Goal: Hardware-accelerated spike sorting

Algorithm: Independent Component Analysis (ICA)

Initially, KiloSort was considered for spike sorting, however, it was decided that it is not the right tool for this application, since it does not allow real-time analysis for channel counts greater than 200. Instead we are now considering Independent Components Analysis (ICA) for blind source signal separation.

The Logic: For both online visualization and offline analysis, hardware-accelerated signal separation of high-channel count streams of neural data is sorely needed. It has previously been shown as a non-real-time implementation that one solution is to pass the data down an analysis pipeline consisting of filtering, blind source signal separation (e.g. ICA), a classification step (e.g. thresholding + SNR criteria) to yield a stream of spikes from neurons with high reliability.

To implement this approach for real-time applications, we will divide the signal separation process into a slow initialization phase and then a real-time online phase with periodic updates. In the first phase a small amount of data (e.g. 60 seconds) is captured and fed into the signal separation algorithm described above on a PC. In the second phase, the resulting mapping is used to filter the incoming channel data (this is fast and can often be expressed as matrix multiplication) on the fly on FPGA. In a third phase, every N minutes the mapping is recomputed on PC, and if significantly different from the map in use on the FPGA, then the map on the FPGA is updated. On the PC, we calculate not only the weights but also threshold values and identify which components are neurons and which are noise or artifacts.

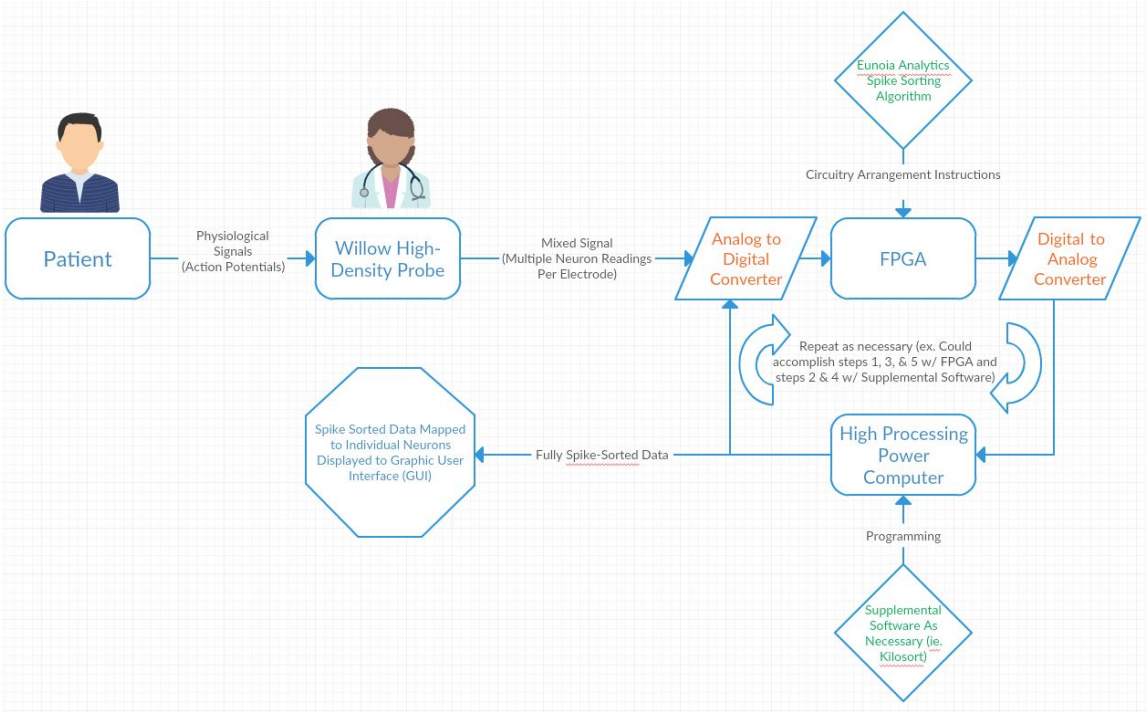
The ICA workflow:

A) filter (highpass usually)
B) ICA to separate 'sources' (the neurons that generate individual spikes)
C) Classification (threshold or SNR criteria to identify spikes)

Within B,

1) PC - use a small subset of the samples to perform ICA and get a square weight matrix
2) PC to FPGA - transfer the weight matrix values
3) PC to FPGA - transfer samples
4) FPGA - matrix multiply between samples and weight matrix
5) FPGA to PC - read back the result to PC memory
6) PC to FPGA - at some time point, update the weight matrix and resend to FPGA if necessary.

The output from step 5 moves to Part C) to be classified, then that data can be visualized in some manner.



NOTE: This visual is slightly outdated and does not represent the current state of system design. However, it captures the general idea behind the project.

Test Data: Here

In the folder you will find

example ICA data
A few example files from a larger corpus of data (see folder_contents.txt). In particular *ICA.h5 contains an example weight matrix (see read_weight_matrix.m).

The *raw.h5 file contains the raw data consisting of the information listed below. It was collected in a experiment where a head-fixed mouse was shown a movie on an ipad (confirmed using a photodiode) and neural activity was recorded by a microelectrode array (MEA) using the Willow system, and simultaneously a single neuron was recorded with a patch pipette using another data aquisition system (DAQ). Thus,

photodiode - a single channel time-series of brightness levels that prove a visual stimulus was

given and when
rawPipette - the single channel of neural data from the patch pipette
rawMEA - the neural data from the MEA as originally captured
syncMEA - a time-shifted version of rawMEA that time-aligns to rawPipette

channel map
A mapping from channel number to recording site on probe in ascii text and python pickle file. See readme for details.

ICA matlab code
Copy of the latest matlab code being used by LeafLabs. See readme.txt for details.