) is filled with 1 bits and TMR0 interrupts disabled. A 1 to 0 transition on the input pin triggers the IOC interrupt and is accepted as start of character. A 0 bit is shifted in, and TMR0 enabled to interrupt in 1.5 bit times. An IOC before TMR0 expires shifts in a new bit value, and TMR0 restarted to interrupt in 1.5 bit times. If TMR0 expires before IOC, a repeat of the last bit value is shifted in and TMR0 is set to 1 more bit times. Either TMR0 or IOC interrupt routine can detect a properly framed character – because initially there were all 1 bits, the overflow bit (bit 7 of RXassem+1) only becomes 0 when the start bit is shifted out of Rxassem.

Although this is not a rigorous method it has proved to be adequate to reliably receive the NMEA messages from the GPS module.

The input pin responds to TTL levels, so is able to operate with 5V or 3.3V signals. The processor UART signals are CMOS 5V so not compatible with a 3.3V signal.

UART

The processor UART is dedicated to the user interface. Its input and output interrupts are not critical so are processed after the soft serial port interrupts. Output interrupts are interrogated before interrupts associated with 1pps timing. Input interrupts are interrogated after them.

1pps detection and processing

There are three possible interrupts associated with the 1pps signal, although they are not all enabled at the same time.

The Timer 1 gate interrupt is interrogated first, it is the normal way to detect the 1pps. The interrupt is not time critical as the gating process is all in hardware.

Next is the Timer 1 overflow interrupt. It is triggered if the 1pps was not detected. If the comparator does not gate Timer 1 then Timer 1 runs to its maximum count and causes the overflow interrupt.

When the gating method is used to detect the 1pps, its arrival time has to be known to less than a ms. To find the first 1pps, the comparator interrupt is enabled, and this establishes the expected arrival time of the next 1pps with sufficient accuracy. The program enables either the gate interrupt or the comparator interrupt, but not both at the same time.

Lost clock

This is the final interrupt that is interrogated. The processor normally derives its clock from the OCXO. It has the ability to fall back to its internal clock if the external clock fails. There is an interrupt handler to detect this condition, it initiates a processor reset. In the startup procedure (after the reset) a missing external clock is signalled by a specific flashing of the LED.

If the interrupt vector is activated (an interrupt is initiated) and none of the handlers detect that it is ‘their’ interrupt, then the program (by design) goes into an infinite loop. This should only occur if the software or hardware is faulty.

NMEA message handler

It is not necessary to receive and decode the NMEA messages. The GPSDO could rely just on the 1pps pulses. However, some GPS modules continue to generate 1pps even though the GPS signals are poor and the module cannot guarantee the validity of the 1pps.

As a precaution, the program decodes NMEA messages specifically to retrieve the status field of the $xxRMC messages, which determines if the module is generating valid data.

The messages also contain ‘nice to have’ data such as satellites in view, UTC date and time. These don’t affect the operation of the GPSDO, but add useful detail to reporting.

The program is also capable of ‘passing through’ the NMEA data to the user interface.

Interruptable threads

The soft serial receiver interrupts are quite short and can interleave successfully with the Timer 2 interrupts. However, once a character is assembled, it is used to assemble a valid NMEA message. Then that message must be processed while continuing to assemble the next message. This requires more processing than is tolerable in an interrupt routine. However, there is insufficient memory to store all the messages for later processing by the mainline so the messages must be processed in a timely manner.

This is achieved by threading. When the processor has an interrupt, significant registers are stored in 8 bytes of memory, and reinstated by a return from interrupt (RETFIE). If this data is saved elsewhere, interrupts can be enabled in the interrupt routine, allowing it to be interrupted. A RETFIE by an intervening interrupt returns not to the initially interrupted mainline, but to the current thread.

This method is used to create two levels of thread. The first deals with characters assembled by the soft receiver, moving them into a buffer. Valid NMEA messages start with a $, end with *xx where xx is a checksum, and have up to 80 characters of data. The character thread assembles this character by character, and after each character is processed, restores the original interrupt data and executes a RETFIE to let the originally interrupted process resume. There is plenty of spare processing cycles to ensure each character completes processing and its thread ends before the next character thread is created. Otherwise a race condition could occur and the program fails.

The exception is when a valid buffer has been filled. A second copy of the interrupt data is taken, the buffer pointer is switched to a buffer already processed (and therefor available to be filled again) and the thread allowed to continue. There is an interlock to prevent a race condition but again there should be plenty of processing cycles to allow a buffer to be processed before the next buffer fills. The thread determines the NMEA data type, extracts any relevant data, reinstates the interrupt data and executes a RETFIE.

Calibration

When first commissioned, the GPSDO determines the actual sensitivity of the OCXO by testing. The OCXO is run 0.25Hz fast, and 0.25Hz slow. The difference in control voltage to do this yields the sensitivity. The actual value stored is: change in the 24-bit value used to generate the control voltage for a change of 1 (25ns) in measured phase per second.

The heavy lifting during calibration is in the subroutine FreqErr. The calling process sets a 24 bit value to be used as the control voltage and calls FreqErr. FreqErr returns a frequency error between the OCXO and GPS.

On entry, FreqErr stores the gated Timer 1 value, then one second later compares the new gated value against the first value. The difference is the error in intervals of 25ns between the two. If the difference is larger than 255 (a frequency error of 64Hz for a 1 second period) then it exits. Otherwise the time to the next comparison is doubled and the process repeats. If the time between storing the first gated value and the latest gated value is 256 seconds the exit is taken regardless of the error. To reach that duration, the frequency error is less than 0.5Hz (otherwise the difference after 128 seconds would have been greater than 255 causing an earlier exit).

Initially, the calibration routine uses FreqErr to determine the frequency errors near the maximum and minimum control voltages. If the OCXO is not faulty, one error will be positive and the other negative (otherwise the process loops indefinitely trying each voltage repeatedly). Once a positive and negative error is established, a loop is entered to refine the control voltages closer to (but not reaching) a voltage where there is no error.

First, a value of control voltage sensitivity is calculated. Initially due to sensitivity changing for different control voltages, this is not very accurate but adequate for refining the calculation. The sensitivity is:

sensitivity S = $(V_1 – V_2)/(E_1 – E_2)$
expressed as change of 24-bit value to get 0.25Hz change in frequency
where V is voltage and E is frequency error at that voltage.

A nominal voltage for zero error can then be calculated

$V_{zero} = V_1 – E_1 * S$ ($V_2 – E_2 * S$ is equally valid)

To refine the calculation, the error with the largest absolute value is selected, and a new control voltage is calculated to try to halve the error.

$V_{test} = V_{zero} + (E_x \backslash 2) * S$

The test is then repeated until E_1 and E_2 are expected to be ±0.25Hz (in the program, E_1 and E_2 approach a value of ±1, being a phase change of 25ns per second).

If the actual value of E_1 and E_2 differ by 0.5Hz within about 1%, the sensitivity is presumed valid. The process then repeats the measurements until 16 valid sensitivity values have accumulated. The average is stored as well as $(V_1 + V_2)/2$ which is a reasonable value for V_{zero}.

The control algorithm used after calibration

The control algorithm applies the stored V_{zero} control voltage, then stores a gated Timer 1 value (in a variable called RTanchor). All subsequent gated values are compared to it, the difference being the phase errors in 25ns intervals (90° of phase for a nominal 10MHz signal).

If the phase error is larger than 127 it is considered to be too large for the control algorithm to process, and it restarts. This equates to a frequency error of approximately 16Hz, which is normally encountered when the OCXO is warming up but not when it reaches running temperature. Even when the frequency error is less than 16Hz, the calculated control voltage to correct the frequency may be outside the allowable range so a restart is necessary.

The algorithm compares the accumulated phase errors for two successive periods. The length of the period ranges from 1 to 2^n seconds. The software could handle n up to 15 (about 9 hours), in practice a value between 7 and 10 gives the best results in most cases.

To calculate a correction, both a frequency error and a phase error are required. It is assumed the frequency was constant for the two periods (the control voltage was not changed). If there is a frequency error the phase error changes throughout the two periods, so an estimate is required of the phase error at the time of calculation (the end of the second period).

For the purpose of explanation, let the accumulated phase errors for two successive periods be A and B. From these, a calculation is made of the frequency error, and an estimated accumulated phase error at the time of calculation.

The frequency error is (B – A)/(measurement period of A and B in seconds)^2

The calculation is in quarter cycles. Divide by 4 to get the error in Hz.

The program does not do any division unless a correction is required. To detect an excessive frequency error, testing the absolute value of (B – A) against a guard value is sufficient.

To yield a frequency error, (B – A) is divided by measurement period twice. Once because A and B are totals, and the calculation requires averages. Once because the phase change has taken place over the measurement period which may be many seconds, and needs to be converted to frequency (phase change per second).

The phase error at the end of the two measurement periods is (3B – A)/((measurement period of A and B in seconds) * 2) in 90° increments.

The term (3B – A)/2 can be better understood as (A+B)/2+(B-A)
(A+B)/2 is the average of the two accumulations and assumed to be the phase error at the mid point.
(B-A) is the change from the mid point to the end of the two measurement periods
Division by the measurement period is needed to convert totals to averages

Given the frequency error and the phase error, how is the correction calculated? The sensitivity of the OCXO to control voltages is known, call it S.
The correction for a frequency error is simply: - error * S

The phase correction is not so straightforward. It requires introducing a frequency error for an arbitrary period of time. Applying a small error corrects over a long period, correcting with a larger error reduces the correction time. The implementation aims to correct the phase error in 2 measurement periods, about the time the next correction is expected to take place. The correction is:

(-phase error * S)/(measurement period of A and B in seconds * 2)

The actual change in control voltage is just the sum of the two corrections.

Multi level accumulation

The previous section describes the calculation, mentioning the collection period in passing. In theory, a collection period can be chosen that balances OCXO variations in the long term against GPS data variations in the short term. In practice, events occur that should be dealt with before the end of the chosen period, especially when the GPSDO is first powered up. To detect and deal with any problem, the errors are accumulated and tested in levels. At the lowest level (Level 0), A and B are the phase errors for 2 successive seconds. At the next level, A and B are the sum of 2 successive 2 second periods. Each level doubles the number of seconds used for accumulation. For example, at level 8 the items are each the accumulation of 2^8 or 256 phase errors, and calculations for the level occur after 512 seconds.

The logic for the accumulation is relatively straightforward. It is guided by a count of the seconds over which accumulation occurs, starting at second zero as the second after the last correction. This count is inspected as a binary number: for example, second 27 is inspected as 11011.

Each second:
1. Place the phase error for the current second in E (for Error), set the level counter L to zero.
2. Counting from the rightmost bit of the counter inspect bit number L.
3. If the inspected bit and all bits in the counter to the left (more significant bits) of the inspected bit are zero then this is the first value received at this level. Set the “previous accumulated value” A(L) = E and go to instruction 6.
4. Otherwise there is a previous accumulated value at this level. Calculate the errors using the previous accumulated value A(L) and E as the two values. Test the calculated errors against predefined limits. If neither limit is exceeded, swap values A(L) and E. If a limit is exceeded, then initiate a correction (which restarts the accumulation process).
5. If the inspected digit is 1, add A(L) to E, increase the level L by 1, return to instruction 2.
6. If the level is for the chosen maximum period (for example, if applying corrections at a maximum of every 1024 seconds, the chosen level is 9) then initiate a correction, otherwise exit the accumulation routine.

Note that step 4 compares errors against guard values. It is this comparison that allows unexpected events to be detected before the chosen period is complete. The guard values are chosen to trigger correction for a significant event in a short time but are more tolerant of less severe deviations. The algorithm ensures that successive seconds are compared every second, every 2 second accumulation is tested every 2 seconds, every 4 second accumulation is tested every 4 seconds and so on.

Of particular interest is the trigger for level 6 (128 seconds) – it is set so it triggers if the phase error is more than 5. This is approximately one cycle deviation of 10MHz in 100 seconds, an error of about 1 part per billion. If the GPSDO reaches this level and does not exceed the guard values it is assumed to be running within its specification (better than 1ppb).

Zero cross detection

If a correction is initiated by exceeding a guard value, the cause is unknown and the applied correction may not be appropriate for the event. This can cause the phase correction to be excessive with the phase going quickly from one extreme to the other. To prevent this, if a guard triggered correction occurs, a zero cross detection is enabled. If, during the correction, a 1 second reading indicates a phase error of the opposite sign to the initial error, the correction is terminated and a new correction applied. A phase error of zero is not regarded as phase reversal.

The zero cross detection is not enabled if the previous correction was at the chosen maximum period.

Guard triggered corrections are common when the GPSDO has a cold start. It takes several minutes for the OCXO to become stable. The longer the GPSDO runs, the less likely a guard triggered correction is required. It is desirable to choose the longest maximum period that avoids frequent triggering. This is usually determined by the quality of the GPS signal. For stable signals from a well positioned antenna, a 1024 second period is achievable. For poorer signals, 512 seconds is usually satisfactory.

If a shorter period than 512 seconds is chosen, the controlled frequency is more likely to be affected by short term variations of the GPS. And less samples per period lead to more uncertainty in the averaging process.

Drift calculation

The quartz crystal used in an OCXO can exhibit a small change over time, such that with a steady control voltage, there is a small but constant change of frequency. This is quoted for the OSC5A2B02 as less than 0.5ppb per day. Although tiny, this change can have an effect when long measurement periods are used.

To negate this effect, the GPSDO has a drift compensation method. It may not be needed, but will not adversely affect performance. It could be useful with more highly specified oscillators and GPS units, when longer averaging periods can be used.

The method is straightforward. Each second, the 24-bit control voltage is added to an accumulator. A new accumulator is started each day. Each day, the following is calculated:

increment = ((((accumulator today)-(accumulator yesterday))/86400)*65536)/86400)

86400 seconds in a day

$65536 = 2^{16}$

The increment is 2^{16} times the change required each second. In practice it is a small number so the 24 bit number representing the control voltage requires changing by 1 at intervals of many seconds.

To determine when to increment/decrement the control voltage by 1, the increment is added to a 16 bit value. If the increment is positive, eventually the 16 bit number will overflow, indicating it is time to add 1 to the control voltage. If the increment is negative, the 16 bit number will overflow every time except when it is time to subtract 1 from the control voltage.

User interface

Much of the program is devoted to the user interface. It is unnecessary for proper functioning of the GPSDO but provides a lot of information which may be of interest to the user.

Input characters from the user are detected and stored by the UART RX interrupt routine. All user commands are two characters, the arbitrary convention for the program is a TAB followed by an alphanumeric. Any character could be substituted for TAB, it was chosen for the fact it is easily located on most keyboards.

The input routine does not echo. It changes flags so further output is suspended (but any data currently in the output buffer will be sent).

The input characters are processed in the subroutine WaitAsecond. This subroutine is called from many places in the mainline after the mainline has finished whatever task was required and is waiting for the end of the current second. Any characters are echoed (TAB is echoed as >) and the alphanumeric command is decoded and acted on.

Output to the user is dumped by various routines into an 80 character output buffer. This is output by the UART TX routine at 9600 baud. Some output (history) is more than 80 characters so has to be output in chunks. This is part of the processing in WaitAsecond, which outputs one chunk a second until output is completed.

Programming considerations

For most calculations arithmetic is limited to 32 bit signed integers and mostly uses subroutines written by Peter Hemsley with some modification. The two accumulators for drift calculation are 48 bit, the difference is calculated by a custom subtraction and is not processed further if it is larger than 32 bit signed (in practice, the difference always has less than 32 significant bits).

The variables for multi level accumulation algorithm are 16 bit signed. Because of the guard test, values that could overflow 16 bits cannot be generated. Only the A value is stored for each level, the passed accumulator E is the B for the calculation at each level, and if the level doesn’t trigger a correction, becomes A for the next calculation at that level. So each level takes 2 bytes of RAM and 4 bytes for the guard limit values.