Application Programming Interface

The purpose of this document is to specify the programming interface for the RCB-LVDS 802.11n 2.4 GHz Wi-Fi wireless electrophysiology data acquisition module. The scope of this document encompasses the commands to the module, reading status from the module, and format of the data stream from the module. Example Matlab® and Octave (www.gnu.org/software/octave) commands are provided.

This document assumes that the module has been configured properly as a station on the local network (see RCB-LVDS User Manual). Communication with the module on the network is accomplished using TCP/IP, HTTP, and UDP. Throughout this document, it is assumed that the address of the module on the network is 192.168.1.93. The user may change this address using the web interface as described in the RCB-LVDS User's Manual.

The RCB-LVDS module is intended to be interfaced with Intan Technologies RHD2000 series interface chips.

Operational Concept

The RCB-LVDS module provides a network interface for an Intan Technologies RHD2000 series electrophysiology interface chip. Multiple modules may exist on the network (with distinct IP addresses) and will operate independently. Status is read from a module by HTTP request. Commands are sent to the module by HTTP Post requests. Data is streamed from the module via UDP. There is no requirement that the UDP destination be the same computer as that from which commands are sent; neither is there a requirement that all commands originate from the same computer.

The two basic modes of the module are Idle and Streaming. Switching between the two modes is accomplished by HTTP Post command. Some status functions are different when in the two modes, and some commands are available only in one mode or the other.

Data streaming is the principal purpose of the RCB-LVDS system. Careful attention has been applied in the design of the module and operation to ensure consistent deterministic sample rates of the channels. The module streams data by composing a repeating sequence of RHD2000 conversion commands that is continuously sent to the RHD2000. The sequence consists of two slots reserved for auxiliary commands, followed by one amplifier conversion command for each enabled channel. Disabled channels are not converted. The entire group of commands (sequence) is sent within one analog-to-digital conversion sample period. If the sample rate is 20460 Hz, the conversion commands all take place within the sampling period $T_s = 48.875$ us. The two auxiliary command slots are themselves composed of sequences, each consisting of 60 commands (programmable by the application). These aux command sequences typically consist of reading auxiliary ADC channels on the RHD2000, temperature, supply voltage, or the read-only registers. These auxiliary sequences cycle every 60 T_s seconds.

Sample Rate Determination

SPI Bit CLK
Sample rate is directly related to the frequency of the SPI bit clock (SCLK). SCLK is derived from a 40MHz oscillator. SCLK is limited to integer divisions by at least 3 of this 40 MHz clock: 13.3 MHz, 10 MHz, 8 MHz, 6.67 MHz, etc.

Offset
Depending on the GUI Selected Sample Rate the SLCK divisor will be Even or Odd. If the Divisor is Even then Offset = 200ns. If Divisor is Odd then Offset = 187.5ns.

Actual Sample Rate
During streaming, a 16-bit channel sample requires Offset + 16.5 bit clock periods. The total number of channel samples per sequence is the number of enabled channels plus two auxiliary time slots. Aux data and Vdd are transmitted in the extra 2 time slots.

SPI CLK = 40Mhz / SPI Divisor
Actual Sample Rate = 1 / ((2 + Channels) * (Offset + 16.5 / SPI CLK))

Selected Sample Rate (Hz)	Max # channels	Offset (s)	SPI Divisor	SPI Clk (Hz)	Actual Sample Rate (Hz)
1000	32	187.5E-9	71	563.38E+3	997.855
1250	32	187.5E-9	57	701.75E+3	1241.003
1500	32	187.5E-9	47	851.06E+3	1502.517
2000	32	187.5E-9	35	1.14E+6	2011.061
2500	32	200.0E-9	28	1.43E+6	2503.129
3000	32	187.5E-9	23	1.74E+6	3039.976
3333	32	187.5E-9	21	1.90E+6	3323.363
4000	32	187.5E-9	17	2.35E+6	4084.967
5000	32	200.0E-9	14	2.86E+6	4922.471
6250	32	187.5E-9	11	3.64E+6	6224.712
8000	32	200.0E-9	8	5.00E+6	8403.361
10000	32	187.5E-9	7	5.71E+6	9564.802
12500	32	187.5E-9	5	8.00E+6	13071.895
15000	32	200.0E-9	4	10.00E+6	15898.251
20000	32	187.5E-9	3	13.33E+6	20639.835
25000	16	187.5E-9	5	8.00E+6	24691.358
30000	16	200.0E-9	4	10.00E+6	30030.030

www.dspw.io

Status

The RCB-LVDS module runs an HTTP server at port 80. Status of the module is obtained by requesting a document from the server with a standard HTTP request.
http://192.168.1.93/intan_status.html
The response is not a static file, but is dynamic newline-delimited ('\n') 8-bit text that is generated by the module at the time of the request. Not all lines are used in this application. An example is shown in Table I.

Table I. Typical *intan_status.html* response.

Line Number	Sample Text Response from RCB-LVDS	Comment
1	Rev: 1383 Modified 2017/06/16 13:42:06	*Revision Information*
2	1382:1383 Mixed revision WC COMPILED:2017/06/20 08:29:55	
3	Unknown Token	*Reserved*
4	ffffffff 6	*Channel Mask and Aux Mask, Hex*
5	Voltage is 3.300000	*Battery Voltage*
6	Unknown Token	*Reserved*
7	Unknown Token	*Reserved*
8	Unknown Token	*Reserved*
9	e800e900ea00eb00ec00fc00fd00fe00ff005000500000000000000000000000000	*Intan Read-Only Registers Contents*
10	192.168.1.148:5001	*UDP Stream Destination Address:Port*
11	4	*Tx Power Amp Backoff, dB*
12	13333333	*SPI Clock Rate*

A Matlab® or GNU Octave command to read the status is as follows:
```
[s, status] = urlread('http://192.168.1.93/intan_status.html')
```
The text will appear in Matlab variable 's'. The status can also be read within a browser simply by entering the URL.

Lines 1 and 2: Revision

These lines report firmware revision information.

Line 4: Channel / Aux Masks

This line reports the current amplifier channel mask and auxiliary channel mask, in little-endian hexadecimal, separated by a space (0x20). The 32 possible amplifier channels are each represented by a bit in this mask; channel 0 is in the lsb, channel 31 is in the msb. If the bit is set, then that channel is sampled and its data is sent in the UDP stream. If a bit is clear, then that channel is not sampled and no data for that channel is sent in the UDP stream.
The auxiliary channel mask should always be 6, indicating two channels of auxiliary data, enumerated 1 and 2.

Line 5: Battery Voltage

Unregulated battery voltage is measured at the time of the HTTP status request, reported in volts. (This is a separate mechanism from the Supply Voltage Sensor on the RHD2000, channel 48.)

Line 9: Intan Read-Only Registers

If the module is in the Idle mode, then the Intan chip is powered-up and the read-only registers are read in the following order, then powered down:
40, 41, 42, 43, 44, 60, 61, 62, 63
The results are reported in 16-bit (4-digit) hexadecimal format with no delimiters. The first eight hex digits are flush digits from the RHD2000 and should be ignored by the application. See RHD2000 data sheet for more information.
If the module is in Streaming mode, the results of the most recent read-only operation are reported, rather than issuing a new sequence to the RHD2000 chip, which would disrupt the streaming sequence.

Line 10: UDP Destination

This is the currently programmed UDP destination IP address and port number. The module will send streaming data to the port at this address.

Line 11: TX Power Amp Backoff

This is the current power backoff of the Wi-Fi power amplifier, in dB (i.e. attenuation). Maximum transmit power is at 0 dB backoff, minimum is at 15 dB backoff. Recommended value for best battery conservation and data integrity is 4 dB backoff. At 0 dB backoff, transmit power is roughly +14 dBm.

Line 12: SPI Clock Rate

This is the currently programmed SPI clock rate (bits per second) for communication with the RHD2000. Maximum value for proper operation is 13333333 bps.

Lines 3, 5, 6, 7, 8: Unused / Reserved

These lines are reserved for future use, and should be ignored by the application. The literal response is "Unknown Text\n", including the newline character (hex 0x0A).

Commands

Command Format

All commands to the RCB-LVDS module are text sent by HTTP Post method. The examples all use the Matlab® or GNU Octave `urlread` function.

SPI Stream On/Off

To turn the SPI stream on, use the following HTTP Post method:
```
[s, status] = urlread('http://192.168.1.93', 'post', ...
    {'__SL_P_ULD', 'ON'});
```
This command will power up the RHD2000 chip, send the previously specified (by other post methods) initial register settings, perform a calibration, then begin streaming the amplifier channel data and auxiliary channel sequence data at the specified SPI bit rate. This command also performs the required 100 us delay between power-up and programming, and the required 100 us delay between programming and calibration.

To turn the SPI stream off, use the following HTTP Post method:
```
[s, status] = urlread('http://192.168.1.93', 'post', ...
    {'__SL_P_ULD', 'OFF'});
```
This command will stop the streaming of data and power-down the RHD2000.

Set Channel Masks

This command sets the channel and auxiliary masks. This command determines which channels are sampled as well as what is actually sent over the UDP link. The format of the masks is hexadecimal digits, 8 digits representing 32 amplifier channels, little endian, and one representing auxiliary sequences. Auxiliary channel mask must be set to 6 for proper operation.
Example: enable channels 0 through 17, and auxiliary sequences 1 and 2:
```
[s, status] = urlread('http://192.168.1.93', 'post', ...
    {'__SL_P_U00', '3ffff 6'});
```

Set Auxiliary Channel Sequence

This command is used to configure the sequence of commands written to the RHD2000 during an auxiliary slot while streaming. Format for string:

nkkddddddddddddddddddddddddddd...

where:

n is the aux sequence number to be assigned (0, 1, 2)

kk is the (decimal) index into the sequence (00, 01, ..., 59)

dddddd... is the value to assign into the array, possibly more than one, 4 hex digits, no whitespace separators.

A maximum of fifteen 4-digit hex words can be programmed with a single Post. Example: program the sequence 0x1200, 0x1300, 0x1400 to sequence 2, beginning at index 7:

```
[s, status] = urlread('http://192.168.1.93', 'post', ...
    {'__SL_P_U01', '207120013001400'});
```

The sequence length is fixed at 60 time slots before repeating (see User's Manual). Thus, to fully populate a sequence, a minimum of four calls must be made to this method.

Note: For compatibility with the GUI and Intan Technologies wired system, the sequence must be offset by one slot: index 00 of the desired sequence must be programmed to the second element in the sequence (01); the last index (59th) of the desired sequence must be programmed to the first element in the sequence (00).

Set UDP Destination

This command sets the destination IP address and port number to use for the next UDP streaming operation. The format is *n.n.n.n:p*, where *n* is a decimal number 0 through 255 inclusive, and *p* is a decimal number. Example: program the destination address to port 1234 at address 192.168.1.222:

```
[s, status] = urlread('http://192.168.1.93', 'post', ...
    {'__SL_P_UUU', '192.168.1.222:1234'});
```

Set Wi-Fi Transmit Power Level

This command sets the Wi-Fi transmit power backoff. This is an attenuation level in dB, such that increasing numerical value is decreasing power. Optimal battery life is achieved with a value of 4. Valid range is from 0 to 15. Example: set backoff to 4 dB:

```
[s, status] = urlread('http://192.168.1.93', 'post', ...
    {'__SL_P_UPA', '4'});
```

Set SPI Bit Rate

This command sets the requested SPI bit rate to use during streaming, in bits per second. Actual bit rate will be an integer division of 40 MHz. Do not program a bit rate faster than $40 \times 10^6 / 3 = 13333333$. Example: set bit rate to 13333333 bits per second.

```
[s, status] = urlread('http://192.168.1.93', 'post', ...
    {'__SL_P_URB', '13333333'});
```

The application may control this value in combination with the channel mask to select a channel sample rate. See section "Sample Rate Determination."

Set Intan Initialization String

This command can be used to explicitly set the initialization command sequence sent to the RHD2000 chip on power-up that is programmed to registers 0 through 21. The format for the string is two hex digits for each register, up to a total of 44 characters. Example: specify register contents for registers 0 through 21 as: 0xde, 0x02, 0x04, 0x02, 0x9c, 0x00, 0x00, 0x00, 0x16, 0x00, 0x00, 0x17, 0x00, 0x10, 0x7c, 0xff, 0xff, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00:

```
[s, status] = urlread('http://192.168.1.93', 'post', ...
    {'__SL_P_UII',
'de0204029c00000016001700107cffffffff00000000'});
```

Reboot Module

This command will reboot the RCB-LVDS module. Example: reboot module.

```
[s, status] = urlread('http://192.168.1.93', 'post', ...
    {'__SL_P_UBB', 'reboot'});
```

Set Digital Input

The streaming data packets contain fields for 16 bits of digital input state. Use this Post method to manipulate the states during streaming. The digital channels are one bit each, enumerated 0 through 15. When represented as a group, they are arranged in little-endian order.

Assign All Bits

Mxxxx

This command assigns the value of all 16 bits at once, using four hexadecimal digits, little endian. Example: set digital inputs 3 and 12 through 15, clear all other bits.

```
[s, status] = urlread('http://192.168.1.93', 'post', ...
    {'__SL_P_UDI', 'Mf008'});
```

Toggle Bits

Xxxxx

This command toggles selected bits, using four hexadecimal digits. Example: toggle digital inputs 3 and 12 through 15, don't change all other bits.

```
[s, status] = urlread('http://192.168.1.93', 'post', ...
    {'__SL_P_UDI', 'Xf008'});
```

Or Bits

Oxxxx

This command sets selected bits, using four hexadecimal digits. Example: set digital inputs 3 and 12 through 15, and leave all others unchanged.

```
[s, status] = urlread('http://192.168.1.93', 'post', ...
    {'__SL_P_UDI', 'Of008'});
```
Note: The command character is upper case 'O', hex 0x4f.

And Bits

Axxxx
This command clears selected bits, using four hexadecimal digits. Example: clear all inputs except 3 and 12 through 15.
```
[s, status] = urlread('http://192.168.1.93', 'post', ...
    {'__SL_P_UDI', 'Af008'});
```

Set Bit

Sdd
This command sets one selected bit. Example: set digital input 3.
```
[s, status] = urlread('http://192.168.1.93', 'post', ...
    {'__SL_P_UDI', 'S3'});
```

Clear Bit

Cdd
This command clears one selected bit. Example: clear digital input 12.
```
[s, status] = urlread('http://192.168.1.93', 'post', ...
    {'__SL_P_UDI', 'C12'});
```

Toggle Bit

Tdd
This command toggles one selected bit. Example: toggle digital input 15.
```
[s, status] = urlread('http://192.168.1.93', 'post', ...
    {'__SL_P_UDI', 'T15'});
```

Data Stream Packets

Data is streamed from the module to the UDP server in packets, and every packet has a header that describes the packet content. UDP packet delivery over the network is not guaranteed, but the header contains sufficient information to enable the UDP server (Application) to maintain synchronization with the module, even if some data will be missing due to lost packets.
Packet length is dynamic depending on the parameters, but is limited to no more than 1480 bytes, including header. The exact number of data and ordering are completely specified in the header of each packet.
It is important to note that while packet contents may change from one packet to the next, the packet header structure will always accurately describe the contents.

Header Structure

The header structure is shown in Fig. 1.

```
/*
 * Structure for UDP packet data header
 */
typedef struct udpHdr{
      uint8_t    magic;            // = 0xc5, magic number to identify
this packet type
      uint8_t    sod;        // offset into this packet for start of
data
      uint8_t    mac[6];     // mac address of data source
      uint32_t   sn;         // sequence number of this packet
      uint64_t   reserved;   // reserved field
      uint32_t   spiBitRate;     // actual bit rate programmed to SPI
interface
      uint32_t   chanMask;   // bitmask of channels in this packet
      uint8_t    auxMask;    // bitmask of aux channels in this packet
                             // number of samples per interval is bitcnt
                             (chanMask) + bitcnt(auxMask)
      uint8_t    auxPhase;   // sample phase of aux channel sequence
      uint16_t   numTs;          // number of sample periods in this
packet

                             // total number of 16-bit samples in this
                             packet is numTs * (num chan + aux)
      uint16_t   vbat;       // latest battery voltage
      uint16_t   digIn;          // state of digital input bits (IR,
GPIO)
} udpHdr_t;

#endif /* SPI_PP_H_ */
```

Fig. 1. UDP packet header structure.
The fields are described below, with offsets specified in bytes. Note that multi-byte fields are stored little-endian.

Offset 0: magic
This byte is always 0xC5 to identify the packet type.

Offset 1: Start of Data
This byte contains the offset within this packet of the start of data. Nominal value is 36. Application should always use the value of this field to begin parsing data.

Offset 2: MAC of Source
These six bytes contain the MAC address of the module that generated the packet.

Offset 8: Sequence Number

This 32-bit unsigned integer is the sequence number of the packet. The module increments this number by one every packet, and it can be used by the application to detect if / how many packets have been lost. This counter restarts at 0 for every new Streaming request.

Offset 20: SPI Bit Rate

This 32-bit unsigned integer field reports the actual SPI clock rate used to generate the packet data, in units of bits per second.

Offset 24: Amplifier Channel Mask

This 32-bit field indicates which amplifier channels are present in this packet.

Offset 28: Auxiliary Slot Mask

This 8-bit field indicates which auxiliary slots are present in this packet.

Offset 29: Start Phase of Auxiliary Sequence

The auxiliary channel slots are sequences that span 60 sample periods, programmed by the application. This 8-bit field indicates the index into this sequence (0 – 59) of the first occurrence of an auxiliary slot sample in this packet.

Offset 30: Number of Sample Periods in this Packet

This 16-bit unsigned integer field indicates how many sample periods are represented in this packet.

Offset 32: Unregulated Battery Voltage

This 16-bit unsigned integer reports the latest value of the unregulated battery voltage on the module. Conversion to volts is accomplished by the following operation:
```
float b = (0xfff & (p->vbat >> 2)) * 1.467 / 4096 * 62.0 / 15.0;
```

Offset 34: Digital Input State

This 16-bit field reflects the state of the digital input bits at the end of the epoch represented by this packet. The digital input bits therefore have a time resolution of numTs sample periods.

Data Format

All the data are 16-bit words, little-endian. The data begins at byte offset p->sod in the packet, and consists of p->numTs groups. Each group has the same order and format, and there are no separators between groups.
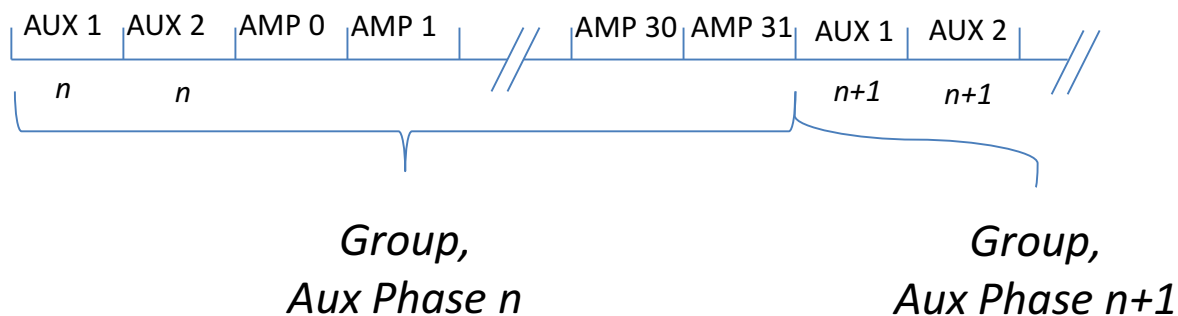
| AUX 1 | AUX 2 | AMP 0 | AMP 1 | // | AMP 30 | AMP 31 | AUX 1 | AUX 2 | // |
| n | n | | | | | | n+1 | n+1 | |

Group,
Aux Phase n

Group,
Aux Phase n+1

Fig. 2. Illustration of data, sequence of groups for auxMask = 6, chanMask = 0xFFFFFFFF. The group consists of 34 words, or 68 bytes.
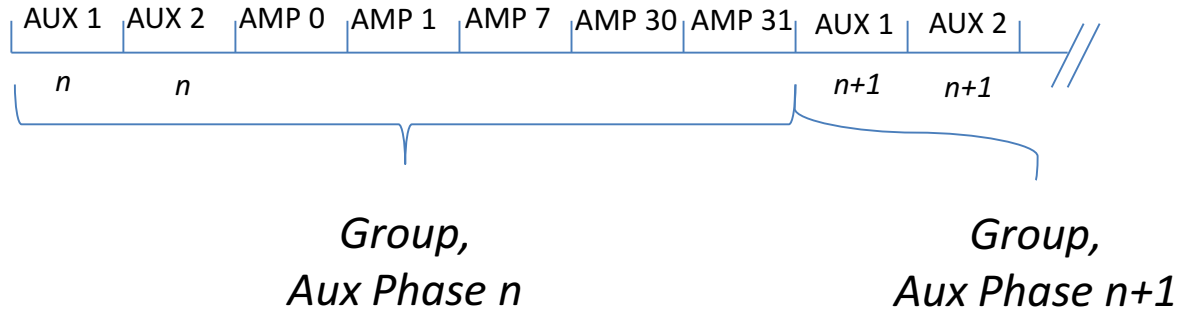
AUX 1	AUX 2	AMP 0	AMP 1	AMP 7	AMP 30	AMP 31	AUX 1	AUX 2	
n	*n*						*n+1*	*n+1*	

Group,
Aux Phase n

Group,
Aux Phase n+1

Fig. 3. Illustration of data, sequence of groups for auxMask = 6, chanMask = 0xC0000083. The group consists of 7 words, or 14 bytes.

Group Structure

Each group is comprised of auxiliary slots followed by amplifier channel slots. The group is completely specified by the auxiliary mask and channel mask.

Auxiliary Slots

The auxiliary slots are specified by the auxiliary mask value (p->auxMask). One datum is present for each set bit in p->auxMask, in order from lsb to msb. Thus, if the p->auxMask is 6 (0110_2), then the first datum in the group is from auxiliary sequence 1, and the second datum is from auxiliary sequence 2. (Recall that auxiliary slots are selected from 60-element sequences. All auxiliary slots are synchronized with the same phase, p->auxPhase. Successive groups have consecutive auxiliary phases.)

Amplifier Channel Slots

The amplifier channel slots are specified by the amplifier channel mask value (p->chanMask). One datum is present for each set bit in p->chanMask, in order from lsb to msb. Thus, if p->chanMask is 0xffffffff, then the first datum after any auxiliary data is amplifier channel 0; the second datum is amplifier channel 1; etc., the 32nd datum is amplifier channel 31.