MODEL A6-1PPS

1PPS Locking Module

USER'S HANDBOOK

Quartzlock UK Ltd
179 Junction Road
Burgess Hill
West Sussex
RH15 0JW
England.

Tel: +44 (0) 1444 232 967
Web: www.quartzlock.com
E-mail: quartzlock@quartzlock.com

Contents

1 Safety Considerations

1.1 General

This product and related documentation must be reviewed for familiarisation before operation. If the equipment is used in a manner not specified by the manufacturer, the protection provided by the instrument may be impaired.

1.1.1 Before Applying Power

Verify that the product is set to match the available line voltage and the correct fuse is installed.

1.1.2 Before Cleaning

Disconnect the product from operating power before cleaning.

WARNING

Bodily injury or death may result from failure to heed a warning. Do not proceed beyond a warning until the indicated conditions are fully understood and met.

CAUTION

Damage to equipment, or incorrect measurement data, may result from failure to heed a caution. Do not proceed beyond a caution until the indicated conditions are fully understood and met.

1.1.3 This equipment must be earthed

An uninterruptible safety earth ground must be maintained from the mains power source to the product's ground circuitry.

WARNING

When measuring power line signals, be extremely careful and use a step down isolation transformer whose output is compatible with the input measurement capabilities of this product. The product's front and rear panels are typically at earth ground. Thus, never try to measure AC power line signals without an isolation transformer.

WARNING

Instructions for adjustments when covers are removed and for servicing are for use by service-trained personnel only. To avoid dangerous electrical shock, do not perform such adjustments or servicing unless qualified to do so.

WARNING

Any interruption of the protective grounding conductor (inside or outside the instrument) or disconnecting of the protective earth terminal will cause a potential shock hazard that could result in personal injury. Grounding one conductor of a two conductor out-let is not sufficient protection.

Whenever it is likely that the protection has been impaired, the instrument must be made inoperative and be secured against any unintended operation.

If the instrument is to be energised via an autotransformer (for voltage reduction) make sure the common terminal is connected to the earthed pole terminal (neutral) of the power source.

Instructions for adjustments while the covers are removed and for servicing are for use by service-trained personnel only. To avoid dangerous electrical shock, do not perform such adjustments or servicing unless qualified to do so.

For continued protections against fire, replace the line fuse(s) with fuses of the same current rating and type (for example, normal blow time delay). Do not use repaired fuses of short-circuited fuse holders.

1.2 Voltage, Frequency and Power Characteristics

Voltage 14 – 30V DC (12 – 14V DC no on board OCXO) TP3 & TP4 0Vdc or J1

Power characteristics 200mA Typical (50mA Typical no on board OCXO)

1.3 Environmental Conditions

1.3.1 Temperature

Operating (ambient)	-10°C to +55°C (-65 to +65 op)
Storage	-40°C to +85°C

1.4 Cleaning Instructions

To ensure long and trouble operation, keep the unit free from dust and use care with liquids around the unit.

Be careful not to spill liquids onto the unit. If the unit does get wet, turn the power off immediately and let the unit dry completely before turning it on again.

Never spray cleaner directly onto the unit or let liquid run into any part of it. Never use harsh or caustic products to clean the unit.

1PPS Locking Module A6-1PPS

2 Scope

The Quartzlock A6-1PPS is a board level product designed to lock a 10MHz stable oscillator, either an OCXO or a rubidium, to the 1PPS time mark signal generated from a GPS receiver. The module can be programmed for a wide range of controlled oscillator parameters, and GPS receivers. The controlled oscillator can be either on or off the board. A stable 1PPS time mark is generated from the controlled oscillator. This can be adjusted to any offset from the GPS 1PPS in 1ns steps.
The control algorithm used is designed to give optimum control results and the fastest possible acquisition from switch on.

Operating Procedure

2.1 Introduction

This module is designed to lock a 10MHz stable oscillator, either an OCXO or a rubidium, to the 1PPS time mark signal generated from a GPS receiver.

There are a number of serious problems involved in this, as the data rate by definition is only one measurement per second. In order to get sufficient frequency resolution to correct the oscillator, a very long averaging time would be required.

Because the 1PPS time mark is a fast rise time logic signal, the only measurement that is feasible is to time tag the incoming 1PPS edge relative to a local clock driven by the controlled oscillator. By calculating the rate of change of the arrival time over a suitable averaging period, the frequency offset of the controlled oscillator can be calculated. An alternative strategy would be to set the time of the first 1PPS arrival as the zero phase of a phase detector with a range of ± 0.5s. This is equivalent to ± Pi radians. A phase lock loop would then provide a very slow control of the oscillator.

In both systems the timing accuracy and resolution of the incoming 1PPS is important. Modern GPS receivers provide a 1PPS output jitter of between 1us RMS for a navigation receiver, to less than 7ns RMS for a special timing receiver operating in position hold mode. It is desirable that the timing resolution of the module should be better than this, as otherwise quantization noise would enter the averaging process and degrade the performance of the system. It would only be possible to compensate for this by increasing the averaging time. A suitable specification for time resolution is ± 1ns.

To achieve this directly would need a 1GHz clock. A much more suitable method is an analogue time interval expander. This device has been used in many designs of frequency counter starting with the Racal 1992. The principle is that an error pulse is generated which has a width equal to the time between the incoming edge to be timed, and the next clock pulse. For example, with a 100ns clock, the error pulse will have a width of between 0 and 100ns. This error pulse is then used to charge a capacitor or integrator. The capacitor or integrator is then discharges at a much slower rate, say 1/1000 of the rate. The resulting stretched pulse is then measured using the available clock pulses. The improvement in resolution equals the ratio of the discharge to charge rate. For the example above the resolution will be 100ps.

The next thing to consider is the choice of the control algorithm. This must provide an appropriate control bandwidth so the short term stability of the controlled oscillator (Allen variance) is optimised over a wide range. The ideal bandwidth will vary considerably between a low cost OCXO, and a rubidium.

One option is to use a simple phase lock loop. This would be a type 2 second order loop (i.e. with an integrator in the loop filter) with a zero to give suitable phase margins for optimum dynamic performance. However one problem with a phase lock loop is that it must reduce the initial phase error to zero by changing the frequency of the VCO. With the very long loop time constant necessary to remove the effect of the GPS time jitter, the eventual settling of the loop could take several days. It is also difficult to extract measures of performance from the loop, for example it is difficult to estimate the current frequency error of the VCO. It was felt that a frequency control loop would settle quicker. For a frequency standard we do not mind operating with a fixed phase offset, and there is no need to reduce this to zero.

One possible method of extracting frequency offset from phase data is a quadratic least squares fit on a block of data. This is a standard method for extracting phase offset, frequency offset, and frequency drift from phase difference information. Having extracted the offset frequency, we can then make a correction to the controlled oscillator to remove the offset. If the control constant was known exactly, there would be no under or overshoot. The problem with this method is that we do not know how large to make the block of data that we analyse. The reliability of the fit is given by the correlation coefficient, and ideally this should be monitored on a continuous basis. What is required is a continuous least squares process. This is of course, a Kalman filter, and this was the eventual method selected for implementing the control algorithm.

The Kalman filter will be briefly described in general in a (hopefully) simple way, and then the specific implementation for our problem will be described in more detail.

A block diagram of a Kalman filter is given in figure 1. It is basically recursive estimation, based on noisy measurements, of the future "state" of a system. The system is defined as a "state vector" and a "state transition matrix". The system in our case would be the controlled oscillator that we wish to predict, and the state vector would contain the phase offset, frequency offset, and frequency drift variables. The "state transition matrix" defines the differential relationship that exists between the state variables over one time increment. The concept of a system driven by noise processes is important here. If our Rubidium had absolutely constant drift, its output phase would be known for all time once the initial drift, frequency offset and phase offset had been determined. Data gathered a year ago would have as much validity as data an hour old. If the Kalman filter is given this model of the Rubidium, the results are identical to the least squares fit of all the data. Of course the quadratic least squares fit assumes that the Rubidium can be modelled by three constants.

A more realistic physical model would allow the drift to vary. If this varied in a deterministic way, we should add a further term to the state vector to reflect this deterministic process. However if the variation was random, we can tell the Kalman filter that this is so. Note that the filter is only optimum for white Gaussian noise processes. However in our case we can model the noise of the Rubidium oscillator more accurately by adding white Gaussian noise to each term in the state vector. If we add some uncorrelated noise to each term in the state vector, we end up with white phase noise, white FM noise, and random walk FM noise due to the single and double integration in the model. This is shown in figure 2.

The measurements are also assumed to be contaminated with Gaussian white noise. In our case we only have one measurement that is phase offset. We do not know that the main contributor to measurement noise, the GPS receiver, is either white or Gaussian. However this is a limitation of the simple Kalman filter that we intend to use. If we are sure of the characteristics of the measurement noise, we can include this knowledge by adding more terms to the state vector. We are then essentially including the known aspects of the measurement in the system model.

As well as the state vector, the Kalman filter maintains a matrix that gives the current variances (mean square error) of the quantities in the state vector. These give us current estimates of the likely errors in the state vector, in our case variances of phase offset, frequency offset, and frequency drift. These will be very useful for display to the user. They also have another use, which will be demonstrated later. In effect they control the "bandwidth" of the filter. As more data comes in, the variances decrease, and the filter gives more weight to the current estimate (which represents the complete history of the data), and less to the current measurement. The measurement variance, which we have to tell the filter, also affects the "bandwidth". If we tell the filter that the measurement is noisy, it reduces the bandwidth.

So far we have considered the Kalman filter as a device for analysing the incoming data in an optimum way. However we need to control the Rubidium oscillator, and reduce the frequency offset to zero. An elementary method would be to write periodic corrections to the Rubidium control DAC, and wait for the Kalman filter to track out the resulting discontinuity in the measurements. However there is a much better way. If we adjust the frequency offset term in the state vector at the same time that we correct the Rubidium, the filter will ignore the correction, and no extra settling time will be required. In effect we are defining the model of the Rubidium to have a frequency discontinuity at a particular time, and provided the real Rubidium has that discontinuity, the Kalman filter will see no difference between the model, and what the measurements are telling it about the real system.

Using this technique, we can correct the Rubidium as often as we like. However if we are uncertain as to the exact value of the control constant, then the correction will undershoot or overshoot the model. Another trick that can be useful is if we know that there is a measurement discontinuity, but we do not know how large it is. An example would be if the GPS signal disappears for any reason. When satellites were reacquired, there could be a phase discontinuity between the GPS 1PPS and the locking module internal clock. Although we cannot tell the filter the amount or direction of the discontinuity, we can tell it that its current estimate of phase is completely unreliable. We do this by adding a large number to the appropriate term of the error covariance matrix. The filter then gives maximum weight to the measurements to reacquire the phase as quickly as possible, however as it thinks its frequency is still accurate, it does not give excessive weight to the rate of change of phase measurement, and the frequency covariance hardly rises.

The Kalman filter can predict ahead if measurement data fails. In this case both the state vector and the error covariance matrix will be updated. The previously estimated value of drift will update the frequency offset automatically. Frequency corrections can be made in the usual way. The error covariances will rise to reflect the lower confidence in the predictions as time passes. When measurements resume, the filter will automatically recover and the error covariances will start to fall. Thus the user is always aware of the reliability of the frequency output. If an unknown phase step is expected on resumption of measurements, then the phase variance should be augmented as previously described.

2.2 Technical details of design

The design is based around a PIC18F6723 microcontroller. This is a high end controller with 5 capture/compare modules and 4 timer/counters. The time interval expander is tightly integrated with the processor internal peripherals to produce an economical design. The basic timing resolution is 400ns (one processor cycle at a 10MHz clock frequency). The time interval expander extends the resolution by 2000 times. In order to avoid the problems of expanding a pulse of zero width, one cycle of the 10MHz clock (100ns) is added to the time error pulse. This gives an unexpanded pulse width of 100ns to 500ns. After expansion, the pulse is 200us to 1ms. This is timed by the 400ns clock to give a basic ±200ps resolution.

A time interval expander must be calibrated as otherwise a glitch will be produced when the time error pulse rolls over from 500ns to 100ns, and vice versa. This is caused by the expansion ratio not being exactly the expected 2000 times. The expansion ratio may drift with time and temperature.

As the incoming 1PPS only needs measuring once per second, the dead time is used to calibrate the time expander. The hardware generates exact pulses of 100ns and 500ns by gating from the 10MHz clock. These are expanded and measured. The calculated end points of the expanded pulse are used to correct the real measurement of the incoming 1PPS. This auto calibration operates continuously.

The control of the OCXO or other controlled oscillator uses a precision tuning voltage derived from D to A convertors. Two 16 bit DACs are used, with the output of the fine tune DAC divided by 256 and added to the output of the coarse tune DAC. This gives effectively 24 bit resolution with an overlap between the coarse and fine tune DACs. A software normalisation process ensures that the fine tune DAC is used for tuning most of the time. Only when the controlled oscillator has drifted out of range of the fine tune DAC would the coarse tune DAC need adjusting, with the chance of a very small glitch in the tuning voltage. A precision, low noise, voltage reference is used to supply the DACs.

The microcontroller is provided with an RS232 interface. A simple set of control codes enable monitoring and set up of the controlled oscillator parameters to accommodate a wide range of controlled oscillators. A Windows front end program will use the control codes to enable the operation of the PLL to be monitored with real time graphs of performance measures.

2.3 Software design

In normal operation the auto calibration performs calibration cycles every 20ms. The approximate time of arrival of the next 1PPS input pulse is known, so the calibration cycles are paused while the 1PPS is measured. The raw measurement of the arrival time is corrected for the actual expansion ratio and is scaled to lie in the range -500.000000 to +499999999 ns relative to the internal clock.

The first valid 1PPS edge to arrive after reset is used to zero the internal clock. This makes the arrival time initially close to zero, and avoids problems with lack of precision in the floating point calculations which follow.

The corrected time tag is sent to the Kalman filter routine which runs once every second. The estimate of the controlled oscillator phase, fractional frequency offset, and drift (the state variables) is updated by the new measurement. Also updated is the error covariance matrix which provides an indication of the accuracy of the estimate of the state variables.

After update of the filter, the frequency correction for the controlled oscillator is calculated. This is done by scaling the Kalman frequency offset estimate by the known (programmed) tuning slope of the oscillator. The correction is then added to the frequency control register of the oscillator.

The tuning voltage is divided between the coarse and fine tune DACs as follows: When normalisation is performed, the fine tune DAC most significant 8 bits are set to mid point (80h). The least significant 8 bits of the fine tune DAC are set to the least significant 8 bits of the tuning word. The coarse tune DAC is then set to provide the final tuning voltage. During all subsequent tuning, only the fine tune DAC is used over its 16 bit range. If the range is exceeded, the normalisation procedure is repeated.

A state machine provides control of locking. After reset the last value of the frequency control register, which has been stored in EEPROM on a regular basis, is restored. This will retune the controlled oscillator to very nearly the correct frequency. The Kalman update is disabled and the software waits for the following all to occur (state 0):

a) Rubidium reference warm up input to go low OR OCXO supply current to drop below a threshold showing the Rubidium/OCXO has warmed up

b) A 1PPS input capture has occurred

The software then requests a reset of the internal clock. (State 1) This will normally occur on the next 1PPS to be received.

Once a clock reset has occurred, the Kalman filter tracking is started, however frequency corrections are not made to the controlled oscillator. (State 2) Each capture must be within 50us of the first capture, otherwise the reset state is re-entered. After 100 successful captures, state3 is entered provided the performance monitor, MEANFREQERROR is below a threshold.

The performance monitor, MEANFREQERROR is calculated as follows:

> The mean of the Kalman frequency offset estimate is calculated by means of a 5th order exponential filter. (In the pre lock state the mean may not be near zero, i.e. there may be a constant offset between the controlled oscillator and GPS time)

After each iteration of the Kalman filter, the current deviation is calculated by subtracting the current frequency offset estimate from the running mean. This value is squared, and divided by the predicted variance from the error covariance matrix that is maintained by the filter. This normalises the actual deviation that is seen by the predicted deviation from the filter. (The predicted deviation only depends upon the system and measurement noise parameters NOT on the actual behaviour of the system.)

The normalised deviation in then filtered in a 4th order exponential filter. During warm-up the performance measure will be high, indicating that the controlled oscillator is still drifting fast, relative to its predicted steady state performance. When the controlled oscillator is stable, and the Kalman filter has settled, the performance measure will drop below a threshold. At this point frequency corrections will be started. (State 3)

In state 3 corrections are made to the controlled oscillator. The filter and oscillator will continue to settle, until the performance monitor falls below a second threshold. At this point the lock indicator is switched off. (State 4)

The following parameters set up the Kalman filter to match the controlled oscillator:

a) Oscillator noise parameters:

S1	variance of random walk FM noise
S2	variance of white FM noise
S3	variance of white phase noise
OC1	oscillator tuning constant in fractional frequency/volt
OC2	maximum oscillator tuning voltage in volts, assuming 0V minimum

b) 1PPS noise root variance (a function of the GPS receiver used)

R	measurement noise root variance in seconds

These parameters are programmed over the RS232 interface, and are stored in non volatile memory.

The oscillator noise parameters may be obtained from a measured Allen variance curve using a MathCAD modelling program.

2.4 Interface

The RS232 interface enables setup and monitoring of the 1PPS.

RS232 configuration 9.6kbaud, 8bits, no parity, no handshake) RX CONN4 Pin 4 TX CONN4 Pin 6.

The basic control codes are defined in appendix A. The purpose and operation of the control codes are considered in this section.

2.4.1 Basic

All control codes operate on fixed string lengths, there is no terminator character, i.e. "enter" is not required.

All control codes start with two letters (upper case only) which defines the control group. In each group the code may be followed by a "?", a "+" or an additional letter (upper case) with a variable length hexadecimal string. In a few cases only the initial two letters are required, in which case the microcontroller will acknowledge with a single carriage return character (0Dh) after the initial two characters.

The "?" as the third character (query command) will return a hex string representing the current setting of the parameters appropriate to the control group, followed by a carriage return.

The "+" as the third character will enter the query command into the internal repeat stack. The command will then be repeatedly processed at a preset rate. The operation of the command repeat facility is described later.

The control groups which accept an additional letter are intended to update internal parameters, and are described fully under each control group. After updating a parameter, the controller will reply with a carriage return and an automatic repeat of the query form of the control group.

If an invalid code or string is sent that cannot be parsed, the controller will reply with "!" "carriage return", and will clear the input buffer.

2.4.2 Detailed use of interface commands.

WARNING

By incorrect use of the interface, it is possible to get the controller into a state where the 1PPS will not work. The key command to avoid is "EU" (EPROM update) which writes any changed parameters into the EPROM memory. Provided this is not done, a power on reset will always restore the previous values. An alternative to the power on reset is the command "SR" (software reset).

2.4.3 Command list

OS (overall status)

This command monitors and controls the state machine which controls the locking and monitoring of the 1PPS. It also controls various test modes which are more concerned with product development than user adjustment. However there is one function which needs to be set up using this command for a new application. This is the tuning voltage span.

"OS?" returns 6 hex strings of different lengths. The control code definition section has a full specification of these.

Test status byte

The test status byte switches on and off various functions for test purposes. To change a particular bit, a new hex string must be calculated. One of the most important bit switches is bit 7, which when set disables the automatic lock control state machine, and enables the 1PPS to be manually controlled.

IMPORTANT The test status byte is maintained in EPROM. If the command EU (EPROM update) is used when the test status byte has been manually altered, the unit may not work after the next power on reset. If in doubt, the test status byte should be set to 00hex, and the EU command executed.

Lock status byte

The lock status byte monitors the operation of the state machine. Bits 0, 1, 2 indicate the current internal state. Bits 4, 5 are read only bits which indicate the state of the OCXO and PLL. Bits 6, 7 are automatically controlled by the state machine, but may be set and reset by the user for test purposes. In order to avoid overwrite by the state machine; bit 7 of test status should be set.

Output status byte

The output status byte monitors the operation of the 1PPS time tag and the Kalman phase estimate.

Tune voltage span (1 byte)

The tune voltage span can be set to suit the OCXO. A value of 00h will give a span of 0 to 10V, and a setting of FFh a span of 0 to 5.8V.

After setting the value should be stored in EPROM using the "EU" command.

OCXO current (2 bytes).

These bytes are read only and give the filtered value of the OCXO supply current. The range is 0 to 500mA. The digital filter has a 3dB bandwidth of about 5mHz, so the value responds quite slowly to a change in OCXO current. The value is compared to a threshold to detect OCXO warm up. When the current falls below the threshold, the state machine will move to state 1 and try and lock the loop. The OCXO threshold is not adjustable without reloading the firmware.

Run Time (2 bytes)

The run time bytes are the power on time of the microcontroller since the firmware was programmed. This is in units of 18.2 hours. This parameter cannot be changed using the interface. (but see "ER" command).

PO (1PPS Output)

This command monitors the current state of the 1PPS output. It also enables user adjustment of some of the parameters.

“PO?” returns 4 hex strings of different lengths.

Modulo50 counter byte

20ms range 0 to 49.

Main counter (2 bytes)

400ns range 0 to 49999.

Delay setting byte

100ns range 0 to 4 delay 50 to 450ns.

Vernier byte

0.5ns range 0 to 255.

PD (1PPS Output)

This command returns the current user 1PPS offset in decimal ns.

TO (Timetag)

This command returns the current time tag parameters

"TO?" returns 5 hex strings of different lengths.

Last time tag (4 bytes)

First capture (2 bytes)

Second capture (2 bytes)

Second capture – First capture (2 bytes)

Modulo50 capture byte

PM (Performance Measure)

This command returns the last time tag, the last Kalman phase estimate, measurement error, filter performance, multiplier and frequency estimate.

"PM?" returns 1 decimal, 2 floating point and 3 hex strings of different lengths

Last time tag (decimal)

Last time tag in decimal ns 9 digits.

Last Kalman phase estimate (floating point)

Measurement error (3 bytes)

(Kalman phase estimate minus the current time tag)^2 and filtered in an 8th order exponential filter. Units are ns^2.

Filter performance (2 bytes)

The square of the current fractional frequency estimate divided by the Kalman filter P22 element of the error covariance matrix multiplied by 2048. The normal running value is 0 to 100.

Multiplier (2 bytes)

The constant used to multiply the S1 system noise parameter (random walk FM). This results in a recalculation of the Q matrix (system covariance matrix). This adjusts the filter for expected increased drift rate of the controlled oscillator

EU (EEPROM update)

This command writes all the parameters that are currently in volatile memory to EEPROM memory.

These include:

clock registers
test status byte
tune voltage span

The clock registers are automatically written to EEPROM every 18.2 hours providing the PLL is locked

SR (software reset)

This causes a reset equivalent to a power on reset without the need to cycle the power

ER (EEPROM read)

This is a read from any part of the EEPROM, either as ASCII characters or as hex numbers

The command "ERC0000" will dump the entire EEPROM memory to the interface.

This command is usually used by a Windows application which can be used to read and write to the scratchpad part of the EEPROM, where serial number and other production data is stored.

EW (EEPROM write)

This command is usually used by a Windows application for updating the scratchpad part of the EEPROM

RI (repeat interval)

This command is usually used by a Windows application to send continuous data. Any command with the permitted form XX+ may be written to a repeat stack which repeats the command XX? at a rate determined by the RI command. Commands may be mixed in the repeat stack.

The command RID clears all commands in the repeat stack

The command RI0aa where aa is one byte sets the repeat interval in units of 50ms.

3 Configuration of board

In order to configure the A6-1PPS board for a new controlled oscillator (whether mounted on or off the board), the following must be considered:

3.1 Board Connections

In order to aid the use of the A6-1PPS below is the signal description for the connectors, test points and links.

Identifier	Pin	Type	Name	I/O	Description
CONN1		MCX	EXT IN	I	External reference oscillator input
CONN2		MCX	1PPS IN	I	1PPS input (CONN5 not fitted)
CONN3		MCX	1PPS OUT	O	1PPS output (CONN6 not fitted)
CONN4		MCX	RF OUT	O	10MHz RF Output
CONN5		BNC	1PPS IN	I	1PPS input (CONN2 not fitted)
CONN6		BNC	1PPS OUT	O	1PPS output (CONN3 not fitted)
CONN7		BNC	RF OUTPUT	O	10MHz RF output
J1	Pin	6mm	VCC	PWR	Voltage supply input
	Outer	6mm	GND	PWR	Power supply ground
JP1		FPC	uP		Factory use only
JP2	1	D9M	LOCK	O	1PPS lock signal
	2		GND	PWR	Power supply ground
	3		GND	PWR	Power supply ground
	4		RXD	I	DPLL input refer to manual for proprietary messages
	5				Not used
	6		TXD	O	1PPS output refer to manual for proprietary messages
	7				Not used
	8		1PPS IN	1	1PPS input
	9		1PPS OUT	O	1PPS output
LINK1	1	LINK	INT		On board OCXO
	2		EXT		External oscillator
Test point	1		TEST	O	Test output
	2		1PPS IN	I	1PPS input
	3		VCC	PWR	Voltage supply input
	4		GND	PWR	Power supply ground
	5		1PPS OUT	O	1PPS output
	6		TEST	O	Test output
	7		CAP		Capture
	8		INT	O	Integrator output
	9		OSC CTRL	O	External oscillator control voltage
	10		DAC	O	Tune DAC
	11		I	O	On board OCXO Current Monitor

4 Specification

CONTROLLED OSCILLATOR REFERENCE INPUT

FREQUENCY:	10MHz
INPUT LEVEL:	100mV PP to 5VPP (oscillator off board)
INPUT IMPEDANCE:	500 OHMs
OUTPUT LEVEL:	13±2 dBm (oscillator on board)

1PPS INPUT

LEVEL	5V TTL/CMOS positive edge
WIDTH	10us minimum
INPUT IMPEDANCE	1000 Ohms

1PPS OUTPUT

LEVEL	5V TTL/CMOS positive edge
WIDTH	10ms
PRESET OFFSET	-500000000 to +499999999ns in 1ns steps
TIMING BASELINE	selectable between fixed (minimum jitter) or Kalman phase estimate (maximum accuracy)

EXTERNAL TUNE VOLTAGE

0 to SPAN, where SPAN is software adjustable between 5.8V and 10V

POWER SUPPLY:

14 to 30V	(on board OCXO is used)
12 to 30V	(no on board OCXO)

CURRENT CONSUMPTION:

150mA	typical (on board OCXO)
50mA	typical (no on board OCXO)

LOCK INDICATOR

On	not locked
Off	locked, low phase error
Short flash every second:	locked, high phase error

INTERFACE

9.6kbaud, RS232, PC compatible

The control performance depends very much on the quality of the controlled oscillator and the source of the 1PPS synchronizing signal. For these reasons it is difficult to quote absolute performance figures.

The following cases are typical:

a) Controlled oscillator: Rubidium

1PPS source: passive hydrogen maser (essentially no 1PPS jitter)

Result:	Allen variance
100s	$1x10^{-12}$
1000s	$3x10^{-13}$
10000s	$1x10^{-13}$

b) Controlled oscillator: Rubidium

1PPS source: Navsync GPS receiver in position hold mode

Result	Allen variance
100s	$1x10^{-12}$
1000s	$1x10^{-12}$
10000s	$8x10^{-13}$

5 Appendix A – RS232 Control Codes

RS232 control codes (all values following command or returned from the microcontroller are hexadecimal unless stated) * = backed up in EEPROM

OS **Overall Status**

OS? returns overall status bytes:

aa bb cc dd eeee ffff

	aa	*is test status byte: bits set:*	*bit0:*	*50PPS output (default 1PPS)*
			bit1:	*50PPS input (default 1PPS)*
			bit2:	*disable auto cal*
			bit3:	*disable 1PPS output*
			bit4:	*disable 1PPS input time tag*
			bit5:	*disable frequency correction (Kalman filter updated)*
			bit6:	*disable Kalman filter update*
			bit7:	*disable state control*
	bb	*is lock status byte: bits set:*	*bit0, 1, 2*	*state (0 to 7)*
			bit3	*normalise DACs (cleared automatically)*
			bit4	*OCXO warmed up*
			bit5	*loop locked*
			bit6	*NU*
			bit7	*zero clock on next 1PPS capture*
*	*cc*	*is output status byte: bits set:*	*bit0*	*1PPS base = last input time tag*
			bit1	*1PPS base = Kalman phase estimate otherwise 1PPS base is zero*
			bit2, 7	*NU*
*	*dd*	*is reference tune span*	*00 to FF*	*(00 = 10V, FF= 5V)*
	eeee	*is OCXO current*	*0000 to FFFF*	*0 to 500mA*
*	*ffff*	*is running time*	*0000 to FFFF*	*unit 18.2 hours*

OSTaa write new test status byte

OSLbb write new lock status byte

OSPcc write new output 1PPS status byte

OSSdd write new reference tune span byte

PO **1PPS OUT**

PO? returns current status of 1PPS output

aa bbbb cc dd

aa	*modulo50 counter (20ms) range 0 to 49*
bbbb	*main counter (400ns) range 0 to 49999 (C34Fh)*
cc	*delay setting (100ns) range 0 to 4 (delay 100 to 500ns)*
dd	*vernier (0.5ns) range 0 to 255*

POAaa write new modulo50 counter

POBbbbb write new main counter

POCcc write new delay setting

PODdd write new vernier

PD **1PPS OUT**

PD? returns current user 1PPS offset in decimal ns

(-)aaaaaaaaa

* *(-)aaaaaaaaa* *last user 1PPS offset,* *-500,000,000 to 499,999,999 ns*

PD(space)(string)(carriage return) inputs new user 1PPS offset in floating point format e.g. to set the offset to 500ns delay "PD .000000500"

TO **TIMETAG**

TO? returns last time tag parameters

aaaaaaaa bbbb cccc dddd ee

aaaaaaaa	*Last time tag (32 bit)*
bbbb	*first capture*
cccc	*second capture*
dddd	*second capture-first capture*
ee	*Modulo50 capture*

TO+ write TO? to command repeat stack

PM **PERFORMANCE MEASURE**

PM? returns last time tag in decimal ns, and other performance indicators

(-) aaaaaaaaa b (fp string) cccccc dddd eeee f (fp string)

(-) aaaaaaaaa	*Last time tag in ns (9 digits)*
b (floating point)	*Last Kalman phase estimate*
ccccccc	*Mean square measurement error (ns squared) (decimal integer)*
dddd	*Filtered performance indicator 0 to 32768 (decimal integer)*
eeee	*S1 multiplier (decimal integer)*
f (floating point)	*Mean of frequency estimate*

PM+ write PM? to command repeat stack

Notes:

The performance indicator is the square of the current fractional frequency estimate divided by the P22 element of the error covariance matrix multiplied by 2048. The normal running value is 0 to 100

The mean square measurement error is the (Kalman phase estimate minus the current time tag) ^2 and filtered in an 8th order exponential filter. Units are ns^2.

The S1 multiplier is the constant used to multiply the S1 system noise parameter (random walk FM). This results in a recalculation of the Q matrix (system covariance matrix). This adjusts the filter for expected increased drift rate of the controlled oscillator.

The mean of the frequency estimate is the Kalman frequency estimate filtered by a 5th order exponential filter

KP **Kalman filter error covariance matrix**

* KP? returns current error covariance matrix

P11	*P12*	*P13*	
	P22	*P23*	
		P33	*in floating point format*

KPxx (space) (string) (carriage return) inputs a new floating point value, where xx identifies matrix element

KP+ write KP? to command repeat stack

KQ **System covariance matrix**

KQ? returns current system covariance matrix

Q11	*Q12*	*Q13*
	Q22	*Q23*
		Q33

Note Q matrix is calculated from system noise parameters on reset, and also in states 2, 3 and 5

KX **System state vector**

* KX? returns current system state vector

X1 *X2* *X3*

Note X1 is current phase estimate, X2 is current frequency estimate, and X3 is current drift estimate

KX+ write KX? to command repeat stack

KS **System noise parameters**

* KS? returns current system noise parameters

S1 *S2*

Note S1 is the level of random walk FM noise, and S2 the level of white FM noise. These are fixed parameters that describe the oscillator being modelled.

KSx (space) (string) (carriage return) inputs a new floating point value where x is the matrix element

KZ **Measurement parameters**

* KZ? returns the latest measurement, and the current measurement noise parameter

Z *R*

Note Z is the last measurement in seconds, and R is the expected variance of the measurement noise in seconds squared.

KZ1 (space) (string) (carriage return) inputs a new R

OC **Oscillator control parameters**

* OC? returns latest oscillator control constant and maximum tuning voltage

OC1 *OC2*

Note OC1 is oscillator tuning constant in fractional frequency /volt, and OC2 is the maximum tuning voltage in volts, assumed 0V minimum.

OCx (space) (string) (carriage return) inputs new OC1 or OC2 where x is the parameter

OT	**Oscillator tuning control**	
	OT?	returns tuning word and DAC values
	aaaaaa bbbb cccc	
*	*aaaaaa*	*24 bit tuning word*
	bbbb	*coarse tuning DAC value*
	cccc	*fine tuning DAC value*
	OTTaaaaaa	inputs new tuning word
	OT+	writes OT? to command repeat stack
CA	**Auto cal parameters**	
	CA?	returns auto cal parameters
	aaaa bbb cccc dddd	
	aaaa	*filtered CAL100ns value*
	bbbb	*filtered CAL500ns value*
	cccc	*last CAL100ns value*
	dddd	*last CAL500ns value*
EU	**EEPROM update (backed up values from RAM)**	
ED	**load EEPROM and RAM with default values**	
SR	**Software Reset**	
ER	**EEPROM read**	
	ERCaaaabb	*returns bb bytes from starting address aaaa as ASCII characters*
	ERNaaaabb	*returns bb bytes from starting address aaaa as hexadecimal numbers (character pairs)*
EW	**EEPROM write**	
	EWCaaaabbccccc------c	
		writes bb characters to starting address aaaa. Correct number of characters must be included in string
	EWNaaaabbcccc------c	
		Writes bb bytes to starting address aaaa. Character pairs cc etc are interpreted as hexadecimal numbers.
RI	**Repeat Interval**	
	RI?	returns command repeat interval
	aa	
	aa	*8 bit command repeat interval multiplier. Range 1 to 255. Command repeat interval is 40ms x aa*
	RI0aa	write new command repeat interval
	RID	cancel command repeat and clear command repeat stack