Guide for implementation of Host software for SpikerBox HID USB device

This document explains how to implement communication layer for the Backyard Brains HID USB SpikerBox for firmware version V0.09. This communication layer will be used for devices Neuron SpikerBox and Muscle SpikerBox with HID USB capability.

This communication has been implemented for the purpose of Spike Recorder open source software that can be found on GitHub:
https://github.com/BackyardBrains/BYB-Neural-Recorder/tree/hidusb
Example C++ class that implements communication with HID USB SpikerBox can be found on:
https://github.com/BackyardBrains/BYB-Neural-Recorder/blob/hidusb/src/engine/HIDUsbManager.cpp

Implementation is based on HIDAPI Multi-Platform library for communication with HID devices. Library files (hidapi.h, hid.c, hidapi.dll and hidapi.lib) can be found in “Support” and “Engine” directory of Spike Recorder GitHub project at:
https://github.com/BackyardBrains/BYB-Neural-Recorder/tree/hidusb/support
https://github.com/BackyardBrains/BYB-Neural-Recorder/tree/hidusb/src/engine

Example of devices firmware implementation can be found in GitHub repository:
https://github.com/BackyardBrains/SpikerBoxPro

SpikerBox hardware capabilities

SpikerBox HID USB hardware has two recording channels with sample rate of 10kHz and resolution of 10bits per sample. In addition to recording channels made for recording of analog electrophysiological signals it has two logic level inputs used to signal onset of experiment’s important events to the Host application.

SpikerBox has one USB endpoint of HID-Datapipe device class. That endpoint is used to transfer sample streams from two recording channels and to pass custom communication messages in both directions from the Host software to the SpikerBox and from the SpikerBox back to the Host software.

Host software will have to deal with two different protocols. First one is HID-Datapipe protocol which is UART-like unformated datastream over HID interface. Payload data inside every HID-Datapipe data frame will be formatted according to the SpikerBox custom protocol.

SpikerBox custom protocol

Sample stream is a byte stream that is divided into frames. Each frame contains one sample for every recording channel. In current version of hardware we have two recording channels so each frame will contain two samples. Each sample is 10 bit value and we use two bytes for every sample. So, in current two channels SpikerBox one frame will contain 4 bytes.

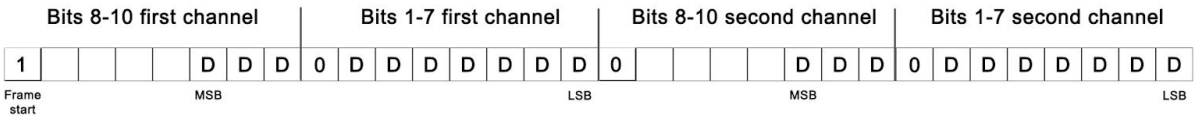

Image 1. Structure of one frame for two channels communication. Boxes marked with "D" represent data bits and empty boxes represent bits that are not used.

In order to mark beginning of the frame we reserve most significant bit of every byte for the frame flag. If this bit is set to 1 current byte represents start of the frame. (Take a look at *Image 1* for details). Ten bits of one data sample are divided in two bytes in such a way so that first byte contains 3 most significant bits and second byte contains 7 least significant bits.

Custom messages are currently used for 7 different purposes:

- start/stop of sampling stream
- to inform Host about firmware version, hardware version and SpikerBox device type
- to pass information about events
- to pass information about maximal sampling rate and number of channels
- state of power rail in device (ON or OFF)
- start firmware update procedure
- to inform Host about type of device connected to hardware I/O interface

(Note that in future versions of firmware we will add more custom messages but all messages will follow same syntax and protocol)

Syntax of one message can be divided to five parts:

1. *type of message*
2. *colon separator*
3. *value of message*
4. *semicolon terminator.*

in that same order.

Below are listed all messages that are supported by firmware V0.09:

Start streaming sample data

Message sent by the Host:

start:;

After receiving this message the SpikerBox hardware will start streaming sampling data.

Stop streaming sample data

Message sent by the Host:

h:;

After receiving this message the SpikerBox will stop streaming sampling data.

Inquiry for hardware version, firmware version and hardware type

Message sent by the Host:

?:;

After receiving this message SpikerBox will reply with three messages that contain information about hardware version, firmware version and hardware type:

FWV:[firmware version string];HWT:[hardware type string];HWV:[software version string];

example:

FWV:0.01;HWT:NEURONSB;HWV:0.01;

Current we have two hardware types *NEURONSB* (Neuron SpikerBox Pro) *and MUSCLESB* (Muscle SpikerBox Pro). Every implementation of Host software should check for firmware version, hardware version and hardware type and inform user about unsupported hardware and guide user to update Host software if update is available.

Inquiry about state of power rail on device

V:;

After receiving this message SpikerBox will reply with message that contains information about state of power rail in device.

PWR:[number that represent logic state];

Because device is connected to Host computer via USB connector digital controller of device can work even when device is turned OFF or battery is empty since it will be powered through USB connection. But amplifiers and analog part of device circuit will not be active if device is turned OFF. Response to inquiry about state of power rail is message with two possible message value (0 or 1):

- **PWR:1;** which means that device's power is turned ON and
- **PWR:0;** which will be sent back to Host if power is OFF

Start firmware update procedure

update:;

After receiving this message controller on device will prepare for update of firmware. For SpikerBox we use controllers from TI MSP430 family and MSP430 USB Software Field Firmware Update Process that is explained in details in Texas Instruments manual "USB Field Firmware Updates on MSP430™ MCUs":
http://www.ti.com/lit/an/slaa452c/slaa452c.pdf

Event message

Message sent by the SpikerBox:

EVNT:[event number];

example:

EVNT:2;

Current hardware and firmware version supports two logic level inputs that can trigger two different events. That events will produce two different message: **EVNT:1;** and **EVNT:2;** Note that in future version of firmware we expect to add more capabilities to the SpikerBox hardware so the event message values will be expanded accordingly.

Type of device connected to hardware I/O interface

When new device is connected or disconnected to expansion I/O interface SpikerBox will send custom message that will contain information about type of device connected to SpikerBox:

BRD:[board type number];

Example:

BRD:2;

Value of the message will be number that represents type of the board. Details on decoding type of the board and description of modes of operation of device can be found in document: “SpikerBox Pro Hardware SDK” on BackyardBrains website.

Maximal sample rate and number of channels

Message sent by Host:

max:;

After receiving this message the SpikerBox will respond with information about maximal sample rate and maximum number of channels supported:

MSF:[sample rate in Hz];MNC:[number of channels];

Example:

MSF:10000;MNC:2;

Current version of SpikerBox firmware supports only sampling at 10 kHz and will always send stream of data from both channels. In future version of firmware one can expected that this will be configurable using new types of messages sent from the Host software.

Since stream of samples from recording channels and custom messages are sharing the same HID channel all messages sent from SpikerBox are embedded in escape sequences. The same is not true for the messages sent from the Host software to the SpikerBox. Escape sequence that signals beginning of the message block can appear on HID communication channel at any time. So the Host software should be capable of detecting escape sequence and extract message from stream of data if SpikerBox is streaming recording data. Even if SpikerBox is not streaming recording data to Host software the messages sent from the SpikerBox to the Host software will still be embedded in the escape sequence.

Escape sequence that signals start of one or several messages is array of 6 bytes. Below are given the 6 bytes in hexadecimal representation:

```
0xFF 0xFF 0x01 0x01 0x80 0xFF
```

Array of bytes that signals end of block of messages is 6 bytes long and is given below in hexadecimal representation:

```
0xFF 0xFF 0x01 0x01 0x81 0xFF
```

HID-Datapipe protocol and HIDAPI library

When user connects SpikerBox hardware to USB port of the computer the SpikerBox will be automatically recognized as HID USB device.
To list all available HID devices connected to Host computer use **`hid_enumerate`** function from the HIDAPI library.

Example:

```
devs = hid_enumerate(0x0, 0x0);
```

The **`hid_enumerate`** will return list of hid_device_info structures. That list holds description of all HID devices connected to user's computer. In order to find Backyard Brains' SpikerBox one should filter list using Backyard Brains' Vendor ID (VID) that should be equal to hexadecimal value **0x2047** and Product ID (PID) that should be equal to hexadecimal value **0x3e0**.

In order to connect to SpikerBox one can use **`hid_open`** function from the HIDAPI library.

Example:

```
handle = hid_open(BYB_VID, BYB_PID, NULL);
```

First two arguments should be set to Backyard Brains' VID and PID and function will return hid api device handle structure.

After connecting to the device one can send data messages to SpikerBox using **`hid_write`** function and receive recording data stream and messages from SpikerBox using **`hid_read`** function.

The HID datapipe interface is based on USB's interrupt transfer type. Every 1ms one USB frame is transferred from SpikerBox to Host software. Each frame received on the Host software side will have 64 bytes. First byte represents type of the frame (this will be constant value when using **`hid_read`** function in HIDAPI library) and can be ignored. Second byte represents length of data (in bytes) in payload and it should be used when extracting data from payload block. And rest 62 bytes represent actual data formatted according to the SpikerBox custom protocol.

Implementation of communication protocol should have separate thread that reads data from HID device (using **`hid_read`** function) at least every 1ms. For more details on how to read data from HID device please refer to HIDUsbManager example class implementation in the Spike Recorder project.

In order to send data to SpikerBox device one should use **`hid_write`** function. This function has three parameters. First parameter of function should be handle structure of connected HID device. Second parameter is data buffer that follows HID specification. First byte in the defines type of HID data frame and should always be set to **`0x3F`** value. Second byte represents length of payload data and should be always be set to **`0x3E`** value which equals to 62 in decimal representation. Buffer should be always **64** bytes long even if the data content (custom message) is shorter than 64 bytes. Third parameter of **`hid_write`** function represents length of buffer and should always be set to value **64**.

In order to close communication with the SpikerBox HID device use **`hid_close`** function from the HIDAPI library.

One typical communication session with SpikerBox looks like this:

1. Get information about connected devices using **`hid_enumerate`**
2. Connect to SpikerBox using BackyardBrains' VID and PID and **`hid_open`** function
3. Start reading thread that reads (**`hid_read`**) HID device packets at least every 1ms
4. Ask for firmware version and hardware type by sending **`?:;`** message
5. Ask for maximal sample rate and number of channels using **`max:;`** message
6. Start sampling and stream of data using **`start:;`** message
7. Receive sampling stream of data and event messages and display it to user
8. Stop sampling and stream of data using **`h:;`** message
9. Close connection with device using **`hid_close`** function