BlueFlyVario
Hardware Settings Manual

v1.5 - Jan 2016

Contents

Introduction

The BlueFlyVario is a flight instrument for measuring vertical speed and altitude based on atmospheric pressure. It consists of a simple hardware device that measures the atmospheric pressure 50 times every second, with a resolution that enables the measurement of altitude differences as small as 10 cm.

A number of prototype models of the BlueFlyVario are available. See www.blueflyvario.com for details.

This manual describes the settings which are stored on the BlueFlyVario, how to set them, and what they control. These settings are known as the hardware settings

Hardware Settings

Overview

The Microchip PIC microcontroller on the BlueFlyVario has an onboard memory (EEPROM) which is used to store hardware settings. The hardware settings are read from the onboard memory when the device is powered on. When a hardware setting is changed by the user the value is used by the onboard code immediately, and the setting is stored in the onboard memory.

Settings can be altered by powering on the BlueFlyVario Hardware and connecting to a host computer or mobile device. The host device needs to run an application which connects to the BlueFlyVario using a serial protocol. This might include any of the following methods:

BFV Hardware	Host Device	Host Device Application
BlueFlyVario Bluetooth Versions	Android Device	BlueFlyVario app
BlueFlyVario Bluetooth Versions	Android Device	Third Party app which supports hardware settings
Any BlueFlyVario version	Desktop or Laptop computer running Windows or Linux, with an Bluetooth adapter or USB to Serial converter	BFV Desktop Java Application
Any BlueFlyVario version	Desktop or Laptop computer running Windows or Linux, with an onboard Bluetooth adapter or USB to Serial converter	Terminal Application such as RealTerm (using the raw serial protocol described in the serial communication section)
Any BlueFlyVario hardware which can connect to a device which supports LK8000	Any device which supports LK8000 and can connect to the BlueFlyVario TTL or Bluetooth versions	LK8000 (most recent versions as of Mar 2014)

A comprehensive list of the hardware settings is included as Annex A.

General Settings

A number of general settings are self explanatory:

useAudioWhenConnected (default = false). If true the hardware audio will sound when the device is connected via Bluetooth. *Note: This setting is not used on non-Bluetooth versions of the BlueFlyVario.*

useAudioWhenDisconnected (default = true). If true the hardware audio will sound when the device is not connected via Bluetooth. *Note: This setting is used on non-Bluetooth versions of the BlueFlyVario as the overall setting to control if audio is on or off.*

positionNoise (default 0.1). This setting controls one of the parameters for the Kalman Filter that is built into the hardware code. A higher value gives less sensitivity by smoothing out the noisy pressure measurements. Try 1.0 to see the difference.

secondsBluetoothWait (default 180). This setting controls how many seconds the hardware will keep its bluetooth radio on while waiting for a connection. *Note: This setting is not used on non-Bluetooth versions of the BlueFlyVario.*

greenLED (default = true) [from v9]. This setting controls if the greenLED will light for each lift beep.

Auto Power Off

From v10 the BlueFlyVario models incorporate a button for soft power on/off instead of the slide switch used on previous models. If the vario is not moved vertically for a period of time then it will shutdown. The setting ***heightSensitivityDm*** controls how many decimetres the vario must be moved in order to reset the auto power off flag. By default that is set at 20dm (equals 2m). After ***heightSeconds*** of no movement greater than ***heightSensitivityDm*** the vario will shutdown.

Output Modes

From version 7 the BlueFlyVario can output pressure and vario data in a number of modes. This makes the vario compatible with a wider range of applications.

outputMode (default = 0) - Sets the output mode. The available output modes are:

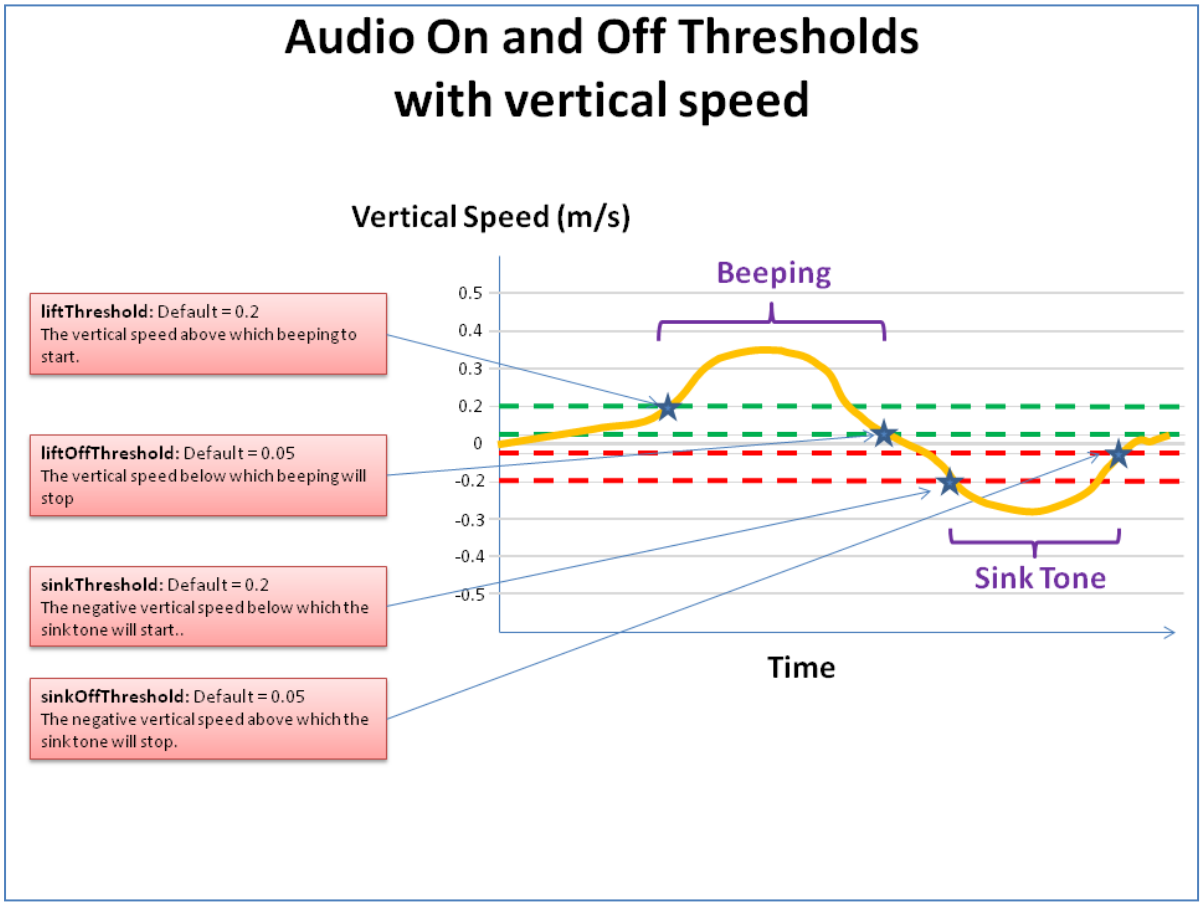
- 0 - The standard BlueFlyVario output mode. This sends raw pressure measurements in the form: "**PRS XXXXX\n**": XXXXX is the raw (unfiltered) pressure measurement in hexadecimal pascals.
- 1 - The LK8EX1 output mode for use with LK8000. This sends pressure and vario data in the form: "**$LK8EX1,pressure,altitude,vario,temperature,battery,*checksum\r\n**": pressure is sent as a decimal integer number of pascals, altitude is not sent (99999 is sent instead), vario is the decimal integer vertical climb rate in cm/s, temperature is in degrees Celsius (1 decimal place), and battery is the battery voltage of the on-board battery (2 decimal places).
- 2 - The LXWP0 output mode for use with a range of apps: **"$LXWP0,loger_stored (Y/N), IAS (kph), baroaltitude (m), vario (m/s),,,,,,heading of plane,windcourse (deg),windspeed (kph)*checksum\r\n** ": The BlueFlyVario only has a partial implementation of this sentence. It only outputs the baroaltitude and vario (all other fields are blank). Note that baroaltidude is determined from filtered pressure using the outputQNH setting.
- 3 - The FlyNet protocol: **"_PRS XXXXX\n**": In this case XXXXX is output as the filtered pressure stream. The filtering parameters used are those from the other hardware settings.
- 4 [from v9 hardware] - No output.
- 5 [from v11 hardware] - Custom BFV sentence: This sends a NMEA like sentence in the following format: **"$BFV,pressure(Pa),vario(cm/s), temp(deg C), battery(%),pitotDiffPressure(pa)*checksum\r\n"**. Pressure (the filtered pressure as an unsigned integer), vario (the filtered vario as an signed integer) and temp(signed float) are always sent. Battery % (unsigned integer) is only sent for models which include a battery; otherwise "0" is sent. pitotDiffPressure (signed integer) is only sent when the hardware setting usePitot is enabled.
- 6 [from v11 hardware] - Custom BFV extended sentence: This sends a NMEA like sentence in the following format: **"$BFV,pressure(Pa),vario(cm/s), temp(deg C), battery(%),pitotDiffPressure(pa), volts(V)*checksum\r\n"**. As per mode 5, but also includes the raw battery voltage. Note that this output mode is most likely to change as the hardware evolves.

outputFrequency (default = 1). Sets the frequency of output sentences from the BlueFlyVario. The BlueFlyVario hardware runs on a 20ms cycle (50 cycles per second). If ***outputFrequency*** is set to 1 then the hardware will send a sentence on each cycle. If set to 2 it will send a sentence every second cycle and so on (if set to 50 is will send a sentence every 50th cycle, i.e. once per second). You might use this with the LK8EX1 output mode to send a sentence five times per second (set to 10).

outputQNH (default = 101325). See outputMode = 2 above.

Audio Thresholds

The sound on the BlueFlyVario hardware switches on and off based on the measured vertical speed. The settings ***liftThreshold***, ***liftOffThreshold***, ***sinkThreshold*** and ***sinkOffThreshold*** control when the sound comes on and off. The graph below describes how this works and what the defaults are. It is ok to set the on and off thresholds to the same value, but you will get funny results if the off threshold is higher than the on threshold.



Essentially what happens as you enter lift is that the vario senses a decreasing pressure. Note that there is a slight delay between the movement and the sensed lift due to the way the filtering works. An updated value of filtered lift gets calculated from each pressure measurement, 50 times a second (or every 20 ms). The first time the lift goes above the ***liftThreshold*** a beep commences for a duration controlled by the beep cadence formula (see below). That beep will last for as long as the beep cadence, then a period of silence is commenced, this time based on the most recently measured lift duration. Only at that time 'beeping' could end, but if the lift continues to be above ***liftOffThreshold***, then another sequence of beep followed by silence will be played.

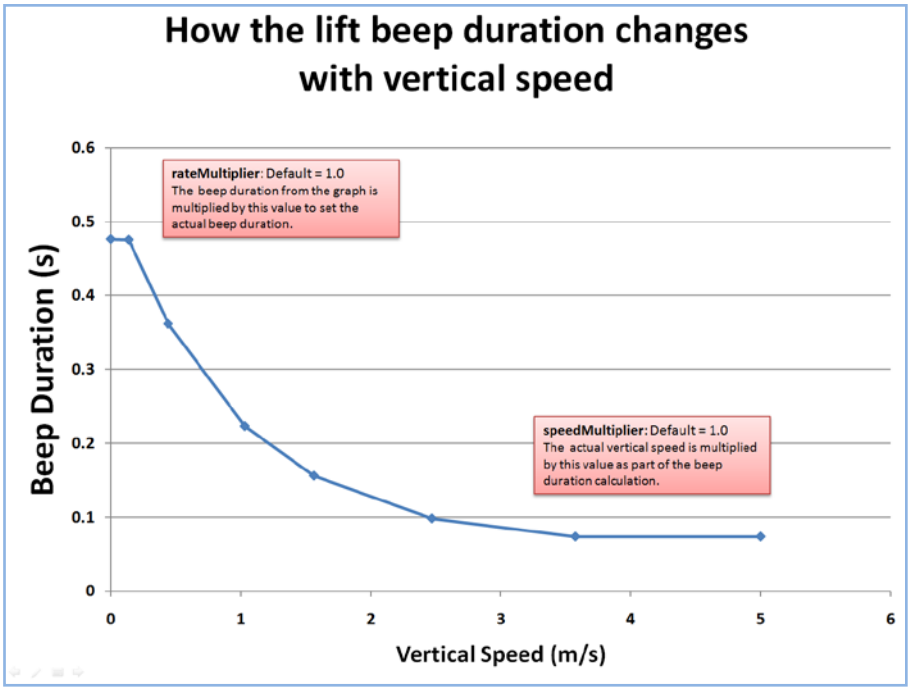
Audio Buzzer

From version 10 (firmware version 10.m04) there is an experimental Audio Buzzer that can be used to indicate light lift. The audio buzzer provides a fast beeping indication. If ***useAudioBuzzer*** is true then it will start when the rate of climb is ***buzzerThreshold*** m/s below the ***liftThreshold***.

Beep Cadence

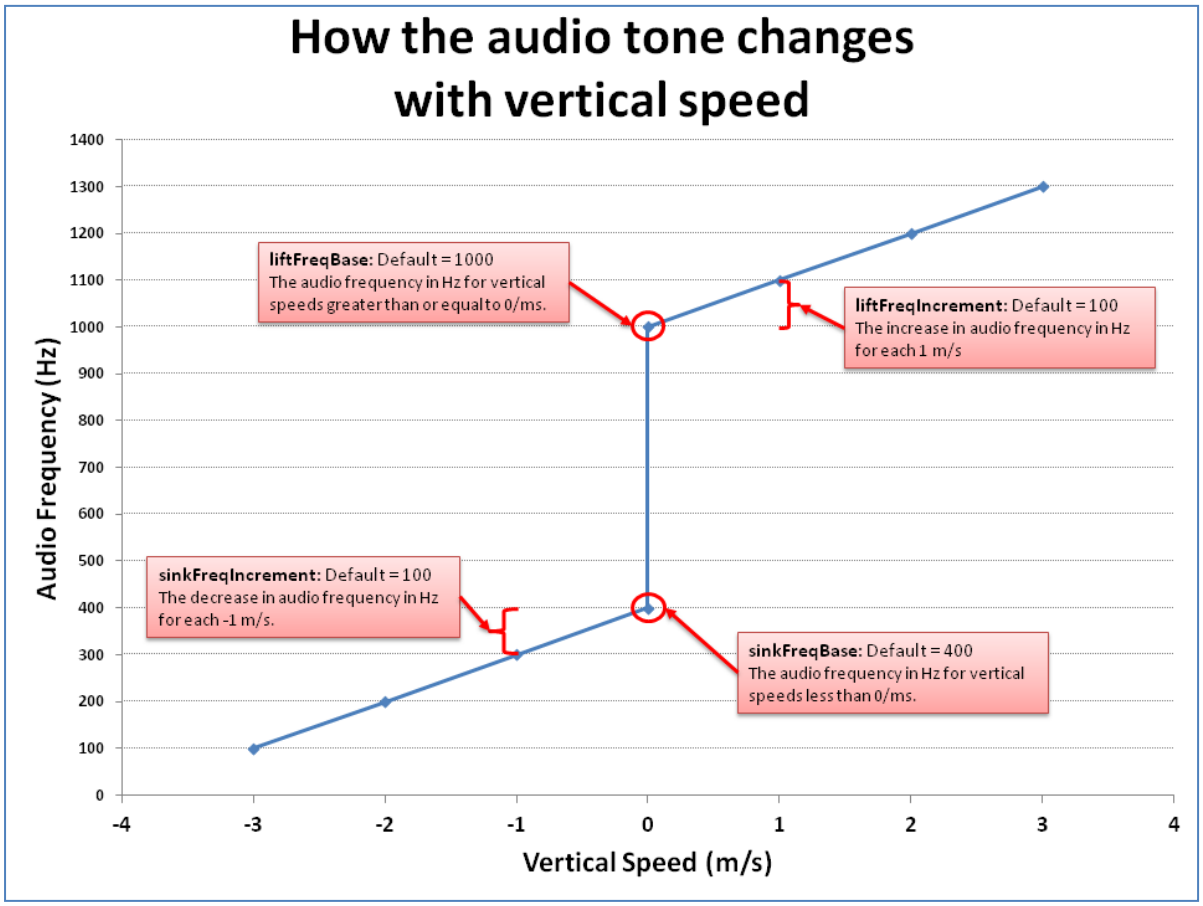
When beeping commences a beep is initiated for a duration based on the measured vertical speed. As soon as the beep stops a silent pause will then be initiated, again for a duration based on the measured vertical speed. The graph below shows the formula used to control the beep duration based on the measured lift. Essentially, it means the faster you are going up, the faster the beeps will occur. Note the ***rateMultiplier*** settings can make it go faster as shown below. A ***rateMultiplier*** setting of 0.5 will make it beep twice as fast.

From firmware 10.6 ***speedMultiplier*** will alter the beep duration by adjusting the actual speed according to the value in the graph below. A ***speedMulitplier*** of 2.0 will make the beep duration half as sensitive to vertical speed changes.



Audio Tone

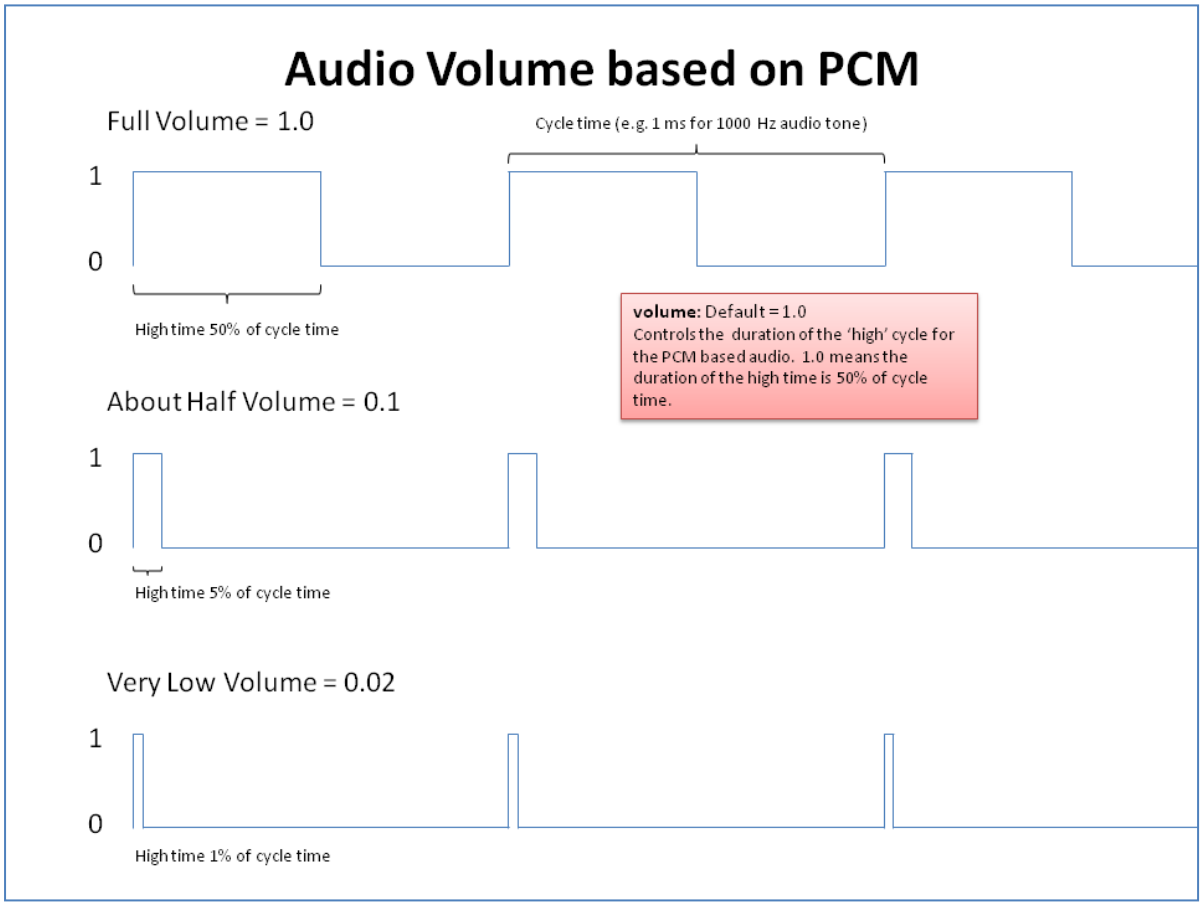
In addition to the lift and sink thresholds and cadence, the audio tone is adjusted based on the measured vertical speed by the four settings ***liftFreqBase***, ***liftFreqIncrement***, ***sinkFreqBase*** and ***sinkFreqIncrement***. The graph below shows how the audio tone (frequency) changes based on these settings, and also shows what the defaults are. The frequency of the sound is constantly being updated to control the tone as the filtered vario value changes. This occurs with every measurement (every 20 ms). The result is changes of tone while beeps are playing based on constantly updated filtered vario values. On occasion the sink tone will sound at the end of a beep if the filtered vario value changes to below the ***sinkThreshold*** while a beep being played.



Note that in the most recent versions of the BlueFlyVario (from v7 onwards), the Audio Frequency cannot be played below 125Hz. If the combination of the settings and the measured vertical speed results in a calculated audio frequency below 125Hz the resulting sound will be clamped to 125Hz.

Volume

The BlueFlyVario uses an electromagnetic transducer. These devices are driven by a square wave from a microcontroller pin (using the inbuilt PWM module of the microcontroller). Using the trick described in the graph below the volume is controlled without needing a variable resistor. Think of the transducer diaphragm being 'kicked' by a high pulse. The more gentle the 'kick' the quieter the sound. The ***volume*** setting is not linear.



Altering the volume setting also subtly alters the audio tone.

Some Notes

The settings selected as defaults are probably not the best for flying, but do allow easy testing of features during assembly. Please provide feedback on what settings work for you and why.

The trick to a well performing vario is to get a combination of ***positionNoise*** and ***liftThreshold*** that works best. If you have almost no filtering (i.e. a low ***positionNoise*** of 0.01) then the calculated vario value will be very noisy. In this circumstance the audio threshold would need to be set at 0.4m/s or higher to avoid errant beeping. These settings would be good if we wanted the vario to be ultra responsive to really jerky movements. However, for flying we bounce around a bit more gently. A ***positionNoise*** of 0.1 with an ***audioThreshold*** of 0.2 seem to work pretty well for most pilots.

Most pilots will probably want to adjust the ***sinkThreshold*** setting to less than their glider sink rate, so it is not on all the time. The ***sinkOffThreshold*** should be adjusted at the same time.

Battery life of the bluetooth model will be affected by the settings chosen. The largest consumers of power are the Bluetooth radio and the electromagnetic transducer. Having both on full time would reduce battery life to less around 8 hours. Most people want the vario to beep when they are in real lift, and the sink tone to only come on when they are in significant sink (-2.0 m/s or so). This will mean that the audio will only be sounding for less than 50% of most flights, which would give over 10 hours battery life when not connected via Bluetooth.

Serial Protocol

Primary Serial Port

A serial port on the Microcontroller (UART 2) is used to output data from the vario and send commands to it. For Bluetooth versions of the BlueFlyVario the serial port is connected to the onboard RN-42 Bluetooth Adapter (or RN4677 from v11 hardware). Bluetooth drivers on host devices set up a virtual serial port so the data transfer is seamless. For the BlueFlyVario_TTL the UART Tx and Rx pins are directly exposed.

The settings for the primary serial port are:

Baud Rate:	57600 (changed to 115200 in v11 hardware)
Data Bits:	8
Stop Bits:	1
Parity:	No Parity
Flow Control:	None
Voltage (for TTL):	Nominally 3.0 Volts

The setting ***uart2BRG*** is used for altering the baud rate of U2 (the main output). This should really only be used with extreme care, but there are a few cases where it is the only way to communicate with a device connected on uart1 (like a GPS) via the microcontroller.
The baud rate is calculated according to the formula:

Baud Rate = 2000000/(uart2BRG+1)

For the default value of 34 this results in a baud rate of ~57143 which is about equal to 57600. The integer setting which is closest to the desired baud rate should be used.

Serial Commands for Settings

Each of the settings described in the Hardware Settings section can be set by sending raw serial commands in ASCII format. The protocol is:

$BXX INT*

The **$** symbol starts a command.

This is followed by the command code **BXX**. A list of all available command codes associated with settings is included in Annex A.

This is followed by the space character.

This is followed by a positive integer **INT** in the range described in the Integer Values section of Annex A. The integer value is converted on the BlueFlyVario in a number of different ways depending on the Type and Factor associated with the setting in the following way:

int:	Converted value = Integer value
Boolean:	Converted value = FALSE if Integer value = 0, and TRUE otherwise
double:	Converted value = Integer value * Factor
int_offset:	Converted value = Integer value + Factor

The ***** symbol finishes a command.

Other Serial Commands

Serial commands include:

$BST* (BlueFlyVario Settings) - This command is responded to with the following information (in version 8 – there are slightly different responses in earlier firmwares):

BFV [*VersionNumber*] \r\n
BST [*followed by a space separated list of each of the settings codes*]
SET [*followed by a space separated list of each of the settings Integer Values*]

$TMP* (Temperature) - This command is responded to with the following information:

TMP [*Integer temperature value * 10*] \r\n

$RST* (Reset) [from v9] - This command resets the module, essentially a hot reboot.

$RSX* (Reset Settings) [from v9] - This command resets the module and reset all of the hardware settings to defaults. It is an extended hot reboot, equivalent to start up with programming pads 2 and 4 shorted.

$SLP* (Sleep)[from v9]. This command sends the module to sleep mode. Note it also sends commands over U1 to send the the PA6H GPS to sleep mode using the PMTK standby command. The microcontroller plus pressure sensor consumes about 0.05mA in sleep mode, but the GPS still consumes about 1.5mA. To wake from sleep mode without a power cycle you just send any character to the module on U2. This will then force a hot reboot, and you are back to the power on state, including the GPS.

$SLX* (Sleep)[from v9 (rev2)]. This command sends the module to sleep mode. Note it also sends commands over U1 to send the the PA6H GPS to sleep mode using the PMTK standby command. The microcontroller plus pressure sensor consumes about 0.05mA in sleep mode, but the GPS still

consumes about 1.5mA. In this sleep mode the module cannot be woken with a serial command. The devices needs to be power cycled to wake it up.

$BSD FFF DDD* plays a tone of frequency FFF Hz for a duration of DDD ms.

- **FFF** can be any frequency from about 150 Hz to about 8000 Hz. Outside of this range the BlueFly speaker does not work well. Note that different volume settings will alter the apparent pitch of the sound. If you send a frequency of 0 then the BlueFly will be silent for the specified duration.
- **DDD** is the duration in ms. You can send any integer up to about 32000 (i.e. 32 seconds). The duration will be rounded up to the nearest 10 ms.
- While the tone is playing other sounds from the BlueFly will be silenced.
- You can send the command while another sound is playing and it will be added to a queue.
- For example, the command $BSD 400 100* will play a tone at 400Hz for 100 ms.

$BSD FF1 DD1 FF2 DD2 ... ... FFN DDN* will play a sequence of sounds the one command.

- The full length of the command including $ and * is limited to 82 characters, so you can fit about 7 to 10 sounds in depending on duration and frequency.
- Note that freq/ms pairs sent (in one command, or in a series of commands) will be put onto the queue and played in the order they were put in. The queue length is 30 10 *[Edit: I updated this in 10.m08]* so you can play some short tunes.
- For example, the command $BSD 400 100 0 150 1000 50* will play a tone of 400 Hz for 100ms, followed by silence for 150 ms, followed by a tone of 1000 Hz for 50 ms.

$BVU* will double the current volume (clamped to the maximum of 1000). You get a short beep each time you send the command to indicate the new volume.

$BVD* will halve the current volume (clamped to the minimum of 1). You get a short beep each time you send the command to indicate the new volume.

$BTN* in 10.m08 which simulates pressing the hardware button on the vario.

Second Serial Port

From version 8 a second serial port is available as an input, and from version 9 you can also send data to it. This serial port exposes UART1 from the PIC Microcontroller. The key parameters are:

Baud Rate:	9600 (by default – see ***uart1BRG*** below)
Data Bits:	8
Stop Bits:	1
Parity:	No Parity
Flow Control:	None
Voltage (for TTL):	Nominally 3.0 Volts

The second serial port is set up to receive NMEA GPS sentences and send them out on UART2 (the standard output). In this way the BlueFlyVario effectively multiplexes the GPS data with the standard output controlled by the ***outputMode***. This allows the BlueFlyVario to output a single stream of data which can be read by XCSoar or other applications.

Data received on the Rx line of UART1 is processed in the following way (if the setting ***uart1Raw*** is false). Any series of ASCII characters starting with **$** and ending with **\n** are recorded as a sentence. The sentence is then scheduled for transmission at the next available 20 ms cycle. Note that it is probably possible to overload the device with too many sentences. Standard GPS output at 1Hz (with about five sentences sent each time) seems to work well.

The setting ***uart1BRG*** (default = 207) controls the baud rate on UART1. The baud rate is calculated according to the formula:

Baud Rate = 2000000/(uart1BRG+1)

For the default value of 207 this results in a baud rate of ~9615 which is about equal to 9600. The integer setting which is closest to the desired baud rate should be used.

From version 9 additional hardware settings control how data is passed between UART1 and UART2:

uartPassthrough (default = true) - This setting is used for passing characters received on U2 through the micro to U1. Due to different baud rates (U2 > U1) the characters are stored in a FIFO buffer of length 100 which is long enough to allows for normal commands to be sent to devices like a GPS. If you need to send bulk data into U2 and have it sent to U1, like a firmware update for a GPS for example, then you will need to adjust the baud rates of U1 and U2 so the buffer does not overflow.

uart1Raw (default = false). Normally sentences received on U1 are processed as described above. When uart1Raw is set to true the data is processed on a character by character basis. Characters received at U1 are passed straight through to U2.. This is used to allow bulk transfer of information that is sent by GPS other than standard NMEA sentences. If you set this to true be aware of the different baud rates (if you have set U1 > U2 then you will get buffer overflows), and also be aware that if the ***outputMode*** is anything other than mode 4 (off) then you will get a very unintelligent multiplexing of the U1 received characters with the pressure output.

Shields

From version 11 you can expand the BlueFlyVario with shields.

BlueFly_v11_Shield_GPS

The BlueFly_v11_Shield_GPS adds a PA6H GPS on UART1 (U1) of the microcontroller. The GPS shield has a few extra components to smooth the power supply to the GPS and provide the PA6H LED

output. Note that the BlueFly button protrudes through the hole in the shield. The GPS works in exactly the same way as the GPS on the TTL_GPS models of the BlueFly:

- Any sentences coming from the GPS which begin with $ and ending with new line (\n) are echoed out on U2 by multiplexing the with the standard BlueFly output.
- The BlueFly does nothing with the GPS information other than pass it through.
- At this stage XCSoar is the only application which reads both the BlueFly output (in a few different modes) and the standard GPS sentences from the same data stream.

BlueFly_v11_Shield_Airspeed

This shield connects the MS4525DO pressure sensor to the BlueFly I2C. The MS4525DO upper port is connected to the pitot pressure and the lower port is connected to the static pressure. The included pitot tube and clear tubing is a cheaply available RC model tube and will need modification at the nose to make it work properly as when it is machined the pitot hole gets a little closed. You will have to tinker with the physical layout of the tube to get it accurate, although note that there is no way to get accurate airspeed below about 10 to 15 km/hr. Some additional notes:

- The BlueFly needs a special hardware setting adjusted. To adjust usePitot send $BUP 1* to the BlueFly via the normal manner (or tick the box in the BFVDesktop application). This makes the BlueFly read this pitot sensor each cycle via I2C and adjust the sent data appropriately. This will be described fully in the manual update; but in the interim note that the LX mode sends the Indicated Airspeed when usePitot is set.
- When the BlueFly starts up with usePitot enabled it immediately begins a pitot calibration. This lights the red LED on the shield, then takes about five seconds of pitot data. The differential pressure measured is averaged and taken to be zero airspeed.
- You can trigger another calibration at any time by pressing the red button on the shield next to where the normal BlueFly button protrudes. It is important that the pitot tube is in no wind when the calibration is underway.
- If you set usePitot, but the shield is not connected, the BlueFly will send an error message.

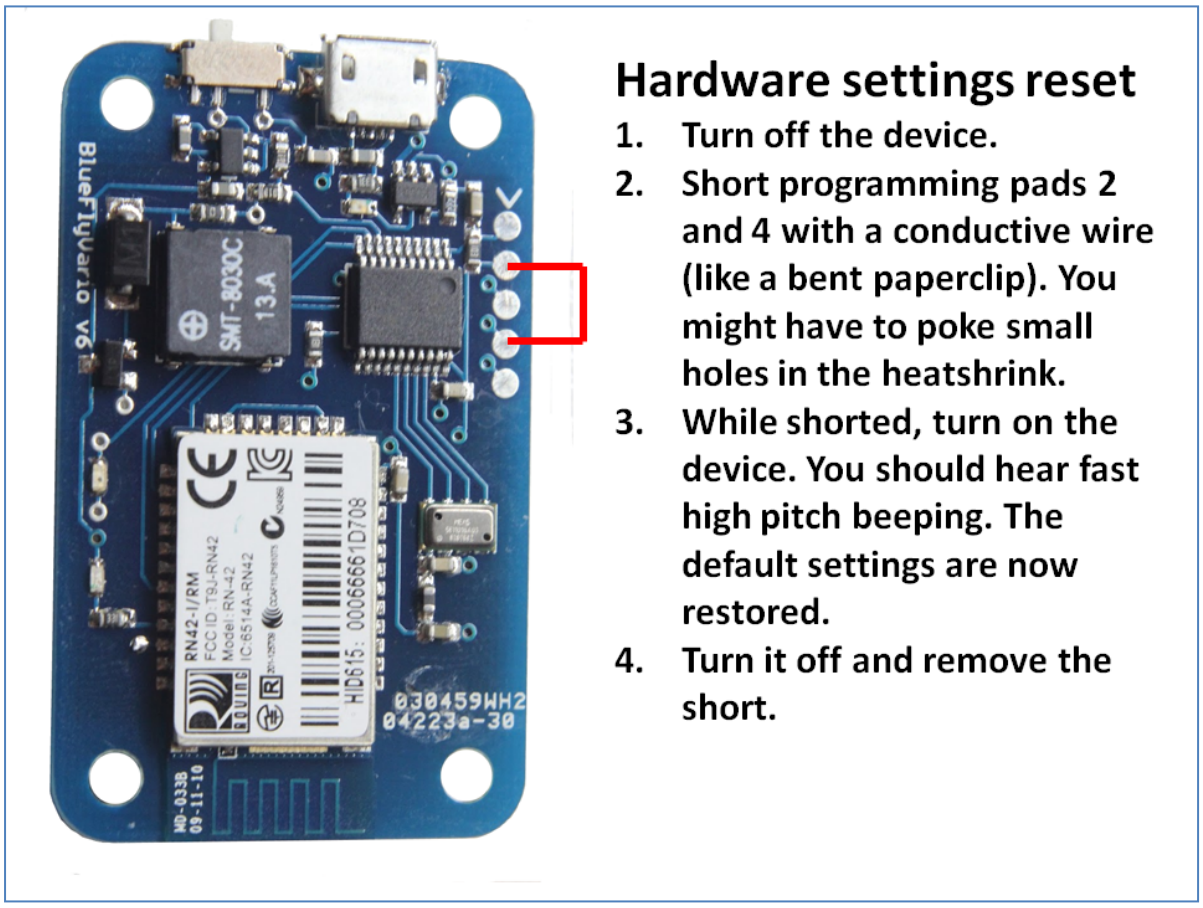
BlueFly_v11_Shield_GPS+Airspeed

This provides the capabilities of both modules in one.

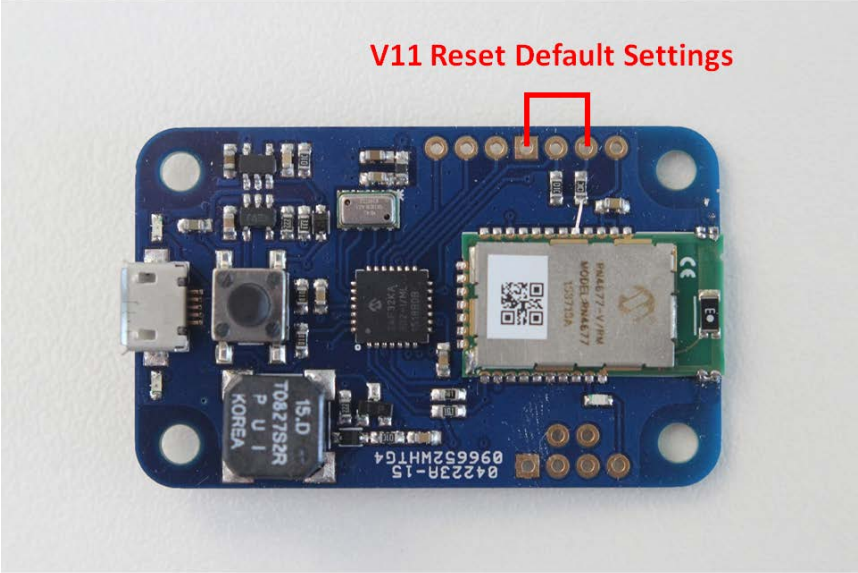
Reset Hardware Defaults

If you really screw up the settings, and for some reason you cannot fix them through the serial connection, then it is possible to reset the hardware settings to their default values using the procedure shown below. You might have to do this if the ***bluetoothWaitTime*** gets set to less than the amount of time required to establish a connection.

For v10 and earlier hardware if programming pad 4 (the forth pad from the arrow which indicates pad 1) is high when the device is turned on, then the reset to default sequence will be entered. Pad 2 is the positive voltage signal so shorting pads 2 and 4, then powering on the device, will achieve hardware reset.



The procedure is similar for v11 hardware, but instead you need to short SCL to Ground.



Firmware Upgrades

From version 10 the BlueFlyVario models include a bootloader so you can upgrade the firmware. The firmware is the chunk of code on the microprocessor which makes the Bluefly wield its magic. Normally you need a special microcontroller programmer to update the firmware, however with the bootloader you can do it over a serial connection without the special programmer.

The ds30loader bootloader

The bootloader is a small section of code which runs before the main firmware code. The BlueFlyVario uses a modified version of the ds30loarder. To use the bootloader you will need the ds30loader gui PC application. You can download it from the support page of the website.

Firmware Versions

The firmware is contained in a .hex file. From version 10 a new numbering system for firmware was implemented. Firmware can be downloaded from the support page of the website. The numbering scheme works like this:

- BlueFlyVario_{DEVICE_TYPE}_v{MODEL_NO}.{TYPE_NO}{SUB_VERSION_NO}.hex where:
  - {DEVICE_TYPE} = Bluetooth, TTL_GPS or USB
  - {MODEL_NO} is associated with a particular series of hardware. (i.e. version 10 for the version 10 models).
  - {TYPE_NO} is associated with the DEVICE_TYPE (Bluetooth = 1, TTL_GPS = 2 and USB = 3).
  - {SUB_VERSION_NO} is the version of the firmware.

For example BlueFlyVario_Bluetooth_v10.102.hex is the second release of firmware for model 10 Bluetooth hardware.

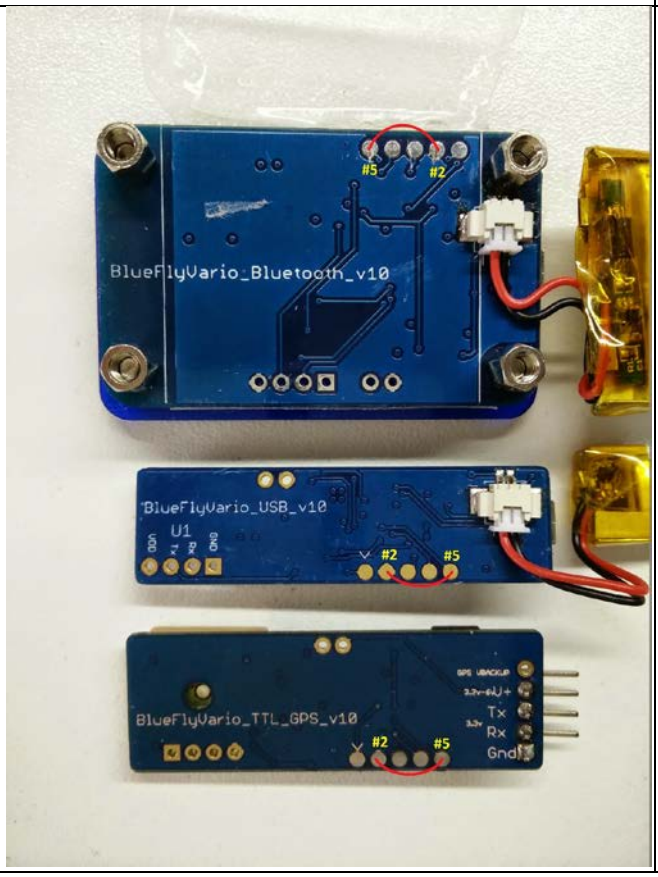
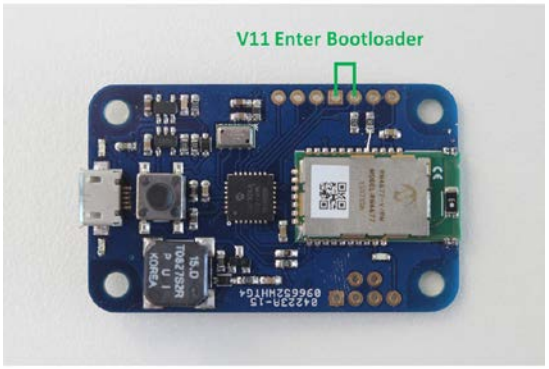
Upgrade Process

Step 1 - Get the software

Get the ds30loader gui application and make sure it runs on your pc. Download the right version of the firmware you will need for your BlueFlyVario model.

Step 2 - Prepare the hardware

You will need access to the programming pads on the pcb so you can short pads #2 and #5 (on v10 hardware) or short SCA to Ground (on v11 hardware). You might need to disassemble the Bluefly so you can access the pads. A small hole in the heatshrink might be sufficient. See the images below for an indication of which pads I am talking about.

V10 hardware	V11 hardware
	

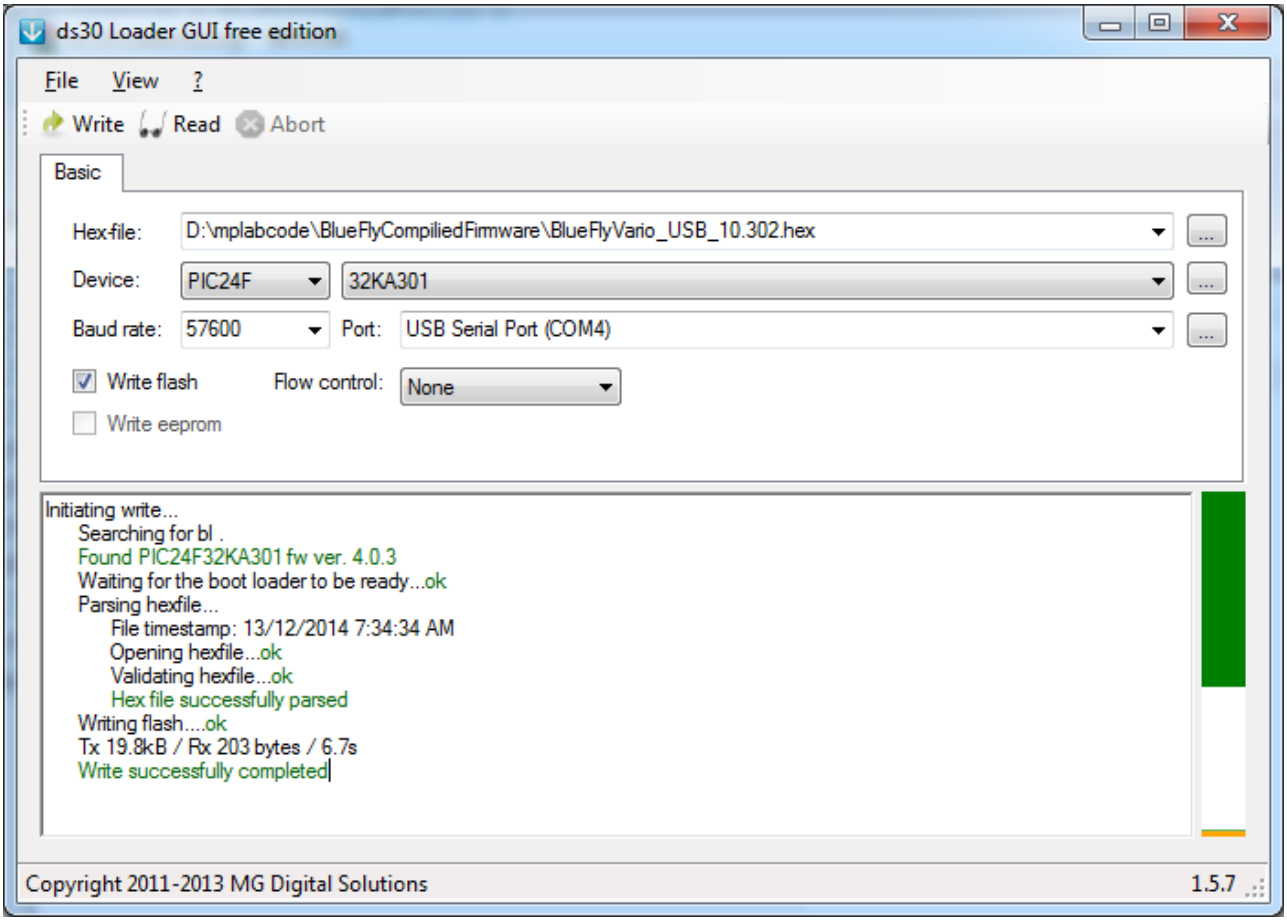
You will also need to establish a serial connection. You can test your serial connection using the BFV Desktop application or a terminal emulator like Realterm:

- **Bluetooth version**: you should add the vario as a device to the PC (you might need to use code 1234 to pair it via the Devices and Setting menu, it depends on your Bluetooth adapter). You will need to work out what serial port it was assigned by looking at its properties.
- **TTL_GPS version**: you will need a TTL to serial adapter. Depending on how you installed the TTL_GPS you might need to solder things. Again you need to work out what serial port the TTL to serial adapter has been assigned.
- **USB version**: the serial port is assigned when the device is first connected to your PC.

Step 3 - Open the ds30laoder application

Open up the ds30loader application. Adjust the settings to match the following:

- Hex file: Browse to the location of the firmware hex file you downloaded (make sure to unzip it)
- Device: PIC24F
- Model: 32KA301
- Baud: 57600
- Port: Whatever serial port you plan to use based on the one you determined in Step 2.
- Write flash: Checked
- Flow control: None



Step 4 - Start up the Bluefly in bootloader mode

You will need to short the pads as described in step 2 then power on the Bluefly module. You should just be able to use a paperclip bent appropriately.

- For the Bluetooth version, press the power on button while the pads are shorted. As soon as the green led lights up solid you can remove the short.
- For the TTL_GPS you will need to power the module using something like a TTL to Serial adapter. Press power then when the green LED lights you can remove the short.
- For the USB version it will power on as soon as you attach it to the USB port of your PC, so make sure the pads are shorted when you attach it (you might need three hands).

If the green led lights up and stays lit you are in bootloader mode. You should not hear any of the normal beeps associated with startup.

After entering bootloader mode you have 60 seconds to complete the next step. If you do not program within 60 seconds the device will exit the bootloader mode.

Step 5 - Program the device

Press the Write button in the ds30loader application. The application should connect to the Bluefly and start programming. It takes about 20 seconds.

When programming is complete the Bluefly will start like normal with the new firmware. You can connect it to the BFV Desktop application to confirm the new version is uploaded.

Step 6 - Update the default settings

After you have completed the update it is recommended that you restore the default settings using the command **$RSX***.

Annex A - Technical Description of Settings

Name	BFV Version	Code	Type	Factor	Integer Values			Converted Values			Message
					Min Value	Max Value	Default Value	Min Value	Max Value	Default Value	
useAudioWhenConnected	6	BAC	boolean	1	0	1	0	FALSE	TRUE	FALSE	Check to enable hardware audio when connected.
useAudioWhenDisconnected	6	BAD	boolean	1	0	1	1	FALSE	TRUE	TRUE	Check to enable hardware audio when disconnected.
positionNoise	6	BFK	double	1000	10	10000	100	0.01	10	0.1	Kalman filter position noise.
liftThreshold	6	BFL	double	100	0	1000	20	0	10	0.2	The value in m/s of lift when the audio beeping will start.
liftOffThreshold	6	BOL	double	100	0	1000	5	0	10	0.05	The value in m/s of lift when the audio beeping will stop.
liftFreqBase	6	BFQ	int	1	500	2000	1000	500	2000	1000	The audio frequency for lift beeps in Hz of 0 m/s.
liftFreqIncrement	6	BFI	int	1	0	1000	100	0	1000	100	The increase in audio frequency for lift beeps in Hz for each 1 m/s.

Name	BFV Version	Code	Type	Factor	Integer Values			Converted Values			Message
					Min Value	Max Value	Default Value	Min Value	Max Value	Default Value	
sinkThreshold	6	BFS	double	100	0	1000	20	0	10	0.2	The value in -m/s of sink when the sink tone will start.
sinkOffThreshold	6	BOS	double	100	0	1000	5	0	10	0.05	The value in -m/s of sink when the sink tone will stop.
sinkFreqBase	6	BSQ	int	1	250	1000	400	250	1000	400	The audio frequency for the sink tone in Hz of 0 m/s.
sinkFreqIncrement	6	BSI	int	1	0	1000	100	0	1000	100	The decrease in audio frequency for sink tone in Hz for each -1 m/s.
secondsBluetoothWait	6	BTH	int	1	0	10000	180	0	10000	180	The time that the hardware will be allow establishment of a bluetooth connection for when turned on.
rateMultiplier	6	BRM	double	100	10	100	100	0.1	10	1	The lift beep cadence -> 0.5 = beeping twice as fast as normal.
volume	6	BVL	double	1000	1	1000	1000	0.001	1	1	The volume of beeps -> 0.1 is only about 1/2 as loud as 1.0.
outputMode	7	BOM	int	1	0	6	0	0	3	0	The output mode -> 0-BlueFlyVario(default), 1-LK8EX1, 2-LX, 3-FlyNet, 4-Nothing, 5-BFVCustom, 6-BFVExtendedCustom.

Name	BFV Version	Code	Type	Factor	Integer Values			Converted Values			Message
					Min Value	Max Value	Default Value	Min Value	Max Value	Default Value	
outputFrequency	7	BOF	int	1	1	50	1	1	50	1	The output frequency divisor -> 1-every 20ms ... 50-every 20msx50=1000ms
outputQNH	7	BQH	int_offset	80000	0	65535	21325	80000	145535	101325	QNH (in Pascals), used for hardware output alt for some output modes - (default 101325)
uart1BRG	8	BRB	int	1	0	65535	207	0	65535	207	BRG setting for UART1, baud = 2000000/(BRG-1) (default of 207 = approx 9600 baud)
uart2BRG	9	BR2	int	1	0	65535	34 (v10) 16 (v11)	0	65535	34 16	BRG setting for UART2, baud = 2000000/(BRG-1) (default of 34 = approx 57600, 16 = approx 115200
uartPassthrough	9	BPT	boolean	1	0	1	1	FALSE	TRUE	TRUE	Check to enable characters received on U2 to be passed through to U1.
uart1Raw	9	BUR	boolean	1	0	1	0	FALSE	TRUE	FALSE	Check to make characters received at U1 are passed straight through to U2 rather than passed through line by line (i.e. $...<LF>) and then multiplexed with pressure output.
greenLED	9	BLD	boolean	1	0	1	1	FALSE	TRUE	TRUE	Check turn on the green LED with each lift beep.

Name	BFV Version	Code	Type	Factor	Integer Values			Converted Values			Message
					Min Value	Max Value	Default Value	Min Value	Max Value	Default Value	
heightSensitivityDm	10	BHV	int	1	0	10000	20	0	10000	20	How far you have to move in dm to reset the idle timeout
heightSeconds	10	BHT	int	1	0	10000	600	0	10000	600	Idle timeout
useAudioBuzzer	10.m04	BBZ	boolean	1	0	1	0	FALSE	TRUE	FALSE	Check to use the experimental audio buzzer
buzzerThreshold	10.m04	BZT	int	100	0	1000	40	0	10	0.4	The value in m/s below the liftThreshold when the buzzer will start.
speedMultiplier	10.m06	BSM	double	100	10	1000	100	0.1	10	1	The lift beep cadence as related to speed -> 2.0 = half as sensitive as normal.
usePitot	11	BUP	boolean	1	0	1	0	FALSE	TRUE	FALSE	Set to use the experimental MS4525DO differential pressure sensor on I2C.