830 Douglas Ave.
Dunedin, FL 34698
(727)733-2447
Fax:(727)733-3962
www.OceanOptics.com

A HALMA GROUP COMPANY

Ventana Data Sheet

Description

The Ocean Optics Ventana series of high-sensitivity spectrometers maintain high sensitivity and throughput for low-light level applications such as Raman analysis or fluorescence without the need for long integration times or high-power excitation sources. This spectrometer uses an HD Volume Phase Holographic (VPH) Grating, low f/# design and anti-reflection coatings to allow more of the coupled light into the spectrometer. The internal optics have been selected to minimize light loss and provide superior image quality. The Ventana models include the following:

- Ventana-785-Raman -- preconfigured for 785 nm excitation Raman measurements; includes a cooled (down to 15 °C) NIR-enhanced, back-thinned detector with low noise electronics
- Ventana-532-Raman -- preconfigured for 532 nm excitation Raman measurements
- Ventana-785L-Raman -- preconfigured for 785 nm excitation Raman measurements with integrated 785 nm laser; free-space coupling to maximize sensitivity; front-end laser matches the $f/1.3$ spectrometer input to maintain maximum efficiency; hard-coated filters used to maximize transmission and minimize Rayleigh scattering
- Ventana-VIS-NIR -- preconfigured with 430-1100 nm bandwidth, 50 micron slit; low f/2; configuration optimized for low signal VIS-NIR measurements (430 to 1100 nm) such as fluorescence; low f/# with custom lens design and optic coatings providing superior image quality and high transmission

All model Ventana spectrometers use the Hamamatsu S11510-1006 back-thinned, reduced etalon detector except for the VIS-NIR version, which uses the Hamamatsu S10420-1006 detector. For complete details on these detectors, visit **www.Hamamatsu.com**.

In addition to the Ventana spectrometers, the Ventana-785-Probe offers a Raman-coupled fiber probe for 785 nm excitation; FC connector (excitation), SMA connector (collection); 11.0 mm working distance; 250-2000 cm-1; comes with 600 µm 0.39 NA detection and 100 µm 0.22 NA excitation fibers to maximize sensitivity.

Features

- HD Volume Phase Holographic (VPH) Grating
- Electrical Performance
  - 16 bit A/D converter
- Communications: USB
- Software compatibility:
  - OceanView (1 complimentary license included with purchase)
  - SpectraSuite
  - OmniDriver (sold separately)
  - SeaBreeze (sold separately)
- Optical input: SMA 905
- Regulations:
  - CE Mark
  - RoHS

Specifications

CCD Detector Specifications

Specification	Ventana 532	Ventana 785	Ventana 785L	Ventana Vis/NIR
Detector	Hamamatsu S11510-1006 Back-thinned NIR-enhanced CCD			Hamamatsu S10420-1006 Back-thinned CCD
Detector active area (mm)	14.336 x 0.896			
Pixel format	1024 x 64			
Pixel size	14 µm square			
Well depth	300 Ke^-			
Read noise	6 e- RMS			
Dynamic range	17000:1 (Typical)			
CCD QE	78% peak, 60% average in Raman			75% peak
CCD Cooling	None	15°C		None

Ventana Spectrometer Specifications

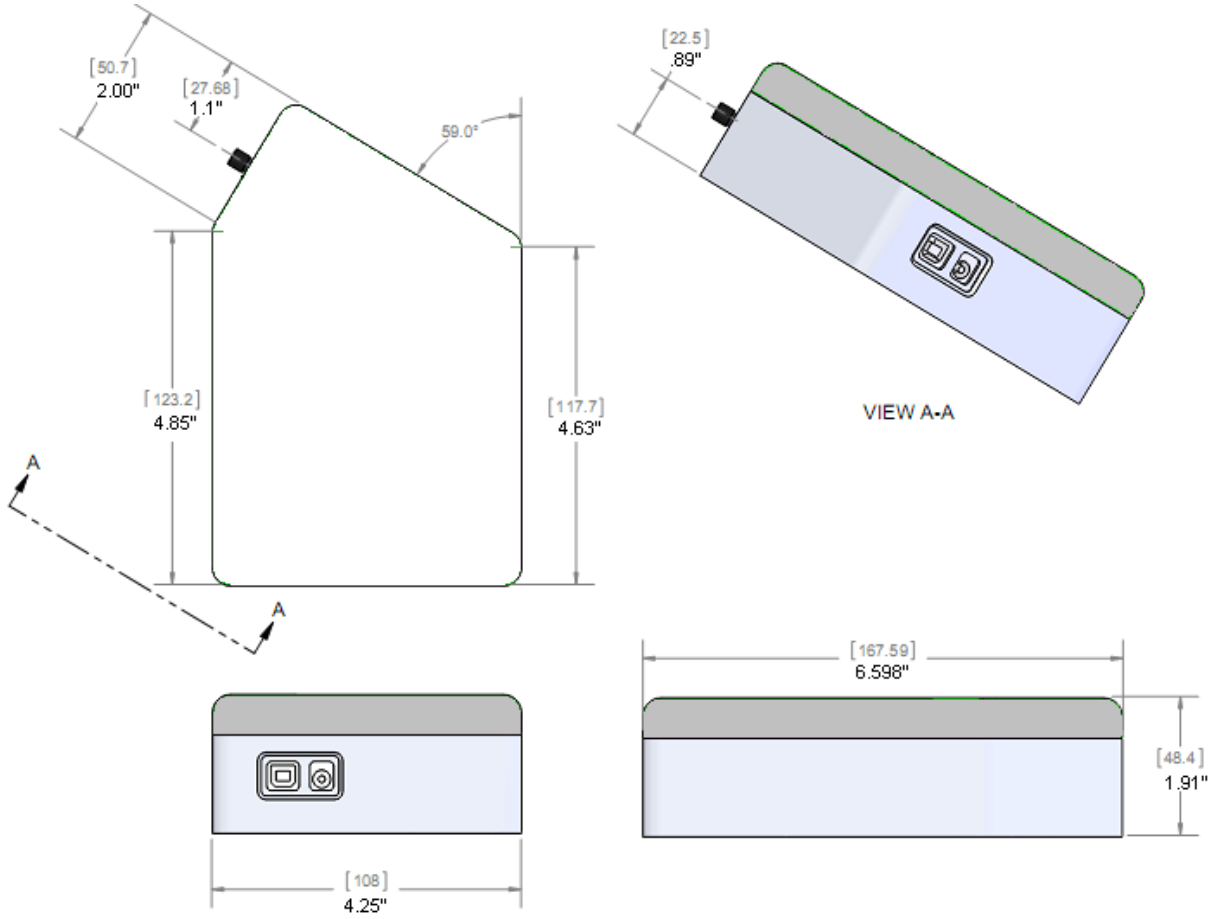
Specification	Ventana 532	Ventana 785	Ventana 785L	Ventana VIS/NIR
Dimensions (LxWxH)	170 mm (6.7 in.) x 110 mm (4.3 in.) x 50 mm (1.9 in.)	125 mm (4.9 in.) x 110 mm (4.3 in.) x 50 mm (1.9 in.)	127 mm (5 in.) x 158.75 mm (6.25 in.) x 50.8 mm (2 in.)	170 mm (6.7 in.) x 110 mm (4.3 in.) x 50 (1.9 in.)
Weight	0.9kg (2 lb)		Spectrometer:1.2kg (2.6 lb) Power Supply: 0.5kg (1.10 lb)	0.9kg (2 lb)
Temperature Operation	0 to 50°C (noncondensing)		5 to 45°C (noncondensing)	0 to 50°C (noncondensing)

Specification	Ventana 532	Ventana 785	Ventana 785L	Ventana VIS/NIR
Power	12 VDC			
Gratings	1600 lpmm HD VPG	1625 lpmm HD VPG		450 lpmm at CWL 580 nm
Diffraction efficiency	85% peak at 610 nm	83% average for S&P		65% average for S&P
Optical input	SMA 905			
Slit Size	50 µm			
Fiber diameter	Optimized for 600µm, NA = 0.39 fiber input			
Resolution (FWHM) cm^{-1}	20 cm^{-1}	10 cm^{-1} @ 810 nm		N/A
Resolution (FWHM) nm	0.6 nm	0.7 nm @ 810 nm		4.0 nm
f/#	1.3			2.0
Signal-to-noise ratio	~550:1 (Typical)			
Integration time	22 ms – 4 min			
Wavelength Range	533 – 690 nm	800 – 940 nm	800 – 940 nm	430 – 1100 nm
Interface	USB 2.0 (no external accessory connector access)			

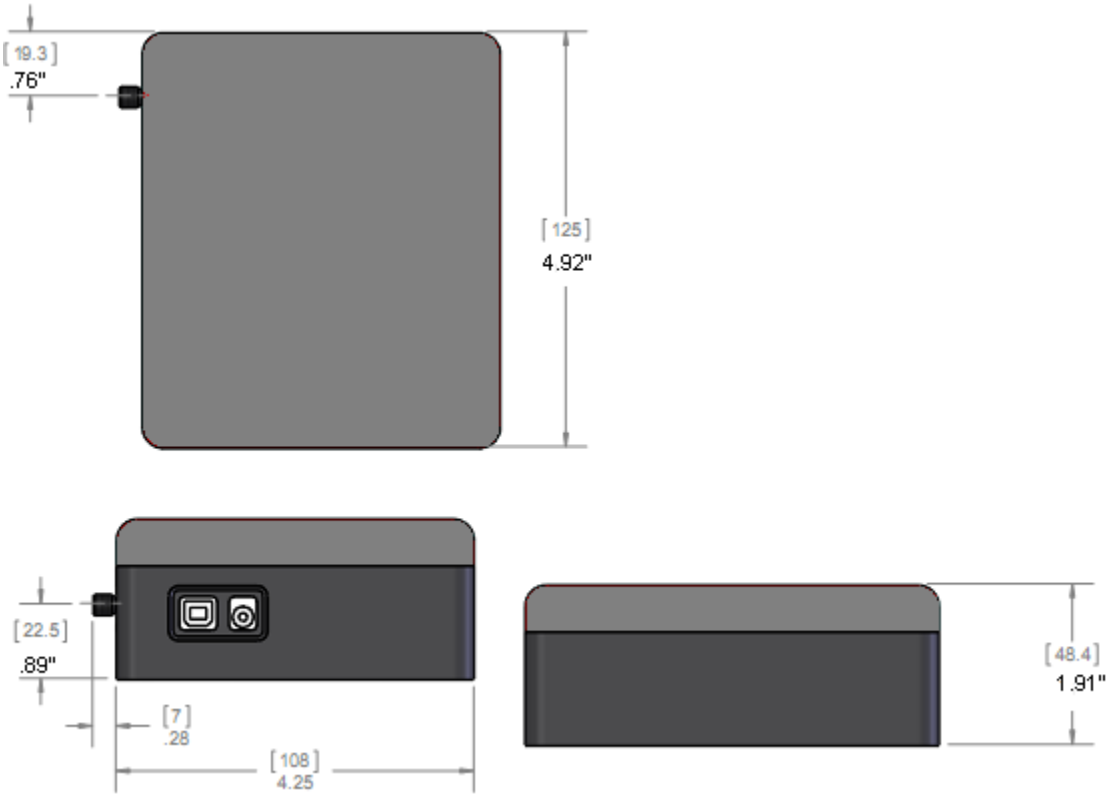
Ventana 785L Laser Specifications

Specification	Value
Working Distance	21.95 mm
Image Spot Size	10 mm
Sample Spot Size	75 mm
Laser type	Single-mode laser diode stabilized with VBG
Laser control	Enable/disable
Laser power stability	< 1% after 30 seconds from being enabled

Mechanical Diagrams



Ventana 532-Raman



Ventana 785-Raman

Ventana Spectrometer Detector

The Ventana contains a Hamamatsu S11510-1106 back-thinned NIR-enhanced CCD. The Ventana VIS-NIR contains the Hamamatsu S10420-1006 detector. The Ventana electronics only support reading out the device as a 1-D array (e.g., all rows are summed together on chip). The structure of the Ventana spectrometer Hamamatsu CCD is shown below. The device has 1024 x 64 active pixels and a total of 1044 x 70 pixels.

Ventana USB Port Interface Communications and Control Information

The Ventana is a microcontroller-based Miniature Fiber Optic Spectrometer that can communicate via the Universal Serial Bus. This section contains the necessary command information for controlling the Ventana via the USB interface. This information is only pertinent to users who wish to not utilize Ocean Optics 32 bit driver to interface to the Ventana. Only experienced USB programmers should attempt to interface to the Ventana via these methods.

USB Info

Ocean Optics Vendor ID number is 0x2457 and the Product ID is 0x5000.

Instruction Set

Protocol Design

The binary command protocol for the Ventana Spectrometer has the following design characteristics:

- Provides information so that the host does not need to know the state of the device to read the message.
- Contains a distinct header and footer to fully bracket transfers.
- Provides an abstract interface to the device. All timing is represented in standard units rather than clock divisors. A specific outcome is achieved via a single mechanism.
- Stores calibration information (wavelength, nonlinearity coefficients, etc.) in distinct commands rather than EEPROM slots.

Ventana Command Protocol

There are two types of messages in this protocol:

- "commands" that do not return any information
- "queries" that cause the device to return information

When developing a device driver that will communicate with the Ventana, the fact that some messages generate a response (including a status indication) and others do not can cause design problems. The simplest approach to creating a driver for this protocol is to have all message types generate a reply. This allows a single message read to be performed after every message write, and if the response indicates an error, then the driver can recover immediately rather than finding the error later when it expects a response to some new query.

The "flags" field in the message header (starting at byte offset 4) has an "acknowledgment (ACK) requested" bit (bit 2). If this bit is set to 1 for every "command", and left at 0 for every "query", then all communications with the Ventana will become predictable read/write transactions. The immediate reply allows the host driver to avoid changing its state until it has received confirmation that the last operation succeeded or failed. This makes driver design much easier than the alternative.

It is recommended that an Ventana protocol driver implement two functions:

- send_command_to_device() which takes a message type and an optional payload, and returns a simple pass/fail result based on the ACK or NACK flag in the response. This should set the "ACK requested" bit in every message it emits;
- query_device() which takes a message type and optional payload and returns a payload (e.g. a byte array) which can be NULL if the response was a NACK. This must not set the "ACK requested" bit because to do so would generate a spurious ACK in addition to the expected response.

By using these two functions to encapsulate all transfers to the Ventana, the programming model is kept very simple.

Message Layout

All multi-byte fields are little-endian (LSB first). Each message in the binary protocol is laid out as follows:

1. A 44-byte header
2. An optional payload
3. A 16-byte checksum block
4. A four byte footer

The header, checksum, and footer are 64 bytes total. For simple messages, the command or response is embedded in the header so only a single packet is required. For more complex messages, the header and footer add a single USB packet as overhead to the transfer.

Header

The message header is structured as follows:

Offset (Bytes)	Field	Size (Bytes)	Valid Values	Notes
0	Start Bytes	2	0xC1C0	Chosen for the following reasons: - High bits set, so not likely to occur in ADC data less than 16 bits wide or message type fields - When concatenated with the tail of a previous message, creates a distinct sequence Note: Do not reverse this byte order.
2	Protocol Version	2	0x0000 – 0xFFFF	Initially set to 0x1000. The host should only send messages known to be supported in the reported version of the protocol. Subsets of this protocol may be specified for other devices using a version less than 0x1000. The device may reject messages with a specified protocol it does not recognize.
4	Flags	2	0x0000 – 0xFFFF	Bits in this flag are assigned as follows: Bit 0: response to earlier request (message type is set equal to request type). Set by device. Bit 1: acknowledgment (ACK) if previous message included request for ACK. Set by device. Bit 2: acknowledgment (ACK) requested. Set by Host. Bit 3: negative acknowledgment (NACK). May be sent if previously sent message type is unknown or otherwise invalid. Message type and regarding fields will be set to the type that caused the error. Set by device. Error Number field contains reason for NACK. Bit 4: exception occurred. Indicates that although the message itself was valid, the device encountered a hardware problem that may have invalidated the result. Error Number will be set to explain, if possible. Set by device. Bit 5: The message protocol used by the caller is deprecated. If set, an older version of the protocol has been detected (version less than 0.1100). Set by the device.

Offset (Bytes)	Field	Size (Bytes)	Valid Values	Notes
6	Error Number	2	0x0000 – 0xFFFF	Only set by the device. Indicates whether the previous request was successful. Only set to be non-zero if at least one of the following flags is set: NACK or exception. Only one value can be set, even if multiple errors were detected. Values: 0: Success (no detectable errors) 1: Invalid/unsupported protocol 2: Unknown message type 3: Bad checksum 4: Message too large 5: Payload length does not match message type 6: Payload data invalid 7: Device not ready for given message type 8: Unknown checksum type 9: Device reset unexpectedly 10: Too many buses (Commands have come from too many bus interfaces) 11: Out of memory. Failed to allocate enough space to complete request. 12: Command is valid, but desired information does not exist. 13: Int Device Error. May be unrecoverable. 100: Could not decrypt properly 101: Firmware layout invalid 102: Data packet was wrong size (not 64 bytes) 103: hardware revision not compatible with firmware 104: Existing flash map not compatible with firmware 255: Operation/Response Deferred. Operation will take some time to complete. Do not ACK or NACK yet.
8	Message Type	4	0x00000000 – 0xFFFFFFFF	Each message type represents a command. See *Message Types*.
12	Regarding	4	0x00000000 – 0xFFFFFFFF	Arbitrary host-defined data. Any response generated by the device will have the same value in its Regarding field. This can be used by the host to match responses to requeVentana if transactions are split up.
16	Reserved	6		For future expansion.
22	Checksum Type	1	0x00 – 0x01	Valid types: 0: no checksum (must still provide a block of 16 bytes after the payload, but they can be zero). 1: MD5 (fully fills the 16 byte checksum block)
23	Immediate Data Length	1	0x00 – 0x10	Total number of bytes used in the Immediate Data field (see below).

Offset (Bytes)	Field	Size (Bytes)	Valid Values	Notes
24	Immediate Data	16		Provides an alternative to specifying a payload so commands with small operands can fit within a single USB packet. If this field is used, the number of bytes containing valid data must be set in the Immediate Data Length field, and there is no payload. If a payload is used, this field is ignored.
40	Bytes Remaining	4	0x00000000 – 0xFFFFFFFF	Includes the payload, if any, plus the checksum and footer. This is included for RS-232, such that a constant-sized header could be read (including this field) followed by another read for the remainder of the message. Payload length must be computed as this field minus the checksum and footer length. Ventana may reject any message too long for it to process internally and return a NACK.

The header can be represented as a C struct as follows (assuming that the int type is 32 bits long):

```
struct ooi_binary_protocol_header {
    unsigned char start_bytes[2];     /* = { 0xC1, 0xC0 } */
    unsigned short protocol_version;  /* = 0x1000 */
    unsigned short flags;
    unsigned short errno;
    unsigned int message_type;
    unsigned int regarding;
    unsigned char reserved[6];
    unsigned char checksum_type;
    unsigned char immediate_data_length;
    unsigned char immediate_data[16];
    unsigned int bytes_remaining;
};
```

Payload

After the standard header, a payload may be provided. The payload contains data required by the given message type. The format of the data within the payload depends on the message type. Note that a payload is not required if operands will fit in the Immediate Data field of the header. The length of the payload must be computed from the Bytes Remaining field in the header, given that the checksum and footer are of a constant length:

Payload length = Bytes remaining – 20

The Ventana may populate the Immediate Data field or the Payload as appropriate for the length of the data being returned, regardless of the message type. Host programs decoding this protocol should always be capable of checking both areas in any response.

Checksum

A 16-byte block must appear after the payload (if any) to contain checksum data. This block is required even if no checksum is used (according to the Checksum Type field) or if the checksum does not require the full 16 bytes. The unused parts of the block are for padding to ensure the message length is consistent. This protocol does not support checksums longer than 16 bytes, e.g. SHA, but the intent of the checksum is to detect bit errors, not to prevent tampering or to provide cryptographic assurance. The checksum may not be necessary for USB but may be useful for buses that do not provide data integrity guarantees, such as RS-232.

If a checksum is used, it will be computed starting with the start byte of the header and continuing through the last byte of the payload. The length of the checksum and footer will not be included in the checksum (i.e., for MD5, which includes the total data length as a salt value).

Footer

After the checksum block, a 4-byte footer is provided. The footer has a constant value of 0xC5C4C3C2. Do not reverse the order of the footer.

Message Types

The binary protocol divides up the 4.29 billion possible message types into categories and subcategories in a hierarchy. The most significant bits represent the more abstract categories, while the least significant bits represent subcategories and the commands. The 32-bit message type is split into three blocks, 0xXXX YYY ZZ, as follows:

XXX: top-level category or feature. 4096 of these may exist.

YYY: subcategories within the feature. 4096 of these may exist for each category.

ZZ: specific commands for the subcategory. 255 of these may exist for each subcategory.

The top-level categories (XXX) are initially defined as follows.

0x000: General device characteristics

0x001: Spectrometer feature (control of detector and ADC, pixel calibrations and corrections)

0x002: GPIO feature (configuration and control)

0x003: Strobe features (single and continuous strobe timing)

0x004: Temperature

The subcategories and commands for each of these categories are described in the tables that follow. Input and output data lengths that can be computed from the header (Bytes Remaining field) are not shown. All multi-byte integer types will be returned in little-endian format (least significant byte first). All data will be carried over Endpoint 1or Endpoint 2 IN and OUT unless otherwise stated.

Message Examples

The following is an example of how the Set Integration Time message type (0x001 100 10) can be constructed based on the information provided in this data sheet:

Header

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0xC1	0xC0	0x00	0x10	0x00	0x00	0x00	0x00
Start bytes		Protocol version		Flags		Error number	

Byte 8	Byte 9	Byte 10	Byte 11	Byte 12	Byte 13	Byte 14	Byte 15
0x10	0x00	0x11	0x00	*x*	*x*	*x*	*x*
Message type (0x00110010)				Regarding (user-specified)			

Byte 16	Byte 17	Byte 18	Byte 19	Byte 20	Byte 21	Byte 22	Byte 23
						0x00	0x04
Reserved						Checksum type	Immediate length

Byte 24	Byte 25	Byte 26	Byte 27	Byte 28	...	Byte 39
x LSB	*x*	*x*	*x* MSB	0	0	0
Integration time (immediate data)				Unused		

Byte 40	Byte 41	Byte 42	Byte 43
0x14	0	0	0
Bytes remaining			

Optional Payload	Byte 44…Byte 59	Byte 60	Byte 61	Byte 62	Byte 63
Not used for this command	Checksum	0xC5	0xC4	0xC3	0xC2
		Footer			

The following is an example of how the Get And Send Corrected Spectrum Immediately message type (0x001 010 00) can be constructed based on the information provided in this data sheet:

Header

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0xC1	0xC0	0x00	0x10	0x00	0x00	0x00	0x00
Start bytes		Protocol version		Flags		Error number	

Byte 8	Byte 9	Byte 10	Byte 11	Byte 12	Byte 13	Byte 14	Byte 15
0x00	0x10	0x10	0x00	*x*	*x*	*x*	*x*
Message type (0x00101000)				Regarding (user-specified)			

Byte 16	Byte 17	Byte 18	Byte 19	Byte 20	Byte 21	Byte 22	Byte 23
						0x00	0x00
Reserved						Checksum type	Immediate length

Byte 24 ... Byte 39
unused

Byte 40	Byte 41	Byte 42	Byte 43	No payload	Byte 44...Byte 59	Byte 60	Byte 61	Byte 62	Byte 63
0x14	0x00	0x00	0x00		Checksum	0xC5	0xC4	0xC3	0xC2
Bytes remaining						Footer			

Response

Header

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0xC1	0xC0	0x00	0x10	0x01	0x00	0x00	0x00
Start bytes		Protocol version		Flags		Error number	

Byte 8	Byte 9	Byte 10	Byte 11	Byte 12	Byte 13	Byte 14	Byte 15
0x00	0x10	0x10	0x00	*x*	*x*	*x*	*x*
Message type (0x00101000)				Regarding (user-specified)			

Byte 16	Byte 17	Byte 18	Byte 19	Byte 20	Byte 21	Byte 22	Byte 23
						0x00	0
Reserved						Checksum type	Immediate length

Byte 24 ... Byte 39
unused

Byte 40	Byte 41	Byte 42	Byte 43	Payload (2048 bytes of spectral data)	Byte 2092…Byte 2107	Byte 2108	Byte 2109	Byte 2110	Byte 2111
0x14	0x08	0x00	0x00		Checksum	0xC5	0xC4	0xC3	0xC2
Bytes remaining						Footer			

General Device Commands

Message Type	Purpose	Input Data	Output Data	Notes
0x000 000 80	Get hardware revision	N/A	Unsigned byte	This value is sensed from the hardware itself. Request has no payload. Reply is a single byte.
0x000 000 90	Get host interface firmware revision	N/A	Unsigned short	Firmware version as binary coded decimal. The same value should be available through the USB descriptor as the bcdDevice field. Request has no payload. Reply is a 2-byte integer (LSB first) of the revision.
0x000 000 91	Get FPGA firmware revision	N/A	Unsigned short	Firmware revision as a binary coded decimal for FPGA
0x000 001 00	Get serial number	N/A	String	

Spectrometer Commands

Message Type	Purpose	Input Data	Output Data	Notes
0x001 010 00	Get and send corrected spectrum immediately	N/A	Pixel values (integers)	This returns the intensity of every pixel on the detector as LSB, MSB as soon as it is available. There is no payload in the request. The reply has 2048 bytes of payload. The pixel intensities are corrected for temperature drift and fixed-pattern noise.
0x001 100 00	Get integration time (µs)	N/A	Unsigned 32-bit integer	
0x001 100 10	Set integration time (µs)	Unsigned 32-bit integer	N/A	Input is 4 bytes for time in µs. Order is LSB, …, MSB No reply. The minimum is 10.

Message Type	Purpose	Input Data	Output Data	Notes
0x001 101 00	Get trigger mode	N/A	Unsigned byte	
0x001 101 10	Set trigger mode	Unsigned byte	N/A	Input has 1 byte in the request for trigger mode. Valid trigger modes: Mode 0 (default): integration begins as soon as possible after request Mode 1: integration or trigger delay begins with a rising trigger edge Mode 2: integration is internally triggered to synchronize with continuous strobe No reply.
0x001 801 01	Get wavelength coefficient	Unsigned byte	IEEE single-precision	Request has 1-byte input data for coefficient index starting with wavelength intercept at index 0. Reply has 4-byte float (LSB first).
0x001 801 11	Set wavelength coefficient	Unsigned byte, IEEE single-precision	N/A	Input is the order of the coefficient to set (indexing starts with wavelength intercept at index 0), followed by an IEEE single-precision float. No reply.
0x001 811 01	Get nonlinearity coefficient	Unsigned byte	IEEE single precision	Request has 1-byte input for coefficient index to retrieve. Reply has 4-byte float (LSB first).
0x001 811 11	Set nonlinearity coefficient	Unsigned byte, IEEE single precision	N/A	Input is the order of the coefficient to set, followed by an IEEE single-precision float. No reply.
0x001 831 01	Get stray light coefficient	Unsigned byte	IEEE single precision	Input is the order of the coefficient to retrieve
0x001 831 11	Set stray light coefficient	Unsigned byte, IEEE single precision	N/A	Input is the order of the coefficient to set, followed by an IEEE single-precision float

Message Type	Purpose	Input Data	Output Data	Notes
0x004 200 00	Get TEC temperature	N/A	IEEE single-precision float	Result is degrees Celsius.
0x004 200 10	Set TEC enable	Unsigned byte	N/A	0 = Disabled, 1 = enabled. Sets whether thermo-electric cooler attached to detector is enabled.
0x004 200 11	Set TEC setpoint	IEEE single-precision float	N/A	Specifies the setpoint, in degrees Celsius, of the TEC.