VideoRay Communication Protocol

Version 1.05

Preamble:

Pre-PRO-4 VideoRay ROV use a simple 2 packet communication protocol for external host control of the vehicle. The host is responsible for transmitting a single control packet, and the ROV responds with a single response packet. This protocol is documented in Pro3_PC.doc.

While the PRO3 protocol is very easy for a host to implement and is near optimal with respect to thruster control response time it has some drawbacks:

1) Does not allow for easy expansion to control new devices. (there are some free bits in the control packet)
2) Does not allow for additional data response. Only two 16-bit words are sent back, these can be multiplexed to provide additional data. There is an additional reserved "type" byte.
3) Is not terribly multi party tolerant. Packets are not checksummed, there is no addressing, etc.
4) Is a completely custom protocol which does not leverage external devices

A new communication protocol has been implemented to overcome the above issues, while at the same time maintaining the low control latencies inherent in the PRO3 protocol.   The design goals for the new protocol:

1) Packetized
2) Addressable (individual and groups)
3) Checksum protection of packet header and data payload
4) Easy to implement on memory constrained devices
5) Allow minimal control latencies
6) Prefer a small set of commands
7) Leverage external devices where possible
8) Allow for external device control with/without an ROV in the loop
9) Stateless parsing of response packets (all info is in the packet, the host does not need to remember what packets it has sent)

The PRO4 ROV system makes it easy to hang additional devices on it's primary tether communication bus as well as a secondary peripheral bus. The physical layer for both of these buses is RS-485 (see http://en.wikipedia.org/wiki/EIA-485). Examples of the types of devices which will be connected to the various buses are:

1) PRO 4 ROV (possible more than one on the same bus)
2) Lights
3) KCF Smart Tether Nodes
4) Pan/Tilt Cameras
5) Various Manipulators
6) Additional Thruster Packets (Omni-directional)
7) Custom Sensor Packs (Radiation Detector, cathodic protection probe, etc.)
8) Altimeters

The PRO4 protocol is essentially a slightly modified version of the protocol used by the Futaba RS301CR/RS302CD servo motors. The PRO4 protocol should be able to exist on the same physical bus as any device which conforms to either the PRO4 or the Futaba protocol. Devices which do not conform may require a protocol converter to interface safely.

The PRO4 protocol is a hybrid Control and Status Register (CSR) Architecture. This maintains the spirit of the Futaba protocol and also has the added benefit of being compact and scalable across disparate devices.

A CSR Architecture essentially implements each end device as a memory map of registers. Data can be written into these registers and/or retrieved from them. A special register can be defined to accept arbitrary commands, if a more traditional command/response system is desired.

The PRO4 protocol can be considered a hybrid since the response packets may or may not consist of contiguous memory mapped locations. This allows for bandwidth saving as multiple packets are not required for non- contiguous locations.

The fact that the protocol is a CSR architecture means that an host program developer will require a memory map for the device to be controlled in addition to the protocol specification.

The protocol is also not inherently bi-directionally routeable There is only a destination id in the packet header. It is assumed that either having fully routeable packets is unnecessary, or provisions will be made in device firmware for intelligently routing of packets, or an embedded extension could be used.

The protocol while being size symmetric (packet header is the same size in both directions) the protocol is not semantically symmetric. In particular the first bytes of the payload data in response packets indicate a device type.

PRO4 Communication Protocol:

Physical Layer:

RS-485, Half-duplex. 115200, 8, N, 1.
Biasing and termination may depended upon application.

While the protocol is PHY agnostic. The described physical layer has been standardized on the PRO4 and various new accessories.

Packet Format:

The PRO4 packet consists of a 7 byte header, arbitrary length payload data, and a final checksum.

The length of the payload is restricted by the fact that the length is a single byte. Larger data packets can be handled by packetization or extending the protocol. Although longer packets are strongly discouraged as they will degrade overall system latency.

PRO4 packet, each field is a single 8-bit byte.

Start of Packet (Byte 1)	**Start of Packet (Byte 2)**	**Network ID**	**Flags ***	**CSR Address**	**Length**	**Header Checksum ***	**Payload Data****	**Total Checksum**

*indicates usage is significantly different than Futaba protocol.
**The first byte(s) in the Payload Data of a response packet is a VideoRay designated device type descriptor.

Synchronization:

The protocol uses two bytes for a start of packet synchronization.
These bytes are Identical to those used in the Futaba protocol.
There are essentially two types of packets, requests and response packets.
0xFDDF are response packets.

Request:	**0xFAAF**
Response:	**0xFDDF**

Some semantics of the header fields changed dependent upon the packet being a response or not.
For example in response packets the **Network ID** field is the responding device's id, otherwise it is the id of the destination device.

Network ID:

This is destination of the target device(s) for request packet.

This is the Node ID of the transmitting device for a response packet.

Network ID's can be in the range of 0 to 127 (0x0 – 0x7F) in order to individually address devices.

The destination can also be 128-254 (0x80 – 0xFE) in order to perform a group device multicast.
Classes of devices will be given individual group codes. This allows for up to 126 groups.

The destination can also be 255 (0xFF) which indicates a broadcast to all connected devices.

In summary:

0x1	**-**	**0x7F**	**Individual Device NODE ID**
0x80	**-**	**0xFE**	**Group multicast**
0xFF			**Full Broadcast**

Devices should not send any response to multicast or broadcast packets.
The exception is the Device Enumeration request discussed below.

Device ID 0 should NEVER be used, as this would cause conflicts with Futaba servos incorrectly parsing packets as LONG packets.

FLAGS:

The FLAGS byte is used as a bit field to designate the type of response requested.

The Futaba protocol allocates the bits in the FLAG byte in a specific manner. This protocol differs from that definition.

0x00	**No response packet**
0x01-0x7F	**Device Specific Packet Type (Defined by device developer, may be cardinal number or a bit field)**
0x80-0xFF	**Contiguous Data starting at CSR Address low 7 bits indicate length of data to return**

So in essence if the msb is set then flag bit can be masked to get the length of desired response data. (length = flag_byte & 0x7F;) An 0x80 indicates that all data from CSR address to the end of the device CSR register file (address 0xEF) should be sent.

A Flag of 0x80 should NOT in general be used to read data above the device specific CSR (0xF0 and up) since there are multiple special length registers. However it is standard practice use a flag of 0x80 when reading the entire CONFIGURATION DATA register.

A response to a Contiguous read always has 0x80 in the FLAGS byte.

CSR Address:

The address field in a device specific response is set to 0x0.

This field identifies the 8-bit address of the device memory map register.

It is recommended that all devices use a little endian (lsb in lowest address) This matches futaba, and what the AVR compilers seem to prefer.

For packets that write or read multiple bytes, the CSR Address designates the start address.

10 addresses (0xF0 – 0xFF) are reserved for device independent use.

0x0-0xEF	**Device Specific Memory Map**
0xF0	**Custom Command Register. Data sent to this register is interpreted as a multi-byte command, and not a CSR memory r/w. It is intended that this address be used for devices which wish to implement a command type protocol rather than a CSR protocol. For example a PRO3 emulation mode would use this address. See embedding external protocols below.**
0xF1 - 0xF4	**Unallocated Reserved**
0xF5-0xF6	**CONFIGURATION DATA SIZE**
0xF7	**CONFIGURATION DATA REGISTER**
0xF8 - 0xFA	**Software Version**
0xFB	**Node ID (0-127)**
0xFC	**Group ID (1-126)**
0xFD	**UNIQUE ID REGISITER (SEE BELOW) An 16 byte register for reads and 17 bytes for writes**
0xFE	**Reboot register LSB Write 0xDE to reboot**
0xFF	**Reboot register MSB Write 0xAD to reboot**

The registers above 0xF7, 0xFD, 0xFE, 0xFF tend to be "special" and are not meant to be accessed by general contiguous reads and writes.

It is intended that each of the "special" registers are only ever written to individually.

Writing the pattern 0xdead to the addresses 0xFE-0xFF causes the device to reboot after a suitable delay. The reboot sequence should be written in a single transmission packet, starting at address 0xFE:

Example Reboot Packet:
0xFA 0xAF 0x01 0x00 0xFE 0x2 0xA8 0xDE 0xAD 0x73
A write of the word 0xdead to the addresses 0xFD-0xFD with no response requested.

The Configuration Data includes the BootloaderID data and any Hardware configuration data.
This is typically stored in EEPROM on the device.

A read of the UNIQUE ID Register returns the device identification. This is primarily used during device enumeration (see below). But can be read at anytime.   This register contains factory configured identification such as serial number and model number. A read is always <=16 bytes

A write to the UNIQUE ID register allows for the setting of the Node ID. This Node ID can be set either with a write to the Node ID register or with a write to the UNIQUE ID register. When setting the Node ID via a write to the UNIQUE ID register 17 bytes should be written. The first 16 bytes should match the value returned by a read of the UNIQUE ID register (the actual device ID) followed by the desired Node ID.. It is envisioned that a broadcast write to the UNIQUE ID register would be used to set node ID's when multiple devices are on the bus.

Length:

This is the length of the data payload. The count of bytes between the Header Checksum and the total checksum.

A length of 255 (0xFF) is reserved to indicate an extended length packet.

A length if 0 (0x0) indicates that the request packet is a read only packet. A packet with FLAGS==0 and Length==0 while syntactically valid, makes no sense.

Extended Length Packets:

Packets larger than the standard size (254 bytes) are marked with 0xFF in the header length byte field.

The actual data payload length is contained in two bytes after the header checksum. The Extended Length word is sent LSB first.

These two length bytes are followed by a checksum byte. The standard payload data follows. The checksum is calculated in the same manner as the other checksums, it is the XOR of the Extended Length word.

PRO4 extended length packet, each field is a single 8-bit byte.

Start of Packet (Byte 1)	**Start of Packet (Byte 2)**	**Network ID**	**Flags ***	**CSR Address**	**Length**	**Header Checksum ***	**Extended Length (LSB)**	**Extended Length (MSB)**	**Extended Length Checksum**	**Payload Data****	**Total Checksum**

*indicates usage is significantly different than Futaba protocol.
**The first byte(s) in the Payload Data of a response packet is a VideoRay designated device type descriptor.

Header Checksum:

This is the XOR checksum of all six bytes from the first start byte to the length byte inclusive.

In the Futaba protocol this is the Count byte which indicates the number of servos being written to via a long command. This byte is always 1 for a short command. Only short commands can be used where possible when the Futaba protocol is being used on the same physical bus.

Payload data:

This is either the data to be written to the device or data sent as a response. As such it is highly context dependent's. It is recommended that all data transmitted is always little endian (least significant byte first)

Payload data in the initiating packet (I.e from a host to a device) can always be considered a write. In the common case the payload data represents data which should be written into the device memory map starting at the CSR Address, and continuing for length bytes.

The Payload data in the request packets are just the data to be written. There is no additional meta data included.

The Payload data in the response packets is prepended with a device type identifier. If this byte is 0xFF then the second byte is the extended device type identifier, and so on. Device type identifiers are issued by VideoRay LLC.

See appendix for current list of designated device types.

Total Checksum/Payload Checksum:

This is the bitwise XOR of ALL bytes from the first start byte to the last payload data byte inclusive.

This matches the Futaba checksum byte exactly.

Note: This is also just the checksum of the Data Payload since the first 6 bytes of the header XOR with the header checksum is 0.

Protocol:

Device Enumeration: NOT IMPLEMENTED YET

It is desired for a facility to exist where a host can automatically enumerate all attached devices in a short period of time.

Upon power-up devices which support automatic enumeration will transmit their enumeration data. The enumeration data is equivalent to a response packet from memory map addresses 0xFB, 0xFC, 0xFD. The length of the data payload for this packet is 18 bytes.

To minimize collisions during enumeration the device will attempt to generate a random number upon power up (either through a seed recorded in non-volatile memory, or using the lsb of some adc channel to generate a random seed). The device will use this random number to determine a transmit time, bounded between 0.5 and 2 seconds. The device will update this time on each transmission, essentially randomizing it's unprompted communication. The device will update this time upon detecting any communication on the bus. The device will cease transmission upon ANY write to it's specific UNIQUE ID Register.

This randomized transmission protocol should also be used when a broadcast enumeration request is made (a packet requesting data from any address between 0xFB-0xFD inclusive)

It is possible that devices developed by external developers may not support device enumeration. In this case the device developer should specify some mechanism for detecting devices and setting their nodal id's.

Typical Usage:

In typical usage the host will send a request packet to the device. If the LENGTH is greater than 0 they bytes in the PAYLOAD DATA will be written into the device registers.

The FLAG byte is set in the requesting packet to inform the device as to what data should be sent back as a reply. If the value of the FLAG byte is less than 0x80 then it's meaning is device dependent.

If it is greater than or equal to 0x80 then the response is essentially a memory dump of the device memory map space.

EXAMPLES:

To Be Added

Encapsulating External Protocols:

In order to make it easier on external device developers provisions have been made to easily encapsulate an existing protocol.

The CSR Address 0xF0 has been reserved for command encapsulation. Any Data Payload written or read from this address is assumed to be an encapsulated protocol.

The FLAGS byte should be set at 0x80 for encapsulated packets.

The standard Network ID should be used where applicable. If an ID is embedded in the custom protocol then a Network ID of 0xFF is permissible, although system testing to insure that there is no conflict between different devices is required.

The first byte in the PAYLOAD Data of the response is not required to be a VideoRay designated Device type. However it would be preferable if devices would conform to this convention.

VideoRay Designated Device Types:

ALL Device types are defined in protocol_pro4_device_types.h