# HPSDR USB Data Protocol

## Revisions

| Rev | Date | Changes | By |
|---|---|---|---|
| 1.0 | 8th June 07 | First Release | VK6APH |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

## Protocol Overview:

- The USB data consists of 512 byte packets
- The sample rate from the receiver A/D converter to the PC is selectable between 48/96/192kHz at 24 bits
- The sample rate from the microphone to the PC is 48kHz at 16 bits
- The sample rate from the PC to the speakers/headphones is 48kHz at 16 bits
- The sample rate from the PC to the I/Q transmit audio is 48kHz at 16 bits
- Control signals that are high priority are sent each 512 block, lower priority data is sent less frequently

## Functions required:

- PTT active
- Dot/dash key active
- A/D sampling speed 192/96/48kHz
- NCO Frequency
- Penelope Open Collector outputs
- Mercury Pre-amps and attenuator

## Protocol

The protocol consists of a 512 byte frame comprising a sync sequence, Command & Control data and ADC or DAC data.

A frame length of 512 bytes is used since this is the maximum number of bytes that the FIFO in the FX2 USB interface can hold.

High priority control data is sent as part of each frame e.g. PTT command/request. Lower priority data is sent as available on a predefined schedule e.g. NCO frequency.

**Sync Sequence**

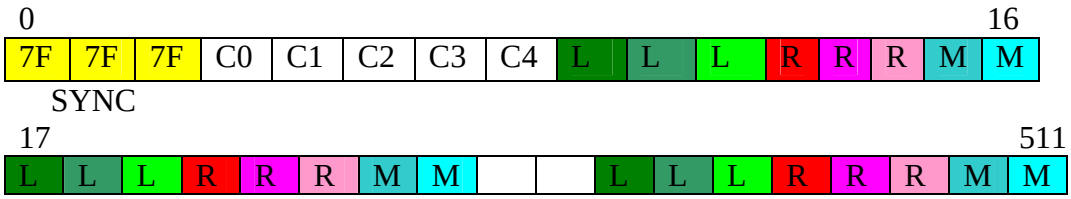
This consists of a three byte sync sequence comprising <0x7F><0x7F><0x7F> . The sync sequence is sent at the start of every 512 byte frame.

**Protocol – From HPSDR to PC**

Since the Janus ADC uses 24 bits per sample and the Microphone/Line ADC 16 bits per sample then, in order that an integer number of Left/Right samples will be included in the 512 byte packed, the maximum number of samples is

(512 – 8) = 63 x (3 + 3 + 2) bytes i.e. 63 24 bit left/right samples and 63 16 bit Mic/Line In samples

This provides 8 Bytes to transfer status data from the FPGA to the PC. The first characters in the 512 byte packet will be sync which is 0x7F7F7F. Since 3 bytes are used for the sync character 5 bytes are used to send Command & Control data to the PC.




Where Cn is a Command/Control Byte and

| | |
|---|---|
| L | Bits 23 – 16 of Left sample |
| L | Bits 15 – 8 of Left sample |
| L | Bits 7 – 0 of Left sample |
| R | Bits 23 – 16 of Left sample |
| R | Bits 15 – 8 of Left sample |
| R | Bits 7 – 0 of Left sample |
| M | Bits 15 – 8 of Mic/Line sample |
| M | Bits 7 – 0 of Mic/Line sample |

**Note:** When sampling speeds of 96kHz and 192kHz are selected the microphone is still sampled at 48kHz but the same data is repeated twice (96kHz) or four times (192kHz).

**Command & Control**

**C0**

```
0 0 0 0 0 0 0 0
              | |
              | +------------ PTT/DOT ( 1 = active, 0 = inactive)
              +-------------- DASH      ( 1 = active, 0 = inactive)
```

C1 – C4 are reserved for future use.

**Protocol – From PC to HPSDR**

The PC sends two audio streams to the HPSDR. These are

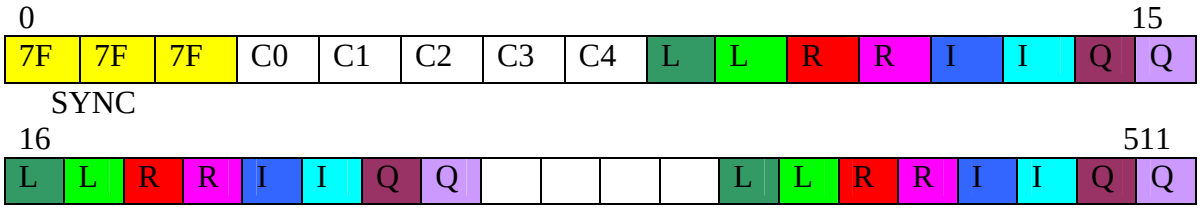
1. 48kHz 16 bit Left/Right received audio
2. 48kHz 16 bit I/Q

Since the received audio is also used to monitor the transmitted audio then these two streams must be available simultaneously.

Since the DACs use 16 bits per sample then, in order that an integer number of Left/Right and I/Q samples will be included in the 512 byte packed, the maximum number of samples is

(512 – 8) = 63 x 4 x 2 bytes i.e. 63 Receiver L/R samples and 63 I/Q L/R samples

This provides 8 bytes to transfer status data from the PC to the FPGA. The first characters in the 512 byte packet will be sync which is 0x7F7F7F. Since 3 bytes are required for the sync character 5 bytes are used to send Command & Control data.




Where

| | |
|---|---|
| L | Bits 15 - 8 of Left Rx audio sample |
| L | Bits 7 - 0 of Left Rx audio sample |
| R | Bits 15 – 8 of Right Rx audio sample |
| R | Bits 7 – 0 of Right Rx audio sample |
| I | Bits 15 - 8 of I Tx audio sample |
| I | Bits 7 - 0 of I Tx audio sample |
| Q | Bits 15 - 8 of Q Tx audio sample |
| Q | Bits 7 - 0 of Q Tx audio sample |

**Command & Control**

NOTE: Bits 7-1 of C0 form an address that determines how C1-C4 should be decoded. C0 is varied round-robin fashion so that all addresses are sent in sequence.

**C0**

```
0 0 0 0 0 0 0 0
                |
                +------------ MOX ( 1 = active, 0 = inactive)
```

**C1**

```
0 0 0 0 0 0 0 0
              |  |
              +  +------------ Speed ( 00 = 48kHz,  01 = 96kHz, 10 = 192kHz)
```

**C2**

```
0 0 0 0 0 0 0 0
|               | |
|               | +------------ Mode ( 1 = SSB/CW, 0 = All other modes)
+---------------+-------------- Open Collector Driver on Penelope
```

**C3**

```
0 0 0 0 0 0 0 0
|             | | |
|             | |  +---------- Mercury Preamp1 (1 = on, 0 = off)
|             | +------------- Mercury Preamp2 (1 = on, 0 = off)
+-----------+--------------- Mercury Attenuator (000000 = 0dB, 111111 = 31.5dB)
```

C4 reserved for future use

**C0**

```
0 0 0 0 0 0 1 0
                |
                +------------ MOX ( 1 = active, 0 = inactive)
```

**C1, C2, C3, C4 --------** NCO Frequency in Hz (32 bit binary representation – C4 holds MSB and C1 LSB)

**Implementation**

The Windows version of this protocol has been implemented using LibUsb-Win32 see http://libusb-win32.sourceforge.net/

Libusb-win32 is a port of the USB library libusb to the Windows operating systems (Win98SE, WinME, Win2k, WinXP). The library allows user space applications to access any USB device on Windows in a generic way without writing any line of kernel driver code.

The Audio data stream runs as Bulk Data (i.e. not Isochronous). Data is read from (LibUsb) Endpoint 0x86 and written to Endpoint 0x02.

The code implementing the protocol can be found in HPSDR SVN. The most interesting modules are

svn://206.216.146.154/svn/repos_sdr_hpsdr/trunk/KD5TFD/PowerSDR/HPSDR-1.6.3/Source/KD5TFD-VK6APH-Audio/io-thread.c
(frames packets to/from PowerSDR/OzyJanus)

and

svn://206.216.146.154/svn/repos_sdr_hpsdr/trunk/KD5TFD/PowerSDR/HPSDR-1.6.3/Source/KD5TFD-VK6APH-Audio/OzyIO.c
(talks to libusb)

Control via the SDR 1K's parallel port is implemented with FX2 firmware responding to Vendor control messages on the control pipe (Endpoint 0). The PC side implementation of this can be found in:

svn://206.216.146.154/svn/repos_sdr_hpsdr/trunk/KD5TFD/PowerSDR/HPSDR-1.6.3/Source/Console/OzySDR1kControl.cs

The bulk of the FX2 implementation is in:

svn://206.216.146.154/svn/repos_sdr_hpsdr/trunk/KD5TFD/Ozy-FX2-SDR1K-Control/Initial/src/sdr1kctl.c

and

svn://206.216.146.154/svn/repos_sdr_hpsdr/trunk/KD5TFD/Ozy-FX2-SDR1K-Control/Initial/src/hpsdr_main.c

ENDS.