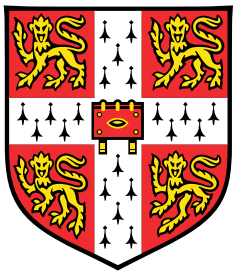
A Neural Signal Processor for Low-Latency Spike Inference

Using Hardware-Software Codesign



Chongxi Lai

Department of Physiology, Development and Neuroscience
University of Cambridge

This dissertation is submitted for the degree of
Doctor of Philosophy

Clare Hall July 2020

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 60,000 words and has fewer than 150 figures.

Chongxi Lai
July 2020

Abstract

A Neural Signal Processor for Low-Latency Spike Inference (by Chongxi Lai)

This thesis describes the development of a system that can assign identities to a population of single-units, in multi-electrode recordings, at single-spike resolution with low-latency. The system has two parts. The first is a Field-Programmable Gate Array (FPGA)-based Neural Signal Processor (NSP) that receives raw input and generates labelled spikes as output, a process referred to as real-time spike inference. The second is a piece of software (Spiketag) that runs on a PC, communicates with the NSP, and generates a spike-sorted model to guide the real-time spike inference. The NSP provides clocks and control signals to five 32-channel INTAN RHD2132 chips to manage the acquisition of 160 channels of raw neural data. In parallel, the NSP further filters, detects and extracts extracellular spike waveforms from the raw neural data recorded by tetrodes or silicon probes and assigns single-unit identity to each detected spike. A set of Python application programming interfaces (APIs) was developed in Spiketag to enable the communication between the NSP and the PC. These APIs allow the NSP to obtain a model from the PC, which holds parameters such as reference channels, spike detection thresholds, spike feature transformation matrix and vector quantized clusters generated by spike sorting a short recording session. Using the spike-sorted model, the NSP performs data acquisition and real-time spike inference simultaneously. Algorithmic modules were implemented in the FPGA and pipelined to compute during 40 ms acquisition intervals. At the output end of the FPGA NSP, the real-time assigned single-unit identity (spike-id) is packaged with the timestamp, the electrode group, and the spike features as a spike-id packet. Spike-id packets are asynchronously transmitted through a low-latency Peripheral Component Interconnect Express (PCIe) interface to the PC, producing the real-time spike trains. The real-time spike trains can be used for further processing, such as real-time decoding. Several types of ground-truth data, including intracellular/extracellular paired recordings, synthesized tetrode extracellular waveforms with ground-truth spike timing, and high-channel-count silicon probe recordings with ground-truth animal positions during navigation were used to validate the low-latency ($\sim$1 ms) and high-accuracy (as high as state-of-the-art offline sorting and decoding algorithms) of the NSP's real-time spike inference and the NSP-based real-time population decoding performance.

I would like to dedicate this thesis to my beloved parents, wife and son.

Acknowledgements

Finishing this thesis was incredibly hard for me. However, I am lucky to have received much support during my PhD. First, I am greatly indebted to my parents and my wife accompanied me through all the joys and struggles with me. Their accompany and encouragement prevents me from giving up in those hard times.

I came into neuroscience as an electrical engineer worked in industry with little self-taught neuroscience knowledge. The Cambridge-Janelia PhD program has awarded me with unique research experience. During my graduate study in Cambridge, my supervisor Prof. Ole Paulsen, spent much efforts and time teaching me on various subjects in neuroscience. I want to thank him for his support and teaching. His rigorous but open-minded attitude on science will always influence me. Without his intellectual support, I will spend much more years to acquire the necessary and in-depth knowledge to achieve my research goals in neuroscience.

In other years of my graduate study, I worked with my Janelia mentor Dr. Tim Harris. My thesis project was to engineered a real-time signal processor that can identify population single-unit activity, at a single-spike resolution, during the data acquisition with tetrodes and silicon probes. It was a quite challenging project that not only requires much of my hard work on trial-and-error and also tons of resources. It would not have been possible without the continuous support of Dr. Tim Harris. He gave me the academic freedom while providing enough resources and ensuring that I stay on course and do not deviate from the core of my research goal. I am hugely grateful for Dr. Albert Lee, who has also supervised on my thesis and other projects. He always offered sharp intellectual input and practical suggestions on the project and the writing of this thesis.

I am grateful for the people who gave support on this project. This includes Brian Barbitis, Aarón Cuevas López, Yixin Chi, Dr. Nakul Verma, Dr. Lakshmi Narayan, Dr. Brian Lustig, Dr. Shinsuke Tanaka, Dr. David Hunt and Dr. Mladen Barbic. Their contribution are specified in this thesis when it comes to details. I also would like to thank Dr. Sue Jones and Dr. Erik Snapp, for their support for my graduate study, thesis writing and career suggestions. Also, I would like to thank Michelle Quiambao, our lab coordinator, for the daily supporting on purchasing equipment and all sorts of lab activities.

Table of contents

Nomenclature

Acronyms / Abbreviations

ALU Arithmetic Logic Unit

API Application Programming Interface

ASIC Application Specific Integrated Circuit

AXI Advanced eXtensible Interface

BMI Brain-Machine Interface

BRAM Block Random Access Memory

CLB Configurable Logic Block

DBSCAN Density-Based Spatial Clustering of Applications with Noise

DMA Direct Memory Access

DPRAM Dual-Port RAM: a RAM that can be read/written by two circuits via two ports

DPS Density Peak Search

DSP Digital Signal Processor

EDA Electronic Design Automation

EPSP Excitatory Post-Synaptic Potential

FET Field Effect Transistor

FF Flip Flop

FIFO First In First Out

FIR Finite Impulse Response Filters

FMC FPGA Mezzanine Card

FPGA Field Programmable Gate Array

FSM Finite State Machine

HDBSCAN Hierarchical Density-Based Spatial Clustering of Applications with Noise

HDL Hardware Description Language

HLS High Level Synthesis

IC Integrated Circuit

IDE Integrated Development Environment

IIR Infinite Impulse Response Filters

IO Input and Output

IP Intellectual Property: here specifically means a preconfigured FPGA digital circuit

IPSP Inhibitory Post-Synaptic Potential

ITDP Input-Timing-Dependent Plasticity

kNN k-Nearest-Neighbors

LFP Local Field Potential

LTP Long Term Potentiation

LUT Look Up Table

LVDS Low-Voltage Differential Signalling

MAC Multiply Accumulate

MEC Medial Entorhinal Cortex

MOSFET Metal Oxide Semiconductor FET

MUA Multi-Unit Activity

MUAP Multi-Unit Activity sampled and encoded in 33-bit where the last bit represents the Peak

NMDA N-methyl-D-aspartate receptor

NSG Neural Signal Generator

NSP Neural Signal Processor

PC Personal Computer

PCA Principal Component Analysis

PCB Printed Circuit Board

PCIe Peripheral Component Interconnect express

PFC Prefrontal Cortex

PLL Phase Locked Loop

PXI PCI eXtensions for Instrumentation

RAM Random Access Memory

RTL Register Transfer Level

SPI Serial Peripheral Interface

STDP Spike-Timing-Dependent Plasticity

SUA Single-Unit Activity

SWR Sharp Wave-Ripples

VQ Vector Quantization

VR Virtual Reality

Chapter 1

Introduction

Mammalian brains are composed of tens of millions to hundreds of billions of neurons. These neurons are distributed across brain regions. Most neurons generate action potentials, also known as spikes, to communicate with each other. Importantly, these spikes are the universal unit of information exchange inside the brain; they carry information about sensation, movement and cognition. To understand how the interactions between neurons generate behavior is one of the primary goals of neuroscience. In order to achieve this, we need to understand how information is encoded, represented and transformed by patterns of spikes. In other words, we need to understand the neural code the nervous system uses to communicate and compute. Particularly, we need to understand how the spike patterns are produced and received by neural circuits and how the neural circuits compute with spikes. Neuroscience has made tremendous progress on understanding what information spikes convey by correlating the recorded firing pattern of a single neuron or a population of neurons with various sensory and behavioral variables.

However, understanding the causal role of the neural code, i.e., how the neural code emerges and adapts to produce behavior, remains challenging. In order to understand how neural circuits produce and receive rapidly evolving spike patterns and how neural circuits compute with spikes to generate behavior, precisely timed closed-loop perturbation conditioned on specific patterns of spikes is required. This thesis aims to design and develop a real-time neural signal processor (NSP) to identify spikes in multi-electrode recordings along with a operational software to communicate with the NSP and to create a model for guiding the signal processing in the NSP hardware. This hardware-software co-designed system can (1) record high-channel-count extracellular data with tetrode arrays and silicon probes, and (2) detect and respond to a population of single-units at single-spike resolution in a low-latency manner.

Here is the agenda of the first chapter:

The first section (section 1.1) will give a brief review of the history of bioelectricity measurements of action potentials, followed by an introduction to the multi-electrode technology for recording multi-units and the spike sorting method for isolating single-units.

The second section (section 1.2) will introduce the neural coding problem - What do spikes and their patterns mean? Specifically, two general coding schemes, the population rate coding and population temporal coding, will be introduced. This section reviews the knowledge to justify the motivation of this thesis and its direct relevance to neuroscience.

The third section (section 1.3) will introduce and justify the aim of the thesis, to design and develop a tool to recognize and respond to a population of single-units at single-spike resolution in a low latency manner. This aim is motivated by its application for investigating several aspects of the neural coding problem. An analysis is provided on how this system fits into different experiment paradigms and under what conditions the technical specifications such as single-unit, single-spike, or low-latency are required. Thought experiments are designed to illustrate the analysis.

The fourth section (section 1.4) will introduce prior related works and highlight what is unique in this thesis, along with its limitations.

1.1 Bioelectricity measurements of neural activity

1.1.1 History

In 1791, Luigi Galvani published a series of pioneering experiments (Galvani, 1791). In one experiment, he wired the leg muscle of a freshly dead frog to a metal rod hanging on the roof and pointing to the sky. When lightning struck nearby, the dead frog leg contracted as if it came to life. To explain the discovery, Galvani dubbed it 'animal electricity': the animal uses the biologically generated electrical signal to move its body. Volta repeated Galvani's experiment, but he had an opposite interpretation: that it was the contact of two different metal pieces causing the electricity flow and the muscle twitch is an artifact (Piccolino, 1997). They were both right. The fact that nerves indeed use electricity took many decades to prove. Noteworthy, in their time, the physical nature of 'electricity' itself was far from clear. In 1799, Volta pushed his idea into engineering by inventing the Voltaic pile, the first battery, by stacking layers of zinc, copper and moist pasteboard together. Voltaic piles soon revolutionized the electrical lab and industry because it provides steady 'electricity'. The tool to precisely measure 'electricity' came after it. Around the 1820s, the galvanometer was invented to detect the current and was refined by Leopoldo Nobili in 1825 to detect a quite small current flow (Nobili, 1825). From 1821 to 1831, Michael Faraday used a Voltaic pile,

galvanometer and coils to conduct a series of groundbreaking experiments that dramatically deepened the understanding on 'electricity' (Faraday, 1832).

The Nobili galvanometer used by Faraday, however, is neither fast nor sensitive enough to capture the time course of small, rapidly changing 'bioelectricity' signals. This did not stop the earliest pioneer electrophysiologists and many variants of galvanometer were developed (Collura, 1993). Here I briefly mention two of them because of their direct involvement in landmark studies of neuroscience. The first came in 1868. Julius Bernstein, with the help of Emil du Bois-Reymond, made a device named a differential rheotome[1]. Julius Bernstein used the device he invented to measure the first described waveform of 'action potential' from the muscle nerve at 2000 Hz (Bernstein, 1868; Schuetze, 1983; Seyfarth, 2006). The second one came in 1873, when Gabriel Lippmann invented a new method to measure rapidly varying potential difference signals, later named a Lippmann electrometer (Lippmann, 1873). It uses the voltage under measurement to control the position of mercury[2] in a fine capillary, which provides the analog readout of the fast-changing voltage signal. In 1876, the Lippmann electrometer was used to record the first electrocardiography (ECG) signal (Marey, 1876), the bioelectrical signal generated by the heart. Later, it was improved and employed to be able to record individual action potentials in the nervous system (Lucas, 1912).

These early measurements of nerve bioelectricity are neither from a single muscle fibre or a single neuron, which require more advanced technology to pick up the extremely small voltage signals. Vacuum tube amplifiers, invented in 1906, set the stage. They can amplify feeble signals by thousands of times and simultaneously allow high input impedance such that the electrodes used to collect the signal can be made small. Several years later, Keith Lucas and his student Edgar Douglas Adrian, at Cambridge University, built an apparatus that integrates the vacuum tube and Lippmann electrometer to record fast-varying signals at the microvolt level (Adrian, 1926; Lucas, 1912). In 1926, Adrian successfully applied the apparatus for measuring the pattern of action potentials generated from the frog muscle nerve fibre, even during its movement[3], in response to various conditions (Adrian, 1926; Adrian and Zotterman, 1926a,b). Using this technology (Fig. 1.1 a), Adrian revealed a remarkable characteristics of the action potential: once generated, all action potentials have

[1] Julius Bernstein's solution was to build a mechanical sampling circuit, a wheel rotating at high speed. When pointing to the right angle, it allows the current flows into the galvanometer for sub-millisecond, thus sampling is possible.

[2] It was known by then that applying voltage can cause a mercury electrode in sulfuric acid to move. Thus with the calibration, one can turn the mercury position into the readout of the voltage.

[3] In Adrian and Zotterman 1926 paper, they write "Non-polarisable electrodes of the Ag, AgCl, NaCl, gelatine type are connected to the nerve by short lengths of moist carpet thread embedded in the gelatine. These allow some movement of the nerve to take place (and this is bound to happen when the muscle is extended) without producing changes of potential in the electrometer."

the same amplitude and duration regardless the stimulus intensity (Fig. 1.1 b), and regardless of whether it is generated from a sensory fibre or a motor fibre, and regardless of whether it is elicited by touch, light or motor stretch. The bioelectrical signal that causes the muscle to twist and those causing perceptual feelings are identical! It is the all-or-none action potential. The question is how the information is encoded into these all-or-none signals. Adrian summarized it as what we now know of as the problem of neural coding (see section 1.2).

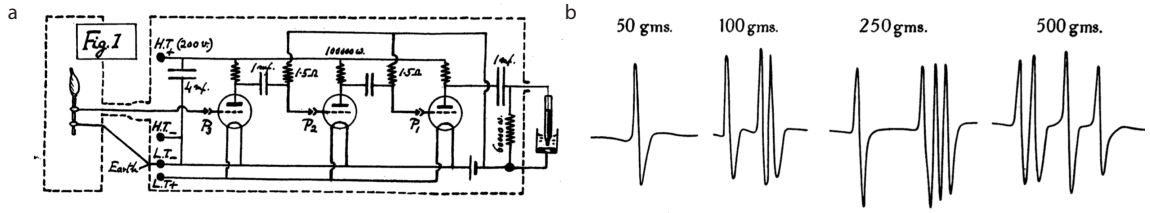


Fig. 1.1 **Adrian's instrument and recorded action potentials with different stimuli**: **a,** The apparatus involves several Vacuum tubes and a Lippmann electrometer. **b,** The size of individual action potentials, recorded from a single muscle fiber, does not change with the stimulus, meaning it is an all-or-none signal. However, the frequency of the action potentials changes with the intensity (weight) of the stimulus.

The next big advancement was recording the entire time course of the all-or-none action potential from a single neuron. This requires measuring the potential difference between the inside and the outside of the single cell plasma membrane. But the major difficulty was that the size of most of cells was too small to put the electrode inside. In this context, the giant axon of squids became an animal model of choice for the electrophysiologists because it is big enough to facilitate experimental access. In 1939, Hodgkin and Huxley managed to place a microelectrode[4] into a squid giant axon and recorded the first single-trial intracellular action potential with absolute amplitude (Hodgkin and Huxley, 1939). Later, they built a quantitative model to explain how an action potential is developed from step by step ion-specific conductance changes (Hodgkin and Huxley, 1952).

Coincidentally, soon after the Hodgkin and Huxley landmark discovery, starting from the 1950s, the world became populated with transistors[5]. The semiconductor industry quickly

[4]In Hodgkin and Huxley 1939 paper, they write "The micro-electrode consisted of a glass tube about 100 μ in diameter and 10-20 mm. in length; the end of the tube was filled with sea water, and electrical contact with this was made by a 20 μ silver wire which was coated with silver chloride at the tip."

[5]The invention of vacuum tube was essential for early breakthroughs in bioelectricity measurement of neural activity. But their successor was about to send a much bigger and everlasting shockwave to the world. In 1947 at Bell Laboratories, the transistor was successfully demonstrated to switch and amplify electronic signals. The credit goes to William Shockley, John Bardeen and Walter Brattain. The MOSFET (Metal Oxide Semiconductor Field Effect Transistor) came in 1959 and represented the core element for VLSI (large-scale-integrated-circuit), or often called chips. In the 1960s, vacuum tube was gradually and completely replaced by MOSFET.

took off, and Moore's Law kicked in. Neuroscientists started to buy instruments from Tektronix (founded in 1946), Texas Instruments (founded in 1951), National Instruments (founded in 1976) and other high-tech companies. Well-calibrated equipment is sold to every neuroscience lab. They produce the chips for signal amplification, sampling, acquisition and more. These chips keep getting smaller and lighter, which opened the possibility for neuroscientists to record while the animal was freely moving.

On this background, since the 1950s, much of the academic effort in measuring the action potential had been directed to engineering better micro-electrodes. The reasons are simple. First, neuroscientists wanted to record action potentials from single neurons when the animal is awake, behaving and even freely moving. This led to the development of metal micro-electrodes that has tip sizes of micrometres, with high stiffness, such that it can be invasively inserted into the targeted position in the brain and placed close enough to a single neuron without damaging the cell or being damaged (Hubel, 1957). Second, neuroscientists wanted to record the intracellular signal and the signal from ion-channels (Neher and Sakmann, 1976), which led to the development of the glass patch-pipette. The patch-pipette is mostly for intracellular recording and metal microelectrodes are mostly for extracellular recording.

Metal micro-electrodes all came in the same basic form: an insulated metal wire. To be specific, it is an exposed metal tip (a few to few tens of micrometers in diameter) with a long insulator-coated shank. Many different materials have been adopted to test their mechanical (durability, high Young's modulus, etc) and electrical (impedance, signal-to-noise ratio or SNR, etc) properties for in-vivo single-unit recording: lacquer coated tungsten electrodes (Hubel, 1957), lacquer coated stainless electrodes (Green, 1958), glass coated tungsten electrodes (Baldwin et al., 1965), formvar-coated nichrome electrodes (O'Keefe and Bouma, 1969), and many more. These microelectrodes, along with a microdrive to position them independently, allowed recording from single neurons in a precise location in the central nervous system of mammals for hours, thus sophisticated in-vivo experiments became possible.

Scientific break-throughs followed immediately. In 1959, David Hubel integrated his tungsten electrode with a customized micro-positioned implant and recorded "Single unit activity in striate cortex of unrestrained cats" (Hubel, 1959). Single unit in their title means a putative single neuron. In the same year, Hubel and Wiesel discovered that the neurons in cat primary cortex respond to specific visual features such as size and orientation (Hubel and Wiesel, 1959).

In the 1960s, John O'Keefe developed a method to record multiple single-units stably from freely-moving rat hippocampus CA1, during navigation using multiple independently

adjustable microelectrodes[6]. Furthermore, O'Keefe invented a novel approach to subtract the signal from an adjacent electrode to maximally reduce the motion-associated noise caused by freely moving rats[7]. These technology improvements contributed to the discovery, made by John O'Keefe and his student Johnathon Dostrovsky, of hippocampal place cells (O'Keefe, 1976; O'Keefe and Dostrovsky, 1971), which generate action potentials at specific locations during navigation (see more in section 1.2).

From the 1960s to now, neuroscience has witnessed a dramatic paradigm shift from single neuron recording to a population recording. By population recording, not only can neuroscientists monitor multiple single neurons simultaneously but also they can analyze the interaction of these neurons. The enabling bioelectricity measurement technology behind this, abbreviated as multi-electrode recording, is to implant distributed multiple micro-electrodes in a brain region. In the following section, I will introduce multi-electrode recording technology and the analysis methods for single-unit isolation.

1.1.2 Multi-electrode recording and single-unit isolation

To make bioelectricity measurements from multiple neurons simultaneously, one generally records in the extracellular medium where the voltage fluctuation caused by a spike (i.e., an extracellular spike) can be measured at some distance[8]. Although it is generally accepted that the measured voltage traces from extracellular electrodes reflect the transmembrane current (Buzsáki, 2004), the nature of the ionic and electrical properties of the extracellular medium is not completely understood. A direct method to ensure the waveform of extracellular spikes is a trustworthy read out of neuron activity is to relate it to the intracellular action potentials, a thoroughly studied ground truth signal. Early simultaneous intra- and extracellular recording performed by Buzsaki et al. (Buzsáki et al., 1996) and Henze et al. (Henze et al., 2000) provided several key insights about the nature of extracellular spike waveforms. First, the extracellular spikes detected near the soma indeed reflect the intracellular action potentials with one to one correspondence (Fig.1.2 a). Second, the filtered extracellular waveform (1

[6]In O'Keefe and Dostrovsky 1971 paper, they write "A small lightweight microdrive which carries up to 8 glass-insulated platinum-plated tungsten microelectrodes is permanently fixed to the rat's skull. Set screws on the manipulator allow any electrode or combination of electrodes to be moved independently of the others (O'Keefe and Dostrovsky, 1971)."

[7]In O'Keefe and Dostrovsky 1971 paper, they write "Maximum rejection of muscle and movement artefacts is obtained by feeding the signals from two adjacent microelectrodes into a high input impedance differential FET preamplifier mounted directly on the microdrive."

[8]The distance extracellular spikes can propagate might be dependent on many factors such as species, resistance of the extracellular medium, size of the cell, morphology of the cell, ion-channel distribution on the cell membrane. Empirically in rodents, it was suggested that the maximum detectable distance of an extracellular spike is less than 150 micrometers (Buzsáki, 2004).

Hz to 3 kHz) approximates the negative first derivative of the intracellular voltage (Fig.1.2 b) in the initial phase before the peak of the extracellular waveform. After the negative peak, the filtered extracellular voltage increases slower than the negative first derivative of the intracellular voltage. Third, the extracellular spikes recorded from multiple adjacent electrodes corresponding to an intracellular spike varies by distance, an important feature for separating the waveforms from multiple neurons (Fig.1.2 c). Moreover, it is known interneurons have narrower action potential width than pyramidal neurons in-vitro and in-vivo. The simultaneous intra- and extracellular recording suggests that the extracellular spike width can be used to classify pyramidal neuron and interneuron based on the shape of the waveforms. Notably, although applying multiple independently adjustable micro-electrodes (O'Keefe and Dostrovsky, 1971) was done long before the paired ground-truth recording, the ground-truth recording confirmed the previous notion that not only can we trust and use the extracellular waveform to read out spiking activities but also we can exploit the difference of the waveforms and cluster them into single neurons with basic cell type information. In addition, the ground-truth recording provides valuable insight for designing algorithms to separate the collective waveforms into single-units.

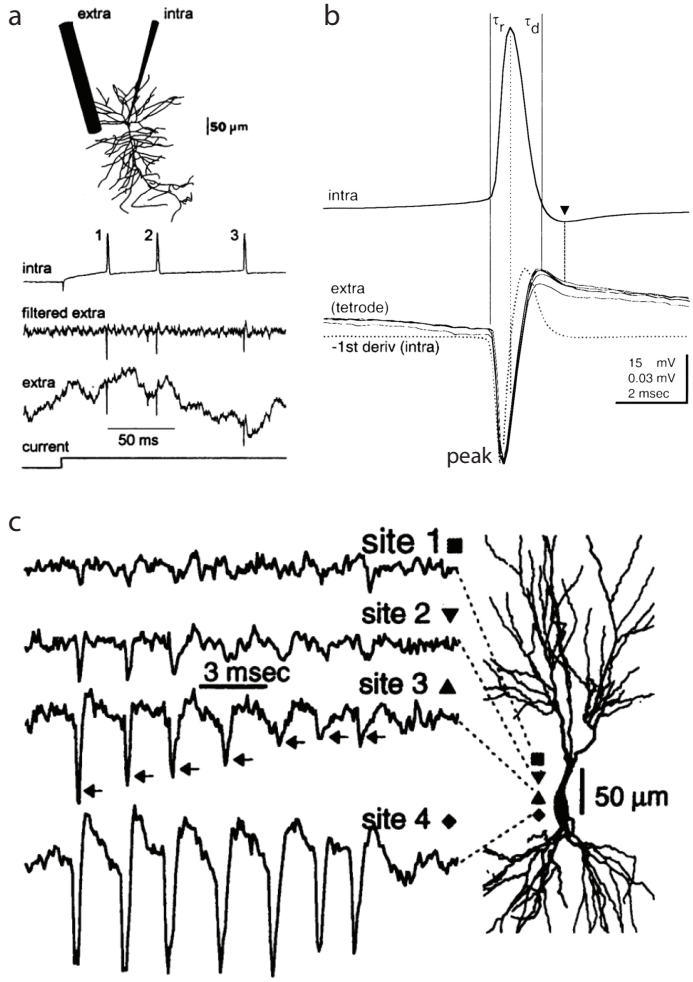


Fig. 1.2 **Extracellular spike waveform as a readout of the intracellular action potential**: **(a)** A paired intracellular and extracellular recording, from pyramidal neurons, shows one to one correspondence of a single spike for both intracellular and the extracellular waveforms. **(b)** The intracellular (upper) and the corresponding extracellular (lower) spike waveforms (multiple black traces). The extracellular waveform is identical to the negative first derivative (dotted line) of the intracellular waveform before the extracellular peak but then deviates after the peak. **(c)** The extracellular waveforms recorded from multiple electrodes along the dendrite-soma-axon axis. The burst firing causes a bigger decrease of the successive extracellular spike waveforms recorded from the electrodes near to the dendrite than from the electrodes near to soma. Figure is adapted from (Buzsáki et al., 1996; Henze et al., 2000).

Unless every single micro-electrode is independently and carefully positioned such that each electrode only monitors one single-unit, the recorded voltage traces from some channels would be more likely to contain extracellular spikes from multiple putative neurons, namely multi-units. A fundamental question is how to separate the multi-units into multiple single-units. Before answering that question, one should first ask the geometry of the micro-

electrodes. Specifically, how far are these electrodes positioned with respect to each other? There are two possible answers. One possibility is that the distance between electrodes is bigger than the distance extracellular spikes can propagate in the extracellular medium, such that one spike can only reach one electrode at most. Alternatively, if the inter-electrode distance among the adjacent electrodes is smaller than the distance that a spike can propagate, the same spike can contribute to signals on more than one electrode.

Both types of devices are called multi-electrodes, and both are suitable for recording multi-units. They differ in important ways when separating the single-units. The technique of isolating multiple single-unit spike trains from a multi-unit recording is spike sorting. Spike sorting separates single units by the shape of extracellular spike waveforms. The early development of spike sorting methods dates back to the 1960s (Gerstein and Clark, 1964; Keehn, 1966). Due to the multi-electrodes at that time having large inter-electrode distance, a single spike could not be captured by more than one electrode. Hence the spike sorting at that time was limited to single channels and had to be performed channel by channel. If several neurons are around a single channel with similar distance, then the spike waveforms from these neurons can be similar as well. Hence this channel-based spike sorting largely limited the accuracy of spike sorting and the yielded number of single-units.

In order to solve this problem, spike sorting with low inter-electrode distance micro-electrodes was first proposed by McNaughton et al. (McNaughton et al., 1983) and described in detail by O'Keefe and Recce (O'Keefe and Recce, 1993) using tetrodes (Reece and O'Keefe, 1989). With a group of low inter-electrode distance electrodes, each single spike can be captured by more than one electrode. A slight variation of spatial relationship between a neuron and an electrode-group can produce a quite different combination of waveforms. This feature remarkably increases the ability to distinguish single-units (Gray et al., 1995), which is validated by an analysis of the paired intra- and extracellular ground-truth data (Harris et al., 2000). Specifically, Harris et al. (2000) had shown that some single-units can only be seen using group-based spike sorting rather than channel-based spike sorting, because the extracellular waveforms from different neurons can have quite similar shape and amplitude on one channel, which was suggested in Fig.1 of O'Keefe and Recce (1993).

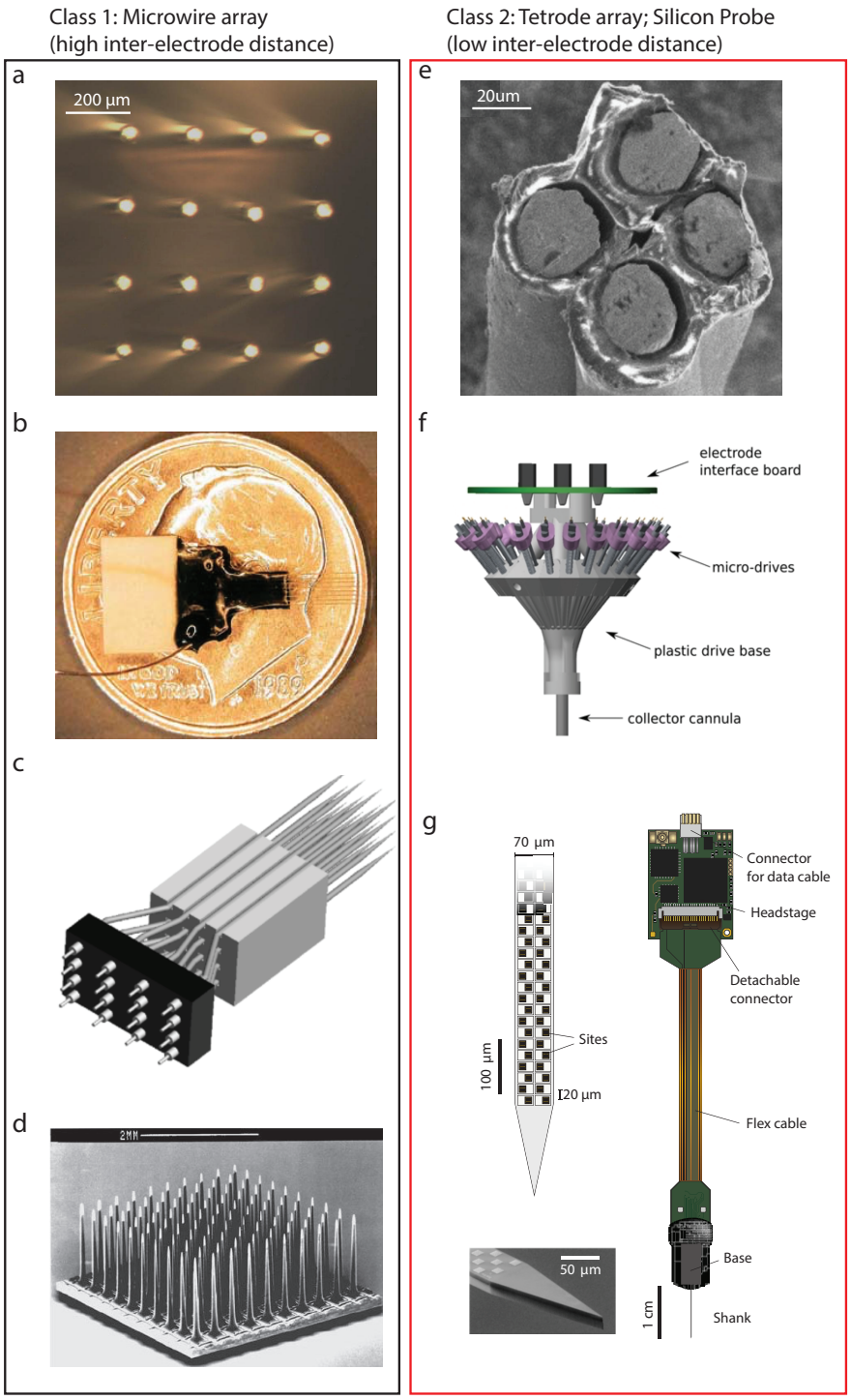


Fig. 1.3 **Two classes of micro-electrode devices**: **(a)** 4x4 Fixed microwire array from Innovative Neurophysiology. **(b)** 2x8 Fixed microwire array from Innovative Neurophysiology. **(c)** A manually assembled 4x4 tungsten microwire array. **(d)** A Utah electrode array (UEA), 100 electrodes in a 2.8 mm square grid (More UEAs use a 4.2 mm square substrate). **(e)** SEM scan of a tetrode. **(f)** Assembled micro-drive tetrode array containing 21 micro-drives that each drive a single tetrode. **(g)** Neuropixels probe with probe packaging, including flex cable and head-stage for bidirectional data transmission. 960 electrodes densely tiled along a 10-mm long, 70 $\times$ 24 µm cross-section shank.

In this thesis, I refer to the large inter-electrode distance among electrodes (> 150 µm) as the Class 1 multi-electrode device (Fig. 1.3 a,b,c,d), and refer to the small inter-electrode distance among a group or groups of adjacent electrodes (< 50 µm) as the Class 2 multi-electrode device (Fig. 1.3 e,f,g). Some examples of both types of devices, which are actively used in current research, are shown in Fig. 1.3. One of the most representative class 1 device is microwire array (Barna et al., 1981; Kim et al., 2006; Nordhausen et al., 1996; Takahashi et al., 2005), in which the distance between electrodes is usually hundreds of micrometers (Fig. 1.3 a,b,c,d). The class 2 devices come in various forms (Fig. 1.3 e,f,g). Notably, the number of electrodes in a group changes with the choice of the recording device, from a group of two electrodes in stereotrodes (McNaughton et al., 1983), a group of four electrodes in tetrodes (Reece and O'Keefe, 1989) and a group of four to eight or more electrodes in modern silicon probes (Buzsáki, 2004; Einevoll et al., 2012; Jun et al., 2017b). Notably, tetrode arrays and modern silicon probes (Fig. 1.3 f,g) contains many groups of adjacent electrodes to maximize the total number of single-units. With the advent of the Neuropixels probe (Jun et al., 2017b), a single shank silicon probe that has 960 sites densely tiled on a 10 mm long shank, a neuroscientist can record more than 400 single-units from cortex, hippocampus, thalamus, basal ganglia, etc. simultaneously within a single session from a single animal. Using multiple single shank Neuropixels probes, Steinmetz et al., published a single paper in which they recorded 30000 neurons in 42 brain regions from 10 mice over 39 sessions during a simple visual discrimination task (Steinmetz et al., 2018).

Although with clear and confirmed advantage on single-units accuracy and yield, the presence of tetrodes and modern silicon probes (class 2) does not put an end to the use of microwire array (class 1). The reason for that is in some cases, the single-units accuracy and yield might not be critical for the goal of the study. For example, the microwire array is still the major recording device used in the field of spike-based brain-machine interface (BMI) in recent years. Both the low-channel-count microwire array (Fig.1.3 a,b,c) and the high-channel-count Utah array (Fig.1.3 d) are used to record in both rodent and primate cortical regions, from which the real-time 'neural units' or 'units' are detected and used to control a computer cursor, robotic arm or other external actuators. In some BMI literature, the 'neural units' or 'units' specifically means threshold crossings on each electrode (Carmena et al., 2003; Christie et al., 2014; Fraser et al., 2009; Golub et al., 2018; Hochberg et al., 2012; Sadtler et al., 2014; Santhanam et al., 2006; Schwarz et al., 2014; Sussillo et al., 2016). For several reasons to be introduced in later sections, single-units might not be important for the performance of primate cortical BMI tasks. In other BMI literature where the adaptation of the single-unit's firing rate during learning is the research interest, the term 'units' denotes the isolated single-units by channel-based spike sorting (Athalye et al., 2018; Ganguly et al.,

2011; Gulati et al., 2017, 2014; Koralek et al., 2012; Neely et al., 2018). When single-units are required, the class-2 device could have been a better solution because the single-unit accuracy and yield matters.

Nonetheless, over decades, the BMI closed-loop signal processing systems had been built around microwire arrays (Donoghue, 2008; Lebedev and Nicolelis, 2017; Schwartz, 2004; Schwarz et al., 2014). This renders the real-time signal processing for the class-2 device as a technological gap to be filled by this thesis (and other motivational reasons will be analyzed in later sections).

A straightforward method to separate the single-units from multi-unit recording with class-2 devices is group-based spike sorting, illustrated in Fig.1.4. The first step of spike sorting is to preprocess the raw voltage trace, which includes bandpass filtering and denoising. The preprocessing is conducted channel by channel (Fig. 1.4 $b \rightarrow c$, $\rightarrow$ means after processing, b becomes c). Following the preprocessing is threshold-based spike detection (Quiroga et al., 2004). Spike detection is also conducted channel by channel to identify the relevant temporal segments that contain extracellular spikes (Fig. 1.4 $c \rightarrow d$). Once the relevant temporal segments around the peak of the extracellular spike waveforms are extracted, an alignment algorithm is required to ensure that the negative peak of the stacked waveforms are aligned along the time-axis (Fig. 1.4 $d \rightarrow e$). Upon finishing the spike alignment, the stacked spike waveforms are transformed into a feature space where each point in the feature space represents a group of waveforms (Fig. 1.4 $e \rightarrow f$: in this example there are four channels in a group hence each point in Fig. 1.4 e represents four waveforms from four channels). Feature transformation reduces the dimensionality of the spike waveforms for both further processing and visualization. The features can be computed either using the peak-to-peak amplitude (O'Keefe and Recce, 1993) or more commonly using principal component analysis (PCA) (Harris et al., 2000; Kadir et al., 2014; Lewicki, 1998), or sometimes using wavelets (Quiroga et al., 2004). The distribution of the feature can be multivariate Gaussian (Lewicki, 1998) or non-Gaussian (Fee et al., 1996; Shoham et al., 2003) due to various kinds of distortion on the spike waveforms. The distortion can be produced by the extent of the variation in the extracellular waveforms (as shown in Fig. 1.2 c), the overlapping spikes of near-synchronous firing (Pillow et al., 2013) and possible drift of relative position between the electrodes and the source neurons (Einevoll et al., 2012; Rey et al., 2015). Each dataset can have different sources for the specific non-Gaussian deviation in the features. These deviatons pose a major challenge to the next step of spike sorting, to cluster the data points in feature space (Fig. 1.4 $f \rightarrow g$). Clustering was commonly used as the last step of automatic spike sorting procedures. However, it was also proposed that a model can be computed from the clustering (Einevoll et al., 2012), and a classifier can be built based on this model for spike classification.

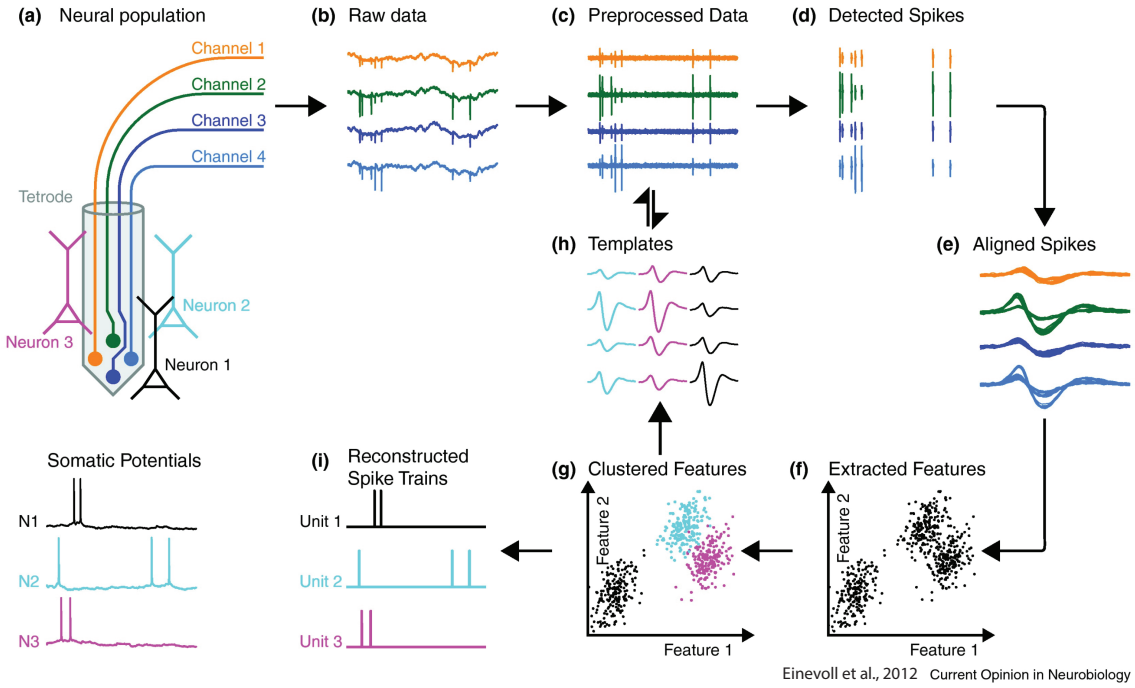


Fig. 1.4 **Overview of spike-sorting process**: From Einevoll et al. (2012):"A group of microelectrodes record changes in the extracellular electrical potential (b) caused by action potentials of neurons in its vicinity (a) Intracellular potentials of three spiking example neurons (N1, N2, N3) are depicted in lower left panel. For the detection and analysis, a bandpass filter is usually used to remove the low-frequency part of the potential (c). The optimal procedure for detection of spikes, especially in multielectrode recordings, is still is an unsolved problem but simple voltage thresholding is commonly used (d). For most spike-sorting procedures the extracted spike waveforms need to be temporally aligned on a common feature like the position of the voltage peak (e) before features are extracted from every waveform (f). The feature extraction is crucial for decreasing the dimensionality of the data to the most informative dimensions. This can be done by, for example, using principal component analysis. Clustering finds the number of clusters and their position in the feature space (g). An individual cluster should contain all spikes of a putative neuron. The average waveform of all spikes belonging to one cluster is called the 'template' of that neuron (h). The outcome of the clustering is often a model of the data (e.g., number of neurons, templates, covariance matrices) that can be used for quality estimation of the sorting result and derivation of a classifier for yet unclassified spikes (e.g., template matching)." Figure is from (Einevoll et al., 2012).

Of note, Fig.1.4 only illustrates one group of electrodes while typically there are many groups of electrodes in tetrode arrays or silicon probes (four electrodes in a group in tetrodes, and arbitrary number of electrodes in a group in silicon probes); therefore, one would perform this process group by group (Einevoll et al., 2012). Dividing the overall problem into subproblems and process one after another is referred to as 'divide-and-conquer' in

engineering parlance. Here, the overall problem is to spike sort the whole data set and subproblem is to spike sort the data from one group of electrodes. This is a strategy that was proposed in (Einevoll et al., 2012) and was re-proposed recently in greater detail (Diggelmann et al., 2018) for high-channel-count class 2 devices. The group-based spike sorting offers computational efficiency for two reasons. First, it limits the complexity of the processing of each group since each group contains only a few electrodes. Second, the processing for each group is identical, and the total complexity is linear in the number of groups. Using multi-CPUs, computation for many groups can be parallelized where each group is passed to an identical basic processing unit. The group-based computing can be particularly efficient for specialized integrated circuits (IC) for the same reason. In the IC design, it is important to have a basic processing unit, of low complexity, that can be used repetitively. However, a drawback of group-based spike sorting is that the grouping of electrodes can be ambiguous for silicon probes where there is no clear boundary between electrode groups, and some spikes can be detected by more than one group (Einevoll et al., 2012). The recently developed offline spike sorting packages often treat a whole shank or all electrodes as a single group. To deal with the overlapping spike issue, the global reconstruction error optimization method that employs template matching was proposed and implemented (Ekanadham et al., 2014; Pachitariu et al., 2016). The templates generated from these algorithms are the spike waveforms produced from putative neurons, and the spike waveforms here includes all the channels rather than a local group (the template waveform can be zero over most of the channels). The concept of group is no longer important for spike sorting procedures that process all electrodes altogether. Using fast GPUs, all the recently developed spike sorting packages achieve automatic spike sorting in less than the recording time (Chung et al., 2017; Jun et al., 2017a; Lee et al., 2017; Pachitariu et al., 2016; Yger et al., 2018). Despite the considerable progress that has been made to accelerate the speed of automatic spike sorting on high-channel-count silicon probe data, this process still requires intensive visual inspection and manual curation (Rossant et al., 2016). More detailed analysis of the challenges and approach to spike sorting that is relevant to this thesis will be introduced in Chapter 2.

1.2 Neural coding

With the recording and analysis technology described in the previous section, one can now record from hundreds of neurons simultaneously. But what is the ‘meaning’ of these spike trains? What ‘message’ do they transmit? This is the problem of neural coding (Perkel and Bullock, 1968; Rieke et al., 1999).

The ‘meaning’ can be hard to measure. Claude Shannon (Shannon, 1948) quantified the ‘meaning’ of ‘messages’ using ‘information’ as bits/sec and employed an encoding/decoding framework to build a mathematical model for general communication systems, in which the semantic ‘meaning’ of a specific ‘message’ (i.e., a binary code) is irrelevant. Today, the word ‘code’ is quite popular in many different subjects. It is particularly appealing to neuroscience because the brain is an ultimate communication system in which most of the neurons use spikes to communicate to each other, meaning binary spikes are the basic elements of the ‘codes’ used by nervous system. The analogy between a communication system and the nervous system was made in the early history of neuroscience. It was first proposed by Fred Attneave (Attneave, 1954) that the job of ‘the perceptual machinery is to encode incoming information in a form more economical than that in which it impinges on the receptors’. Later Horace Barlow expanded this idea and formulated it into the efficient coding hypothesis (Barlow, 1961). In parallel, David Marr (Marr, 1969) used the word ‘codon’, in a more general sense, to describe the possible presynaptic spike patterns that cerebellar neurons respond to and related it to the space of ‘features’ or ‘representations’ that neurons use to store and carry information. Meanwhile, Perkel and Bullock (Perkel and Bullock, 1968) summarized several important aspects of the neural coding problem. They pointed out that spikes serve as more than communication because though the spikes carry information, they also adapt during learning and carry out computations to generate behavior. Therefore, codes in nervous systems (i.e., neural codes) should not only be measured as ‘bits/sec’ as in Shannon’s communication systems, as the specific functions of neural codes to learning, memory and behavior are important. Although tremendous progress has been made in neuroscience in the last few decades, many aspects of the neural coding problem remain unsolved (Panzeri et al., 2017). Particularly, how to search for the behaviourally relevant neural codes using modern neural recording and perturbation technology, and understanding the causal relationship between hypothesized neural codes and behavior, are still pressing issues under intensive discussion and debate (Jazayeri and Afraz, 2017).

The overarching goal of this thesis is the development of a system with single-unit and single spike resolution to facilitate exploring many aspects of the neural code and its role in behavior, rather than solving a specific neuroscience question. In this section, I will give a broad and concise review of the neural coding problem from its early history to the current understanding, serving as the foundation for introducing the motivation (tool development) of this thesis and for analyzing the objectives and constraints of tool development. Some of the points to be introduced in this section were already introduced in previous sections because some great discoveries were enabled by the advancement of the relevant technology. The following discussions are centered on ‘what do spikes mean’.

1.2.1 Feature selectivity and the population rate code

Spikes carry information about sensory stimuli. Edgar Adrian (Adrian, 1926, 1928; Adrian and Zotterman, 1926a) laid down the earliest fundamental work related to the neural code. He discovered that the firing rate of individual stretch-receptor neurons increases proportionally to the weight applied to the frog muscle fibre (Fig. 1.1). Hartline (Hartline, 1938, 1940) found that a frog optic nerve fibre could be excited by casting light on a small area of the retina. He called this area the receptive field, using a term first introduced by Sherrington (Sherrington, 1906) in the tactile domain. Barlow, Adrian's student, developed the concept of feature selectivity by recording the frog retinal ganglion cells (Barlow, 1953). Feature selectivity was later strongly reinforced by the observations of Hubel and Wiesel (Hubel and Wiesel, 1959, 1962). They found the firing rate of single cells in cat visual cortex is tuned to the orientation of a bar in the cell's receptive field.

Spikes also carry information about motor parameters. Edward Evarts pioneered the study of tuning property of single primary motor cortical (M1) neurons in awake monkey during motor tasks (Evarts, 1966, 1968a,b). He showed that an M1 neuron's firing rate increases when the monkey pulled a lever and decreased when pushed. Using the same single-unit recording technique, Apostolos Georgopoulos et al., (Georgopoulos et al., 1982) made a milestone discovery that primate M1 neurons exhibit broad tuning to the direction of arm movement.

Spikes even carry information related to cognition. John O'Keefe and Jonathan Dostrovsky made a ground-breaking discovery of a cell type named hippocampal place cells (O'Keefe, 1976; O'Keefe and Dostrovsky, 1971). A place cell increases firing rate when rats navigate through a specific location in the environment. Many of the place cells fire when the animal traverses through a location, regardless of the sensory input and motor output, showing that it is the cognitive location that is encoded in spikes generated by place cells. The receptive field of a place cell is called a place field. Later it was proposed that place cells are used to form a cognitive map (Tolman, 1948), which encodes the relationship between places/items to support latent learning and flexible navigation (O'Keefe and Nadel, 1978). More navigationally selective cell types were found in other brain regions. Head direction cells (Ranck, 1984), in presubiculum, are tuned to the animal's head direction. Grid cells (Hafting et al., 2005), in the entorhinal cortex, are tuned to a grid pattern in the two-dimensional environment. In addition, it is well known that the hippocampal-entorhinal system is required for memory-based navigation. For example, the discovery of the place cell was followed by a lesion study showing a place learning but not cue learning deficit by removing the fornix, major afferent and efferent pathway of the hippocampus (O'Keefe et al.,

1975). Later, Morris et al., provided further evidence that hippocampal lesions impair the allocentric[9] and memory-guided navigation (Morris et al., 1982).

The early evidence of the feature selectivity at the single neuron level raised the question of whether the precise information can be decoded from the activities of neuron ensembles. The idea of the linear decoder using population activities of neurons was first proposed by Humphrey et al. in 1970. The authors successfully confirmed that various motor parameters could be decoded from the firing rates of motor neurons using linear regression (Humphrey et al., 1970). Later, Georgopoulos et al., constructed the "population vectors"[10], the weighted averages of single neurons' firing rate, that can predict the direction in which a monkey moves its arm (Georgopoulos, 1994; Georgopoulos et al., 1986). It was later found that both speed and direction can also be decoded from these population vectors (Schwartz, 1993, 1994; Schwartz and Moran, 1999). As multi-unit recording technology became popular in the 1990s (Nicolelis et al., 1993; Reece and O'Keefe, 1989; Wilson and McNaughton, 1993), the population decoding of behavioral variables using multiple simultaneously recorded neurons has been demonstrated in many different regions beyond primate motor cortex. In the sensory cortex, Hung et al. successfully decoded an object's identity among 77 objects using a small population of neurons in monkey inferior temporal (IT) cortex (Hung et al., 2005). In the hippocampus, population decoding was employed to reconstruct the spatial location in a maze of a rat during navigation, which is a cognitive percept in the brain. This was achieved by decoding using the firing rate of ensembles of neurons in the hippocampus recorded with a tetrode array (Wilson and McNaughton, 1993). Later, the Bayesian decoding method was introduced to accurately predict the rat's position in various shapes of mazes (Zhang et al., 1998a). During navigation, the decoded spatial distribution of possible 'mental positions', from spikes fired by several place cells, can be much smaller than the place field of a single place cell; thus the decoded position can precisely describe where an animal is. The Bayesian decoding remains the most commonly decoding algorithm used today in hippocampus literature.

Being able to decode specific information from a population of neurons consolidated Hebb's view of cell assembly organization (Hebb, 1949) as a fundamental principle of neural coding[11]. This hypothesis elicits at least three basic questions: how did neural assemblies

[9] Allocentric describes a process using object-to-object relationship whereas egocentric denotes the self-to-object relationship. Allocentric navigation depends on the memory of object-to-object relationships, which requires a cognitive map (O'Keefe and Nadel, 1978).

[10] Note the early development of the population vectors decoding method still employed the single-electrode neurophysiology, and the 'population vectors' were artificially constructed from single neurons recorded at different days.

[11] According to Hebb, an assembly is "a group of neurons that are repeatedly active at the same time and develop as a single functional unit, which may become active when any of its constituent neurons is stimulated".

adapt and change during learning, does the brain actually use these cell assemblies for behavior, and can the brain volitionally access and control the assemblies. The first and second questions are still far from being solved, while the last was partially validated by BMI studies (Fetz, 2007; Nicolelis and Lebedev, 2009). Initiated by the classic single neuron operant conditioning (Fetz, 1969; Olds, 1965), and built upon the motor decoding algorithm developed by Georgopoulos and colleagues (Ashe and Georgopoulos, 1994; Georgopoulos, 1994; Georgopoulos et al., 1982, 1986), BMI studies set out to test whether the animal can 'volitionally control' (Fetz, 2007) external devices using the population firing rate of motor neurons. The test was successful. Pioneering work demonstrated that rodent, primate, and human can indeed move a cursor on the screen or a robotic arm in a closed-loop manner by controlling the firing rate of a set of motor neurons (Carmena et al., 2003; Chapin et al., 1999; Hochberg et al., 2006; Santhanam et al., 2006; Serruya et al., 2002; Taylor et al., 2002; Velliste et al., 2008; Wessberg et al., 2000). Several interesting observations emerged from these early BMI experiments. First, it was surprising to Chapin et al. that the animals stopped moving in a few trials, and yet still successfully used their brain activity alone ('brain control mode') to move the external lever to get their water reward (Chapin et al., 1999). Second, the analysis showed that the directional tuning curves of single-units in M1 changed during the brain control mode (Carmena et al., 2003; Taylor et al., 2002), indicating learning updates the neural code. In addition, Carmena et al. found that the animal can learn to control the firing rate of the electrode (multi-units) as well as single-units in a BMI task (Carmena et al., 2003). Exploiting the discovery that the animal can learn to control multi-units in motor cortex[12], later BMI studies started to use the 'neural units' for building the decoder, where 'neural units' means threshold crossings of each electrode (i.e., multi-units) (Carmena et al., 2003; Christie et al., 2014; Fraser et al., 2009; Golub et al., 2018; Hochberg et al., 2012; Sadtler et al., 2014; Santhanam et al., 2006; Schwarz et al., 2014; Sussillo et al., 2016). The position of the electrodes is random in these experiments, meaning animals can learn to control the external device using randomly selected 'neural units'. This suggests that the brain can learn to reorganize neural assemblies during learning and access these assemblies for real-time BMI control during execution. Later it was shown by Koralek et al., that motor cortical BMI learning is striatal NMDA receptor dependent. Blocking NMDA receptors impairs the ability to learn the BMI task but not the ability to execute the learned BMI task (Koralek et al., 2012). Moreover, 'during learning, strong relations between the activity of neurons in motor cortex and the striatum emerged', indicating a similar neural mechanism as in natural motor learning (Green and Kalaska, 2011; Koralek et al., 2013, 2012). By

[12]But also see (Georgopoulos et al., 2007; Hatsopoulos, 2010). Multi-unit BMI learning might be more feasible in the primate cortex than other brain regions or other species, because the primate motor cortex has a columnar structure where nearby neurons tend to have similar feature selectivity.

artificially defining the neural code to control the external device, BMI experiments offers a unique opportunity to study the learning process that causally leads to the activation of predefined neural assemblies, which in turn directly causes the behavior.

In all of the BMI literature, to my knowledge, whenever the population decoder is employed, the population rate code was presumed by default. The rate code uses time averaging of the spike count to reduce 'noise' (the variability in spike trains). Spike rate is the orthodox coding scheme in neuroscience (Shadlen and Newsome, 1998). However, arguably, the limitation of the rate code resides in its slowness (Rieke et al., 1999; Thorpe et al., 2001; VanRullen et al., 2005). The rate code is slow because the information is stored in the first order statistics of the spike trains, the rate. A high firing rate neuron (100Hz) can only fire one spike in 10 milliseconds, and for this spike to propagate to its downstream region it will take a few to a few tens of milliseconds more depending on the specific brain regions. It will take another few tens of milliseconds for the downstream layer to estimate the firing rate of this neuron. If every layer of neurons has to estimate the firing rate from its upstream input, it is hard to account for many behaviours that happen within just few to few tens of milliseconds (Carr, 1993; Gerstner et al., 2014; Rieke et al., 1999) (see chapter 7.6 in Gerstner et al. (2014)). This naturally opens the possibility that the brain could also use other, faster coding schemes to exchange information among populations of neurons and give rise to behavior.

1.2.2 Relative spike timing and the population temporal code

One of the long-standing mysteries in neuroscience is whether milliseconds precise spike timing carries additional information that contributes to neural computation and behavior (Abeles, 1982, 1991; von der Malsburg, 1981, 1999). This is a topic of intensive debate and is still not fully resolved (Brette, 2015; Ferster and Spruston, 1995; Fetz, 1997; Gerstner et al., 2014; Hopfield, 1995; Jacobs et al., 2009; Kayser et al., 2009; König et al., 1996; London et al., 2010; Panzeri et al., 2017; Shadlen and Movshon, 1999; Shadlen and Newsome, 1995, 1998; Singer, 1999; Singer and Gray, 1995; Softky, 1995; Stevens and Zador, 1995; Thorpe, 1990; VanRullen et al., 2005).

Christoph von der Malsburg first proposed "The Correlation Theory of Brain Function", in which he extended the flexibility of Hebb's assembly (von der Malsburg, 1981). In his theory, the activity of neurons can transiently synchronize together, in a few milliseconds, to create dynamic cell assemblies, meaning an individual neuron is free to join another assembly in the next moment to construct a different code. The flexible transient partnership among neurons creates a vast coding space that operates rapidly via short-term plasticity (also see the interpretation of (Shadlen and Movshon, 1999)). Moshe Abeles formulated the idea of

coincidence detection and highlighted the propagation aspect of synchronous activity. He proposed that synchronous firing could facilitate information propagation in a multi-layered feed-forward network efficiently, and named the model a 'synfire chain' (Abeles, 1982, 1991). Precise spike timing is essential for this type of hypothesis[13] and has a quite different nature from the instantaneous firing rate. Two patterns of spiking activities can be defined as a temporal code (Theunissen and Miller, 1995): first, spike trains that store information in its special spiking pattern, such as bursts[14], that is not captured by the mean spike counts (rate); second, a population of spike trains that stores information in the relative timing of a set of neurons. The latter draws a clear boundary between the temporal and the rate coding scheme. The temporal code emphasizes the relative and dynamic relationship of spikes from single-units. It is an immediately appealing idea given the definition. First, it transmits information instantaneously hence is much faster than the rate code, significantly decreasing the communication latency of the nervous system (Gerstner et al., 1996; Thorpe, 1990). Second, the number of possible spike patterns across ensembles of neurons is considerably higher than the number of possible rate codes in a given time window. Thereby it substantially expands the bandwidth available in the nervous system (Izhikevich, 2006; Thorpe, 1990; von der Malsburg, 1981).

Nonetheless, the possible advantages do not mean the brain uses the temporal code. When considering this idea, an immediate question arises, is the temporal code physiologically possible? The evidence at the cellular level supports it. First, it has been known since the 1960s that neurons respond differentially to the same rate patterns that have different temporal patterns of spikes (Segundo et al., 1963). Second, pyramidal neurons are sensitive to the relative timing of their synaptic inputs (Branco et al., 2010), and synchronized synaptic inputs (in a few milliseconds) can cause dendritic spikes (Stuart and Spruston, 2015), that produce both non-linear dendritic integration and plasticity (Mel, 1999). Third, a brain slice study showed that the neuron could generate precisely timed spikes with submillisecond variability when receiving the fluctuating intracellular (Mainen and Sejnowski, 1995) or extracellular (Pouille and Scanziani, 2001) stimuli. This suggests the variability of spike trains observed in-vivo (Shadlen and Newsome, 1995, 1998) need not arise from noise at the cellular level (but see (London et al., 2010)). Importantly, the temporally structured activity could facilitate the propagation of information (Diesmann et al., 1999; Salinas and Sejnowski, 2001). Moreover, another critical fact in physiology is that the latency between cortical neuron pairs is broadly

[13] see also (Izhikevich, 2006) that describes computation with spikes using precise timing but not synchrony as proposed in synfire chain model.

[14] An example type of burst is a 'complex spike', which is a stereotyped single-unit firing pattern that contains two to six action potentials with short interspike intervals of a few milliseconds each. In general, bursts are ubiquitous in various brain regions. Given its special role in synaptic plasticity and information processing, the burst itself has been suggested as a unit of the neural code (Lisman, 1997).

distributed yet is reproducible with sub-millisecond precision (Swadlow, 1985, 1988, 1992). As Izhikevich poses, "why would the brain maintain different delays with such precision if spike-timing were not important?" (Izhikevich, 2006). Finally, various well studied forms of plasticity are dependent on the timing of spiking (Suvrathan, 2019). For example, spike-timing-dependent plasticity (STDP) can be introduced by repetitive and precisely timed firings of pre and postsynaptic neurons (Bi and Poo, 1998; Dan and Poo, 2004; Froemke and Dan, 2002; Magee and Johnston, 1997; Markram et al., 1997; Sjöström et al., 2001; Zhang et al., 1998b); Long term potentiation (LTP) (Bliss and Lømo, 1973) can be readily induced if a priming burst, arriving at the synapses of the CA1 neuron, is followed by another burst within between 100 to 200 ms (Davies et al., 1991; Larson and Munkácsy, 2015; Larson et al., 1986); LTP can also be induced by simply stimulating two pathways to a target region one after another within a few tens of milliseconds delay, known as input-timing-dependent plasticity (ITDP) (Basu et al., 2013; Cho et al., 2012; Dudman et al., 2007; Leroy et al., 2017). Although spike timing is not everything for plasticity (Spruston and Cang, 2010), it is often necessary and efficient for introducing the plasticity. The physiological facts listed here suggest that the nervous system could have evolved to allow neural circuits to generate highly patterned population of spikes (i.e., temporal codes) to exploit molecular mechanisms for learning and behavior.

To quantify a relative spike timing code, a reference signal for timing is required. It can be a reference neuron or be a reference assembly, or more commonly be the extracellular oscillation (Sejnowski and Paulsen, 2006). The low frequency (<300 Hz) of the extracellular oscillation, known as the local field potential (LFP), can be further decomposed into many functional sub-frequency bands (Buzsáki, 2015; Buzsáki and Wang, 2012; Colgin, 2016; deCharms and Zador, 2000; Panzeri et al., 2014; VanRullen et al., 2005). Although the biophysical mechanism of generating the LFP is not fully understood, it largely reflects "overall synaptic activities and excitation-inhibition interplay in a large local volume"(Buzsáki et al., 2012; Einevoll et al., 2013), thereby naturally providing a common reference signal for the population spike timing (Panzeri et al., 2014).

1.2.3 Temporal codes in the hippocampus and hippocampal-cortical communication

Despite the many forms that a temporal code can take, among the most reliable and robust temporal codes discovered so far is the hippocampus theta phase code (O'Keefe and Recce, 1993). The phase code was discovered when O'Keefe and Recce used tetrodes to record many single place cells in the rat hippocampus and examined the relationship between spike

times of individual place cells and the phase of the theta (4-12Hz) oscillation. O'Keefe and Recce did not just check the overall preferred spike-phase using the spike-triggered average LFP as in earlier studies (Buzsáki et al., 1983), but how the spike-phase relationship changed from one theta cycle to another. What they saw was that the place cell fires at progressively earlier phase of theta, a phenomenon named phase precession (O'Keefe and Recce, 1993). Since place cells often fires bursts when the rat traverses a place field, phase precession was related to the fact that "individual cells had a higher inter-burst frequency than the theta frequency" (O'Keefe and Recce, 1993). The correlation between the spike-phase and position is much higher than the correlation between spike-phase and the spike-rate (Huxter et al., 2003) (Fig. 1.5 a), suggesting independent coding of phase and rate. Compared to use of the rate code alone, the position decoding can be 40 percent more accurate if the spike-phase is included (Jensen and Lisman, 2000). Importantly, when adopting the population perspective, the phase code allows the partially overlapping place cells to form a compressed sequence to encode past, current, and future locations within a single theta cycle. This is termed a theta sequence, and its compression ratio can go up to 10:1 in an individual cycle (i.e., ten cells with partially overlapping place fields sequentially fire in one $\sim$125 milliseconds theta cycle) (Foster and Wilson, 2007; Skaggs et al., 1996). Consequently, the temporal pattern of hippocampal theta phase coding allows for at least two types of plasticity. First, the time interval between adjacent spikes within a compressed sequence is around 10 ms, which is a time scale that permits repeatedly induced STDP (Dan and Poo, 2004). Second, the bursts interval of the place cells with adjacent and partially overlapping place field is approximately one theta cycle, which permits rapidly induced theta-burst LTP (Larson and Munkácsy, 2015). The STDP potentially explains the facts that both correlation between phase and position (Mehta et al., 2002) and theta sequences (Feng et al., 2015) become more robust with experience. Moreover, decoding population spikes from theta sequences clearly shows that the 'mental position' of the animal sweeps from behind to ahead relative to its current position during locomotion, and each sweep is organized by an individual theta cycle (see Feng et al. (2015):Fig 1a). This suggests that during navigation, the individual theta cycle is a unit of the neural code that, with the population sequences it contains, reflects the temporal relationship among assembly-represented places/objects from the past to the future. Indeed, it was demonstrated that when rats experienced backwards travel on a train, the theta sequences reverse their order to maintain the past-current-future relationship (Cei et al., 2014). Although the causal role in behavior is still unclear, phase precession and the resulting theta sequences may provide animals with information about what is just past and what is to come. Meanwhile, phase precession and theta sequences may also serve as a backbone for encoding and retrieval of episodic memory where the temporal link between

events is critical (Hasselmo et al., 2002; Jaramillo and Kempter, 2017; Lisman, 2005; Lisman and Jensen, 2013; Tulving, 2002).

If theta sequences are truly related to the episodic memory, it is important to check whether the encoded temporal structure is preserved after the experience, particularly during sleep. Skaggs and McNaughton first reported that the temporal order of pairs of cells is preserved during slow-wave-sleep (SWS) after awake experience (Skaggs and McNaughton, 1996)[15]. Later, Lee and Wilson provided clear evidence that goes beyond pairs of cells. They found the reactivation of sequences of a large number of place cells during slow-wave-sleep, reflecting the awake behavioral trajectories with the same order but at a much faster speed (20-fold) (Lee and Wilson, 2002). The observed hippocampal replay events during sleep are associated with the sharp-wave-ripple (SWR) events, another well characterized LFP pattern that contains a large polarity deflections and a fast oscillatory component (110-250 Hz) (Buzsáki et al., 1983; O'Keefe, 1976; O'Keefe and Nadel, 1978). The finding that sleep replay hippocampal sequences can encode awake experiences in a highly compressed manner resonates with what had been postulated before, that the hippocampal SWR events are involved in memory consolidation (Buzsáki, 1989; McClelland et al., 1995). The reactivation of sequential firing was later reported during awake behavior, where both reverse and forward replays are observed when the animal remains stationary (Diba and Buzsáki, 2007; Foster and Wilson, 2006). When the animal runs from one end of a linear track to the other end, the forward replay is more likely to occur at the beginning of the running and the reverse replay is more likely to occur when the animal reaches the other end and stops (Diba and Buzsáki, 2007) (Fig. 1.5 b). Like sleep replay, the awake replays are also associated with the SWR and a surge of population firing activity. What is the possible function of the awake replay event? Accumulated evidence indicates that, during pauses in exploration, the SWR replay sequences participate in more than memory consolidation (Davidson et al., 2009; Gupta et al., 2010; Karlsson and Frank, 2009). Pfeiffer and Foster showed that in 2D navigation, the awake forward replay can predict the immediate future trajectory of the animal (Pfeiffer and Foster, 2013). Later, it was further demonstrated that awake reverse replay, but not forward replay is modulated by the amount of reward (Ambrose et al., 2016). These studies suggest that the reverse replay is specifically involved in reinforcement learning, and forward replay is specifically involved in planning.

[15] In the paper, the authors speculated that "this memory for temporal order of neuronal firing could be produced by an interaction between the temporal integration properties of long-term potentiation and the phase shifting of spike activity with respect to the hippocampal theta rhythm."

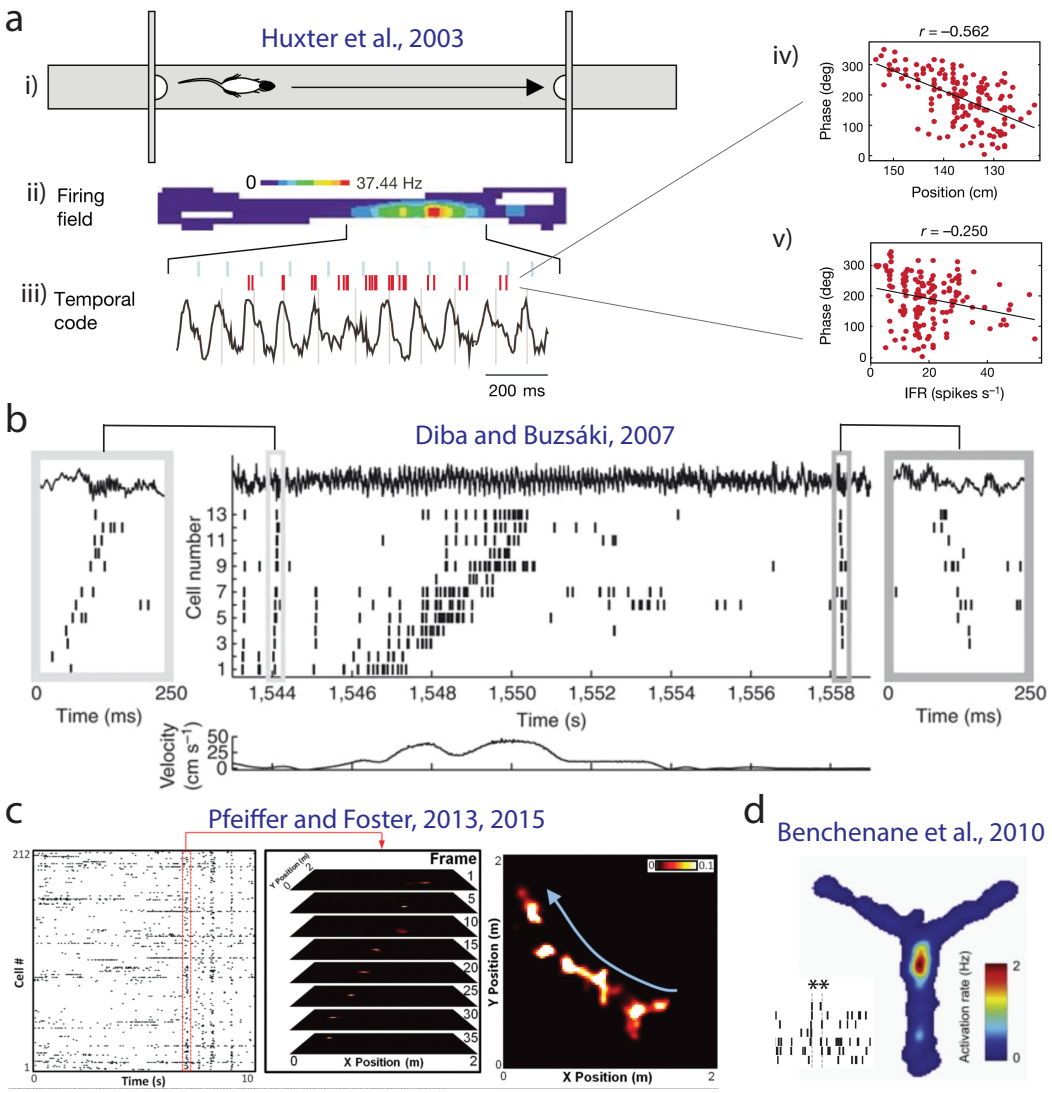


Fig. 1.5 **Hippocampal temporal code**:**(a)** from (Huxter et al., 2003): (i) animal runs from one end of a linear track to the other end. (ii) firing field of a place cell. (iii) theta rhythm and single-trial spike timing (in red) of the place cell. (iv) position-phase relationship from multiple traversals. (v) rate-phase relationship from multiple traversals. **(b)** from (Diba and Buzsáki, 2007): spike raster of 13 neurons before, during, and after a single lap with associated LFP (top). The forward replay occurs before the animal starts running (left panel), the reverse replay occurs when the animal reaches the other end and stops (right panel). **(c)** from (Pfeiffer and Foster, 2015): The spike raster of 212 neurons (left) during 2D navigation and an example of the decoded 'mental trajectory' (right), connected by the decoded likelihood distribution of the mental position from each frame (middle) in an awake replay event. The frame (middle) is a 20 ms decoding window advancing at 5 ms intervals. **(d)** from (Benchenane et al., 2010): The rate of the prefrontal cortex (PFC)-hippocampus transiently synchronized assembly (denoted by star in the left raster plot) peaks at the stem of the Y maze (rate map in colour coded), where the animal needs to make a decision, presumably entrained by the emerged PFC-hippocampus theta coherence. Such inter-region theta coherence predicts the behavioral performance.

To see the highly time-compressed 'mental trajectory'[16] of awake replay events (Fig. 1.5 c), one has to use a decoding procedure that employs a small decoding window (∼25 ms) advancing at an even smaller interval (3-5 ms) (Ambrose et al., 2016; Davidson et al., 2009; Pfeiffer and Foster, 2013, 2015). In an individual decoding window, some single-units fire nearly synchronously while each single-unit only contributes a few spikes[17]. It is the selective but flexible transient partnership among a large number of neurons that allows the rapidly evolving 'mental trajectory', which is the core feature of the temporal coding scheme proposed by Malsburg (von der Malsburg, 1981).

Although here I use examples from the hippocampus to illustrate the forms of temporal codes, both theta and SWR events are not restricted to computation inside the hippocampus and can be used to facilitate the reciprocal information transfer between neocortex and subcortical regions and hippocampus (Buzsáki, 1989, 2015; Colgin, 2016; Place et al., 2016; Sirota et al., 2008). The cortical-hippocampal and subcortical-hippocampal synchrony were found to be specifically correlated with behavior in medial entorhinal cortex (MEC)-hippocampus (Yamamoto et al., 2014), prefrontal cortex (PFC)-hippocampus (Benchenane et al., 2010), and amygdala (LA)-hippocampus (Seidenbecher et al., 2003), by quantifying either the oscillation coherence between two regions or the occurrence rate of the cross-region assembly during behavior (or both). For example, Benchenane et al. showed that the transient (PFC)-hippocampus theta synchrony peaked in the segment of the maze where animals need to access their memory to make decisions (Fig. 1.5 d) (Benchenane et al., 2010; Yamamoto et al., 2014).

To summarize, various forms of temporal code have been found in the hippocampus and hippocampal-cortical communication. The hallmark of the temporal codes is a flexible transient partnership among a large number of neurons with a specific relative spike timing pattern. Previous BMI experiments have shown that animals can volitionally control the firing rate pattern of multiple single-units or multi-units (Fetz, 2007; Lebedev and Nicolelis, 2017; Nicolelis and Lebedev, 2009). Likewise, on the way to understanding the relationship between temporal codes and behavior, we should also ask whether the animal can volitionally control the temporal code based on the information we posit the code carries. For example, forward replay has been considered as a hypothesized neural code for planning (Buzsáki, 2015; Foster, 2017). Yet the previous study (Pfeiffer and Foster, 2013) does not rule out the

[16] In Pfeiffer and Foster (2013), the authors defined the decoded 'mental trajectories' that depict clear 2D spatial trajectories as 'trajectory events'. The 'trajectory event' has "a mean duration of 103.6 ms, and path lengths that ranged from 40.0 cm to 199.1 cm". Also see individual examples of these 'trajectory events' in their supplementary movies in this paper.

[17] According to this decoding scheme, replay events cannot be regarded as a rate code because such a short decoding window is not sufficient to reliably express and estimate the rate of any neuron but is sufficient for the downstream circuits to respond to selective transient synchrony of a large population of neurons.

possibility that the awake SWR forward replay is purely spontaneous and is not under the control of the animal, yet somehow biases the future navigation. A BMI experiment could be used to test whether the animal can cognitively manipulate the forward replay to navigate in the virtual world (i.e., the animal teleports itself through a trajectory from one location to another location by generating a forward sequence with a clear 'mental trajectory'), in which the motor engagement can be removed entirely. The same question can also be applied to the theta sequences: whether an animal can volitionally control the temporally ordered sequences of single spikes and burst of spikes from a large number of single-units. The same question can be even generalized to a bursting spike pattern or an arbitrary spike patterns from a large number of single-units. Independent of the information of the cells we recorded in the natural behavior: can the animal learn to generate arbitrary predefined temporal codes? If they can, which patterns are hard to learn and which patterns are easy?

In order to verify the volitional control over various forms of a hypothesized or artificially defined temporal code, a general BMI tool that can in real time detect and respond (e.g. by providing reward feedback) to single spikes from multiple single-units is required. Single-units and single-spike resolution are necessary because this is essential to temporal codes. In some experiments, we may not only care about detecting a specific combination of spikes from a specific set of neurons (Benchenane et al., 2010; Fujisawa et al., 2008; Hirabayashi and Miyashita, 2005; van de Ven et al., 2016; Yamamoto et al., 2014), but also we might care about whether a specific neuron produces single spikes or bursts (Lisman, 1997). In addition, because many of the currently hypothesized temporal codes relate to the hippocampus, the desired BMI must be compatible with tetrodes or modern silicon probes that are commonly used in the hippocampus. Such a tool does not exist in the field of neuroscience yet. Hence, we have to build one. The goal needs to be more specific. Do we want this tool only to support an arbitrary temporal code-based BMI experiment, or do we want it to be also compatible with a rate code-based BMI? Does this tool need to be able to process high-channel-count recordings, and how high? Does it need to be low-latency, and how low? Does it also need to keep the functions developed in previous real-time BMI technology such as multi-unit detection and decoding? In the following section, I will further analyze our motivation, specifically for the neural coding problem, and use the analysis to derive the specific goals of our technology development.

1.3 Motivation and analysis

To understand how the nervous system gives rise to behavior is a central goal of neuroscience. For that reason, we need to understand what action potentials, the universal language of

the nervous system, mean. This is related to the neural coding problem I introduced in the previous section. With the advancement of large-scale recording and neuronal activity perturbation technology, many more questions can be addressed. For this thesis, more questions can now be tackled by closed-loop experimental paradigms, but not open-loop. These questions collectively represent three different aspects of neural coding. In this section, I will discuss these three aspects, the related questions, and what technology is needed to investigate each.

1.3.1 Closed-loop BMI to study the neural code adaptation in learning and cognitive control

First, there is the adaptation aspect of the neural codes. If neural codes cannot adapt, they are less useful. How does the brain adapt its neural activity through learning and cognitive control to support adaptive and flexible behavior? This aspect of the question is difficult to answer because the causal link from neural activity to behavior is elusive in most cases (Jazayeri and Afraz, 2017). The BMI experimental paradigm provides a unique advantage in tackling this type of question, by artificially establishing a causal link between the predefined decoder and behavior (Fetz, 2007). For the BMI prosthetic task, the animal/human is required to learn to rapidly adjust the firing rate of a set of artificially selected neurons to control the external devices. During the learning process, the neurons generate new feature selectivity to match the predefined decoder (Carmena et al., 2003; Gulati et al., 2014). It was found that many brain regions are involved in leveraging macroscale innate learning mechanisms to solve this credit assignment problem in the biological nervous system[18] in the biological nervous system (Koralek et al., 2012; Moxon and Foffani, 2015; Orsborn and Pesaran, 2017). Meanwhile, some studies provided fundamental insights into the difference between controlling the existing network and training a new network via credit assignment (Sadtler et al., 2014; Sakellaridi et al., 2019), and other studies investigated how the credit assignment discriminatively treated the task-related single-units and the task-unrelated single-units during sleep (Gulati et al., 2017, 2014). In the closed-loop BMI paradigm, the artificially established causal link (Fig. 1.6 a) between action potentials produced by selected (randomized) neurons and the decoder provides a unique advantage in examining the innate learning mechanism. BMI can also interrogate the adaptive rules of the neural code with single-unit resolution

[18]The term 'credit assignment' is a concept borrowed from AI but is generally used as well in the BMI and neuroscience literature where it relates to learning with feedback from a defined objective. "The concept of credit assignment refers to the problem of determining how much 'credit' or 'blame' a given neuron or synapse should get for a given outcome. More specifically, it is a way of determining how each parameter in the system (for example, each synaptic weight) should change to ensure the objective." (Richards et al., 2019)

(Gulati et al., 2017, 2014), as well as providing a unique route to tease out the different neural adaptation principles, such as reinforcement learning and cognitive control (Sadtler et al., 2014; Sakellaridi et al., 2019). These studies are still rare and limited to the motor cortex but represent an exciting new direction: the adaptation of the neural code to serve not only the slow credit assignment process but also the fast cognitive control process.

A tacit assumption of all these studies has been the population rate coding scheme, meaning the decoders employ only the information in the firing rate of neurons. The first wave of modern BMI studies in the early 1990s has indeed vindicated that population rate coding can be used by the brain (Fetz, 2007; Lebedev and Nicolelis, 2017; Nicolelis and Lebedev, 2009). However, temporal coding has received little attention in the BMI literature. The possibility of extending the BMI decoder design into the realm of temporal coding has been proposed to validate whether an animal can control the precise timing of spikes of a population of neurons (see the section of 'BMI to study neural code' in (Moxon and Foffani, 2015)). In order to test if an animal/human can control or learn to control the relative timing between neurons at millisecond resolution, a real-time neural signal processor (NSP) is needed to track the timing of multiple single-units simultaneously. This NSP must assign the single-unit identity to each detected spike online. Randomizing the selected task-related neurons and tracking how all trained neurons behave during sleep (Harris, 2014), with their cell-type-specific information, will provide insight into the adaptation rules of the temporal code. For these reasons, a real-time NSP operating at single-unit and single-spike resolution would be a valuable component for using BMI to study the adaptation aspect of the neural code.

1.3.2 Closed-loop perturbation to study the inter-regional communication

The second aspect of the neural code is its causal role in inter-regional communication. Neural codes allow information exchange from region/circuit to region/circuit at the mesoscale level, facilitating behavior in which two or more brain regions are involved. However, in natural behavior, the relationship between recorded activity and the behavior is often elusive (Jazayeri and Afraz, 2017). The most powerful tool here is the intervention method. It is widely accepted that the intervention strategy with randomized and controlled variables of interest is "the gold standard to establish causality" (Bollt et al., 2018; Jazayeri and Afraz, 2017; Marinescu et al., 2018). For an experiment to draw a causal statement requires multiple iterations between hypothesis and experiments: (1) collect data, (2) identify the variable of interest X and its hypothesized causal effect Y, (3) create controlled experiments by

randomizing or controlling X, (4) run these experiments, (5) analyze results and validate the hypothesis of a causal link between X and Y. In order to study the causal role of a neural code with a hypothesized inter-regional or inter-circuit communication function for facilitating a behavior, such an intervention study is necessary. The direct neuronal manipulation technologies, intracortical microstimulation (Penfield and Boldrey, 1937; Romo et al., 1998; Salzman et al., 1990) and optogenetics (Boyden et al., 2005; Zhang et al., 2007), not only allow scientists to statistically randomize the variables but also actively probe or perturb the variables of interest at the single spike level. Optogenetics perturbation offers unparalleled control and specificity in a cell-type and projection defined manner with millisecond precision. One can activate or deactivate a specific set of controlled neurons and test if it is sufficient to drive physiological response and/or behavioral change (Cardin et al., 2009; Deisseroth, 2014; Huber et al., 2008; Liu et al., 2012; O'Connor et al., 2013; Rickgauer et al., 2014; Royer et al., 2012; Sohal et al., 2009). However, using optogenetics alone is still a relatively crude method for cracking the hypothesized role of the neural code in inter-region information exchange. It is still not possible to genetically tag the specific functional cell-types such as place cells and grid cells that carry specific information to create the fully controlled neural code with high fidelity using optogenetics alone. One possible solution to solve this difficulty is to use a closed-loop paradigm with real-time monitoring of the on-going activity and transiently perturb the downstream region before a detected neural activity pattern transmitted from the upstream region reaches its target region. The high fidelity of inter-regional communication is kept when no perturbation is delivered, while the controlled variable is created when the perturbation is transiently delivered upon a detected neural activity pattern.

The electrophysiology closed-loop perturbation method so far has only employed LFP as the readout signal (Ego-Stengel and Wilson, 2010; Fernández-Ruiz et al., 2019; Girardeau et al., 2009; Jadhav et al., 2016; Maingret et al., 2016; Rothschild et al., 2017; Roux et al., 2017; Siegle and Wilson, 2014; van de Ven et al., 2016; Yamamoto et al., 2014). This can only test the hypothesized role of all the underlying spiking activity during an LFP state rather than testing a causal role of specific neural code with controlled spike content and cell-type specificity. The time window for stimulation based on LFP pattern recognition rather than spike detection is still relatively long. Minimizing optogenetic stimulation time could be essential for minimizing secondary or off-target effects[19] (Otchy et al., 2015). By quantifying this off-target effect, Otchy et al. found that switching from sustained stimulation (1 s) to a brief stimulation (50 ms) can reduce off-target effects by several fold (Extended Data Figure 2C in Otchy et al. (2015)). This work critically reminds us that minimizing

[19]Here the secondary or off-target effects means the effect that caused by the optogenetic stimulation that is not pathway-specific and perturbs an unintended downstream region.

the optogenetic stimulation time could be important for reducing off-target effects in neural perturbation experiments. A closed-loop perturbation based on a single-spike, rather than LFP, can minimize this perturbation time. The optogenetic stimulation on the downstream target could be reduced to only one or a few milliseconds. Based on the information that spike carries, the perturbion of neuronal integration at the arrival time to downstream dendrite is enabled. Moreover, inter-region/circuit communication often involves the coordinated activity of various genetic cell-types (Luo et al., 2018), highlighting the importance of single-unit resolution for the closed-loop perturbation experiment. For these reasons, a real-time NSP providing single-unit and single-spike resolution with low latency response would be valuable. It could function as a gatekeeper in an inter-region/circuit pathway that intercepts the targeted neural spiking activity pattern (hypothesized neural code) and tests the necessity of such communication for its hypothesized role in behavior (Fig. 1.6 b).

The spatiotemporal constraints on optogenetic inactivation were thoroughly examined at a large scale in cortex (Li et al., 2019). The authors showed the activation of excitatory channelrhodopsins (ChR) expressed in GABAergic neurons is currently the most effective way to locally inactivate a target brain region. The fast-spiking GABAergic neuron responds to light in 1.1 ± 0.2 ms, followed by a drop of firing rate of local pyramidal neurons within 3 ms[20] and reaches a maximum photoinhibition effect at 18.4 ± 1.6 ms. Let us take this into account and imagine a thought experiment. We implant a set of tetrodes or silicon probes in the hippocampus to record from multiple single-units simultaneously. A real-time NSP is used to detect single spikes from single-units with low-latency. The tool reports a predefined relative spike timing pattern within 1 ms after the last spike of a target temporal code, triggering ChR-assisted photoinhibition in a specific downstream cortical region. The fast-spiking GABAergic neuron becomes active in the next 1 ms, starting to impair the ability of the local region to integrate input. The photoinhibition will reach its maximum effect within 18 ms, at which time the output of the target region is silenced. Therefore, we should turn off photoinhibition before it reaches the maximum because our purpose is to specifically perturb the target region from integrating the input. So where is the input now? The previous detected temporal code must go through the axon conduction delay to reach the downstream region, usually taking longer than 3 ms for inter-region communication. With a low-latency (i.e., $\sim$1 ms) single-unit, single-spike detector, and fast mesoscale ChR-assisted photoinhibition, it is possible to prevent an arbitrary downstream region from receiving an arbitrary temporal code, with minimum side effect. Importantly, the faster the real-time

[20] But being cautious about that there is a spatial constraint for the ChR-assisted photoinhibition as the authors reported "Photoinhibition at 2 mm away lagged the photoinhibition at the laser center by 10 ms".

signal processor can detect single spikes from single-units, the more time budget can be assigned to buffer the delay of the feedback constrained by physiology.

1.3.3 Closed-loop perturbation to study the microcircuit processing

The third essential aspect for probing the neural code involves its generation and processing. Neural activity is produced and processed by the interaction of neurons of various cell types within neural circuits. As Luo et al. state "the major goals in systems neuroscience are thus to determine (1) how these neural representations arise in defined cell types and (2) how neural representations in specific cell types relate to behavior. Addressing these challenges requires cell-type-specific recordings and manipulation of neural activity." (Luo et al., 2018). Notably, "neuronal cell types reflect a collection of parameters including cell body location, dendritic morphology, axonal projection, physiological characteristics, developmental history, gene expression pattern, and function." (Luo et al., 2018). To study neural code generation and processing at the microcircuit level with cell-type-specific information, single-unit resolution is necessary. A combination of silicon probes, optogenetics and real-time neural signal processing has been proposed to "allow the monitoring of brain activity at the single-neuron, single-spike level and the targeted manipulation of the diverse neuron types selectively in a closed-loop manner" to study local circuit operation (Fig 1 in Buzsáki et al. (2015)). This paper highlighted several technological advancements on integrating silicon probes and micro-LEDs to enable near-site, multiplexed control over just one or a few neurons (Buzsáki et al., 2015; English et al., 2017; Kim et al., 2013; Stark et al., 2012). Notably, here the goals, as described by Buzsaki et al. are "(1) identification of genetically labelled neurons, (2) physiological characterization and classification of neuron types and (3) testing the causal roles of the identified neurons in the performance of local circuits." (Buzsáki et al., 2015). Importantly, this combination would potentially provide new opportunities to advance or delay the timing of specific individual neurons and further perturb the neural code at an even higher resolution (Buzsáki et al., 2015; Cobb et al., 1995). The single-unit analysis methods, including electrophysiology-based cell-type identification (Barthó et al., 2004; Jia et al., 2018; Li et al., 2019, 2015) and cross-correlogram (CCG), reflects the functional connectivity between neurons with cell-type specificity (Barthó et al., 2004; Buzsáki, 2004). This combination allows us to identify many different circuit motifs and how the functional connectivity among neurons evolves during behavior (Benchenane et al., 2010; Constantinidis et al., 2002; English et al., 2017; Fujisawa et al., 2008; Hirabayashi and Miyashita, 2005; Yamamoto et al., 2014). Closed-loop perturbation at the microscale will provide feedback to stimulate one or a few neurons, either electrically, optogenetically or even using patch-clamp, depending on the specific question and the requirements for feedback latency. Again,

for the real time signal processor, the faster the better. An intervention study at this level could be constructed based on the off-line analysis of the data collected. The hypothesis of how a neural circuit produces and processes activities can be formed from the functional connectivity map, and from which the controlled experiments can be designed. Once the controlled variable is decided, so is the rule of the feedback. Once the causal feedback loop is established, a new circuit motif emerges by artificially connecting or disconnecting neurons via single-unit single-spike low-latency detection and feedback. Additionally, low-latency feedback from neuron-to-neuron can further introduce timing or modulation-based plasticity on top of the artificially established circuit motif. By randomizing the circuit motif with both functional and genetic cell-type specificity, the causal link can be drawn between circuit operation and neural code generation and processing, that is, how the neural circuit computes with spikes.

1.3.4 What is required for all above experimental paradigms?

To summarize, these closed-loop experiments based on real-time neural signal processing, operating at macro-, meso- and micro-scale to investigate different aspects of the neural code, offer unique advantages over correlational open-loop experiments (Fig. 1.6 a). The macro-, meso-, micro-scale are loosely defined according to the spatiotemporal scale that the artificially established causal-loop spans and the number of brain-regions involved.

At a macroscale (Fig. 1.6 b), closed-loop BMI for controlling external devices is employed to study how learning and cognitive control differentially affect the adaptation of neural code, and how credit assignment differentially changes task-related and task-unrelated single-neurons during sleep. This loop involves many brain regions and the brain's innate learning and attention mechanisms. With single-spike and single-unit resolution for decoding, we can extend the current existing BMI task into the realm of temporal codes and test if and how animals can control the precise relative timing among a population of neurons.

At a mesoscale (Fig. 1.6 c), spike-activity-based closed-loop perturbation is employed to test the necessity of neural code in inter-region communication. The artificially established loop involves two communicating regions or circuits, either bidirectional or unidirectional. The hypothesized role of a targeted code transmitted from one region to another region during behavior can be examined by real-time monitoring of the activity pattern generated in the upstream circuit and rapid perturbation of the downstream circuit before it receives the code. Moreover, with single-spike and single-unit resolution, a large portion of the hypothesized functional role of the transient temporal pattern of activities with cell-type specificity can be examined with minimum off-target effect.

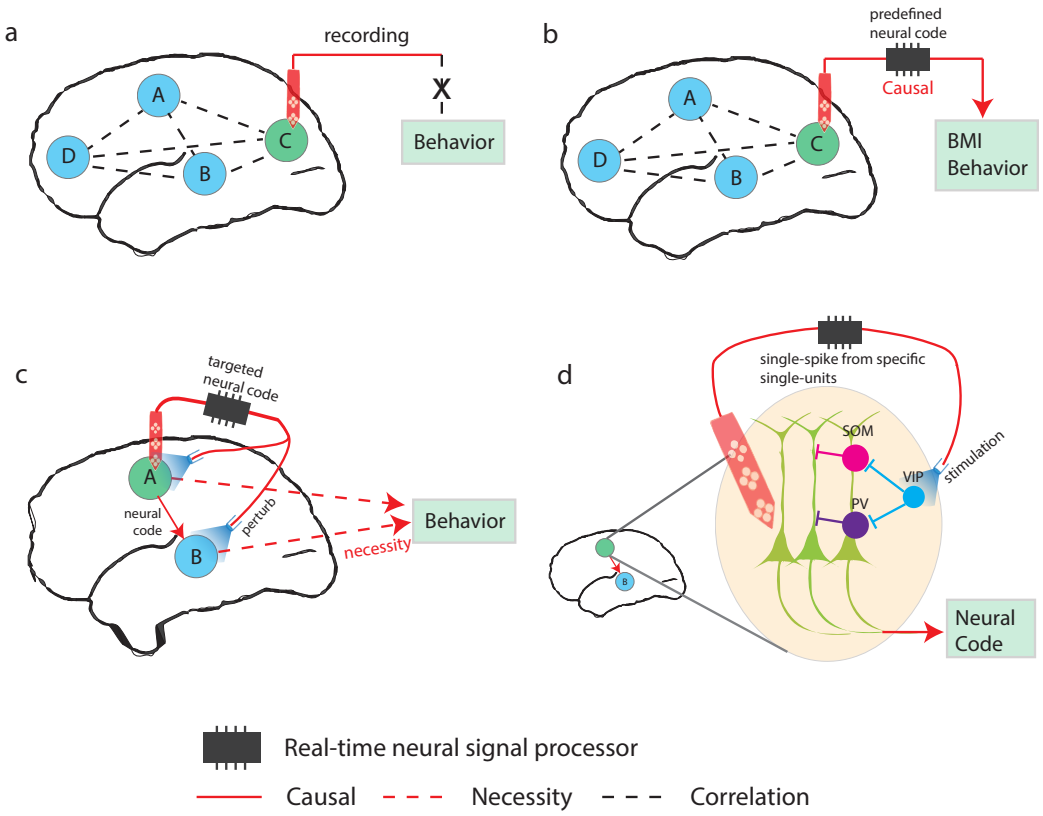


Fig. 1.6 **Correlation and causal study of the neural code with a low-latency Neural Signal Processor (NSP): (a)** Open-loop experiment. The hypothesized neural code in region C and behavior is correlated. **(b)** BMI experiments where the animal has to learn to control the firing pattern of a set of neurons in the recorded region C. A NSP (the black chip connected to a probe) artificially establishes a causal link between the predefined neural code and behavior while the global learning mechanism is involved. **(c)** Closed-loop low-latency mesoscale perturbation to test the necessity of an arbitrary neural code, transmitted from region A to region B, for the behavior. A NSP is used to detect an arbitrary spike timing pattern from multiple single-units. Mesoscale inhibition is triggered by the detection of the hypothesized temporal code. **(d)** The NSP artificially bridges the recorded single neurons and the stimulated neuron in a low-latency closed-loop manner with single-unit and single-spike resolution. A large number of artificial micro circuit motifs can be constructed to study neural code generation and processing with cell-type specificity (VIP, SOM, PV are the names of sub-types of inhibitory neurons; each of them has a distinctive projection and connectivity pattern.).

At a microscale (Fig. 1.6 d), the single-spike from a genetically identified single-unit can be used to trigger the activity of other neurons. This artificially established loop involves cell-type-specific neurons that are embedded in a circuit motif. The hypothesized circuit motif is constructed from both an off-line analyzed functional connectivity map and optogenetic

circuit mapping. Closed-loop in this regard equates to intervening in the circuit motif in a controlled manner. For example, one can artificially connect or disconnect a random hippocampal CA1 place cell projection to a set of interneurons (see Royer et al., 2012) or even a single interneuron to investigate the effect on theta phase-precession. This artificial connection can be established by either optogenetic stimulation or faster electrical/patch-clamp stimulation depending on the required feedback speed. These types of activity-based closed-loop stimulation, with single-unit single-spike resolution, could be useful to probe how the neural circuit gives rise to the neural code, particularly the temporal code, mechanistically.

A real-time NSP is required for all these possible experiments at different scales. Specifically, the single-unit and single-spike resolution with a low-latency response are not absolutely required for every experiment. (1) The requirement for single-unit resolution depends on whether relative spike timing, cell-type, bursting, functional connectivity or other single-unit specificities of single neurons are relevant to the questions. (2) The need for single-spike deterministic timing depends on the importance of precise spike timing information to the question. (3) The need for single-spike low-latency response depends on the speed requirement for the feedback, which is determined by the timescale of the specific phenomenon under investigation. The faster the NSP can assign single-unit identity to a single spike, the more time is saved for further decoding or buffering the inevitable delay for the feedback to take desirable effect. A latency of around 1 ms can be useful for a subset of questions at the macroscale, it becomes more needed at the mesoscale, and it can be necessary at the microcircuit level.

In the next section, the technology development goals for the real-time low-latency NSP will be derived from the analysis presented in this section, with a comparison between our aims and previous works on spike-based real-time closed-loop systems.

1.4 Previous works and the goal of this thesis

From the analysis in the previous section, I converged to what Buzsáki et al. concluded: “To understand how function arises from the interactions between neurons, it is necessary to use methods that allow the monitoring of brain activity at the single-neuron, single-spike level and the targeted manipulation of the diverse neuron types selectively in a closed-loop manner.” (Buzsáki et al., 2015). Such methods can facilitate the causal study of at least three types of experiment paradigms that engage the brain at three different scales (Fig. 1.6). To perform such methods, a real-time NSP that can infer population spiking activity at low-latency with single-unit and single-spike resolution is required.

A variety of spike-based NSPs have been built around microwire arrays to support BMI experiments since the 1990s (Lebedev and Nicolelis, 2017). The spike-based closed-loop experiment has been extensively explored in BMI studies (Lebedev and Nicolelis, 2017). It can be implemented either on-PC or on-chip. The on-PC solution detects spikes with stochastic timing because it usually includes a non-real-time operating system (OS) in the loop. This represents a large portion of recent BMI experiments (Athalye et al., 2018; Gulati et al., 2017, 2014; Koralek et al., 2012; Neely et al., 2018; Sadtler et al., 2014), where the firing rate of single-units or multi-units was used for the BMI behavior, in which the precise and deterministic timing of each spike was not required. Multi-unit BMI studies are most commonly conducted on monkeys with an implanted Utah array. The single-unit BMI experiments in rodents are mostly done using other types of microwire array with much fewer electrodes than the Utah array. Consequently, these experiments used a few single-units (<10), and the single-units were isolated from single channels rather than groups of channels. One exception (de Lavilléon et al., 2015) indeed performed group-based spike sorting to isolate single-units using tetrodes. Nevertheless, this study again adopted a tailored on-PC algorithm when performing online inference and precisely timed feedback was not required. The customized algorithm was based on threshold-crossing (the selected single-unit for real-time feedback had a substantially high amplitude spike on one channel).

On the other side is the on-chip solution, where the acquisition and online processing is conducted on the same chip simultaneously. The on-chip solution can provide deterministic and low-latency processing. The processing chip is a configured or customized integrated circuit (IC), where the input and output are scheduled at the precision of nanoseconds. The same algorithm in an IC runs much faster than on a PC. For an IC, at each single clock cycle, massively distributed digital circuits can implement the desired operations in parallel. In addition, due to its customizable hardware nature, the on-chip solution provides not only the advantage of speed but also scalability. It naturally fits into the application when the real-time processing of thousands of channels is needed.

On-chip spike inference is a newly emerging field in recent years. Navajas et al., performed a simulation to show the feasibility of the on-chip solution for real-time spike inference, in which the authors proposed a two-stage method (Navajas et al., 2014). The first stage builds a spike-sorted model in software, and the second stage downloads the model into the chip to guide real-time inference (Navajas et al., 2014). The Field Programmable Gate Array (FPGA) is an ideal chip to be used here because the FPGA is programmable (FPGAs will be introduced in detail in Chapter 3). FPGA-based single-unit inference has been demonstrated in-vitro (Dragas et al., 2014, 2015; Müller et al., 2013), which demonstrated low-latency single-unit inference capacity on a large population of neurons in culture

and in vitro. However, there are critical differences in the electrode geometry between cultured, in-vitro and in-vivo data. Furthermore, an in-vitro solution does not need to deal with the issues of motion noise correction, chronically lost channels, flexible configuration of electrode group according to the geometry of the electrodes, etc. Recenty, in-vivo on-chip systems, exploiting the power of algorithmic IC's for real-time single-unit inference, have been reported (Luan et al., 2018; Park et al., 2017; Wang et al., 2019). These works demonstrated several end-to-end solutions of channel-based single-unit inference on 128 and 32 channels, where the real-time signal processing was conducted in an FPGA. These works represent a significant step forward for in-vivo low-latency single-unit inference. However, these works still employed channel-based spike sorting. Channel-based spike inference assumes no correlation between spike waveforms from adjacent channels and is naturally compatible with microwire arrays. It is much easier to implement in the FPGA since it does not operate on the combination of waveforms collected from adjacent electrodes. However, it cannot be used with tetrodes and silicon probes and would lead to lower yield and lower accuracy in single-unit inference. Therefore, these recently developed real-time spike inference systems are unlikely to be applied in rodent hippocampus and other brain regions where many neurons are densely packed. The most recent reported progress is the Neuralink ASIC (application-specific integrated circuits) (Musk and Neuralink, 2019). It can simultaneously detect spikes on 3072 channels with 900 nanoseconds latency. However, the Neuralink chip does not employ single-unit inference yet. Instead, it performs spike detection via channel-based threshold crossing as most BMI studies have done, since decoding movement parameters in primate motor cortex does not require single-units. The recent progress of on-chip NSPs is summarized in Table 1.1 (SUA denotes single-unit activity; MUA denotes multi-unit activity).

Table 1.1 Online spike inference systems

	On-Chip Solution
Simulation	(Navajas et al., 2014)
In Vitro	(Müller et al., 2013) (Dragas et al., 2014) (Dragas et al., 2015)
In Vivo (MUA)	(Musk and Neuralink, 2019)
In Vivo (channel-based SUA)	(Park et al., 2017) (Luan et al., 2018) (Wang et al., 2019)
In Vivo (group-based SUA)	This thesis

This thesis aims to develop an FPGA-based NSP for low-latency, grouped-based, single-unit inference with single spike resolution, using the two stage method proposed previously (Navajas et al., 2014). In order to apply this system to the experiment paradigms I analyzed in the last section, it needs to be compatible with high-channel-count class-2 recording devices such as a tetrode array and silicon probes. For that, the FPGA NSP is integrated with a 160-channel acquisition system that supports five INTAN RHD2132 chips[21] that provide the raw digital neural input. The data acquisition and single-unit inference is performed on a same chip (an FPGA) simultaneously. To build the group-based spike-sorted model for the on-chip single-unit spike inference, a software package was developed to perform group-based spike sorting, data visualization and manual curation. The created model is then downloaded to the FPGA to guide real-time signal processing. To tolerate the complexity of recording in a behaving animal, the system also needs to manage the motion-based noise, lost channels and provide a flexible grouping of electrodes. With the spike-sorted model downloaded into the FPGA, our NSP aims to simultaneously perform preprocessing and detect spikes on 160 channels and assign spike identity within 1 ms latency.

From the technique perspective, this thesis does not invent any new recording method, surgery procedure, amplification, or digitization ASIC. It integrates existing solutions for these parts which are widely used in current neuroscience laboratories and focuses on the parts that did not exist yet: the on-chip neural data processing for low-latency population single-unit spike inference and the software to quickly spike sort and operate the FPGA NSP. Therefore, this thesis does not deal with other issues in BMI or neuroscience, although these issues are related. These issues include, the stability of the single-units during chronic recording, power consumption, and, the spike decoder design. Also, although this thesis raises some hypotheses about what kinds of experiments could be enabled by the device, none of these experiments are in the scope of this thesis. A major limitation of this thesis is that our method is subject to the limitations of multi-electrode recording and the spike sorting method (will be further discussed in other chapters). Since the NSP spike inference model (i.e., the spike-sorted model) is generated from the spike sorting procedure, the accuracy of our on-chip spike inference is also bounded by the accuracy of the spike sorting method. I will futhur discuss the limitations in the last chapter.

To achieve the goals described in this chapter, in the last four years, I designed, developed, debugged most of the components and integrated the system presented in this thesis, including the software and the hardware, except where specific comments in text are made to refer the

[21] The RHD2132 is a 32-channel digital electrophysiology recording chip from INTAN Inc. The RHD2132 chips are widely used in in-vivo experiments to sample, amplify, digitize and multiplex the multi-channel raw neural signals collected by tetrodes or silicon probes. The digital output of RHD2132 is sent to the user acquisition system.

work of others, the work conducted with the support of others, or the work built on top of others previous work. In the next two chapters (one for software and one for the FPGA NSP), I will describe the design and development of the system in detail.

Chapter 2

The model generator for real-time spike inference

2.1 Big Picture: Software-Hardware codesign for group-based spike inference

Two features distinguish this thesis from previous work on the development of a real-time spike inference system introduced in the previous chapter.

First, in order to use tetrode arrays and silicon probes to monitor many neurons simultaneously and in order to infer the identity of single-units accurately, our system performs group-based, rather than channel-based, real-time spike inference. As illustrated in Fig. 2.1 (a *vs* b), the group-based spike inference with groups of low inter-electrode distance electrodes benefits by using the correlated (but not the same) spike waveforms recorded from adjacent channels (Gray et al., 1995; McNaughton et al., 1983; Wehr et al., 1999). Group-based spike inference is to my knowledge a missing feature in all related previous works (Luan et al., 2018; Park et al., 2017; Wang et al., 2019), and it holds the key to integrating accurate online spike inference with high-channel-count tetrode arrays and silicon probe recording. Tetrode arrays and silicon probes are widely used in neuroscience. Tetrodes are particularly useful for recording in brain regions where neurons are densely packed, such as the hippocampus (O'Keefe and Recce, 1993; Pfeiffer and Foster, 2013), and silicon probes are particularly useful for recording a massive number of neurons from multiple brain regions simultaneously. In order to be compatible with both tetrode arrays and silicon probes, our real-time spike inference system was built with four electrodes per group. Four electrodes per group also works well with silicon probes. It has been shown with ground-truth data that three to four electrodes with $\sim$25 micrometres spacing are sufficient to distinguish nearby neurons (Henze

et al., 2000; Hunt et al., 2019; Neto et al., 2016)[1]. Also, an extracellular spike can only be detected within approximately two hundred micrometers, which spans a few electrodes, usually less than eight, in state-of-art silicon probes (Buzsáki, 2004; Hong and Lieber, 2019; Jun et al., 2017b).

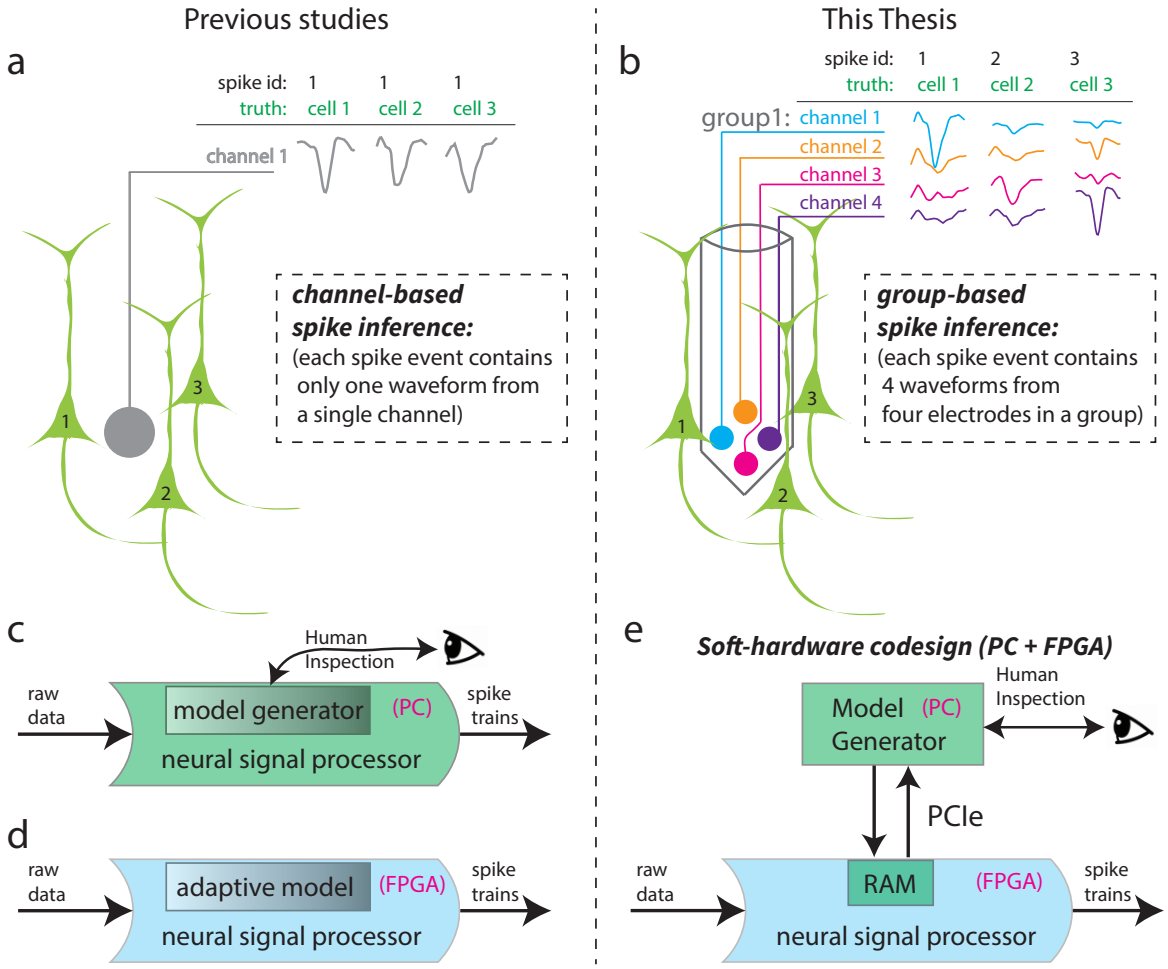


Fig. 2.1 **Software-Hardware codesign for group-based spike inference**: (a) Channel-based spike inference. Spikes from different neurons can be indistinguishable using single channel (grey) if the distances to the nearby neurons (green) are similar. (b) Group-based spike inference. The combination of spike waveforms from a group of adjacent electrodes makes source separation (clustering and classification) easier. (c) On-PC solution for spike inference allows for human inspection and intervention, thus is flexible. (d) Hardware as an on-chip solution using an adaptive spike inference model is fast and efficient but is unlikely to be accurate and makes human inspection difficult. (e) Software-Hardware codesign as a solution for spike inference in this thesis.

[1] In Henze et al. (2000), it was also shown that single electrodes are not sufficient to distinguish spikes from nearby neurons.

A possible problem of using four electrodes as group in the silicon probe is that the same spike could be detected sometimes by two groups and rarely by three groups (Einevoll et al., 2012). However, this redundancy can be easily removed by examining the pairwise cross-correlogram of all reported neurons. If two neurons are an exact or approximate copy, the cross-correlogram shall exhibit a dominant peak at zero.

Second, most previous works applying spike-based closed-loop feedback either focuses on hardware (on-chip Fig. 2.1c) or software (on-PC Fig. 2.1d) for the real-time spike inference, rather than a combination of both. The on-chip solution is quite efficient and can process spikes with minimum latency, but the on-chip solution alone requires the algorithm be implemented in hardware to be adaptive and accurate enough to operate without human inspection and intervention. This, to my knowledge, is currently unlikely to be achieved because the problem of spike sorting is far from full automation. On the other hand, the on-PC solution can be quite flexible and accurate, but is slow. My approach uses a soft-hardware codesign method to achieve both the efficiency of the on-chip solution and the flexibility of the on-PC solution; the real-time spike inference was implemented in an FPGA by applying a spike inference model, and the software to generate the spike inference model was developed in a PC (Fig. 2.1e). To be specific, in my work, the data acquisition and real-time spike inference take place in an FPGA NSP. Such two-stage processing (i.e., building the model in software and then applying the model in hardware) has been proposed using a simulation (Navajas et al., 2014) but the implementation of the NSP and the software to build the model and operate the NSP has not been presented yet. The NSP also includes several intermediate computational steps: filtering, reference subtraction, spike detection, feature extraction, and spike feature classification. The algorithms of these computational steps were implemented in an FPGA, and the intermediate and final processed data was sent to PC. The parameters of the FPGA-implemented algorithms in the NSP are determined in the PC by software developed to build the spike inference model, which I named Spiketag (https://github.com/chongxi/spiketag).

The software part (Spiketag) will be introduced in this chapter, and the hardware part (FPGA NSP) will be introduced in the next chapter. Here I briefly introduce Spiketag at a high-level and leave the logic and methods of its design to the rest of this chapter.

The Spiketag was developed in Python with numerous popular packages such as ‘numpy’, ‘scipy’, ‘pytorch’ and ‘vispy’. The process to construct a spike inference model using Spiketag remarkably resembles a spike sorting pipeline, with some differences that will be introduced in this chapter.

In Spiketag, a backend clustering engine and a multi-perspective 3D visualization and interaction system were developed for the experimentalist to visually inspect, interact with

the data and build the spike inference model progressively. The backend clustering engine was developed to be non-blocking to the frontend, such that automatic clustering and manual curation can be conducted on different electrode groups concurrently. The interactive modules in Spiketag were developed to have the OpenGL rendering capability[2] such that the interaction with a large amount of neural data can be done in 3D with no lag, just as when playing a modern video game. All Spiketag software modules can be used in the Jupyter notebook[3], which allows simultaneous neural data analysis and fast 3D data visualization/interaction. In addition, the Spiketag application programming interface (API) was developed to communicate with the FPGA to read the produced data and configure the parameters into the FPGA RAM through the low-latency Peripheral Component Interconnect express (PCIe) interface (Fig. 2.1e).

2.1.1 Software-Hardware tradeoff

Since software-hardware codesign is used here, an important question to address before development is to decide the tradeoff between software and hardware, that is deciding which task belongs to the software and which task belongs to the hardware. In order to minimize the development in hardware, which is time consuming, the hardware computation was devoted to the tasks that must operate in real-time to achieve low-latency spike inference, such as multichannel filtering, spike detection, transformation and classification (Fig. 2.2 lower panel in blue). In other words, the algorithms in the classic spike sorting pipeline, except the feature extraction and the clustering, were all implemented in an FPGA. On the other hand, the tasks that require data visualization, interaction and data analysis to build and update the parameters for the FPGA operation were developed in the software (Fig. 2.2 upper panel in green). In other words, software was developed to build a spike inference model using spike sorting, except that the sorting steps (blocks in Fig. 2.2 with a star) before feature extraction and clustering were all implemented in hardware.

[2]Open Graphics Library (OpenGL) is an across platform industrial standard that uses graphics processing units (GPUs) to achieve hardware-accelerated real-time rendering. OpenGL has been long applied in game development.

[3]Jupyter notebook is currently used in many neuroscience labs, it dramatically eases the verification and sharing of work in 'data science, numerical simulation, statistical modelling, machine learning, and much more.' (check https://jupyter.org/).

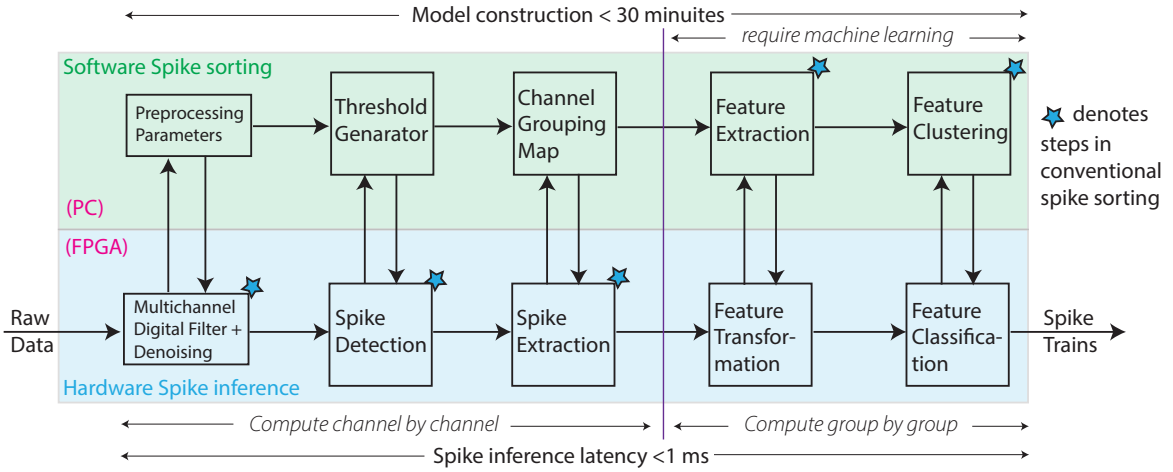


Fig. 2.2 **Software-Hardware tradeoff**: (upper panel in green): software pipeline constructing a spike inference model. Software components in the pipeline read the data output from the hardware and use it to decide the parameters in the spike inference model (lower panel in blue): hardware (FPGA) pipeline to perform real-time spike inference using the model constructed by software. A spike inference pipeline (lower panel in blue) is similar to spike sorting (steps denoted by pentagram) except it changed the feature extraction to feature transformation and the clustering into classification. For each algorithm implemented in FPGA has a corresponding software component to read and write the required parameters. For example, the parameters used by feedforward feature transformation and classification are computed from feature extraction and clustering and then downloaded into the FPGA. The arrows denote the parameter/data flow.

Notably, in the early phase of the spike inference the computation is carried out on each sample channel by channel (Fig. 2.2 before Feature Transformation), but after spike events from adjacent channels are extracted from the data stream and are transformed into the feature space, the computation is carried out group by group. Thus the amount of the computation carried out in the hardware is substantial in the beginning (Fig. 2.2 before Feature Transformation) but is then reduced because spikes are sparse and the number of groups is less than the number of channels. Oppositely, in the software, the parameter construction is easier in the beginning (Fig. 2.2 before Feature Extraction) because we simply need to load and check the intermediate output from the hardware and decide whether to update the basic parameters such as spike detection threshold, but gets heavier in the later phases because using extracted features to construct the feedforward feature transformation and using feature clustering to construct the classification model requires machine learning and user visualization/interaction.

The software development of the Spiketag is described in the following sections of this chapter, and the hardware development of the FPGA NSP is described in the next chapter.

2.2 Spiketag: a fast and interactive spike inference model generator

2.2.1 Challenges in spike sorting and the approach of Spiketag

Since this thesis uses spike sorting to build the real-time spike inference model, the limitations of spike sorting have to be discussed first. Spike sorting used to be very slow; an hour of recording on hundreds of electrodes could cost researchers weeks to sort. Recent work on spike sorting methods has focused on improving automatic sorting speed on high-channel-count neural recording, reducing the time spent on automatic sorting from weeks to less than the recording time by leveraging the efficiency of GPUs (Chung et al., 2017; Jun et al., 2017a; Pachitariu et al., 2016; Yger et al., 2018). However, for all the existing algorithms, the automatic sorting result is usually not reliable. To get a satisfying sorting result, intensive human intervention is still required (Rossant et al., 2016).

The most unreliable link in the automatic spike sorting pipeline is the clustering. Could the situation soon get better and spike sorting be made fully automatic without any human inspection? Unless we choose to discard a large portion of the observable spikes, the answer may be no. Arguably, the reasons for that can be seen by comparing different forms of statistical learning.

Clustering belongs to one of many forms of statistical learning paradigm, namely unsupervised learning, and it is the most difficult form of learning. Why? Because no ground truth can be involved to guide the learning process. In contrast, the supervised learning, semi-supervised learning and reinforcement learning all allow feedback signals from some performance metric computed using the ground-truth; the difference between these forms of learning is how the ground-truth information is acquired, how the feedback signal is computed and the frequency of the feedback[4]. So far, spike sorting has not been formulated as semi-supervised or reinforcement learning, and the amount of ground-truth data is not big enough to convert it into a supervised learning problem. Hence spike sorting still uses pure unsupervised learning which depends on performance metrics that are generally defined without ground-truth information. Unsupervised learning paradigm is generally hard, but it might not be the only and primary reason that prevents reliable automatic spike sorting. Other difficulties are centered around clustering, specifically, clustering the spikes.

[4]Supervised learning system constantly receives feedback from error from a large number of labelled data and use that to drive learning; semi-supervised learning system takes a small size of labelled data but with data augmentation it can generate a large number of self-labelled data for learning; the reinforcement learning posits an acting agent and an environment, from which sparse reward signals are used to guide the learning. All these forms of learning allow feedback signals from a ground truth metric to assign the credit to the system to achieve a good performance.

Clustering aims to find a partition function to separate the spikes into distinct categories according to the shape of the waveforms. However, problems lurk behind this deceptively intuitive goal. First, a theorem has been proven that it is impossible to find a partition function to satisfy three reasonable and desirable properties simultaneously, namely scale-invariance, richness and consistency of the partition on all of the dataset (see Kleinberg (2003) for details, but also see criticisms from Ackerman (2012)). Second, in practice, clustering in spike sorting is particularly unreliable when the data is non-stationary, which happens all of the time due to small electrode drifts, bursting, spike waveform adaptation and overlapping spikes adjacent in time and electrodes (Lewicki, 1998; Rey et al., 2015). The non-Gaussian distributed features (Fee et al., 1996; Rey et al., 2015) produced for these reasons and the possible non-linear decision boundary between clusters can be observed in the individual feature spaces of almost all high-channel-count extracellular recording. In some cases, even when two seemingly separable cluster centers can be clearly observed, their boundary can be contaminated by those out-of-distribution spikes, which make it hard to draw the decision boundary. For all these reasons, even with the most recent spike sorting package, human visual inspection and manual curation is still an inevitable step before spike train analysis (Rossant et al., 2016). Manual curation usually, in practice, takes much longer than the automatic sorting[5].

In short, slowness and unreliability appear to be two major challenges presented in the spike sorting process. The recent development of offline automatic spike sorting packages has tremendously advanced the sorting speed (Chung et al., 2017; Jun et al., 2017a; Pachitariu et al., 2016; Yger et al., 2018). Yet the human sorter still needs to wait tens of minutes until the automatic sorting results return. In order to quickly build a model for online spike inference, we aimed to minimize further the waiting time for the experimentalist to enter the manual curation stage. In addition, the recent progress in spike sorting has not yet solved the second challenge, the unreliability. We admit that current automatic sorting is not sufficient, and software development on fast manual curation is needed as suggested in (Rossant et al., 2016). Such need is more evident when a reliable sorting result is required to build the real-time spike inference model for closed-loop experiments. Therefore, the purpose here is not to build a full-blown, stand-alone and broadly applied spike sorting package. Instead, the goal here is to produce a software architecture to minimize the time needed to build the spike inference model for the hardware computation, and we aimed to offer an efficient 3D data visualization and manipulation interface to allow the human sorter to perform easy data inspection and fast curation. In the end, using our software, the experimentalist should be

[5]Empirically speaking, up to the time I am writing this thesis, most of the spike sorters who deal with roughly an hour of recording of a hundred of channels have to wait at least tens of minutes for the automatic sorting to finish, and then spend several hours on the visual inspection and manual curation.

able to make fast and flexible decisions on how accurate the model shall be for his/her online closed-loop experiment.

The slowness and unreliability in spike sorting remain as the same challenges for building the spike inference model[6]. In the following sections, I will explain the approaches that Spiketag has adopted in greater detail.

2.2.2 A non-blocking architecture in Spiketag

Spiketag distinguishes itself from previous spike sorting packages by applying a non-blocking software architecture. The recent progress in spike sorting focuses on implementing a fast GPU signal processing pipeline to speed up the automatic sorting. This approach is still a 'blocking' approach[7] because the manual sorting has to wait until the automatic process finishes and returns. In contrast, my approach implemented a non-blocking software architecture, in which the manual curation and automatic sorting are conducted concurrently. A key reason that this approach can work is that the high-channel-count electrophysiological recording can be "divided and conquered" group by group (Fig. 2.3b). With non-blocking architecture, the experimentalist can perform manual curation on one group while the backend keeps clustering other groups. To be specific, as soon as the recording is finished, the experimentalist can immediately work on the few groups where the automatic sorting is already finished, while other groups keep clustering in other CPU processes, which might take tens of minutes for all groups to finish. As long as at least one group contains clear cluster structure and can return as soon as the recording is finished, the experimentalist can work on the manual curation on that group without waiting. In practice, a single group's manual curation almost always takes longer than a single group's clustering. Therefore, once the experimentalist starts the manual curation, no further waiting is needed. For this reason, my approach bypasses the slowness of automatic spike sorting in high-channel-count recording and leads to almost no waiting time in real experiments. Here, the non-blocking software architecture is visualized in Fig. 2.3.

[6]Meanwhile, there are other challenges in spike sorting, such as dealing with electrode drift, quantifying the uncertainty of each splitting or merging of the clusters and building quality metrics for spike sorting. These challenges are beyond my scope in this thesis.

[7]See https://en.wikipedia.org/wiki/Blocking_(computing)

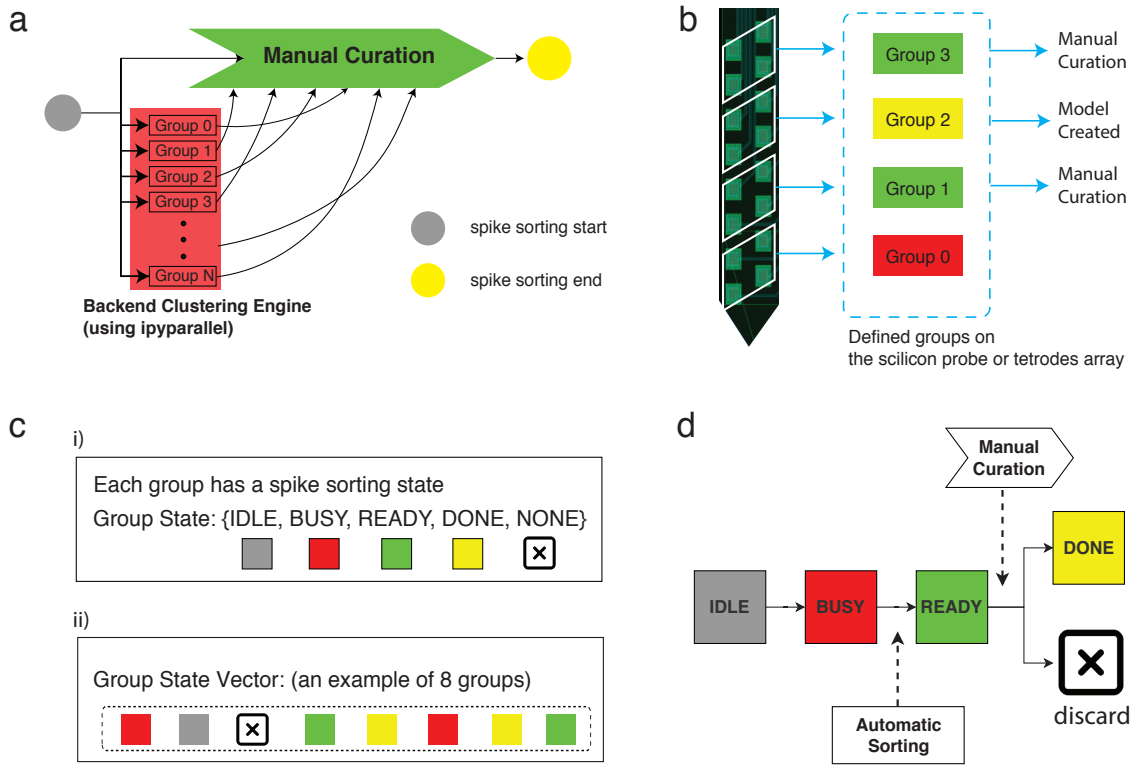


Fig. 2.3 **Non-blocking group-based clustering backend engine in Spiketag**: (a). When spike sorting starts (grey circle), a backend clustering engine starts in multiple CPUs in parallel. The groups in which the automatic clustering is finished will be immediately accessible for manual curation. (b) The group is determined prior to the recording based on the position of the electrodes. The possible software operation on the specific groups is dependent on its state indicated by colour, which is visible to the experimentalist. (c) i. Each group has five possible states: IDLE: the group waits to enter the backend clustering engine; BUSY: the group is undergoing clustering; READY: clustered and ready for manual curation; DONE: manual curation finished and model is created and downloaded into the FPGA; NONE: no usable clusters. ii. An example of the state vector of a probe shank that contains eight groups. The state vectors are visible to the experimentalist. (d) The experimentalist operates the software according to the state of groups. The BUSY group is not allowed to open; the READY group can be opened for manual curation or sent back to the backend clustering engine; The DONE group are the groups used to construct the spike transformation and classification model.

In order to implement the non-blocking architecture, 'ipyparallel'[8] is applied to implement the backend clustering engine. 'ipyparallel' framework allows building the backend engine on multiple CPUs that 'listens for requests over the network, runs code, and returns results', according to its latest official documentation. In my implementation, as soon as the recording

[8](https://ipyparallel.readthedocs.io/en/latest/)

is finished and the spike waveform extracted by the FPGA is saved in the binary file 'spk.bin', spike waveforms would be loaded to the backend clustering engine according to which group it is extracted from. Each individual group has an associated 'state', which can take five values (Fig.2.3d): 'IDLE' means the group is waiting to enter the backend clustering engine; 'BUSY' means the group is undergoing clustering; 'READY' means the group is clustered and ready for manual curation; 'DONE' means manual curation finished and model is created and downloaded into the FPGA; 'NONE' means no usable clusters are found in this group, therefore, the group should be excluded (i.e., the parameters associated with this group will not be downloaded into the FPGA) when building the spike inference model.

A convenient feature of Spiketag is that the state of all groups, which is referred to as group state vectors (Fig. 2.3c), are always visible to the experimentalist so that one can flexibly decide which group to check and interact with. If the experimentalist decides that the initial clustering of a group is not acceptable (e.g. because spike detection thresholds were set to low; or the stochastic automatic clustering result is not good; or because there are putative bad channels in the group) to start the manual curation, then he or she can reset the parameters and send this group back to the backend clustering engine and keep checking other groups that are 'READY'. In the end of the Spiketag spike sorting process, only the 'DONE' groups would be used to build the spike transformation and classification model for FPGA real-time computation.

2.2.3 Spiketag is a multi-view progressive visual analytics application

Since visual inspection and manual curation is inevitable in current spike sorting, hence it is also inevitable in building the spike inference model. One of the essential tasks of our software development is to allow the user to visualize and interact with the intermediate output from the FPGA NSP and flexibly interact with the raw and processed data. Being able to rapidly perform visual inspection and manual curation is demanded by the nature of online experiments, where we need to make sure we can use the data in a minimum amount of time. For instance, if it takes too long to check and modify the spike sorting result, the probe could have already drifted or the animal may not be performing the task optimally. In order to do that, the progressive visual analytics (PVA) paradigm was adopted when developing Spiketag.

PVA has earned growing attention in the age of big data. It places the user in the loop by providing continuous visual feedback and allows users to interactively adjust parameters to produce verifiable intermediate partial results, which in turn progressively give rise to the end results (Angelini et al., 2018; Fekete et al., 2019; Stolper et al., 2014). The PVA method is particularly fit to our software design because we will have to first configure

the parameters for denoising, spike detection, etc, then we will check whether spikes were correctly extracted by the hardware. Once the spikes are extracted from the hardware, we will perform automatic clustering and decide which groups contain good clusters to build the transformation and classification model. When building the spike transformation and classification model, we still need to visually check and manually fine-tune the sorting result group by group. Importantly, when building and transferring the model parameters from Spiketag into the FPGA, the FPGA was progressively configured to generate the intermediate output (Fig. 2.4a), which includes filtered, denoised high-channel-count multiunit activities (mua.bin)[9], spike waveforms (spk.bin) and spike features either with or without the inferred spike identities (fet_clu.bin). From the other side, users interact with Spiketag to load, visually check, interact with and analyze these data to construct and adjust parameters progressively until a satisfying spike inference model is constructed (Fig. 2.4 b).

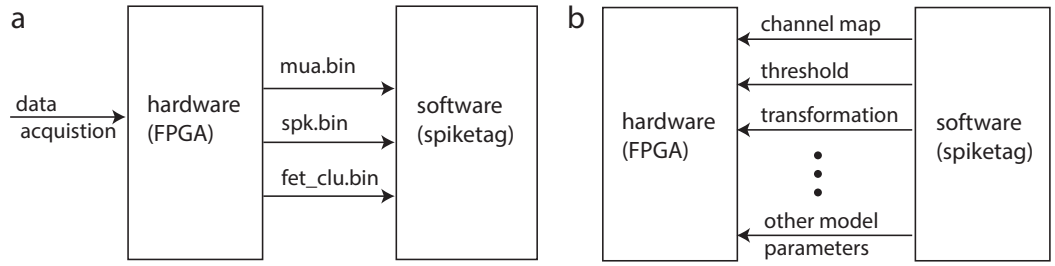


Fig. 2.4 **The intermediate output from FPGA and the model parameters from Spiketag**: (a) The FPGA hardware converts the raw data streams into three intermediate forms of binary data, which are read by the software. (b) The software allows visualization/interaction and analysis of the intermediate form of data generated by the hardware and constructs a spike inference model. The model parameters are downloaded into the FPGA to guide the real-time inference (will be introduced in Chapter 3).

Notably, in traditional spike sorting packages, the intermediate results (e.g. the filtered data, spikes, features) are computed in software after the recording. Here because the FPGA NSP computes in real-time during recording, we do not need to spend software time on the same computation again. A basic task for the software is to load and check the intermediate data output from the FPGA. Particularly we must visually check the FPGA-extracted spikes before starting to perform automatic clustering, which is necessary for our in-lab experience for many reasons. For example, an undesirable situation can occur, once in a while, such as a loose cable, bad channels, error parameter setting, a sudden drift of the probe, power interference, the animal jumps or crashes their head against the wall, etc. Because there are

[9] *.bin denotes binary file. It is the fastest data format to load and can be loaded by 'numpy.memmap' for directly 'accessing small segments of large files on disk, without reading the entire file into memory.'(https://numpy.org/doc/stable/reference/generated/numpy.memmap.html)

too many possible and yet unexpected glitches in animal experiments, visual inspection of the extracted spikes is necessary before using them to build a model. Fig. 2.5 shows an example of using the 'spiketag.view', a developed module containing many different types of views[10] in Spiketag to quickly check the intermediate output data from the FPGA.

[10]Yixin Chi, an ex senior software engineer worked in Alibaba Inc, helped coding with me on many 'spiketag.view' views where his contribution can be tracked (as other open-source software projects) in our Github repository: https://github.com/chongxi/spiketag.

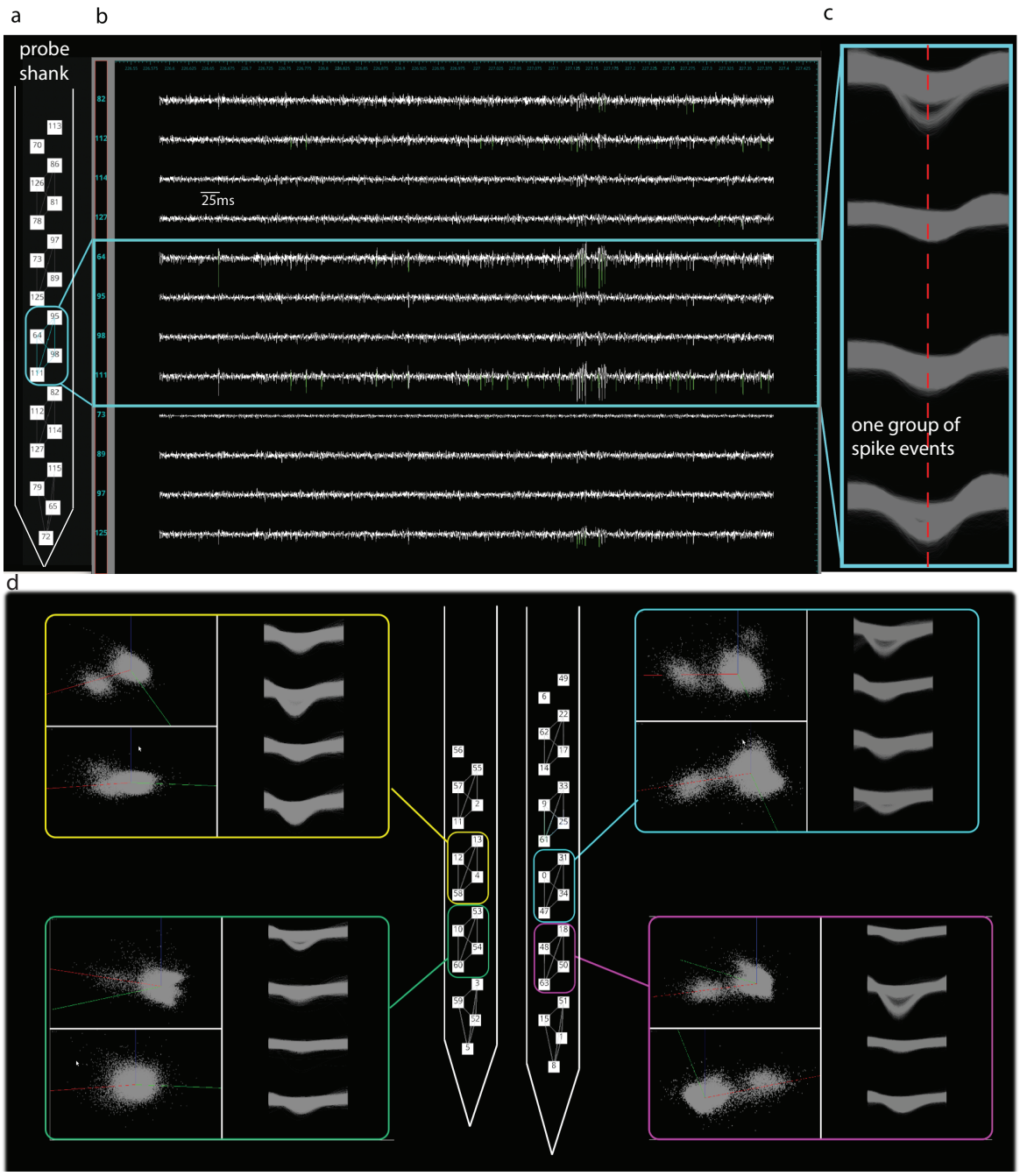


Fig. 2.5 **Using Spiketag to rapidly check the FPGA NSP-extracted spikes:** (a) A Spiketag view named probe view displays one shank of a silicon probe. The white squares denote the electrodes. (b) A Spiketag view named multi-wave view displays arbitrary waveforms from ordered channels, in this case, the multi-channel MUA with spikes (in green). (c) Spike view displays the extracted spike events from a single group. The four waveforms are from the four adjacent electrodes of that group. Deviation from unimodality is clearly seen in the first and fourth channels, indicating the presence of at least one cluster. (d) Feature view displays one or many feature spaces. One spike view can relate to several feature views with a different combination of the feature dimensions (here PCA is used).

To load and render the data as shown in Fig. 2.5, several types of data structure are required, including the data structure to represent the position of the electrodes, the multiunit activities, spike waveforms, spike features and more. Additionally, a key aim of Spiketag is that the views share the underlying states by message passing, through which multiple different views can be flexibly and loosely coupled together in-situ. The user interaction, such as key and mouse input, on a specific rendered view, should be able to modify the underlying data, and the change of the underlying data in turn should notify other views to update.

The above goals call for a clean, simple and modular design of the software architecture. In order to do this, the Model-View-Controller (MVC) design paradigm[11] was adopted, in which the 'spiketag.base' modules handling the data and model were separated from the 'spiketag.view' modules serving visualization and interaction. Specifically, 'spiketag.base' (Fig. 2.6b,c) consists of several modules to handle different types of data structure describing the channel mapping and electrode geometry (in PRB module), the multichannel multi-unit activities (in MUA module), the spike events waveform (in SPK module), the spike features (in FET module) , and the clustering results (in CLU module). Numpy was used here to represent the core data structure in PRB, MUA, SPK and CLU modules. Accordingly, 'spiketag.view' (Fig. 2.6e,f) is consists of modules to allow 2D and 3D OpenGL accelerated rendering directly with data in the instantiated PRB, MUA, SPK and CLU modules, as well as the user interaction to modify the underlying data. Our interactive views are all developed using the Vispy library (Rossant and Harris, 2013) to ensure the high performance of interactive visualization over large datasets using Python programming[12]. Particularly, the fast OpenGL rendering with Python Numpy data array structure is a prominent advantage enabled by the Vispy library.

Fig. 2.6 illustrates the architecture for the interactive visualization in Spiketag, in which the modules in 'spiketag.base' (Fig. 2.6b,c) and the modules in 'spiketag.view' (Fig. 2.6e,f) were bridged by the Vispy event system whenever the user clicks the mouse or presses keys on an instantiated Spiketag view (Fig. 2.6f). The mechanism here is that the user interaction on a Spiketag view can trigger a message (event_in in Fig. 2.6). This message can encode the action of the users: from entering a specific group to moving between groups; from the selection of a single spike to a bundle of spikes; or it can encode the action to split, merge or delete the selected data. The user actions are sequentially processed by a finite state machine

[11] MVC (Model-View-Controller) is a software architectural design pattern that encourages modular organization of software development, where the model (to process the data) and the view (to show the data) are separated.

[12] The standard visualization tools in Matlab or Matplotlib in Python are exetremely slow when it comes to interaction. Vispy has solved the exact problem of interactive visualization over large datasets by leveraging OpenGL-based rendering and an interaction event dispatch system. See more details in http://vispy.org/

in ‘spiketag.base’ system, which can cause the change of the state of the underlying data, e.g. moving to another group, selection of a bundle of spikes, splitting of one cluster, merging two clusters, deletion of some noise etc., and trigger a loopback message (event_out in Fig. 2.6) to update all other Spiketag views in use. All one needs to do is to instantiate the needed views and subscribe them to an intended message. By this mechanism, often referred to as the “publisher-subscribe” design pattern, the subscribed views are updated by events published by a common ‘spiketag.base’ object synchronously[13].

[13]Technically, the rendering update of different views does not happen at the exact computer time, but because OpenGL rendering is at milliseconds level the human eye could not tell.

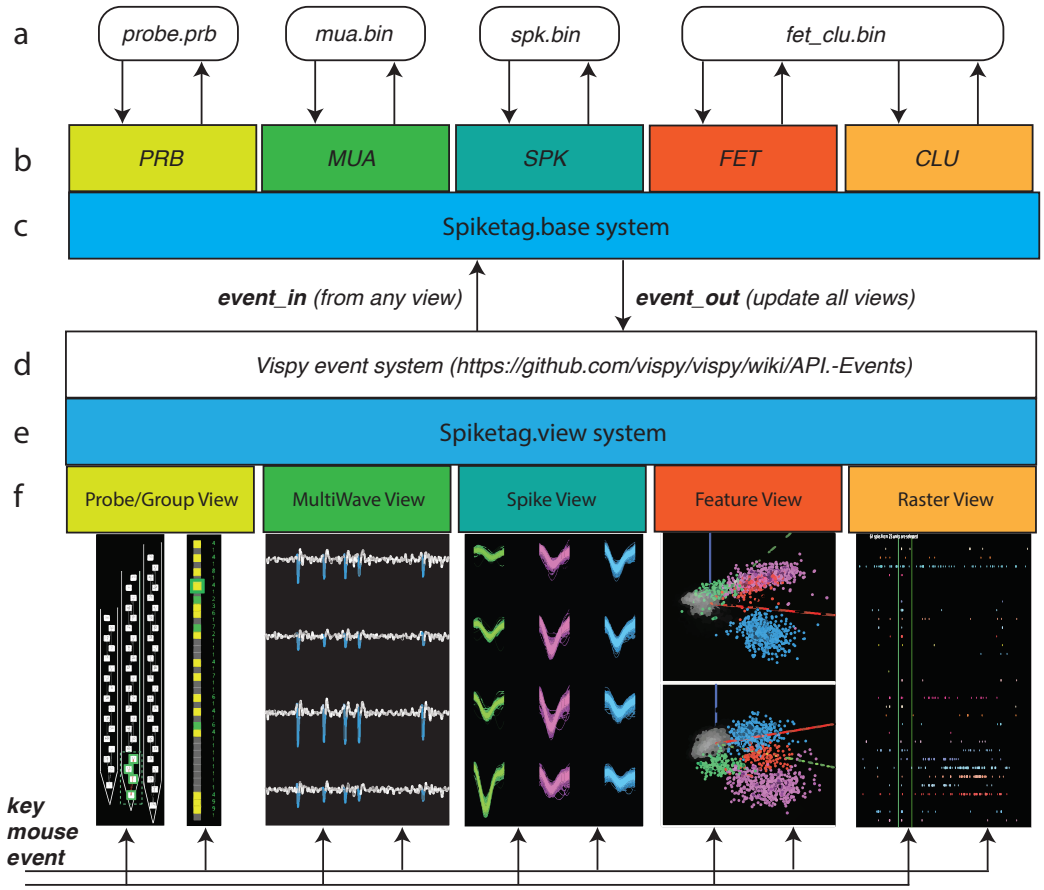


Fig. 2.6 **Multi-view interactive visualization in Spiketag**: (a) The data files the Spiketag loads, analyzes and visualizes. (b) The *spike.base* modules: PRB reads the probe file, MUA reads *mua.bin*, SPK reads *spk.bin*, FET and CLU reads $fet\_clu.bin$. (c) PRB, MUA, SPK, FET and CLU were developed as part of the *spiketag.base* system. (d) The Vispy event system was used here to couple the *spiketag.base* and *spiketag.view* objects. The *event_out* message is sent out to all the subscribed views to update them synchronously. (e) *spiketag.view* includes all the interactive views. (f) some examples of the interactive views. All views can receive mouse and key input and get updated through the event loop (i.e., *event_in*, *spiketag.base*, *event_out*).

We found the multi-view interactive visualization is particularly useful in manual curation. For example, one can easily use mouse dragging to select a bundle of spikes from either spike view or feature view and observe the highlighting of the selected spikes in all other views. For example, being able to select a bundle of spikes in one channel allows one to observe the selected spike waveforms in all other channels, as well as the feature of the selected spikes in the feature space by highlighting with brightness or color. These functions are not possible in a conventional static visualization. Also, one can rotate a 3D feature space and further

examine and fine-tune the selected spikes and type ‘s’ to split a new cluster or type ‘m’ (or left click mouse) to merge to another cluster that the mouse is floating over.

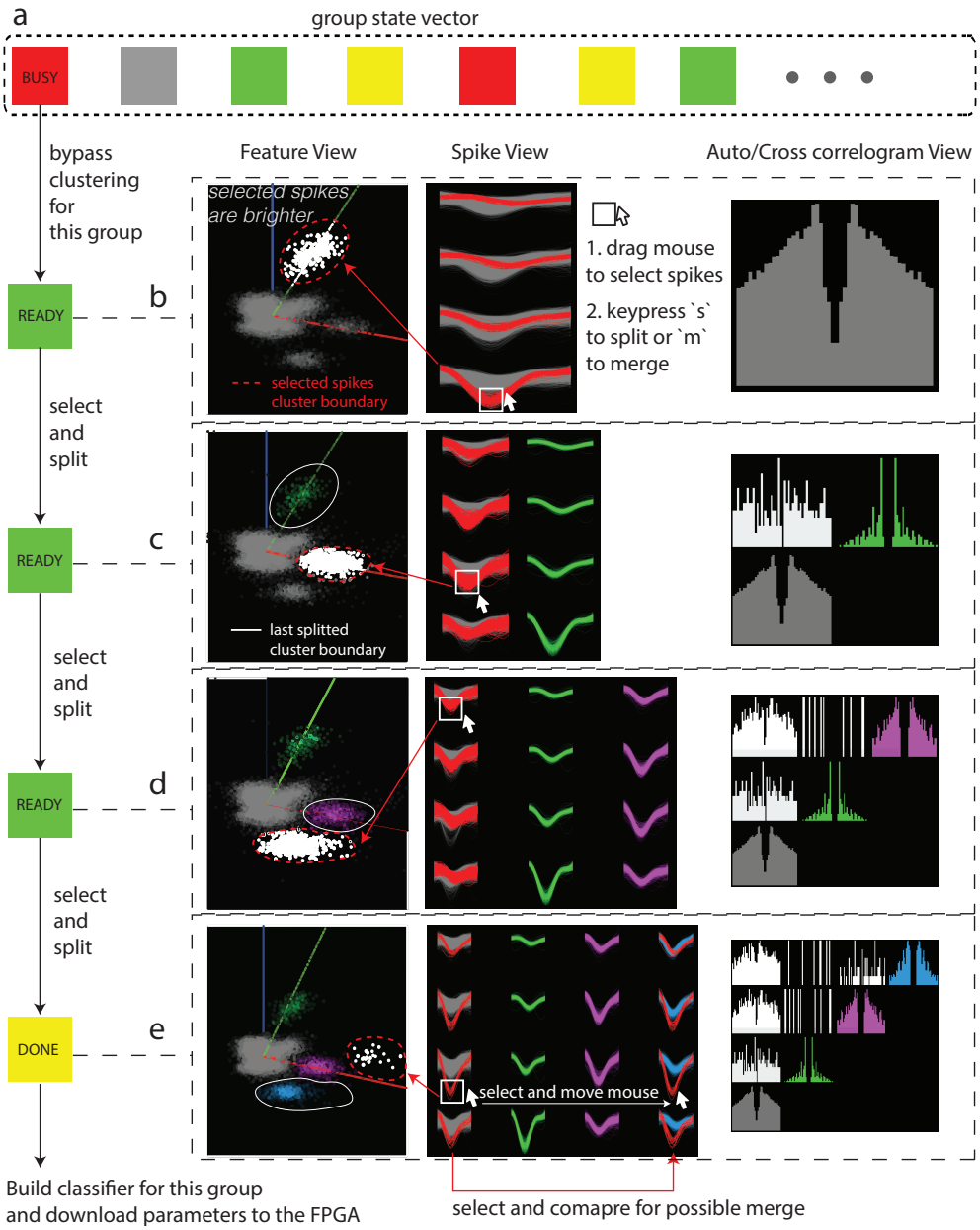


Fig. 2.7 Multi-view interaction in Spiketag to select, split and merge clusters in a group: (a) To illustrate pure manual clustering from the beginning, we entered a group that was still in the BUSY (in red) state and stopped it from automatic clustering by changing its state to READY (in green). By doing this we bypassed the automatic clustering and started manual clustering. (b) A bundle of spikes was selected in the Spike view by dragging the mouse over the waveforms (the white box with a mouse cursor icon). The size of the white box represents the range of the mouse dragging. The spikes were selected at the fourth electrode in this group, but the spike waveforms were highlighted on all electrodes (in red). The selected bundle of spikes was also highlighted in the feature space (left of the spike view) without lag (in White). Then we pressed 's' to split the selected bundle of spikes into a new cluster (green cluster in c). The auto/cross correlogram view was also get updated by the splitting. (c) A new bundle of spikes was selected from the Spike view. This time spikes were selected by dragging the mouse over a region around the third electrode. Simultaneously, the selected spikes in other channels and in the feature spaces were both highlighted. We then 3D rotated back and forth the feature space; the selected feature vectors (in White) look like a multivariate Gaussian and were well separated from other clusters. 's' was pressed to split the cluster again. We then got a new cluster (Purple in d). The simultaneously updated auto correlogram view appears to be reasonable, showing no violations of the refractory period (Purple). (d) The same procedure was repeated again, this time interacting with the first electrode. (e) In the end, one bundle of spikes was left (in red). By first selecting this bundle of spikes and then moving the mouse to other clusters without clicking, we can visually compare the spike shape between any cluster and the selected bundle of spikes. Also, from the feature space, we can see the position of the selected feature vectors (in White) related to other clusters. The experimentalist who performs the experiment has the ultimate flexibility to decide which cluster any selected spikes belong to. The minimum number of selected spikes can be as low as a single spike, meaning we can move a single spike from one cluster to another (although it is not efficient to do that). In this illustration, we simply show the visual effect when mouse moves to the blue cluster. Upon finishing the clustering, we set the state of this group to DONE (yellow square). Spiketag automatically builds the transformation and classification model on those groups tagged as DONE. The process from (a) to (e) took a few tens of seconds.

Fig. 2.7 illustrates an example[14] of pure manual clustering to demonstrate step by step how the multi-view interactions lead to a satisfying clustering result in a single group. In order to interactively go through all groups in the recording, we need to put more views together. A useful feature of Spiketag is that we can flexibly put the views needed for the experiment into a GUI using the Qt package without rewriting much code, since the event driven updating is implemented in 'spiketag.base' and 'spiketag.view' as illustrated in the Fig. 2.6. Fig. 2.8 shows two examples of the composed GUIs we often used. A GUI is essentially a composition of interactive 'spiketag.view' modules. Other users might find a different efficient composition of views given a different dataset.

[14]Dr. Brian Lustig and Dr. Shinsuke Tanaka in the Lee Lab (Janelia) collected this data from a chronically implanted silicon probe in the rat hippocampus. Only one group is shown here for this illustration.

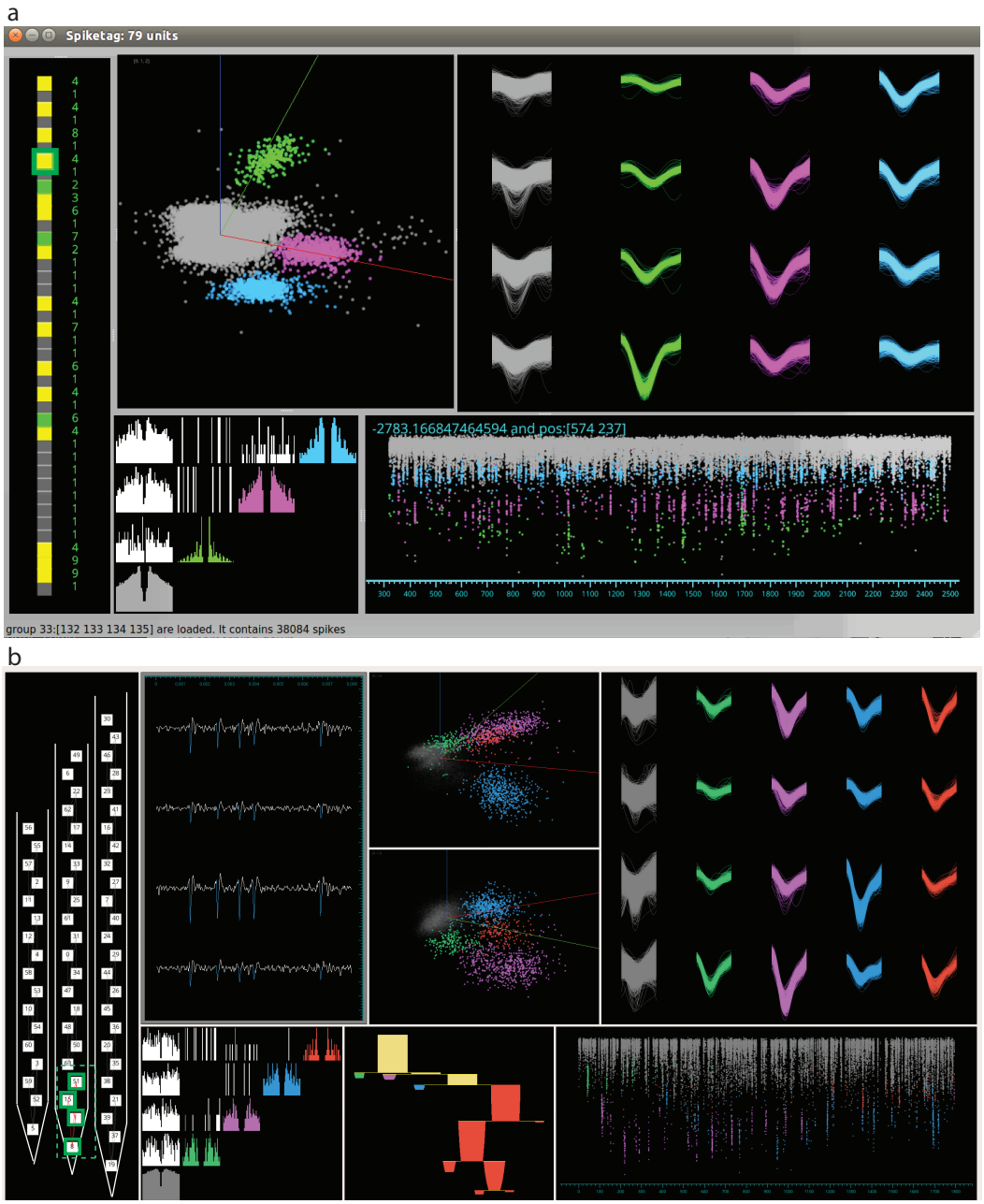


Fig. 2.8 **Flexible Spiketag GUI through arbitrary combination of interactive views:** Two examples of GUIs by flexibly selecting the interactive views. The probe/group view in the leftmost side of both GUIs allows moving from one group to another by either clicking the mouse on the targeted group or using a customizable keypress shortcut. The color of the group denotes its state. (a) The GUI example used to produce the example in Fig 2.7. The vector of the squares is the group state vector. In the lower right panel is the amplitude view, which plots the largest amplitude peak of all four spike waveforms as a function of time. (b) Another example of a possible GUI by simply adding more views. The view between the cross-correlogram and the amplitude view is an interactive visualization of a condensed tree structure used in an automatic clustering algorithm to be introduced in the next section.

A core aim of Spiketag is to maximize flexibility to allow the experimentalist to compose the interactive views they need and the action sequences they would like to use to produce a spike inference model that they think is good enough for their specific experiments.

2.2.4 Spike feature clustering

Pure manual sorting on a group, as shown in Fig. 2.7 is ineffective. Normally one would only enter groups which already have an initial automatic clustering result. Therefore, deploying a clustering algorithm is required in Spiketag. Building a clustering algorithm, or any other unsupervised learning algorithm to separate clusters of spikes, used to be the core of spike sorting. Interestingly, many of the spike sorting algorithms were developed by neuroscientists who work outside the machine learning field. That might be because of spike sorting is quite an old field, while the development of the Python machine learning open-source libraries is quite a recent advancement.

In this context, the development of Spiketag used a quite opposite approach. Instead of developing a new clustering algorithm, we simply allow invoking any developed clustering algorithm in the standard Scikit-learn library[15] or other Python open-sourced clustering algorithms. Because the ground-truth data in the neuroscientific community is still lacking, a quantitative comparison is hard to conduct.

We empirically found in our practice that no clustering algorithm can be used in a fully automatic manner without manual curation. All of them made apparent visually detectable mistakes here and there that can be corrected and fine-tuned using the interactive strategy described in previous sections.

However, the algorithms that provide additional low-dimensional (<3D) information representing the cluster hierarchy of the data stands out when using Spiketag, because the cluster structure information can be exploited in Spiketag for interaction. Given any matrix $X \in \mathbb{R}^{n \times p}$ where n is the number of points and p is the number of features, a space of fewer than three dimensions is produced where the structure of the data can be clearly visualized. In the structure space, the clusters can be easily selected, split and merged, an exciting result of which is the clustering algorithm itself became interactive.

One algorithm of this kind is Density Peak Search (DPS) (Rodriguez and Laio, 2014), which transforms the data into a two dimensional structure-space where the x-axis is referred to as the density ρ of point i by counting the number of points j within distance d_c; the more points nearby the higher the density is:

[15]The clustering package in the scikit-learn machine learning library: https://scikit-learn.org/stable/modules/clustering.html and other clustering algorithms in scikit-learn extension: https://github.com/scikit-learn-contrib

$$\rho_i = \sum_j \chi\left(d_{ij} - d_c\right),$$

$$\chi(x) = 1 \text{ if } x < 0 \text{ else } \chi(x) = 0$$

The y-axis of the structure-space is the minimum distance between the point i and any other point j with higher density, defined as δ:

$$\delta_i = \min_{j:\rho_i<\rho_j}\left(d_{ij}\right)$$

For the highest density point, for which there is only one highest density point for the whole data, δ is the distance to its furthest point.

For each cluster, there exists one and only one density peak where both the ρ and δ are high. Because ρ is represented in the x-axis and δ is represented in the y-axis, the density peak is easily visualized (Fig. 2.9e,g). The assumption made here is the density peak uniquely represents a cluster around it. Fig. 2.9g shows the actual position of the detected density peak in the feature space. Once the positions of the density peaks are known, the number of clusters and the feature points belonging to each cluster are easily decided by the nearest neighbor with higher density (Fig. 2.9j).

The second clustering algorithm I found useful in providing a structure space is Hierarchical Density-based spatial clustering of applications with noise (HDBSCAN). HDBSCAN (Campello et al., 2015) is a variant of the well-known DBSCAN. Specifically, it brings hierarchical clustering into the density-based clustering algorithm DBSCAN. As in DPS, HDBSCAN also produces a matrix encoding the pair-wise distance among points at the first step, but the distance is calculated using a mutual reachability metric. Mutual reachability distance is better than Euclidian distance between two given points in the sense that it makes sure that the sparse points will have larger distance from any other points by:

$$d_{\text{reach}-k}(i,j) = \max\left(\text{core}_k(i), \text{core}_k(j), d(i,j)\right)$$

where the $core_k$ for any given point i is the distance to its kth nearest neighbor, and $d(i,j)$ is the Euclidean distance from point i to point j. With a pair-wise distance matrix, a minimum spanning tree is built using Prim's algorithm, as shown in Fig. 2.9b. The minimum spanning tree is then used to generate a single-linkage dendrogram, which is a diagram representing the hierarchy of the cluster tree. In a conventional dendrogram, a distance threshold d_{cut} is used to cut parent clusters into child clusters. With the dendrogram comes the key concept in HDBSCAN: the minimum cluster size. The idea is that if the number of points to be split

into a new cluster is less than the minimum cluster size, we consider that the points 'fall off from its persistent cluster rather than split'. Armed with this concept, a condensed tree can be generated from a conventional dendrogram where the y-axis of the condensed tree is defined as:

$$\lambda = \frac{1}{\mathrm{d}_{cut}}$$

Since λ is inversely proportional to d_{cut}, the larger the λ is, the more splits will happen. As we increase λ from zero, the whole dataset will first split into two, and then more clusters would appear in the condensed tree (Fig. 2.9d) as λ grows. By analogy, λ behaves as time, the clusters are born and vanish in λ. Any given cluster in the condensed tree, since its birth, keeps losing data points until splitting or vanishing. No data point is left when the cluster vanishes (death). For a cluster to be a real cluster, it has to be persistent enough from its birth to its death under such settings. Therefore, the $stability$ is defined to measure the persistence of a cluster, from their birth to death:

$$\text{stability}(C) = \sum_{p \in C} \lambda_p - \lambda_{birth}$$

Where λ_p is when a data point p falls out the cluster and λ_{birth} is when the cluster is born. The stability of a cluster above a certain threshold is automatically decided as a persistent cluster by HDBSCAN. Only the persistent clusters would appear in the automatic clustering result. Nevertheless, the stability threshold cannot be the same for all datasets. Thus the result often needs to be corrected.

We do not need to solve this problem algorithmically. In Spiketag, the condensed-tree was developed into an interactive view (Fig. 2.8b). It encodes the information about the cluster structure of the data; each leaf of the condensed tree represents a cluster (Fig. 2.9f circled by ellipses). The length of the leaf represents how long it lives without splitting. With interaction, we can select any node by simply moving the mouse there. Because the data is represented by a tree structure, the data points belonging to a specific leaf or node can be hashed. Particularly, the cluster membership can be precisely tuned by first selecting a specific node in the tree and observing the highlighted spike waveforms and points in the feature space, where the manual fine-tuning can be conducted.

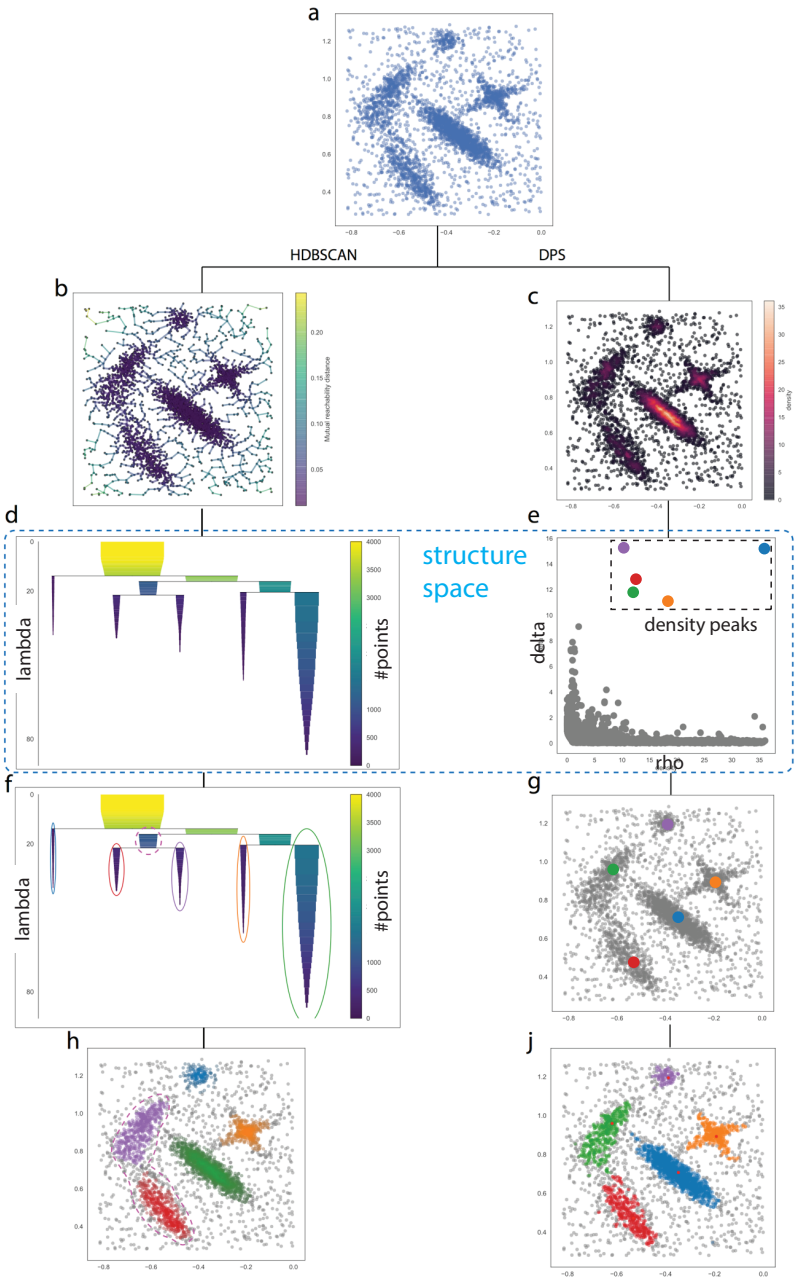


Fig. 2.9 **Clustering with structure information**: (a) The data used to illustrate DPS and HDBSCAN clustering. (b) HDBSCAN constructs a minimum spanning tree. (c) DPS constructs a density map. (d) HDBSCAN converts the minimum spanning tree into a condensed tree where each leaf is a cluster. (e) DPS converts the density map into a 2D plot where density peaks of candidate clusters can be seen. (f) The leaf of the condensed tree is the cluster in (h) with the same color. (g) The position of density peaks of each cluster. (h) HDBSCAN clustering result. (j) DPS clustering result.

Of note, the discussion of these two algorithms does not imply they were used exclusively in Spiketag. Other conventional algorithms such as Dirichlet Process Gaussian Mixture are also often used in practice when the clusters in the feature space are clearly Gaussian[16]. Users can choose an arbitrary clustering algorithm in the "scikit-learn" machine learning library. With a bit of tinkering, the clustering algorithms from other libraries can also be used. However, if the data is noisy and more interactive options are needed, algorithms such as DPS and HDBSCAN provide additional structural information of the data and thus additional convenience for manual curation.

2.3 Software API to communicate with the FPGA via PCIe

Once the clustering and manual curation are finished, a full spike inference model can be built (Fig. 2.2). Once the model is configured in the hardware, the real-time processed data is automatically output from the hardware. The model parameters include the reference channel number for noise reduction, the channel-wise threshold for spike detection, channel grouping map for spike event extraction, PCA feature transformation and a feature classification model for each group[17]. In order to download these model parameters into the FPGA, a set of Python APIs was developed to read/write to/from the memory of the FPGA, which is referred to as the memory-interface API. In order to read out the real-time processed data streams from the FPGA, another set of Python API calls was developed to support reading out both the continuous data stream and data packets from the FPGA, which is referred to as the data-interface API.

The memory-interface API communicates with the FPGA Random Access Memory (RAM), which contains discrete memory locations that are accessible by address (often abbreviated as addr or ADDR). In contrast, the data-interface API communicates with the FPGA First-In-First-Out (FIFO), which has no memory address and is used for transmitting the real-time processed data. To ensure the low-latency communication between the host application and the FPGA with low development cost, both memory-interface and data-interface API were implemented using the Xillybus PCIe framework in Linux (Preußer and Spallek, 2014; Xillybus, 2010). PCIe is a high-bandwidth low-latency interface and is widely used for low-latency communication between FPGA, GPU and other specific hardware-based applications, but it is also quite complicated if the developer needs to handle the protocol of

[16] https://scikit-learn.org/stable/modules/mixture.html

[17] Although these hardware related concepts are briefly mentioned here for their relevance in this section, the detail of (1) The method to construct the model parameters, (2) the hardware-implemented algorithms in the FPGA and (3) the actual hardware device such as RAM and FIFO to communicate with the applications in the PC will be introduced in Chapter 3.

the data transmission. Xillybus is "a straightforward and intuitive PCIe based solution for data transport between an FPGA and a host" (Preußer and Spallek, 2014; Xillybus, 2010) that provides a framework in which all the low-level detail of PCIe data transmission is hidden. The software developer who uses the Xillybus PCIe solution sees the specific actual hardware devices as standard Linux Input/Output (I/O) devices, through which "the user application on the host performs plain file I/O operations" (Preußer and Spallek, 2014; Xillybus, 2010).

In 'spiketag.fpga' modules, the "plain file I/O operations" in the Xillybus PCIe framework were further encapsulated into even simpler Python APIs. In order to make the Python API consistent with hardware development, the names of the customized functions are specifically associated with the names of RAMs or FIFOs in the FPGA. For example, one among many RAMs developed in the FPGA was named as 'mem_16'; the Python API functions associated with this RAM were named as 'write_mem_16' and 'read_mem_16', where the 'write_mem_16' function was used to write to FPGA 'mem_16', and the 'read_mem_16' function was used to read from FPGA 'mem_16'. A straightforward example of interfacing the RAM 'mem_16' and the test of the writing/reading operations are illustrated in Fig. 2.10.

Notably, hardware APIs developers need to pay additional attention to the bit-width of the specific targeted hardware device, which can also be read from the name of the developed API. For example, 'read_mem_16' indicates that the 'mem_16' RAM stores 16-bit values in each memory location. Furthermore, I use fixed-point arithmetic in the FPGA development. This means the API developer has to know how many bits are used to represent the decimal part in the specific RAM or FIFO and convert the parameters/values into the fixed point binary number. These are important details for the developers who want to build their own FPGA APIs. However, the developers who aim to build on top of Spiketag can simply use the developed APIs without attending to these details.

In order to hide these hardware-related technical details from experimentalists and make their work convenient, several highest-level APIs were developed. Here are some examples, in the 'spiketag.model' module, the 'model.compile()' function was developed to configure all the model parameters into the FPGA RAMs and is followed by a checking procedure which would return a warning or error if any parameter is missing; in the 'spiketag.realtime.bmi' module, the 'bmi.start()' function was developed to assign a dedicated CPU process to receive the spike trains from the FPGA. The real-time acquired spike trains via PCIe are by default stored in the 'fet_clu.bin' file and also can be used as input to user-customized decoders. These highest-level APIs were developed to allow experimentalists to write scripts for their specific experiment.

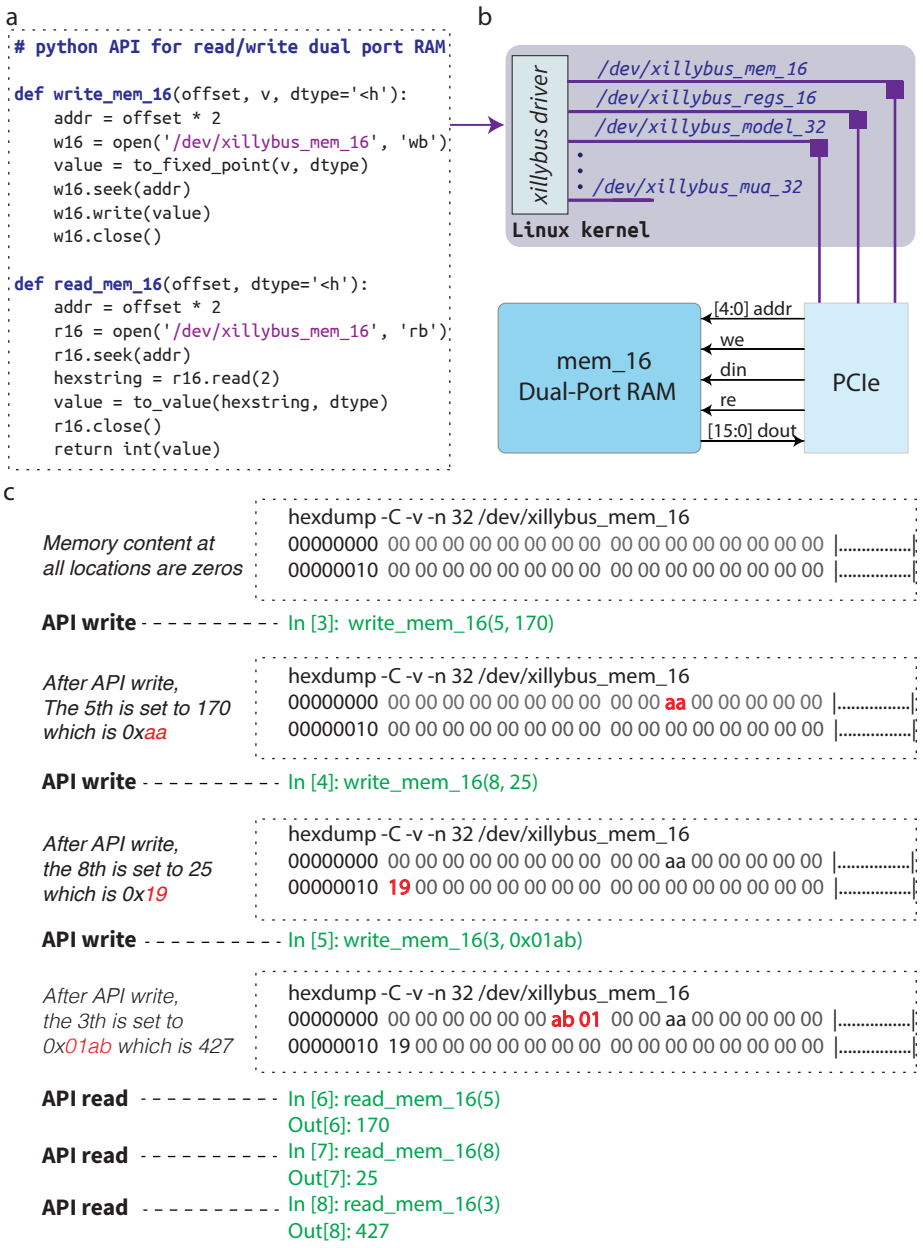


Fig. 2.10 **Python memory-interface API: write/read the model parameters to/from FPGA RAMs**: (a) Sample code of the Python API to write to ('write_mem_16' function) and read from ('read_mem_16' function) a specific memory (RAM) in the FPGA. (b) The Python API uses the Xillybus PCIe driver (purple) to write to and read from a Dual-Port RAM in the FPGA. Each device (e.g., '/dev/xillybus_mem_16') listed in the Linux kernel is associated with an actual FPGA RAM (e.g., 'mem_16'). This RAM ('mem_16') has a 5-bit address (addr) to store 32 16-bit values, which is reflected in the corresponding Python API. (c) An example of writing and reading a RAM ('mem_16') using the Python API. A small Linux utility program 'hexdump' is used to show all of the 32 locations of the RAM ('mem_16'). In the beginning, the values of all locations of this RAM are zeros. After value 170 is written to the location 5 using 'write_mem_16' Python command, 'hexdump' shows that the memory content at location 5 becomes 0*xaa*. The following examples of operations use the same mechanism. The parameter written into the RAM can be read out using the 'read_mem_16' Python function.

To conclude, a Python-based fast multi-view progressive visual analytics application named Spiketag has been developed. Spiketag can be used to generate a spike-sorted model for FPGA-based real-time spike inference, using a process that highly resembles spike sorting with enhanced 3D multi-views data visualization and interaction. In this chapter, I have described the architecture of Spiketag, the underlying libraries it was based on, the method Spiketag uses to perform the backend group-based automatic clustering and non-blocking manual curation[18]. The non-blocking software architecture bypass the waiting period of the automatic spike sorting and is particularly useful for online experiments in which saving time for building a model is important. I have also introduced the API design for PCIe-based FPGA communication. Nevertheless, I have not introduced how the spike inference model is built, for example, how the spike classification model is built based on the clustering result. Since the algorithms using the model parameters to perform the real-time processing are built in the hardware, the algorithms to compute the model parameters, although implemented in Spiketag, are also highly constrained by the hardware. For that reason, both types of algorithms will be introduced in the next chapter.

[18]This makes it different from all other existing sorting packages. While other spike sorting packages aim to minimize the automatic sorting time, Spiketag multiplexes the automatic sorting and manual curation on different electrode groups. When users manually curate one group, the automatic sorting engine works on other groups in the background and delivers the ready signal when finished for each group. In practice, the user's manual curation is always slower than the automatic sorting. Therefore the user never waits for the automatic sorting to return.

Chapter 3

The FPGA-based Neural Signal Processor (NSP)

In the previous chapter, I introduced spike sorting and a newly developed stand-alone software package: Spiketag. Spiketag is designed to support fast, 3D interactive manual curation with simultaneous automatic group-based spike sorting running in the background. It allows the experimenter to complete high-channel-count spike sorting with some recorded data, then utilize the resulting spike-sorted model in the rest of the experiment. The model parameters computed on the PC are then downloaded to the FPGA on-chip memory. The FPGA-based NSP circuits then access these model parameters to perform the real-time data processing. The FPGA implementation of the NSP signal processing pipeline ensures the low latency (<1 ms) response to single spikes of a population of single neurons. In addition, the FPGA-based NSP can simultaneously respond to a few tens or a few hundreds of neurons, depending on how many neurons appear in the spike-sorted model.

The first design goal for the NSP is the flexibility to set its model parameters. The spike-sorted model parameters vary from animal to animal, experiment to experiment, task to task, even session to session. Therefore it is important to be able to flexibly update part or all of the parameters when it is required. The second design goal is the accurate signal processing for the spike inference. The real-time spike inference accuracy should be limited only by the spike sorting accuracy but no other factors. Being able to infer single-units identity using FPGA allows the real-time detection of hypothesized population temporal codes (see chapter 1). The third design goal is the low-latency response capacity. The less than 1 ms latency means the user would know which neuron fires before the spike arrives at any remote downstream region (as the axonal conduction delay across brain regions is usually greater than 1 ms). The final design goal is the high cell-count capacity. Being able to record from more channels and simultaneously identify spikes from more single-units gives

rise to a higher information rate from the brain to the machine, enabling high-bandwidth BMI decoder.

The current version of this NSP was implemented to contain a 160-channel acquisition system and a chain of signal processing circuits inside the FPGA for real-time spike inference on a population of neurons. For example, assuming after the sorting phase, we find 100 neurons from 160 channels of recording, and this gives us a spike-sorted model to transform the 160-channel raw data into the spike identity from 0 to 99. After downloading the model parameters into the NSP, during data acquisition, the NSP automatically reports which neurons fire within the last 1 ms when any neuron among these 100 neurons fires. The real-time identified spikes are converted into TTL pulses to trigger a feedback (e.g. for low-latency optogenetic perturbation). Meanwhile, the real-time produced spike trains from the NSP are sent to the computer for population decoding (e.g. Bayesian population decoding of animal's spatial location).

This chapter describes the design, development, and implementation of the FPGA-based NSP (or referred as to the FPGA NSP).

3.1 Field-programmable gate arrays (FPGAs)

FPGAs are a type of integrated circuit (IC) that can be hardware-programmed for specific algorithms. FPGA's are generally regarded as a programmable algorithmic IC. Just as its name conveys, it has a hardware architecture that supports programming of very large array of logic gates and input/outputs (IO)s. Since the first commercial FPGA invented by the founders of Xilinx in 1984, the modern FPGA (e.g. the Xilinx 7-series FPGA) has gone beyond just having a huge number of logic gates and IOs to include high-speed serial Input/Output (HSSIO), clock management tiles (CMTs), digital signal processors (DSPs), block-random-access-memory (BRAM), and even microprocessors such as ARM processors.

Among all physical elements, the Configurable Logical Block (CLB) is the most homogeneously repeating physical unit in an FPGA (Trimberger, 2015). CLBs are to an FPGA what neurons are to a brain. CLBs are the main logic resources for implementing combinatorial and sequential circuits[1]. In the Xilinx 7-series FPGA, each CLB element is connected to a switch matrix for access to the general routing matrix. A CLB element contains a pair of

[1]Combinatorial and sequential circuits are two types of digital circuits. Combinatorial circuits map the input to the output in a time independent manner, meaning no clock signal is required. Sequential circuits take a sequence of input and generate a sequence of output, the rate of input and output is dependent on the input/output digital clock. The combination of combinatorial and sequential circuits are often used to design finite state machine. The combination of finite state machine and memory component is equivalent to the turing machine and often used to solve arbitrary algorithmic problem in the form of digital circuits. These basic digital design principles apply generally and go beyond FPGA design.

slices. Each slice contains “four look-up tables, eight flip flop/latches as welll as multiplexers” (Xilinx, 2016).

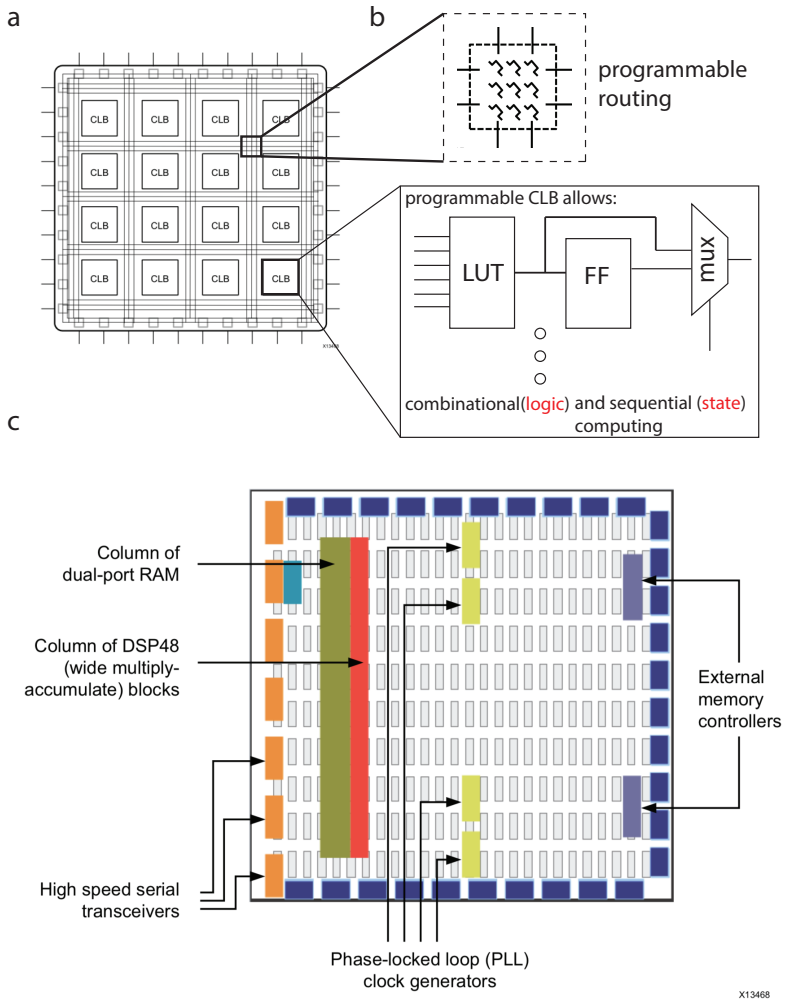


Fig. 3.1 **What is inside an FPGA (Adapted from Xilinx Document X13468 UG998)**: (a) The CLB is a repeated building block of an FPGA. The routing (switch) matrix is used to connect physical elements among different CLBs (which are programmable). Combinatorial and sequential circuits built on top of specific routing are responsible for the specific logic and state dependent computing. (b) Zoom in view of the switch matrix (upper block) that connects CLBs and the inside of an individual CLB (lower block) which contains several types of basic physical devices to enable logic and sequential computation. (c) The modern FPGA heterogeneous structure includes not only CLBs but also various on-chip components: Dual-Port RAM, DSPs and high-speed serial transceivers and large numbers of IOs, etc. (General purpose IOs are not shown here since a modern FPGA usually can have 500 IO pins.)

Among all physical elements inside an FPGA, the below are heavily used in this project:

1. Look-up table (LUT): This element is a collection of memory cells connected to a set of multiplexers which can either perform logic operations or act as distributive memory. The input bits instantaneously map into the output bits, which allows the combinational logic that does not require states.

2. Flip-Flop (FF): This register element stores the last input and releases it in the next clock cycle. These FFs and LUTs together enable sequential logic and state machines, of which the outputs are not only dependent on current input but also the sequence of the past inputs (state).

3. Wires: These wires route physical elements to one another via the switch matrix, which sits at the core of FPGA programming.

4. DSP: An arithmetic logic unit (ALU) embedded into the fabric of the FPGA that supports fast multiply–accumulate (MAC) operations.

5. First in First out (FIFOs): A FIFO can be thought of a one-way tunnel that data can flow through. It is often required when the data moves across two circuits running at different clock speeds[2].

6. Embedded Block Random Access Memories (RAM): The RAM and especially the Dual-Port[3] RAM allows model parameter update and access.

7. Phase-locked loops (PLLs): PLLs drive the FPGA fabric at different clock rates to support slow sampling and fast computation operating simultaneously. Used as clock generators.

8. High-speed serial transceivers: Allows Peripheral Component Interconnect express (PCIe) low-latency and high-bandwidth transmission.

9. Input/Output (IO) Blocks: A large number of programmable IOs (often > 300) allows scalable input into a single FPGA and flexible output extension.

To program an FPGA is to program the switch matrix to connect the physical computational resources into a functioning digital circuit. The state of every element evolves according to the clock cycles, their previous states and their inputs. The computation in the FPGA digital circuits is intrinsically distributed and parallel, which gives rise to very large computational bandwidth. The fact that all of the on-chip memory is accessible at every

[2]This is known as the clock domain crossing (CDC) problem

[3]Dual-Port RAM means these RAMs can be written by one device and read by another device.

clock cycle endows an FPGA with incomparable memory bandwidth versus the modern PC. PC computation is centralized. For a PC to compute, you need to move data between memory and the CPU (Central processing unit). There is a bus between the CPU and memory facilitating the data movement. However, you can only move one piece of data per one clock cycle on the bus which constrains the memory bandwidth. In an FPGA, there are no such constraints (no bus, no centralized computation; physical computational elements are everywhere for to be connected) (Fig. 3.1b). You can move all the data you need during one or few clock cycles from the FPGA on-chip memory to one or many distributed locations, where the computation is performed. This is the big picture of why an FPGA solution is usually much faster than any general-purpose processor, even though the clock rate is lower. The only currently available device that could possibly be faster than an FPGA is an ASIC (application specific IC), which is also intrinsically parallel and distributed but can run with higher clock rate. However, an ASIC cannot be generally programmed.

Using an FPGA to build the NSP has numerous advantages compared to the computer-based NSP. First, Programmable IO allows scalable high-channel-count real-time online electrophysiology data input. A standard FPGA chip normally has more than 500 IO pins. Second, the intrinsic parallel and pipeline capacity of an FPGA allows computation and acquisition to happen on the same chip at the same time. During the time interval between data samples entering the FPGA, the computation is performed on the previous samples. Third, the vast internal memory bandwidth allows bursts of intensive computation when needed. There is no need to move data between the memory and computational units because the computation is carried out in a distributed manner. Fourth, the FPGA solution computes with a much faster rate (few nanoseconds) than the data acquisition rate (tens of microseconds). This allows the spike inference to be finished as soon as the last sample of the spike arrives. Moreover, bidirectional FPGA On-chip memory, especially the dual-port RAM, allows the separation of algorithmic circuit modules and the model parameters the algorithms need. Finally, the dedicated FPGA-PCIe-PC interface allows high throughput and low-latency communication between an FPGA and a PC.

These advantages become even more evident when the data is recorded from a class-2 recording device such as tetrodes and silicon probes (described in Chapter 1, also see Fig. 3.2a), where the operations needed for real-time spike inference are more complicated[4]. In the next section, I will describe the design and development of the FPGA-based NSP.

[4]As described in the first chapter, class-2 recording devices such as tetrode arrays and silicon probes contain groups of low inter-electrode distance electrodes. Spike inference relies on the combinational waveforms recorded from the adjacent electrodes in each group. In contrast, spike inference on class-1 recording devices such as a microwire array is much simpler because it does not require grouping of waveforms from adjacent electrodes.

3.2 Overview of the FPGA-based NSP

3.2.1 Overview and key concepts

The FPGA-based NSP performs real-time spike inference on the raw data recorded from tetrodes or silicon probes (Fig. 3.2a). Fig. 3.2 illustrates the process of sampling eight channels of data from two groups of electrodes, which are predetermined by the experimenter[5]. The overall system includes two parts: an FPGA and a PC. The FPGA conducts data acquisition and applies a spike-sorted model to perform the real-time spike inference; the model is generated on the PC. Three signal processing stages were designed (Fig. 3.2c) to execute the BMI or the low-latency neural closed-loop feedback experiment. In the beginning, the FPGA only conducts data acquisition and data pre-processing (filtering and removing motion noise). This stage can last from tens of minutes to hours to acquire a sortable dataset. After that, Spiketag software on the PC then uses the intermediate datasets to generate a spike-sorted model and download it to the FPGA. The model generation step can take a few tens of minutes. Once the model is in the FPGA, the algorithmic circuits inside the FPGA start to compute, based on the model, to assign the spike identity (i.e., which unit the spikes comes from) in real-time. For different implanted animals or different experiments on different days, a new model is always required.

The data enters into the FPGA sample by sample (Fig. 3.2c). The samples from all the channels at a given acquisition cycle[6] forms a data frame (Fig. 3.2a). The neural signal processing is performed by the algorithmic modules inside the FPGA, from multi-channel filtering, denoising, spike detection, spike waveform grouping, spike feature extraction to spike classification. This end-to-end[7] pipeline takes samples from raw data at the acquisition end and produces the filtered data, the extracted spike waveforms and the feature vectors along with the spike identities at the output-end. The output goes to the PC through a low-latency PCIe interface (Fig. 3.2c). Three signal processing pathways, using different PCIe channels, were adopted to allow communication between the FPGA and the PC (Fig. 3.2c: number in the circles from one to three). The first pathway moves the pre-processed (intermediate) data from the FPGA to the PC for spike-sorting, the second pathway conveys the model parameters from the PC to the FPGA to configure the model inside the FPGA. The

[5]Current design of the FPGA-based NSP supports acquisition from 160 channels (i.e. 40 groups in which each individual group contains four electrodes).

[6]The time interval between data frames is defined as the sampling interval, the reciprocal of the sampling interval is commonly known as the sampling rate (Hz).

[7]end-to-end is an engineering term referring to a system that contains many modules that are chained together, from input end to the output end, to provide a complete solution to a problem.

third pathway transmits the real-time spike identities (id) along with their timestamps, i.e., the spike trains, to the PC.

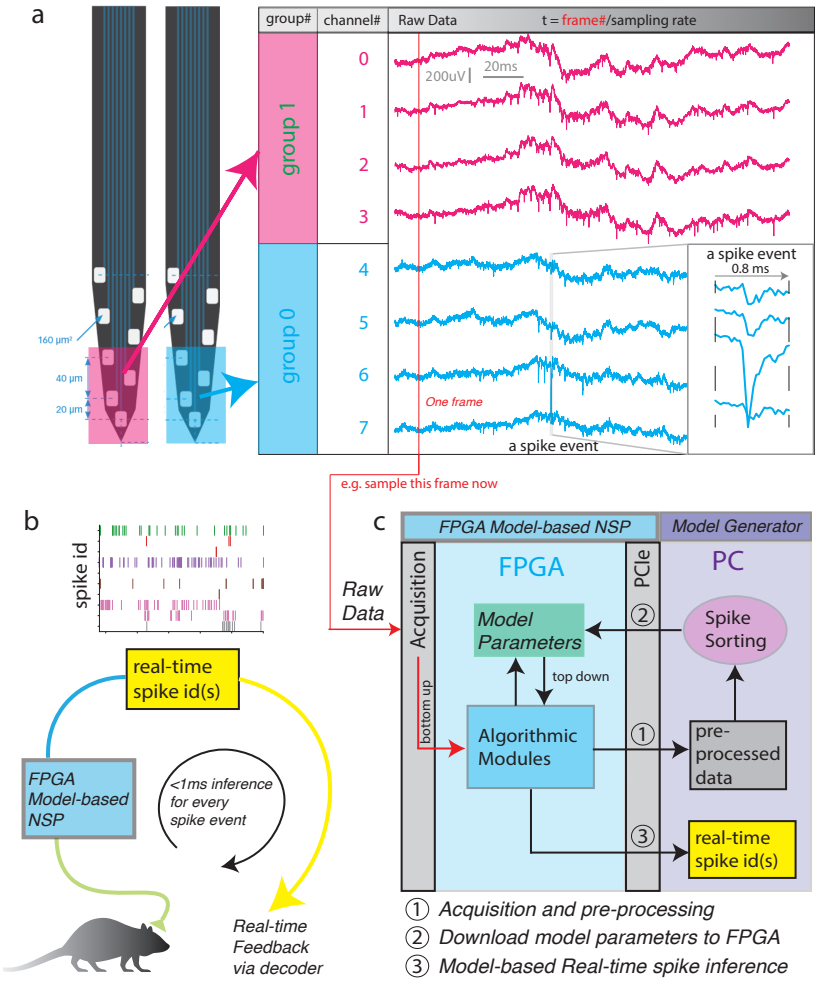


Fig. 3.2 **The FPGA-based NSP with silicon probe recording**: (a) A snapshot of recorded raw data from two groups (pink and blue) of electrodes on the silicon probe(s). Each group contains 4 channels. The voltage waveforms from adjacent channels (0,1,2,3 or 4,5,6,7) are correlated. (b) The FPGA-based NSP takes the raw data and outputs the real-time spike ids, which triggers the feedback directly or feeds a decoder to trigger the real-time feedback. The spike inference loop takes less than 1 ms to complete. (c) A high-level schematic overview of the devices, the building blocks and the signal processing pathways (the numbers in circles). The FPGA contains two building blocks: the model parameters and the Algorithmic Modules. Firstly, the FPGA conducts the acquisition and pre-processing of the raw data (the 1st signal processing pathway). Then the spike sorting software package on the PC uses this data to generate model parameters and downloads them into the FPGA (the 2nd signal processing pathway). With the updated model parameters, the NSP algorithmic modules are involved to contribute to the real-time spike inference (the 3rd signal processing pathway).

Accordingly, there are three software programs running on the PC, which operate in the order below, representing the three signal processing pathways:

1. An acquisition software program adapted from the Open Ephys GUI is used to receive and store the pre-processed (intermediate) data. (Fig. 3.2c: the first arrow)

2. The model generator software program (Spiketag) is used to read the intermediate data, perform the spike sorting and build the model. A set of Python APIs in Spiketag is used to download the model parameters to the FPGA (Fig. 3.2c: the second arrow)

3. The software that contains a decoder that can receive and use the real-time spike id(s) and can further process the real-time spike trains (Fig. 3.2c: the third arrow).

The work divided between PC and FPGA makes the scope of each component clear and allows the development of the FPGA NSP focusing on its dedicated goal: the real-time neural signal processing with simultaneous multi-channel data acquisition to deliver low-latency spike inference. To achieve this goal, three high-level design objectives must be met.

The first objective is low-latency data movement, from raw data acquisition to the result transmission. In order to do that, a set of commercial low-latency I/O interfaces was chosen and calibrated to feed the raw data into the FPGA and transmit the computed result out of the FPGA to the PC. The data movement from a set of INTAN amplifiers to the FPGA and then to PC was first implemented and tested. Then the PC/FPGA PCIe communication was implemented and tested.

The second design objective is fast and accurate neural signal processing. For that, several FPGA algorithmic modules (each is a modularized physical circuit) are designed and developed. The data flows into those physically distributed algorithmic circuits, gets processed, and the results flow out. The specific operations of each circuit are triggered by the arriving data and the index. The index is information associated with the data, indicating where (*channel*# or *group*#) and when ($frame$#) the data originates (see the top of Fig. 3.2a: the # means No.). The association between the data and index are listed below:

1. Each sample has a frame number ($frame$#), indicating when it enters the FPGA. The timestamp for that specific sample is ($frame$#) divided by the sampling rate ($25000Hz$).

2. Each sample has a channel number (*channel*#) from 0 to 159, indicating origin channel of sample.

3. Each sample has a group number (*group*#) from 0 to 39, indicating origin group of sample.

4. Each *group*# has a unique channel hash code (inside the FPGA it is a 64-bit-wide code for every group) which is a 4-tuple. For example: group n: (channel a, channel b, channel c, channel d). The group hash code reflects which channels are adjacent to each other and uniquely decides which detected spike waveforms should be grouped together to form a spike packet for downstream processing.

5. Each data packet (spike packet/feature packet) has a unique *group*#

The index is a very important concept in the rest of the chapter. It provides the spatial (*channel*#, *group*#) and temporal ($frame$#) information of every data sample or every data packet (the concept of data packet will be introduced later). Importantly, the spatial index system (*channel*#, *group*#) bbelongs to part of the model parameters, the channel grouping map, that can be flexibly configured in each experiment; the temporal index ($frame$#) of the data represents the acquisition frame of each sample. In each recording, the $frame$# starts from zero and increases by one after each complete acquisition of a new frame of data (i.e., 160 samples from 160 channels). Once entering the processing pipeline, the indices associated with the data samples determine how a specific circuit operates on the arriving data at specific clock cycles.

This leads to the third design objective: a specification of how the data and index are associated (at which clocks and at which digital lines) and how they are read from and written to the digital flows. In the end, all these digital flows are a pack of fixed width (bits), wires that carry high and low voltages changing from clock to clock. A set of digital protocols (specifications) were designed for each algorithmic module to correctly capture its designated data sample/packet and accompanying index[8]. Such digital specification also holds the key for developing the software API to read the processed data stream and data packet from the FPGA.

In the following sections, I will describe the methods to achieve the design objectives of the NSP - the interfaces, algorithmic modules, and digital protocols respectively - in the context of FPGA design.

3.3 The digital design of the NSP using FPGA

Designing digital circuits with an FPGA requires coding. However, unlike software coding where programming can be done in one or a few places, FPGA design requires many toolkits

[8]This aspect renders programming in an FPGA much harder than in a PC, because the data is always moving in an FPGA while it is static in a PC. In a PC, the customized digital protocol is not required for data processing. But in an FPGA, much effort is needed to calibrate the way for capturing the correct data sample and its index from many concurrently moving data streams.

so that digital engineers can perform a design flow. The source code can then be mapped into physically connected hardware. These must satisfy the timing constraints when digital signals flow through the circuits. My current design used the Xilinx Kintex FPGA and tools to design and develop the NSP.

Xilinx, Inc. provides several design flows for programming an FPGA through an integrated development environment (IDE) called Vivado. The design flows, meaning several tools executed in an order from the Vivado IDE, were used to generate the desired FPGA circuit. In this thesis, two of the design flows were adopted:

First, the Register-transfer level (RTL) flow: this design flow uses the low-level hardware description language Verilog and Xilinx IPs to model the circuit IOs, clock management, etc. The RTL code, e.g. Verilog, describes the flow of digital signals between hardware wires and registers, the logic operations on these wires and registers either dependent or not dependent on the digital clocks, and the clock generation and module IOs, etc.

Second, the High-Level-Synthesis (HLS) flow: A Vivado C-based High-Level Language is used to describe the algorithm. The HLS compiler converts the C-based language into Verilog by taking the programmer's pragmas that provide the guidance of how to map the algorithm into actual physical circuit operation. Once the Verilog is generated, the rest is very similar to the RTL design flow described above.

Both above-described design flows output the hardware description language (HDL) source code and Intellectual Property (IP, this is a term used in the field of digital design which means preconfigured circuits/modules that can be reused as building blocks) that then go through a series of procedures to map the source code and IPs into the real implementation in hardware. These procedures include logic simulation, logical and physical constraints assignment, logic synthesis, implementation, timing closure and bitstream generation. To understand the individual toolkits in the Xilinx FPGA Design Flow, the Xilinx documentation Navigator[9] is recommended. The Xilinx documentation Navigator is a library that contains all the documents providing instructions to utilize tools such as RTL coding, High-Level-Synthesis, IP Packager/Integrator, Simulation, Synthesis, Implementation and so on. These toolkits are Xilinx-specific. To cover the details of these technologies would require several books (see Xilinx documentation Navigator) and is not necessary for understanding the methodology of FPGA NSP design. For this reason, this thesis will not include a tutorial of any specific Xilinx tool. Instead, it will focus on how the NSP is designed and developed, its algorithm, and how an FPGA is used here[10] to enable the high-bandwidth and low-latency spike inference.

[9]To complete an FPGA design, many toolkits will be used; see the Xilinx documentation Navigator (https://www.xilinx.com/support/documentation-navigation/overview.html)

[10]Using FPGAs from venders other than Xilinx could implement something similar.

In the following sections, four aspects of FPGA NSP design will be investigated: first, the NSP digital interface to acquire and move the data; second, the NSP workflow and signal processing pipeline to convert the raw data into the spike identity; third, the algorithmic detail of each individual FPGA NSP module and their latencies; fourth, the digital protocol design to ensure the data and index are handled correctly with nanosecond precision.

3.3.1 The NSP interface

Interfaces, or Input/Outputs (IOs)[11], are the physical hardware connections where the data/signal flows across circuits in a single device or across different devices. Three types of interfaces were developed here.

The first type of interface is responsible for the acquisition of data. A customized Serial Peripheral Interface (SPI) link (section 3.3.1.1), originally developed by INTAN Inc. and Open Ephys for their commercial acquisition system, was modified here to sample the data into the FPGA (Fig. 3.3 a,b).

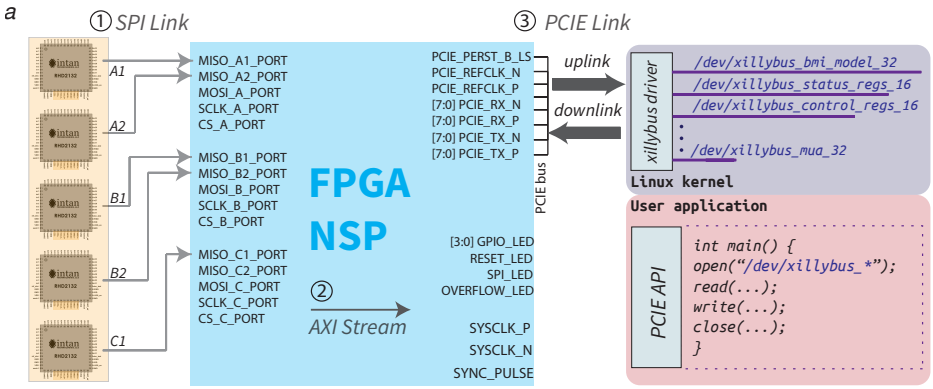
The second type of interface is the internal IOs. These IOs move the data between internal FPGA modules, allowing signal processing with many modules. Each module moves the processed data to its downstream module. The AXI4-stream[12] (section 3.3.1.2), originally designed by ARM Inc, was directly applied between each pair of modules (Fig. 3.3 a,c).

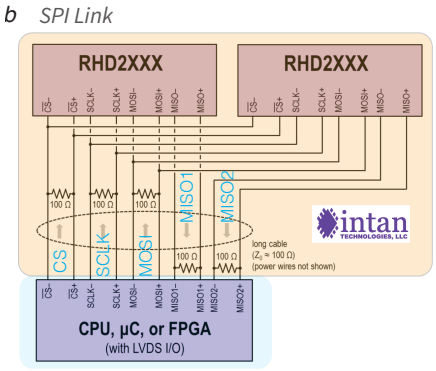
The third type of interface, Peripheral Component Interconnect express (PCIe), enables the communication between the FPGA and the PC. An IP core[13] (section 3.3.1.3) provided by Xillybus Inc (Xillybus, 2010) was adopted to bridge a set of customized FIFO/RAMs in the FPGA and the PC (Fig. 3.3 a,d). The data communication is enabled by several Xillybus PCIe Linux devices. These Linux devices, behaving much like standard Linux IO, can be opened, read from, and written to (Fig. 3.3 a: right side). Each Linux device is either associated with a FIFO or a RAM in the FPGA (Fig. 3.3 d), depending on the purpose of the device. The FIFOs buffer the data flow while the RAMs receive and store the model parameters.

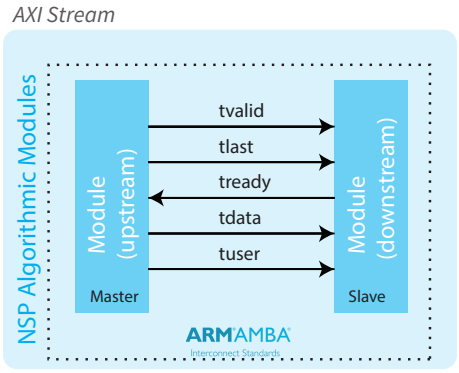
[11] Interfaces, IO, path, link: These terms are usually interchangeable when describing the moving of data from one place to another place. The difference is linguistic. But, dependent on the context, it may mean not only the hardware connections but also the digital protocol/specification, and even software driver (e.g. for PCIe a driver in the PC is necessary), that all together enable the data movement. This section focuses on the hardware aspect, which defines the basic architecture of the system.

[12] AXI stands for Advanced eXtensible Interface. As part of the ARM Advanced Microcontroller Bus Architecture (AMBA), it was designed for on-chip communication. It is widely used in both FPGAs and ASICs.

[13] IP, or IP core, stands for the intellectual property, it is a term that does not convey the meaning well. In the field of FPGA design, IP specifically means preconfigured circuits/modules that can be reused as building blocks.


a
① SPI Link
③ PCIE Link
A1
A2
B1
B2
C1
MISO_A1_PORT
MISO_A2_PORT
MOSI_A_PORT
SCLK_A_PORT
CS_A_PORT
MISO_B1_PORT
MISO_B2_PORT
MOSI_B_PORT
SCLK_B_PORT
CS_B_PORT
MISO_C1_PORT
MISO_C2_PORT
MOSI_C_PORT
SCLK_C_PORT
CS_C_PORT
FPGA
NSP
② AXI Stream
PCIE_PERST_B_LS
PCIE_REFCLK_N
PCIE_REFCLK_P
[7:0] PCIE_RX_N
[7:0] PCIE_RX_P
[7:0] PCIE_TX_N
[7:0] PCIE_TX_P
PCIE bus
uplink
downlink
[3:0] GPIO_LED
RESET_LED
SPI_LED
OVERFLOW_LED
SYSCLK_P
SYSCLK_N
SYNC_PULSE
xillybus driver
/dev/xillybus_bmi_model_32
/dev/xillybus_status_regs_16
/dev/xillybus_control_regs_16
/dev/xillybus_mua_32
Linux kernel
User application
PCIE API
int main() {
open("/dev/xillybus_*");
read(...);
write(...);
close(...);
}


b SPI Link
RHD2XXX
RHD2XXX
100 Ω
CS
SCLK
MOSI
MISO1
MISO2
long cable
(power wires not shown)
CPU, µC, or FPGA
(with LVDS I/O)


c AXI Stream
NSP Algorithmic Modules
Module (upstream)
Master
tvalid
tlast
tready
tdata
tuser
Module (downstream)
Slave
ARM AMBA
Interconnect Standards

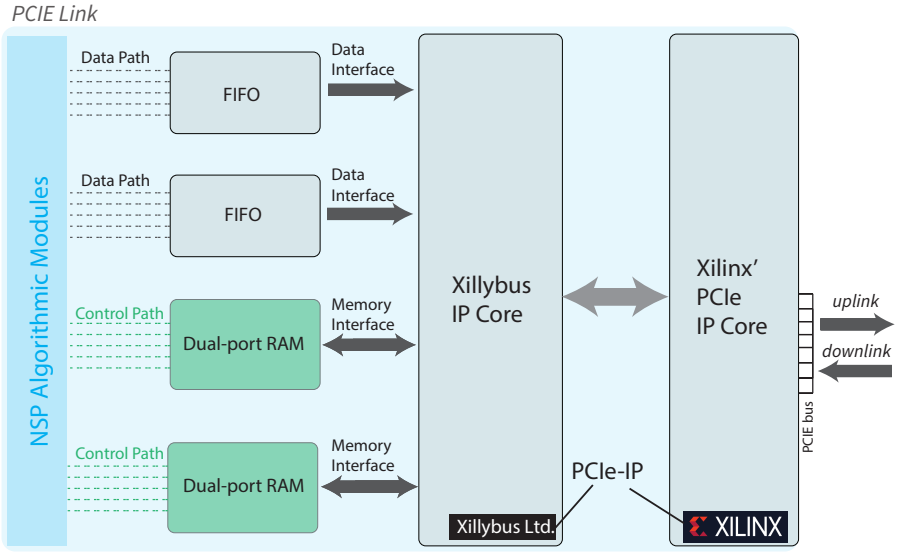

d PCIE Link
NSP Algorithmic Modules
Data Path
FIFO
Data Interface
Control Path
Dual-port RAM
Memory Interface
Xillybus IP Core
Xilinx' PCIe IP Core
uplink
downlink
PCIE bus
PCIe-IP
Xillybus Ltd.
XILINX

Fig. 3.3 NSP interfaces - external and internal IOs: (a) The FPGA NSP uses external IO pins (in black font) to communicate with external devices. The FPGA NSP connects to 5 INTAN chips from, A1, A2, B1, B2, C1, through an SPI Link (Circled number 1), then the data is processed inside by many FPGA modules. Inside the FPGA, these modules move data from upstream to the downstream circuits using an AXI-Stream Link (Circled number 2). The FPGA NSP connects to the Linux PC through the PCIe Link (Circled number 3). The user application reads/writes PCIe data streams (each stream has a name like $/dev/xillybus*$). From the user's perspective, these PCIe data streams are readable/writable Linux devices. From left to right: (Orange background: INTAN chips, Blue background: FPGA NSP, Purple Background: Linux Kernel, Red Background: User application). (b) One FPGA-SPI module supports up to two INTAN RHD2XXX chips (RHD2132 is used in this thesis) through 5 pairs of LVDS signals: two MISO, one MOSI, one SCLK, one CS (highlighted in blue). (c) The internal interface (IOs) between FPGA NSP modules is an AXI4-stream Interface. The data ('tdata') is sent only when the 'tvalid' signal is high. The 'tready' indicates that the downstream module is ready to receive data. (d) The data path through FIFO and the control path through Dual-port RAM were designed to link the FPGA NSP algorithmic modules to the PC. A Xillybus IP Core mediates the low-latency communication between the FIFOs and RAMs inside the FPGA and the PC. (Modified from INTAN, Xilinx and Xillybus development documentation)

The next few subsections (subsections 3.3.1.1, 3.3.1.2, 3.3.1.3) will examine the hardware connections of each interface in detail.

3.3.1.1 The SPI links acquisition chips to the FPGA

Five INTAN RHD2132 chips were used here as the acquisition chips. Each RHD2132 is a 32-channel digital electrophysiology acquisition chip from INTAN Inc. It samples, amplifies, digitizes and multiplexes the raw neural signals collected by the multi-electrodes. The clock input and digital output of the RHD2132 goes through the SPI interface (see more in http://intantech.com/). Five INTAN RHD2132 chips provide 160 channels of input to the FPGA NSP.

In the FPGA, an SPI interface was adapted from INTAN's original design and development. This adapted SPI interface contains three FPGA-SPI modules. One single FPGA-SPI module connects to two RHD2132 chips, via 5 pairs of LVDS signals[14] (Fig. 3.3a:left side, b). These five pairs of signals are: Two MISO (Master In, Slave Out) inputs, each of which receives 1-bit of data from one RHD2132 chip at each sampling clock. One MOSI (Master Out, Slave In) output, sending commands to two RHD2132 chips, for example, to ask for the acquisition of the next 1-bit of data. One SCLK (Sampling Clock) signal, sending sampling clock signals to two RHD2132 chips. The sampled data from to chips was transmitted via

[14]LVDS stands for Low-voltage differential signaling. LVDS signals transmit information using the voltage difference between a pair of wires, which provide robustness against interference. Modern FPGAs host many pairs of IO pins that receive or generate LVDS signals for specific applications.

MISO inputs. One CS (Chip Select) signal, sending active-low chip selection signals to two RHD2132 chips, is used to select input.

A customized acquisition PCB designed by Brian Barbarits in the Harris lab (Janelia) was used to physically bridge the INTAN chips' IO pins and the FPGA IO pins through an FMC (FPGA Mezzanine Card) connector. In theory, the three FPGA-SPI modules implemented in the NSP can communicate with six RHD2132 chips. However, the current version of the customized acquisition PCB design provides IO pins for five INTAN chips (A1, A2, B1, B2, C1 in Fig. 3.3 a:left side).

3.3.1.2 The AXI4-Stream links the internal FPGA modules

Since each internal FPGA algorithmic module is an individual circuit, to move data from one module to another, we also need IO interfaces between these modules. The major type of internal interface, used in this thesis, to move data from one module to another is the AXI4-stream interface (Fig. 3.3 c).

The AXI4-stream interface supports both packet data transmission and continuous data transmission. Five signals were used to support communication between upstream and the downstream circuits (Fig. 3.3 c). The downstream circuit indicates to the upstream circuit that it can receive data when the 'tready' signal is high. Data ('tdata') is only sent, from the upstream to the downstream circuit, when the 'tvalid' signal is high. When transmitting the packet data, the 'tlast' signal is high only with the last data sample of the packet. Notably, the 'tuser' signal provides meta information that can be sent along with data in the same clock cycle, and was heavily used in encoding the index of the data (see the concept of 'index' in section 3.2.1). A data sample and index can require many bits. Both 'tuser' and 'tdata' can use more than one bit while 'tlast', 'tvalid' and 'tready' can only be 1-bit wide.

3.3.1.3 The Xillybus PCIe links the FPGA and the PC

The PCIe links the FPGA to the PC at a different clock rate (250 MHz) than SPI acquisition (25 kHz). Also, the PCIe interface supports both asynchronous and synchronous communication between the computer and the FPGA. In contrast, the SPI link only works in synchronous mode (the RHD2132(s) and the FPGA work synchronously at a regular pace, e.g., every 40 µs for one frame of data, and there is no buffer between the sender and the receiver). The asynchronous PCIe data stream is especially useful for low latency continuous data transmission because some of the data are briefly stored in the Direct Memory Access (DMA) buffer (usually a few microseconds, dependent on the data-bandwidth and buffer size)

before being transmitted to the computer memory. DMA allows data transmission between the PCIe DMA buffer and computer memory without the involvement of CPU time.

The Xillybus IP-core was adopted for handling the PCIe downlink and uplink. Xillybus not only supports a full PCIe data link, including low-latency DMA transmission but also provides a Linux driver that offers user-programmable data streams for direct communication between the FPGA and the PC. The Xillybus IP core was specified and generated with a driver, with which the data stream can be read and written by the tailored applications in Linux. In practice, these programmable data streams were remarkably robust in the implementation of the NSP.

Unlike the SPI and AXI4-stream, to make the PCIe link work requires software APIs. A high-level set of Python-based APIs was built to read and write the specific PCIe data stream that is supported by a Xillybus Linux driver (Fig. 3.3 a:right side). From the perspective of the FPGA developer, it is the FIFO and RAM that is read and written by the Linux devices defined by the Xillybus IP Core. In other words, the device named like $/dev/xillybus*$ is the Linux device that reads/writes to a Dual-Port RAM that resides inside the FPGA and stores the model parameters (Appendix B.0.2) and reads from a FIFO to pull out the processed data (Fig. 3.3 a and d: PCIe Link; and see Appendix B.0.1).

The Python-API accessible FPGA FIFOs and RAMs are the key for the software to communicate with the NSP algorithmic modules. Since the FIFOs are dedicated to data transfer, I collectively refer to the interfaces that transfer data via FIFOs as the data path. Since the model parameters, transferred from the PC to the RAM, control how the FPGA NSP algorithmic modules operate in real-time, I collectively refer to the memory interfaces via RAM as the control path (Fig. 3.3 d). Two categories of interfaces between the NSP algorithmic modules and the FIFOs/RAMs were developed. The first is the data path, in which the NSP algorithmic modules use the FIFOs in the data path to transmit pre-processed MUA, spike waveforms, feature vectors and spike id(s) to the PC. The second is the control path, in which the algorithmic modules use its connection to the Dual-Port RAM in the control path to read the model parameters stored in the RAM and use these parameters to control how the circuits handle the current data. Different algorithmic modules access either different RAM or different ranges of addresses of the same RAM that stores the model parameters used to guide its specific signal processing. From the software perspective, the connection from the PC to the FPGA RAM is referred to as the memory interface because updating the model parameters simply requires the corresponding Python APIs to write the FPGA memory device, the dual-port RAM (Fig. 3.3 d).

To summarize, the interfaces are the backbone of the FPGA NSP design. They constitute the foundation for the design of the NSP algorithmic modules, by controlling:

1. How the data enters into the FPGA through the SPI interface.
2. How the data moves from module to module inside the FPGA through the AXI4-Stream interface.
3. How the parameters that the algorithmic modules need to perform computations are updated from the PC through the PCIe memory interface (Appendix B.0.2).
4. How the computed data output by the algorithmic modules are transmitted to the PC through the PCIe data interface (Appendix B.0.1).

3.3.2 The NSP signal processing pipeline

So far, I have described the hardware architecture enabling the data to move into, within, and out of the FPGA. This section will describe how the data is processed in real-time algorithmic modules inside the FPGA. The general methodology of algorithmic FPGA design will be introduced first, followed by the specific NSP algorithmic design. Three stages of the NSP signal processing pipeline will be described. Real data examples will be used to illustrate the input and output of each algorithmic module.

3.3.2.1 General methods for algorithmic FPGA design

Most algorithms posit a model[15] that contains functions with parameters. The input and parameters together determine the output. The key to FPGA algorithmic design is to explicitly separate the model parameters and the functions into distinct circuit elements.

To build a sophisticated signal processing device, many different algorithms are required at different stages of the signal processing pipeline. In an FPGA, an individual function can be converted into an individual FPGA module. Each module obtains the parameters for its own operation from the on-chip memory (i.e., the dual-port RAM) to generate the desired output. Parameters stored in the Dual-Port-RAM are updated from the PC whenever a new model is constructed, or part of the model is updated. The dual port RAM ensures that the memory locations holding the model parameters can be accessed from both the PC-PCIe port and the FPGA processing circuit port.

Both Hardware description languages (HDLs) and High-level-synthesis (HLS) were used to describe the algorithm in the design of the NSP. While HLS is a better language to

[15]The model selection - how do we know that we have a good model - is a related topic in data science. Here I focus on describing the FPGA design methods, assuming we already have a good enough model for my goal. My task is to implement the algorithm, in the form of hardware, that specifies every single operation to compute that model.

describe the scheduling of operations, Verilog is a better choice to specify how one module is connected to another and how the memory is accessed.

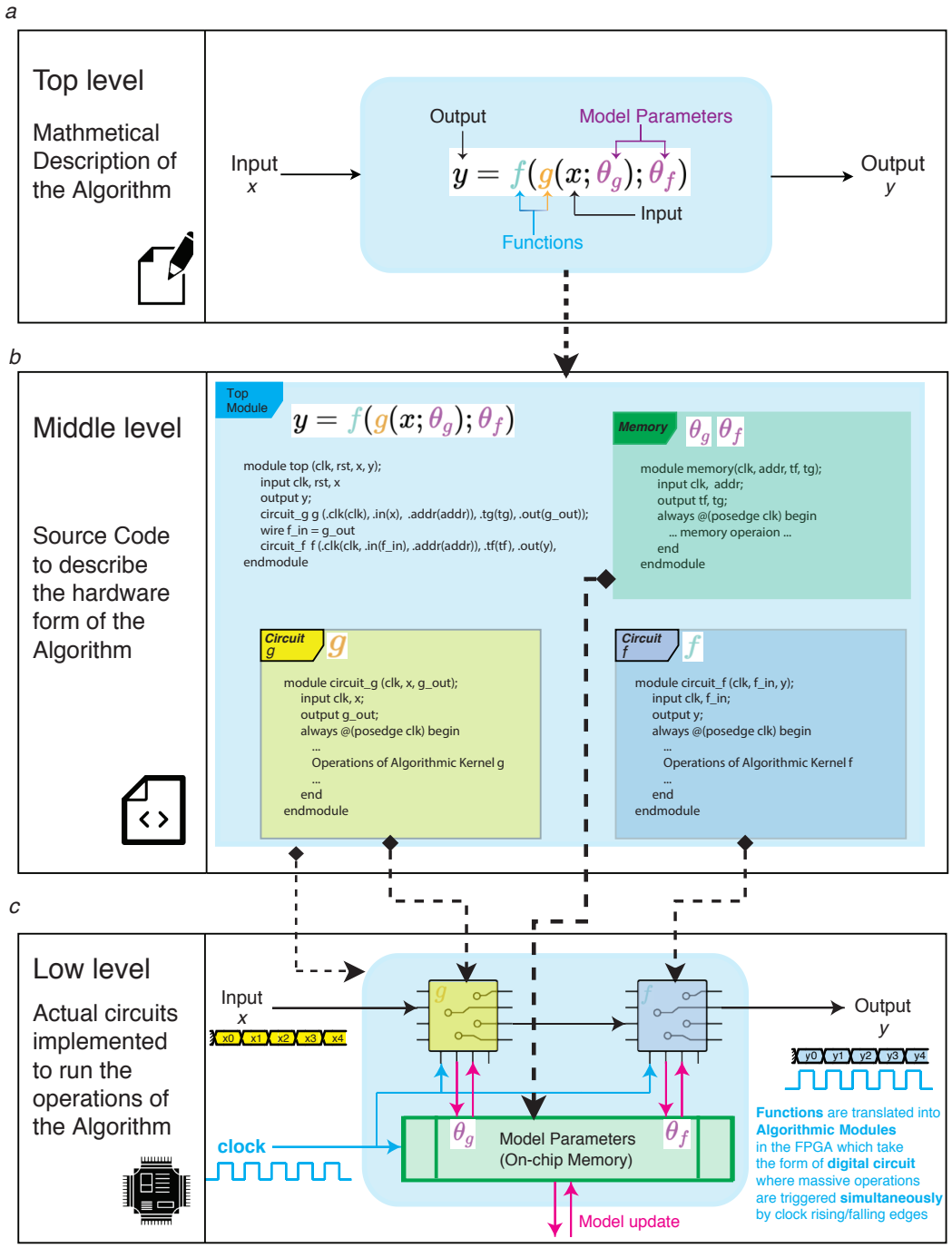


Fig. 3.4 General algorithmic FPGA design methodology: (a) The top level: the mathematical description of the algorithm. Often one algorithm can be decomposed into several functions. Here is an example of a two compositional function algorithm with their model parameters. (b) Two functions: 'f' and 'g' are described in either HLS or Verilog source code as individual modules (circuit), and the model parameters are described using a Dual-Port RAM (On-chip Memory). A top Verilog module defines the internal IOs to connect circuit 'f' and circuit 'g' together. (c) Through the toolkits provided by Xilinx or other FPGA vendors, the source code is converted into the actual hardware. Each algorithm corresponds to an individual connected module/circuit. At this level, the data arrives at circuit 'g' at a certain clock rate. Circuit 'f' fetches its model parameters and uses them to generate the intermediate input into the circuit 'f'. 'f' then fetches the parameters for the intermediate input and generates the final output along with the clock. The final output can be chained into the downstream circuits.

Fig. 3.4 illustrates three levels of algorithmic design in the FPGA: first, write down the algorithm's mathematical description from input to output and explicitly decompose the algorithm into several functions and parameters according to the model assumptions (Fig. 3.4a); second, code the description of the algorithm, that is, describe each function as a circuit/module, the on-chip memory circuit/module and a top module that describes the interfaces between modules and memory, using HLS or Verilog (Fig. 3.4b); third, translate the code into actual hardware, consisting of circuits connected in a daisy chain, where each circuit is interfaced with on-chip memory that stores parameters (Fig. 3.4c).

Another important feature of FPGA algorithmic design is to construct the pipeline. The word pipeline means each module inside the pipeline performs simultaneously, like the workers in a factory assembly pipeline. Both Verilog and HLS provide features to schedule the operations at the clock edges, either from low to high or high to low. Many operations from different circuits can be triggered by a single widely distributed clock. When implementing the algorithm in the FPGA, distributing the clock signals to targeted modules, and scheduling the operations to the clock cycle is very important (Fig. 3.4c).

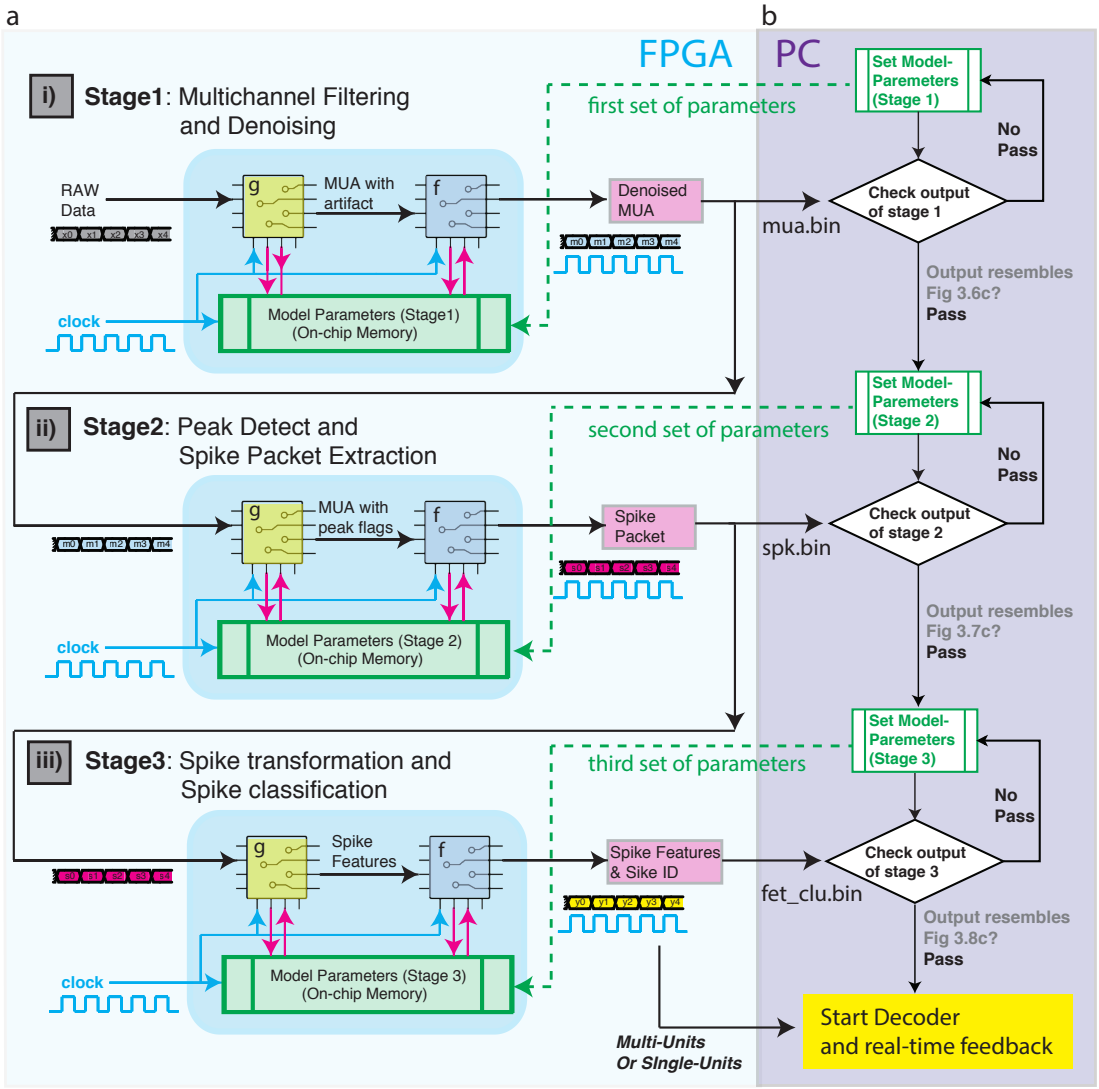
Next, I will introduce the specific signal processing pipeline designed for the NSP's real-time spike inference.

3.3.2.2 Three stages of the NSP signal processing pipeline

The NSP receives the raw neural signal and generates labelled spikes as output. A series of computations is required, including filtering, denoising, spike detection, spike packaging, spike transformation, and spike classification for 160 channels. The NSP requires configuring these modules in sequential order: the parameters demanded by spike detection and packaging are generated by analyzing the data output, filtering and denoising; the parameters demanded by spike transformation and classification are generated by analyzing a large number of

recorded spike waveforms. Accordingly, these modules were separated into distinct signal processing stages, where each stage requires a distinct set of parameters generated by analyzing the output file obtained from the previous stage. Fig. 3.5 illustrates the three stages of the signal processing workflow.

a



c

Signal Processing Stage	NSP algorithmic modules		Output	
	g	f	Signal output	File output
Stage1:	FIR filter	Reference Substraction	denoised MUA	mua.bin
Stage2:	Peak detection	Spike Packet Extractor	spike packet	spk.bin
Stage3:	Spike Transformer	Spike Classifier	spike features&ID	fet_clu.bin

Fig. 3.5 Three signal processing stages of the NSP algorithmic modules: (a) Three signal processing stages of the FPGA NSP, each requiring a distinct set of parameters to operate. Each stage consists of two modules. Each stage consists of two modules. The first stage denoises the MUA, the second stage outputs the Spike Packet and the third stage outputs spike features and spike ids. Each module takes the output from the previous module as input. The form of signals (RAW data, MUA with artefact, Denoised MUA, MUA with peak flags, Spike Packet, Spike Features, Spike ID) at each node along the processing pipeline is illustrated in the next few figures. (b) Three signal processing stages from the PC operation perspective: set parameters for each stage and then examine the output file to check whether the configured modules function correctly. The Spiketag software will take file output from the previous stage and generate parameters and download it into the FPGA (green dashed lines from (b) to (a)) for the next immediate stage. In the beginning, we set the reference channel for each channel (the first green dashed line) and then get the denoised MUA. Next, we set the threshold for each channel (the second green dashed line), then we get the Spike packet. The spike packet is a packet that contains spike waveforms along with the spike timing and the channel and group from it was captured. With many recorded Spike packets, we calculate the parameters of spike transformation and spike classification and download them into the FPGA (the third green dashed line from (b) to (a)), then we get the spike features and spike id. If the software determines that all the modules operate correctly by checking the file output from each stage, the real-time spike-ids would be sent to the decoder or directly used to trigger the feedback. (c) The names of modules, signal output and file output from each signal stage. *mua.bin* is the binary file storing multi-channel denoised MUA, *spk.bin* is the binary file storing all the real-time extracted spike packets, $fet\_clu.bin$ is the binary file storing all the spike feature vectors and the corresponding spike identities (ID).

The three stages of signal processing in the NSP bear much resemblance to traditional spike sorting procedures. The difference here is that we must configure three set of parameters[16], one stage after another, into the FPGA, which requires iterative checking that the modules in the previous stage are correctly configured (Fig. 3.5b). The first set of parameters are the channel order, each channel's group number and each channel's reference channel number[17], enabling the signal denoising by removing the common noise from adjacent reference channels. The second set of parameters are the channel-wise thresholds to detect the spikes from the 160 channels of the denoised MUA. Once the threshold is set, the second stage modules would output spike waveforms directly. After visual inspection of the extracted spike waveforms, the experimentalist might decide to manually change the reference channels or thresholds before entering the next stage. With a sufficient number of spike waveforms, the third set of parameters, the PCA transformation matrix and vector quantized clusters (VQ) for each electrode group, can be computed. After the visual inspection over the final spike

[16]In the later sections, I will dive into the algorithm and discuss: 1. The computation carried out by each module with each set of parameters. 2. How each set of parameters is decided. In this paragraph, however, they are briefly introduced to illustrate the overall workflow of the three stages of pipelined signal processing.

[17]A question is why the parameters for the filter is not here. The answer is the filter coefficients are fixed in the ROM of the FPGA, because we do not need to update them from one experiment to another.

assignment output, the experimentalist can decide whether to start the real-time feedback experiment. If clear drift of clusters appears because of probe drift, the experimentalist might decide to restart the pipeline from the very beginning.

The decisions made by the experimentalist on the parameter settings (Fig. 3.5b) are important for the in-lab performance of the NSP. Next, I will elaborate further on the purpose of each signal processing stage and the typical desired input and output.

3.3.2.3 Signal processing stage 1: multichannel filtering and denoising

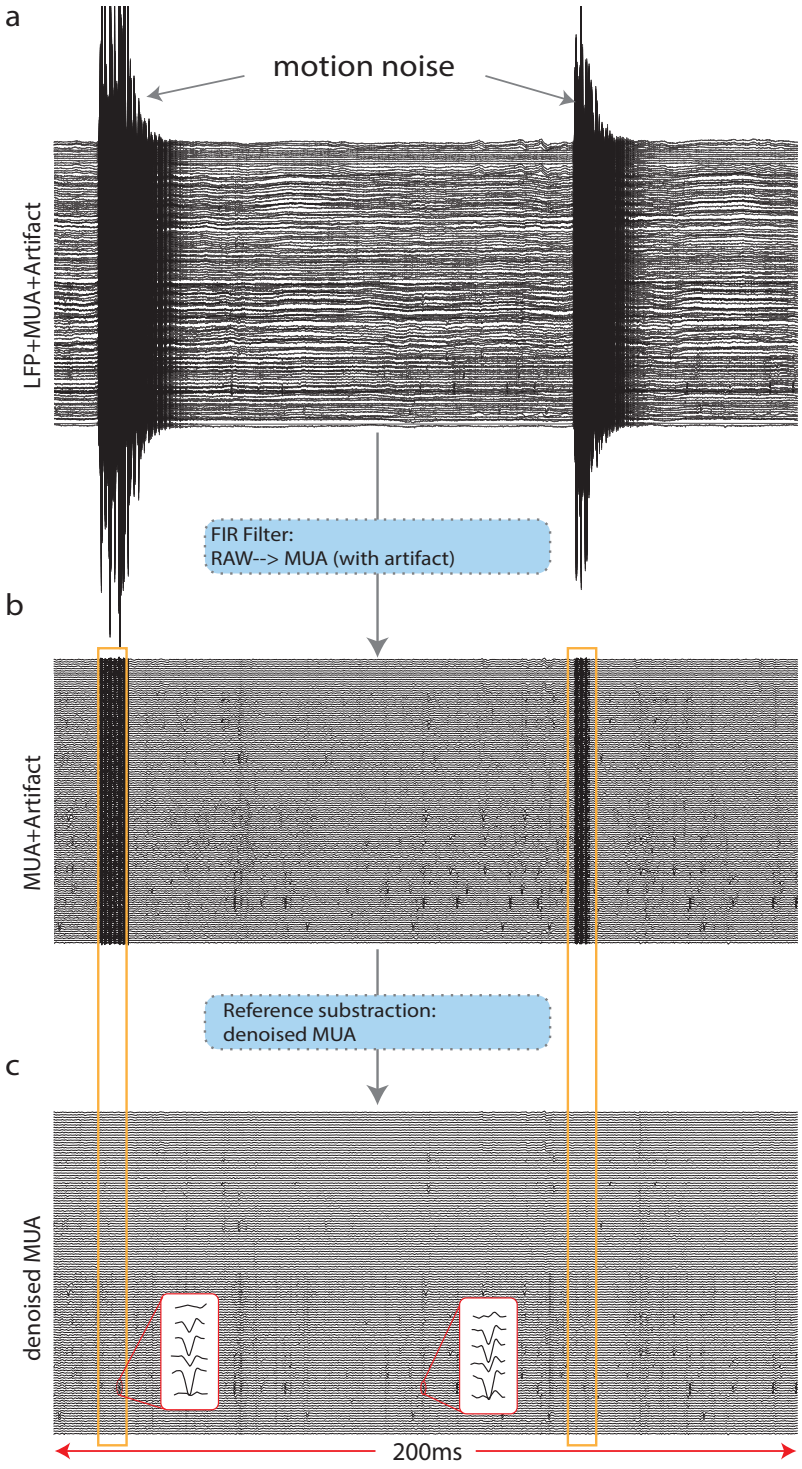


Fig. 3.6 NSP signal processing stage 1: (a) 160 channels of RAW signals. (b) 160 channels of MUA with artefact. (c) 160 channels of denoised MUA by FPGA reference subtraction.

The purpose of the stage 1 is to get the denoised MUA, and store it in the file *mua.bin* (Fig. 3.5: row 1). Analyzing the *mua.bin* gives rise to the next stage of parameters, the spike thresholds. By downloading the spike thresholds into the FPGA NSP, the spike waveforms are output in real-time as a file *spk.bin*.

The stage 1 FPGA processing is implemented with a multichannel finite impulse response (FIR) filter and a multichannel Reference Subtraction module. For instance, RAW data with motion noise (Fig. 3.6 a) become less noisy after the FIR filter; however, it is still mixed with a large artefact during the time period of motion noise (Fig. 3.6b). With reference subtraction, this artifact is almost completely removed (Fig. 3.6c) because this motion artefact is almost identical across every channel. Fig. 3.6c demonstrates that the denoised spike event (Fig. 3.6c, the first spike event) in the previously noisy zone looks very similar to the spike event occurring outside the noisy zone (Fig. 3.6c, the second spike event). Reference subtraction is the digital version of the differential recording between adjacent electrodes, invented by O'Keefe (O'Keefe and Dostrovsky, 1971) to remove the motion artefact from noisy neural data. In my experience, digital reference subtraction eliminates most of the visible motion noise and makes the spike waveforms much more sortable. It will not remove the noise that abruptly changes across adjacent channels. But if that happens often in the data, one might need to recheck the probe, channel order and the recording system.

From the PC perspective, the experimenter needs to operate the software in several sequential steps. First we need to set the parameters for stage 1, which are the channel order, channel group#, and reference channel#. The FPGA can then perform reference subtraction in real-time, which requires subtracting from the sampled data the cached reference sample. Then we start the NSP and we would receive the denoised MUA. After we finish a recording, we check the output by loading the file *mua.bin* using Spiketag software and check if the data is correctly filtered and denoised. In the end, we set model parameters for stage 2 by computing the threshold and downloading the channel-wise threshold for spike detection on every channel. The empirical equation for estimating thresholds from the *mua.bin*, first introduced by Quiroga (Quiroga et al., 2004), was used here. This equation would fail for the bad channels that only collect noise. To cope with bad channels (or loose cables), manually updating the thresholds (or other parameters), through the Python APIs, is always allowed.

After the thresholds are prepared and downloaded to the FPGA, we can start to collect extracted spike waveforms.

3.3.2.4 Signal processing stage 2: spike detection and spike packet extraction

The purpose of stage 2 is to extract spike waveforms from the continuous denoised MUA stream and yield the spike waveform packet. Every detected spike event is wrapped by a spike packet with its index information, including the *frame#* and *group#*. In contrast to the continuous MUA data stream, the extracted spike packet data stream is no longer a continuous data stream. Every time when a spike packet is transmitted from the FPGA to the PC, it is appended to the end of the file *spk.bin* (Fig. 3.5: row 2).

The stage 2 NSP processing is implemented with a Spike Detector and a Spike Packet Extractor.

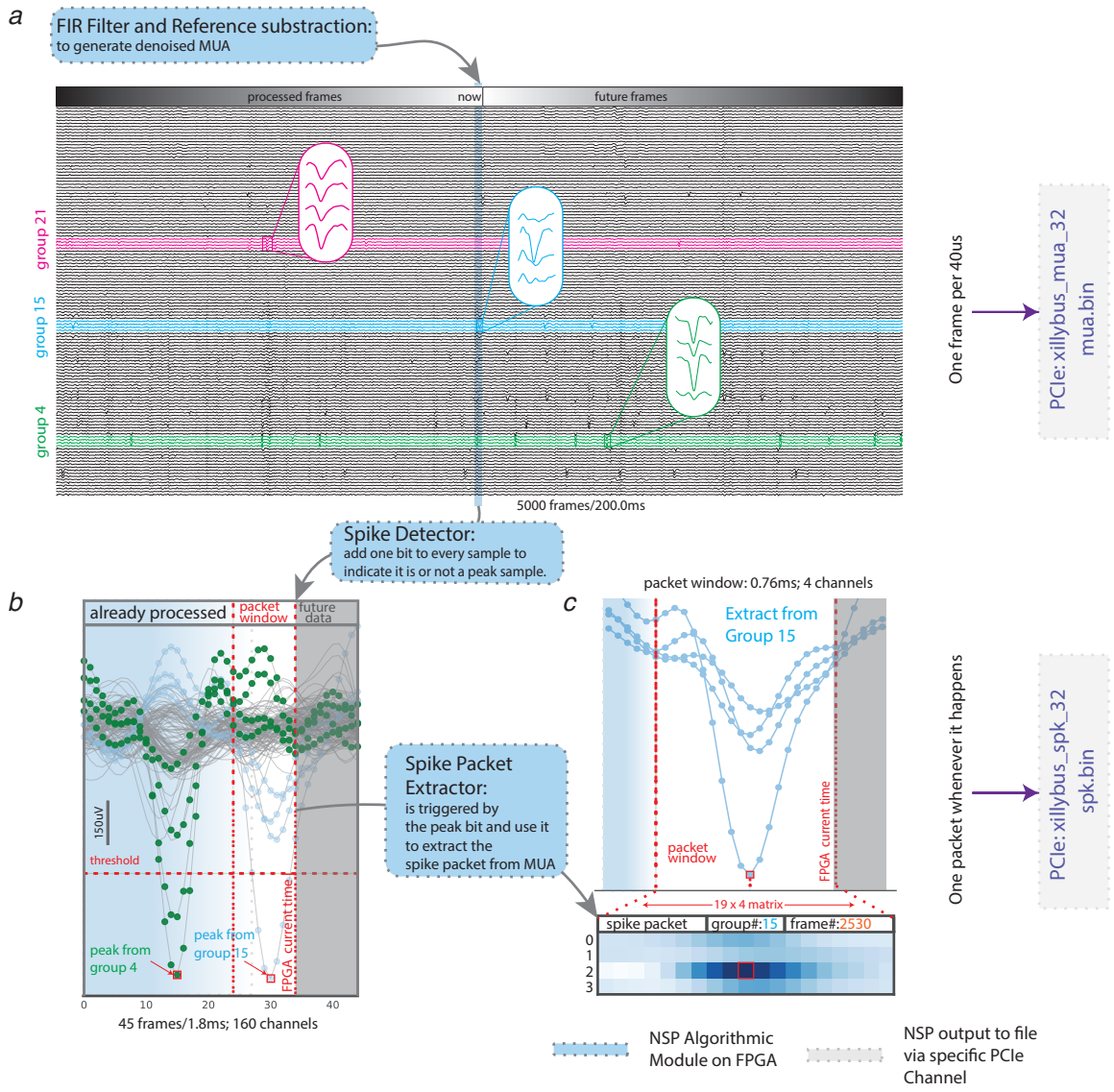


Fig. 3.7 NSP signal processing stage 2: (a) The example output of Stage 1 is the input to Stage 2: 200 ms of denoised MUA recorded from 160 channels, where each set of 4 electrodes form a group. Three spike events from three different groups are highlighted. A spike event from group-15 (blue) is inside the FPGA current processing window. The current processing window is highlighted with a vertical bar. All the samples from 160 channels inside the processing window pass through the memory buffer of the FPGA (the blue vertical bar in the middle). (b) A magnified window that contains all the samples in the processing window. An already processed spike event and a second spike event (the blue spike event in (a)) that is currently in the packet window (between two vertical red dotted lines). To perform the spike detection, the FPGA NSP first computes which sample is a peak sample. Every sample is marked with an additional flag bit where 0 means non-peak and 1 means peak sample (the small red rectangle). The peak sample means a spike event is detected, but not yet extracted. (c) The Spike Extractor only receives spike waveforms from the groups currently containing peak sample. Therefore, we only see four waveforms here. Top: the 0.76ms (that is 19 samples under 25000Hz sampling rate) packet window for spike packet extraction. When the largest peak moves to the middle position of the window, the spike packet extraction process starts. The spike packet extraction process operates group by group, and packages the spike event waveforms as a matrix, with the current group# and the frame# of the largest peak sample. Each spike packet matrix is formed by 4 rows and 19 columns of 32-bit numbers. Every 19 samples (a row in the matrix) consist of a waveform from one electrode out of the four-electrodes group. Bottom: an image view of the spike packet.

The job of the Spike Detector circuit is to determine if the signal has peaked, after crossing the thresholds. The Peak Detector appends all data samples with a single additional bit. Every 32-bit sample becomes a 33-bit sample with the last bit as 1 indicating this is a peak sample and 0 indicating not a peak. Then the 33-bit MUA with peak flag multichannel data stream (Fig. 3.5: row 2, between ‘g’ and ‘f’) enters into the Spike Packet Extractor (Fig. 3.5: row 2, ‘f’).

The job of the Spike Packet Extractor circuit is three fold. First, it selectively filters out the samples that are not from the electrode group where the peaks are detected. Second, inside a 0.76 ms[18] long packet window (the typical captured spike waveforms in 0.76 ms window is shown in Fig. 3.7 c), a state machine will decide which peak is the biggest peak compared to other peaks found among all the channels in the group. Then, when the biggest peak moves to the central location in the packet window, the matrix storing the spike event waveforms is extracted and wrapped with the *group*# and $frame$#. This packet data is called

[18]This is a spike width we found empirically would do well in spike sorting with PCA while being as short as possible. In fact, many experimentalists only use the four peaks from a spike event as feature vectors to perform spike sorting, which requires an even shorter spike window. The spike width of is usually less than 1 ms for pyramidal neurons and around 0.5 ms for interneurons, according to a recent study using a combination of large scale silicon probe recording and optogenetics on thousands of single-units (Li et al., 2019). However, for spike sorting and inference, we do not need the 1 ms long spike waveform. The shorter the spike window is, the less likely that other spikes will appear in the same window. The shorter the spike window is, the shorter the inference latency will be.

a spike packet (Fig. 3.7c lower: the spike packet) and is sent to the downstream circuit via AXI4-stream and to the PC via PCIe. This design ensures the processed spike packet is always aligned such that the highest amplitude sample is centered in the time axis.

From the PC operational perspective, stage 2 involves similar sequential steps as stage 1: First, check thresholds for every channel in the FPGA. Second, start the NSP to receive the output data from both stage1 (*mua.bin*) and stage 2 (*spk.bin*). Then load the file *spk.bin* using Spiketag software and perform spike sorting. Upon finishing the spike sorting, set model parameters for stage 3 (the transformation and classification model parameters) for each group and download them into the FPGA.

After this stage, we are ready to perform real-time spike inference and acquire the real-time spike trains as the final output.

3.3.2.5 Signal processing stage 3: spike transformation and classification

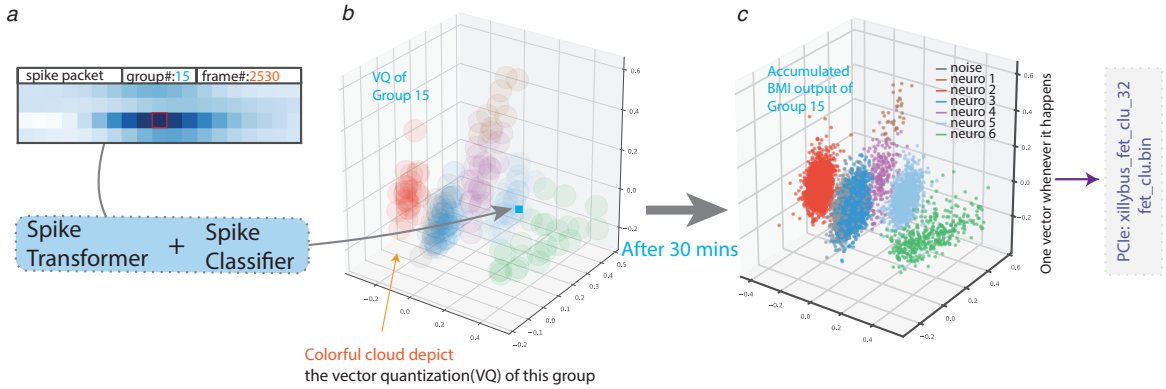


Fig. 3.8 NSP signal processing stage 3: The spike transformer and classifier determines the feature vector and spike identity (id). (a) The spike packet from the previous figure transmitted to the spike transformer and spike classifier. (b) The real-time inference of this specific spike packet. The index of the spike packet shows it was from group 15, hence it is transformed into a feature vector embedded in the feature space of group 15. The vector quantized (VQ) cloud points of group 15 are used to classify the feature vector according to its position in feature space. (c) After 30 min of real-time spike inference, many feature vectors appear in the feature space of group 15. Overall 6 neurons are found in this group, with the other feature vectors classified as noise.

The purpose of stage 3 is to infer the spike features and identities (ids). This is the final stage of real-time spike inference, and the result is stored in the file $fet\_clu.bin$. Meanwhile, the output spike trains can be also used for the real-time decoding.

After we enter stage 3, the index of channels no longer exists; here only the index of *group*# matters. The spike packet index only carries two pieces of information: *group*# (which group the spike event was from) and $frame$# (spike time).

The stage 3 processing inside the FPGA is implemented with a spike transformer and a spike classifier (Fig. 3.5: row 3).

The spike transformer fetches the parameters according to the spike packet *group*#, through which it performs the affine transformation on the spike event waveform and outputs a short 4 dimensional feature vector, which can be visualized in a (reduced) 3D space as a point (Fig. 3.8a,b, the blue point that the grey arrow points to). The resulting feature vector with its *group*# and $frame$# forms a spike feature packet and is sent to the spike classifier.

The spike classifier fetches the parameters for classification according to the *group*# that the spike feature packet carries. The model parameters for classification are calculated through a process called vector quantization (VQ, described later), from which the results can be visualized as many cloud points with different colors meaning different clusters (Fig. 3.8b). The classifier decides the cluster identity of the feature vector according to its position with respect to the VQ cloud model.

From the PC operational perspective, stage 3 involves several sequential steps. First, the transformation and classification parameters for every group in the FPGA were checked. Then the NSP starts and the output data from stage 1, stage 2 and stage 3 was produced. The data files were loaded into Spiketag software to visualize the spike inference result in feature spaces. The desired result is that all the clusters in each channel group are clearly separated. If so, the user customized real-time decoder can be launched.

The final output data of the NSP pipeline is not only spike ids, but also contains $frame$#, *group*# and a feature vector. By designed it this way, loading the $fet\_clu.bin$ is convenient for visually inspecting whether the real-time spike inference yields separated clusters. To illustrate, Fig. 3.8c shows the resulting clusters, from a single group, of 30 minutes of spike inference.

A spike packet will find its group and appear in that feature space as soon as it was detected. The time for a spike packet to be transformed and classified is as short as one microsecond[19]. Such a processing speed of the FPGA implementation minimizes the problem of having several simultaneous spikes detected at the same sampling cycle. To exceed the processing capacity of our FPGA NSP, the recording must generate 1000 spikes within 1 ms

[19]In the FPGA, each operation takes deterministic time. In later sections, a detailed description of the operations of transformation and classification will be introduced.

from 160 channels. Typically, there would be only one to few spikes[20] within such a short time window from 160 channels as illustrated in Fig. 3.7b.

To conclude, the three NSP signal processing stages ensure the experimentalists can iteratively build and check their model step by step, until all the model parameters are configured into the FPGA. On completion of the model building, the NSP would yield real-time spike trains in parallel with the data acquisition. The designed capacity of NSP spike inference guarantees that the NSP can assign spike identity to all spikes appearing in the recording with a short latency.

3.3.3 The NSP algorithmic modules

So far, the overall NSP workflow and the signal processing pipeline have been described. The goal and input/output signal of each algorithmic module in the pipeline have also been introduced. However, some questions remain. For example, how was the filter constructed? How exactly is the spike peak detected? How are the waveforms from adjacent channels grouped together to form a spike packet? How is the transformation matrix computed? How does the VQ model work for classification, and what is the advantage of using the VQ model? What is the latency of each module? These kinds of questions are at the core of the NSP real-time processing and the parameter setting. I will address these questions in this section by examining the specific computation conducted by each module.

3.3.3.1 Bandpass FIR filter: a pseudo-linear phase filter

Filtering is the first step of the NSP pipeline to extract the 500-3000 Hz component, the frequency band of MUA, from the raw neural signal. An ideal bandpass filter should meet four requirements:

1. It should maximize the waveform smoothness (this will facilitate peak detection further down the processing pipeline).

2. It should minimize the waveform distortion (i.e. to preserve the features of the original waveform).

3. It should minimize the latency (i.e. minimize the time for a spike to propagate through the filter, also known as the group delay).

[20] Even in the most extreme case, it is hard to imagine there can be more than 1000 spikes appear in a 1 ms window from only 160 channels. However, even if that would happen, the overflowed spike packet would be buffered in the FIFO and would be processed in the next 1 ms window. The system would fail under one condition, which is for any given 1 ms long window, more than 1000 spikes are recorded.

4. It must be FPGA-implementable.

A causal N-tap[21] finite impulse response (FIR) filter has transfer function (TF) $H(z)$ calculated from the FIR's impulse response $a[n]$ (also called the filter coefficients), given by:

$$H(z) = \sum_{n=0}^{N-1} a[n]z^{-n}, \quad \text{where } z = e^{j\omega}$$

The filter's kth output of channel i is given by:

$$y[k,i] = \sum_{n=0}^{N-1} a[n]x[k-n,i], \quad k = 0,1,2,\ldots$$

where x is the RAW signal and y is the MUA. This computation is usually realized in an FPGA using a simple structure called a multiply–accumulate (MAC), where the impulse response (filter coefficients) $a[n]$ are stored in the FPGA. To understand what a filter does, one must look into the frequency domain, where two frequency dependent quantities pop up: the amplitude response and phase response. The values of the filter coefficients $a[n]$ determine both the amplitude response and phase response. If the impulse response a[n] is either symmetric or antisymmetric, the filter has a linear phase response (Fig. 3.9 ii), meaning the group delay is constant[22]. A constant group delay also implies minimum distortion of the signal in the sense that the amount of time the signal needs to propagate through the device is independent of the signal's frequency components. However, the group delay of a linear phase filter is usually much larger than that of a non-linear phase filter with similar amplitude response. This means the non-linear filter generally has shorter latency. However, the non-linear filter suffers from higher phase distortion. What does phase distortion mean for spike waveform filtering?

[21] N-tap means N coefficients are used. The inputs that multiply the coefficients are commonly referred to as taps. An FIR with N taps requires N multiply-and-accumulate operations.

[22] The group delay is the negative derivative of the phase response. It is a function of frequency and represents the time delay of the amplitude envelope of a sinusoid with certain frequency.

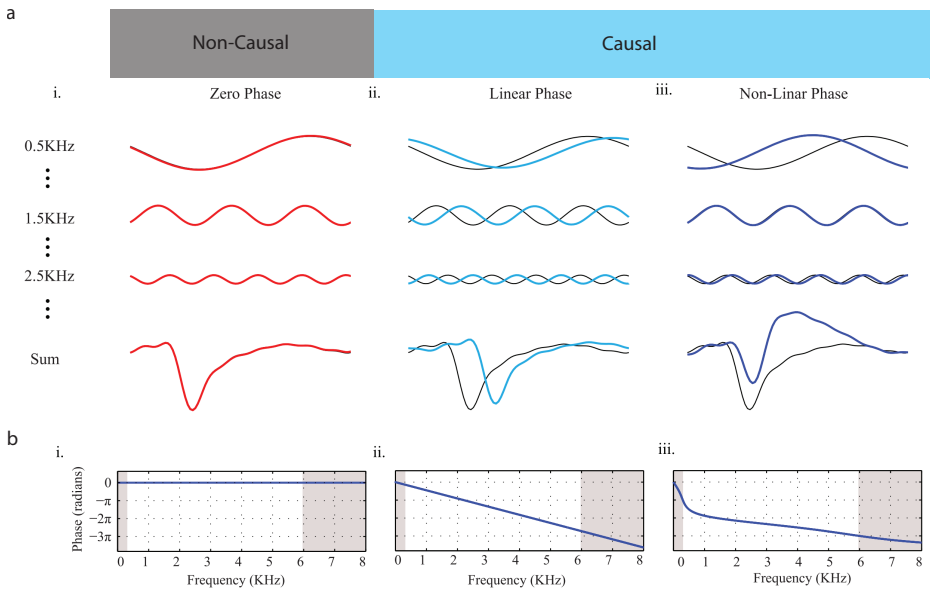


Fig. 3.9 **Filter latency and distortion according to their phase response (adapted from Yael and Bar-Gad, 2017)**: (a) Sinusoidal signals with various frequencies (top) and their summation (spike waveforms at bottom) before (black) and after (colored) being filtered by (i) Zero Phase, (ii) Linear Phase, and (iii) Non-linear phase filter with bandpass at 300-6000Hz. The wider the passband is, the higher the frequency sinusoidal waveforms that will be added and the noisier (non-smooth) the summed waveform this will result in. (b) Phase responses of the filters used in (a). The white band is the passband. The Non-linear filter has short latency but huge distortion.

Yael and Bar-Gad examined the effect of phase distortion on extracellular spike waveforms (Yael and Bar-Gad, 2017). Fig. 3.9 adapted from their paper illustrates the distortion produced by three filters having the same amplitude response but different phase response: a zero-phase filter (Fig. 3.9 i), a linear-phase filter (Fig. 3.9 ii), and a non-linear phase filter (Fig. 3.9 iii). The zero-phase filter has zero group delay (latency). However, a zero-phase filter is non-causal thus is not implementable in the FPGA or any hardware. The non-linear filter introduces a short latency compared to the linear filter. This filter also introduces significant phase distortion (i.e. different frequency components have different time delays), resulting in a distorted spike waveform. To explain, a spike waveform (Fig. 3.9 iii) composed of the summation of many sinusoidal components of different frequencies and filtered by a non-linear filter shows that the component at 0.5 kHz (Fig. 3.9 iii 0.5 kHz) has a longer delay than the component at 1.5 kHz (Fig. 3.9 iii 1.5 kHz). Hence, the resultant waveform - the linear sum of each filtered component - is distorted (Fig. 3.9 iii Sum). The phase distortion is strongly unfavorable in the case where the shape of the waveform is important. This is especially important to consider as there is no guarantee that each input waveform bears the

exact same frequency components. In fact, spikes waveforms suffer from many different kinds of noise (Harris et al., 2000; Lewicki, 1998; Rey et al., 2015), such as bursting, subtle brain movement and electrode drifting, for which the noise can be amplified non-linearly by the phase distortion. Yael and Bar-Gad also proposed a post-correction method to remove the phase distortion in the filtered MUA (Yael and Bar-Gad, 2017), but this method is essentially an offline analysis and hard to implement in the FPGA.

Here I designed a pseudo-linear phase FIR filter, with an asymmetric impulse response, as a solution to simultaneously minimizing the latency (group delay) and distortion (phase non-linearity) without compromising smoothness (in a narrower passband, 500-3000 Hz) of the filtered signal. Because the phase response of the filter is pseudo-linear, it gives rise to a nearly constant and low group delay in the passband and a non-linear phase response in the stopband. In addition, according to the Paley-Wiener Theorem, no causal filter can have zero power in the stopband, nor can the spectrum be perfectly flat in the passband. Therefore, the filter design goal is to optimize the filter coefficients towards a desired amplitude response and an almost linear phase response in the passband. Approximating the desired amplitude response and phase response simultaneously has been extensively studied in digital filter design (Ahmad and Antoniou, 2007; Berchin, 2007). Here the Matlab built-in function 'arbmagnphase' was used[23] to design the asymmetric impulse response coefficients.

First, in the passband, 500 Hz-3000 Hz, the designed asymmetric filter (Fig. 3.10b: red) exhibits pseudo-linear phase response (Fig. 3.10c: red) and comparable amplitude response to the symmetric FIR filter with same number of taps (Fig. 3.10c), yielding a sufficiently smooth spike waveform with 0.84 ms latency.

[23]The Matlab function 'arbmagnphase' asks the designer to specify an arbitrary set of desired amplitude and phase response, it then constructs the FIR coefficients that approximate the input amplitude and phase response.

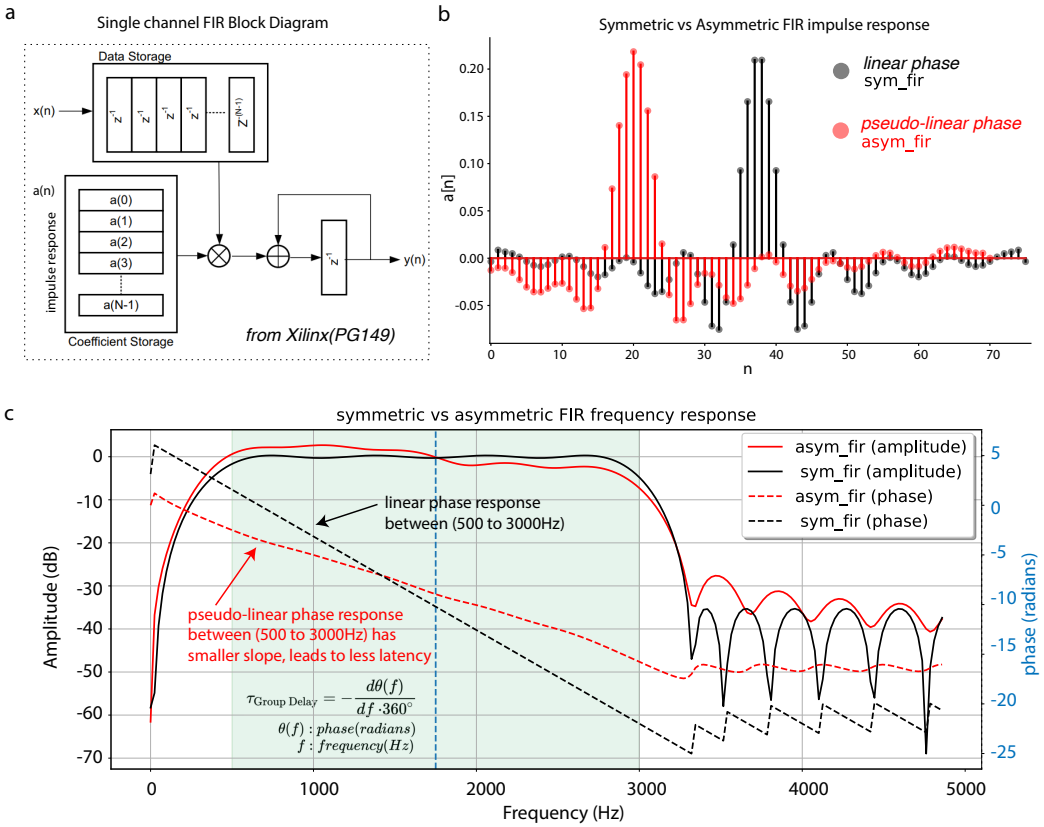


Fig. 3.10 **Filter structure, impulse response and frequency response**: (a) The basic FIR Block Diagram is implemented using the Xilinx FIR Compiler, wherein the multiply–accumulate (MAC) engine is the algorithmic kernel used to compute a single channel FIR (according to Xilinx document PG149). Multiple MACs are utilized for multichannel FIR, with the same set of coefficients. For engineers who use this IP, the major task is to design the impulse response. (b) The impulse responses of two types of FIR filters: the symmetric impulse response filter (black, sym_fir) is a linear phase filter, whereas the asymmetric impulse response filter (red, asym_fir) is a pseudo-linear phase filter. (c) The Amplitude-frequency and Phase-frequency response for the pseudo-linear phase filter (red) and linear-phase filter (black). The pseudo-linear phase filter drops the group delay (the latency) by nearly half while the amplitude response within the range of 500 to 3000 Hz remains similar to the linear phase filter.

The typical output of the designed pseudo-linear phase filter reveals the following. First, the amplitude response specification allows only the MUA to pass (Fig. 3.11a), ensuring sufficient smoothness of the filtered waveforms (Fig. 3.11b). Second, with the same slight distortion of the spike waveform, the pseudo-linear phase filter exhibits shorter latency - 0.84 ms - nearly half of the linear phase filter (Fig. 3.11b).

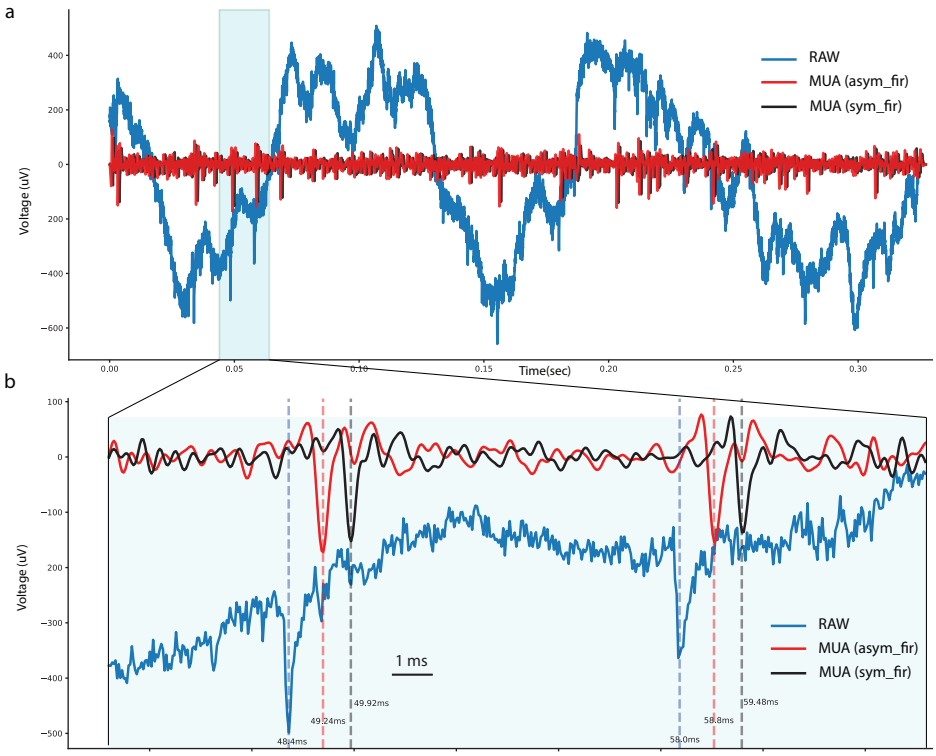


Fig. 3.11 **Examples of the performance of the linear and pseudo-linear phase filters on real data:** (a) Both symmetric FIR (sym_fir) and asymmetric FIR (asym_fir) filters successfully convert the RAW signal into a MUA signal. Here only one channel is shown. (b) The symmetric FIR (sym_fir) filter's spike latency is 1.5 ms while the asymmetric FIR (asym_fir) filter's spike latency is 0.84 ms. Both FIR filters preserve the shape of the spike waveform. The filtered MUA spike waveform is smooth enough for the negative peak detection.

Although FIR design is a highly developed field with many out-of-box packages provided by both MATLAB and Python, and linear phase (symmetric) FIR filters are routinely applied to extract MUA from raw extracellular recordings, applying a pseudo-linear phase (asymmetric) FIR filter to simultaneously minimize the distortion and the latency while ensuring adequate smoothness of the filtered extracellular spike waveform is, to my knowledge, novel in the field of neuroscience.

The multiple pages dedicated to discussing the filter design highlights its importance. Indeed, this filtering module introduces most of the algorithmic latency in the NSP pipeline and thus careful design is required. Furthermore, the resultant multi-unit spike waveform's smoothness can affect the spike detector's performance[24] in the next signal processing stage and is thus crucial to good overall performance of spike inference.

[24]Since the spike detector is based on peak detection, a waveform that is not sufficiently smooth can have multiple local peaks detected around the true peak, which would cause a significant waveform alignment issue.

Before the spike detection, the MUA must be further denoised to mitigate motion artefacts. In the recording of a freely moving animal, large motion artefacts can often be present in the filtered MUA as shown in Fig. 3.6.

3.3.3.2 Reference subtraction: the digital version of differential recording

Differential recording measures the voltage difference between a pair of electrodes, where one electrode is the reference channel of the other electrode. This method has a long history among methods to eliminate motion noise. In the 1960s, John O'Keefe made use of just-invented modern field-effect transistors (FET), which are light-weight and sensitive enough to allow robust differential recording from freely behaving animals[25] (O'Keefe and Dostrovsky, 1971).

Here a 160-channels digital version of the same algorithm was adopted, in which every channel was assigned with a reference channel. Unlike in differential recording, the reference subtraction was conducted explicitly in the FPGA. The algorithm was designed as follows:

$\forall$ current input channel $i \in [0, 160)$, $\exists$ reference channel j

$if(i! = j)$

$$denoised\_MUA[k-1,i] = MUA[k-1,i] - MUA[k-1,j]$$

$else\ if(i == j)$

$$denoised\_MUA[k-1,i] = MUA[k-1,i]$$

The model parameters required for this algorithm are given by 160 2-tuples (i, j), where i is the channel# and j is the reference channel#. The subtraction always acts on the previous frame, therefore the algorithmic latency for this algorithm is 40 microseconds (a single sampling interval). If the parameters are not set, this algorithm would delay the data flow by one frame and output the previous frame of MUA.

The typical test results of the FPGA reference subtraction are shown in Fig. 3.6, in which the motion artefact was largely eliminated. The success of the reference subtraction has two premises. First, the reference channel itself should contain few detectable spikes, especially if this reference is used by multiple channels. If this is not the case, after reference

[25] In the paper (O'Keefe and Dostrovsky, 1971), in which place cells were discovered, the authors wrote:"Maximum rejection of muscle and movement artefacts was obtained by feeding the signals from two adjacent microelectrodes into a high input impedance differential FET preamplifier mounted directly on the microdrive. ... More recently, satisfactory recordings have been obtained by manipulating one electrode into place in the cortical white matter and fixing it there to serve as a reference for each of the other electrodes in turn."

subtraction, one would see copies of inverted spike waveforms on all the channels which use this reference channel. Second, the motion artefacts should be of relatively similar amplitude between the recording channels and their reference channels and these signals should be in phase.

3.3.3.3 Spike detection: append the peak flag to all samples

The threshold for each channel was automatically set according to an empirical equation proposed by Quian Quiroga et al (Quiroga et al., 2004):

$$\text{Thr} = -4.5\sigma_n; \quad \sigma_n = \text{median}\left\{\frac{|x|}{0.6745}\right\}$$

Here the x is a few tens of seconds of multichannel denoised_MUA. After the automatically calculated threshold is downloaded into the FPGA, the FPGA spike detection would operate using the threshold. The Spike Detection's operation, as with all other algorithmic modules, is triggered by the simultaneous arrival of data samples (denoised MUA) and the associated indices (frame#, channel#).

In a first step, the channel# is used to fetch the associated threshold and group#. Then, the threshold is used to carry out the specific spike peak finding computation. The peak finding operation outputs the same MUA stream but with one crucial extra bit added (note this physically adds one more bit associated with each data sample), which is referred to as the MUAP stream, where P denotes peak. This last important bit will be set to 1 only if this data point is determined as the peak relative to its previous data and following data.

In other words, the main job of the spike detector is to find the peak sample below the threshold of each denoised_MUA data stream and flip the peak-flag bit from 0 to 1. To this end, a finite state machine (FSM) was designed containing a state vector of 160 channels that all evolve independently. Fig. 3.12 illustrates the peak finding FSM: When the received data sample from one channel is above the threshold, the state of that channel is set to $s0$ (note that extracellular spikes are generally negative-going, therefore a candidate spike event occurs when the MUA stream goes below the threshold). When going below the threshold (Fig. 3.12), the state jumps from $s0$ to $s1$. It stays at $s1$ if the value keeps going down and jumps back to $s0$ if the value goes back above the threshold level.

The transition from $s1$ to $s2$ occurs when a turning point below threshold (local minimum or spike peak) appears, indicating a spike, under the assumption that the spike waveform is smooth enough, which is ensured by the FIR filter design. If the amplitude continues increasing, we will reach the state $s3$. The transition from $s2$ to $s3$ confirms the formation of a negative peak. This transition causes the last crucial bit (peak-flag bit) of the most recent

$s1$ sample (the local peak sample) to flip from 0 to 1 indicating the occurrence of a spike (Fig. 3.12b). As each sample goes through the spike detector's buffer, its peak-flag bit is by default 0. Hence, only the local peak sample of each channel would have the last bit as 1.

The spike detector FSM ignores transient electronic artefacts (Fig. 3.12b). Sometimes, the sudden drop of the voltage (from a non-spike) will cause the state transition from $s0$ to $s1$, then it quickly goes back above the threshold, which leads to a following state transition from $s1$ back to $s0$. In this case, peak flag would not be labelled.

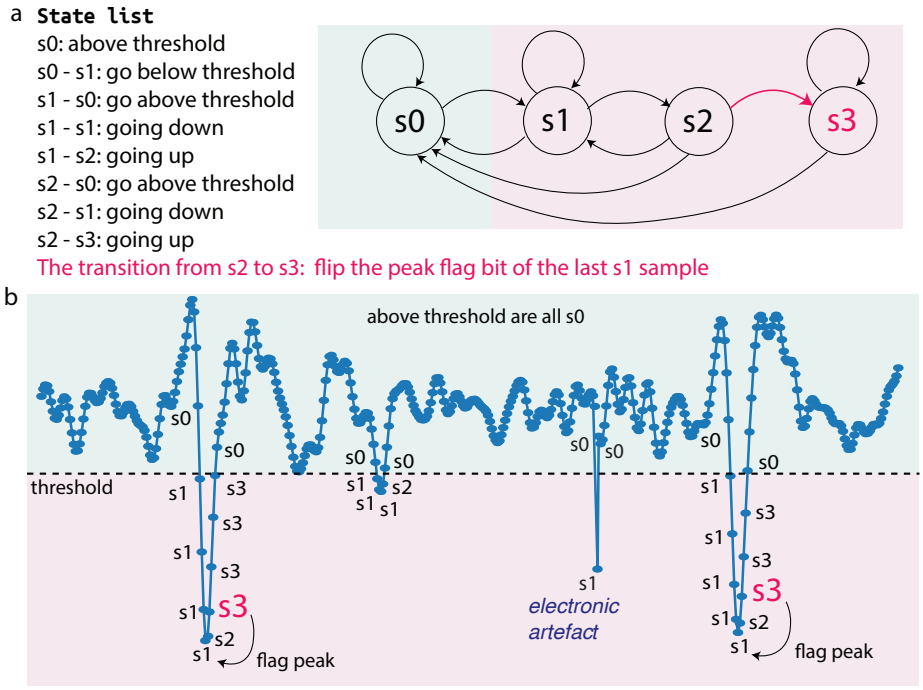


Fig. 3.12 **The FSM of the spike detector for every channel**: (a) The state list and the state transition diagram. (b) An example of spike detection using the designed FSM. Light blue covers the $s0$ states, which means the sample is above threshold. The light yellow covers the samples under the threshold, which can take states $s1$,$s2$ and $s3$. The red arrow is the key state transition $s2$ to $s3$. This transition 'flags' the local peak sample, which is the previous $s1$ sample.

Because the peak samples are always labelled two samples after the true peak (Fig. 3.12b: flag peak operation), the latency of the FSM is 80 microseconds (two sampling intervals). Also, to flip the prior $s1$ sample, the algorithm needs to go backwards in the buffer for two steps. Therefore, the minimum buffer depth for the spike detector is 3. Notably, here we only need to detect and flag the peak when the data streams flow through, and we leave the spike extraction to the next module, which requires a larger buffer depth to load the full spike waveform. Importantly, the spike detector is the last module yielding channel-wise

continuous data streams. After the spikes have been extracted, the NSP will only accept and process the packet data.

3.3.3.4 Spike packet extractor: group-specific buffer for packet extraction

The Spike Packet Extractor draws out a spike packet from the MUAP continuous stream according to each MUAP sample's peak-flag bit. The spike packet is a packet of data that contains the index (frame#, group#), along with the spike event waveforms from that group.

The key component here is a buffer with overall size of $40(groups) \times 19(samples) \times 4(channels) \times 33(bits)$. This corresponds to 40 FIFOs, wherein each FIFO holds 19 samples from 4 channels. Each sample is 33 bits with the last bit indicating whether it is a spike peak. According to the associated group#, the data streams move into the corresponding FIFO. In other words, each FIFO receives four MUAP streams from one specific electrode group (Fig. 3.13 a). The MUAP data streams from 40 groups move through these 40 FIFOs in a shift-register manner: each arriving data sample is first stored in a local register and then moves into a deeper position of the FIFO when four samples from four channels all arrive (Fig. 3.13 a). This can be visualized as the overall four channel waveforms of each group moving (a vertical line consists of 4 samples in (Fig. 3.13 a)) into the FIFO.

The data content of a single FIFO is a 19*4 matrix that represents the candidate spike event waveforms from 4 channels of a group (Fig. 3.13a). This matrix can be captured, packaged with its index (frame# and group#), and sent out as a spike packet (Fig 3.13c). The question is when to start forming the packet such that the spike event waveforms are centered at its largest peak. This is achieved by designing a two state FSM. Each FIFO has an independent FSM. The FSM's state updates when a new frame of data enters the FIFO (Fig. 3.13a)

When the FSM is at state $s0$, it checks whether there is a peak sample at the middle position of the FIFO. If true, it would further check whether that peak is the biggest peak among all peaks in a small window after the middle position of the FIFO (Fig 3.13a: the yellow colored pivotal region). If still true, the $s0$ to $s1$ transition happens. This transition initiates the packet forming process, which takes only nanoseconds. Immediately after packet forming, the state jumps back to $s0$.

This designed FSM guarantees that when spike packet forming starts, the pivotal sample (the largest peak) is always at the center of the packet (Fig 3.13c). Hence, a group of spike waveforms are aligned. To form a packet, the frame# and group# together form the index of the packet, in which the frame# of the packet is the frame# of the pivotal sample of this group. Therefore, the downstream circuits and the PC always have the space-time information about

every packet. Moreover, the processing latency of this module is 4 nanoseconds, because the state transition and packet forming together only take one clock cycle (4 nanoseconds).

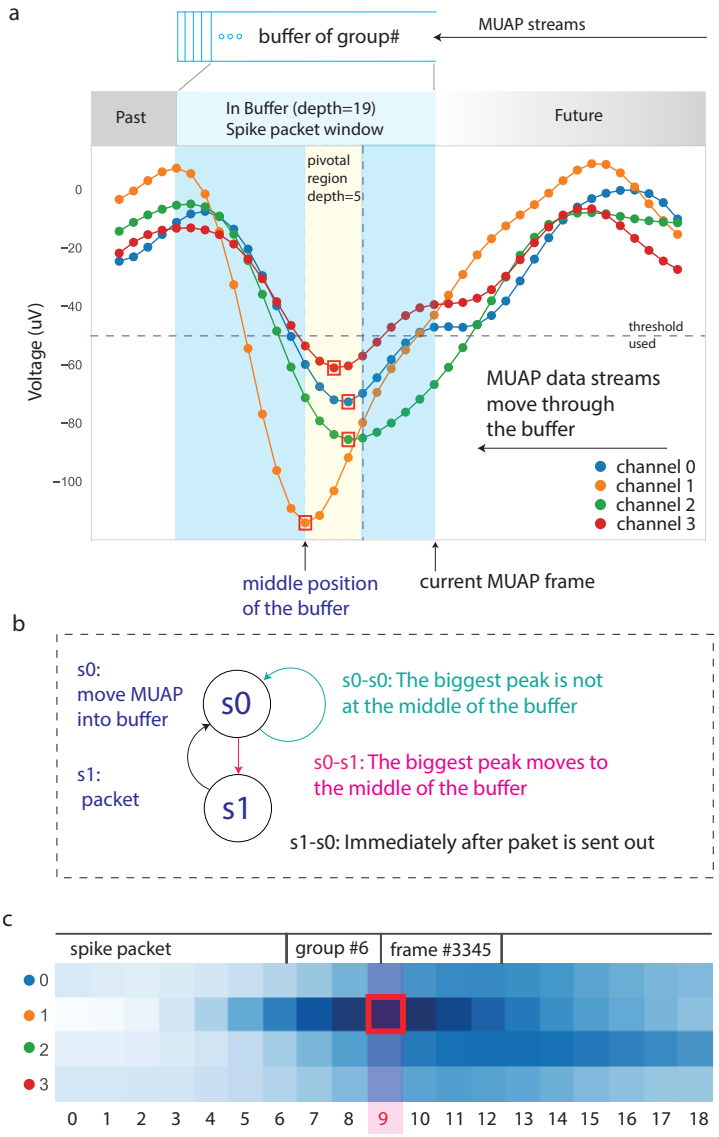


Fig. 3.13 **The spike packet extractor generates spike packets from the MUAP stream:** (a) The buffering process of the spike packet extractor: From right to left, the data moves into the buffer region which contains 40 FIFOs for the 40 groups (only one group is shown). When the largest peak, in the pivotal region (yellow), appears at the middle position of the group buffer, the packet forming is triggered. (b) The FSM of this module has two states: $s0$ for moving the data into the buffer, $s1$ for starting the packet forming. (c) The spike packet of the spike event in (a). The pivotal sample, i.e., the largest peak (red rectangle) is always at the column 9, hence the spike waveforms are aligned.

Every spike packet is sent to the PC and stored in the 'spk.bin' file as well as to the downstream circuit. The downstream circuit, which transforms the spike packet into a feature packet, is triggered whenever a spike packet is received.

3.3.3.5 Spike transformer: normalized PCA

The spike transformer takes the spike packet as the input, uses the group# contained in the packet to fetch the transformation parameters for that group, and transforms the spike event waveforms (19 by 4 matrix) into a 4D-vector. This 4D-vector is accompanied with its index (frame# and group#) to form a feature packet.

The transformation parameters (P, b, a) are constructed using Principal Component Analysis (PCA) of the stored spike waveforms in the 'spk.bin' for each group. The overall spike waveform matrix $X \in \mathbb{R}^{n\times m}$ for each group, where n is the number of spike events and m is the total number of samples of each concatenated waveform (m = 4 channels × 19 samples = 76). After mean subtraction from each data dimension (for PCA to work properly), singular value decomposition (SVD) is applied.

$$X - \bar{X} = USV^T$$

First the transformation matrix P was computed by simply selecting the first four dimensions of the rotation matrix V[26]. Then a mean-subtraction vector b was computed by (b is important because we have done the mean-subtraction when using the historical data matrix X to construct P, hence the newly arrived spike event waveform x also needs to undergo the same mean subtraction):

$$b = -\bar{X}P$$

In the end, we need to normalize the final PCA transformed vector for each group. The scale factor was computed by:

$$a = \frac{1}{\max(XP+b) - \min(XP+b)}$$

This parameter construction process was repeated for every group and downloaded to the FPGA. Once configured, whenever a new spike packet arrives at the transformer, the transformer retrieves the corresponding (P,b,a) according to the group# and then performs the affine transformation on the concatenated spike waveform x:

[26] V is an orthonormal matrix. Multiplying by it will not change the norm of the vector, therefore its function is to rotate the vector.

$$y = a(xP + b)$$

This transforms the 76-D vector x into a 4-D vector y. The exact same index (frame#, group#) of the input spike packet is packaged with y to form a feature-packet as the output. The current FPGA implementation of this algorithm transforms one spike-packet and outputs one feature-packet in less than 500 nanoseconds. Within one sampling cycle, at least 80 spike packets can be processed by the transformer. This processing throughput is much higher than the possible spike rate summed over 40 groups of 4-electrodes.

3.3.3.6 Spike classifier: vector quantization (VQ) + k-nearest-neighbour (kNN)

Spike classifier the last step of the whole NSP pipeline. It takes the spike feature packet and performs the spike classification. The output is a copy of the feature packet appended to the classified spike identity (spike-id packet). If the classification parameters are not configured, the spike identity would be 0 by default. A k-nearest-neighbour (kNN) algorithm was implemented for the spike classification and vector quantization (VQ) was implemented to compress the number of model parameters for classification[27] to make the kNN algorithm FPGA-implementable.

Because of drift, bursting, spike waveform adaptation and many other reasons, the decision boundary between single-unit clusters in feature space is not always linearly separable. All linear classifiers assume the data is linearly separable and is not desirable when there is a non-linear boundary between any two clusters. kNN is a simple non-linear classifier that labels a data point according to its k nearest neighbours' identity and can perform well when the decision boundary between classes are non-linear. It has been shown that kNN generally outperforms (in accuracy) the many types of linear classifiers when simulating real-time spike classification (Navajas et al., 2014). However, the algorithmic complexity of the kNN classification is much higher than other classifiers, as it requires comparing each newly acquired data sample with all existing data samples.

For this reason, an FPGA kNN implementation can be slow and memory intensive. The algorithm needs to access all the labelled data whenever a new data sample is acquired. On-chip memory is one of the most valuable physical resources in the FPGA, thus memory-intensive algorithms are not practical. In order to implement kNN as the spike classifier module in the FPGA, compression of the model (i.e., the labelled data for kNN) is required. Here vector quantization (VQ) (Fig 3.14b to c) was used to compress thousands of points

[27] Here the spike sorting should be already finished (using Spiketag) and thus the cluster model of each group should already be there.

to a few tens or hundreds of points in the feature space. Although highly compressed, the quantized vectors can faithfully reflect the shape of each cluster and draw a highly accurate boundary between clusters, which is not necessarily linear.

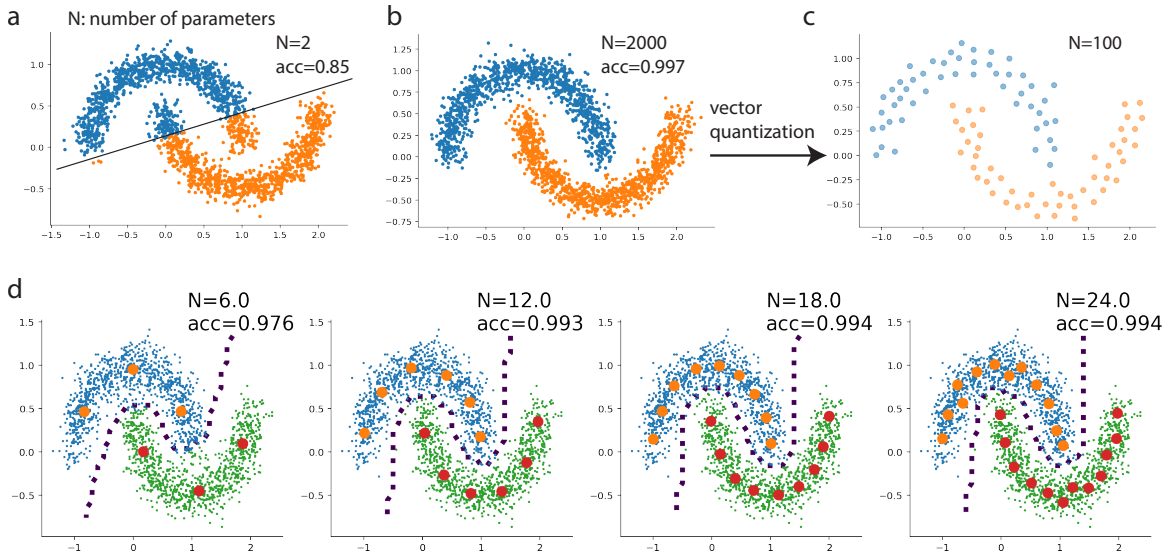


Fig. 3.14 **kNN with VQ compression provides a highly efficient classification model with non-linear decision boundaries:**(a) The linear model learns a hyperplane (black) from the training set and uses it to classify the test set. Both training data and test data were generated from two crescent-shaped clusters (only test data is plotted here). The decision boundary between two moon shape clusters is not linearly separable. Therefore, the linear model performs poorly on this toy example (accuracy = 0.85). (b) kNN performs well on the test data (regenerated from the same distribution) by assigning the data the same label as its nearest neighbour (here k=1). However, the kNN model uses 2000 parameters since the size of the training set is 2000. (c) VQ can compress the model into 100 quantized vectors. Here I apply k-means to generate the quantized vectors cluster by cluster. Hence the number of quantized vectors is controllable. In this example, each cluster is represented by 50 quantized vectors. Although compressed 20 times, the quantized vectors precisely describe the shape of each cluster. (d) VQ compresses the kNN model to 6, 12, 18 and 24 quantized vectors and yet yields a highly accurate classification result on the test data (accuracy = 0.976, 0.993, 0.994, 0.994 respectively). The non-linear decision boundary (dotted line) changes with the choice of the number of parameters (N).

To illustrate, two crescent-shaped clusters containing 2000 points in a 2D space were synthesized[28]. This data was used as the training data, and a test data following the same distribution was used to test the classification accuracy. Fig 3.14 shows the performance of various choices of models on the test data. The typical result of a linear model is shown in (Fig 3.14a): although the number of parameters is as low as two because the linear

[28]In comparison, the spike feature sample from upstream circuits has four dimensions

model uses a hyperplane to assign the labels. The accuracy (i.e., the percentage of correctly classified samples in the test data) is only 0.85. In contrast, the accuracy of kNN (k=1) is high (0.997) due to the non-linear decision boundary. However, because the kNN algorithm simply memorizes the whole training data set and compares the test sample to each sample in the training data, the number of parameters is the size of the training set. In this example, a kNN model contains 2000 points (all points in the training set). A 30-minute 160-channel neural recording can have hundreds of thousands of spikes. Keeping so many parameters in the FPGA can easily overflow the on-chip memory. Even if we use off-chip memory to compare each sample in the training data with every feature packet, this is too expensive for real-time inference.

To address this problem, the VQ algorithm was used to compress the training data into a model with a small number of parameters. Here the k-means algorithm was used as a vector quantizer for each cluster, in which the k-means-generated centroids were used as the quantized vectors (Fig 3.14c,d). Because the number of centroids of k-means is adjustable, we can choose an arbitrary number of quantized vectors for each cluster. For example, Fig 3.14c shows a 100 k-means quantized vectors constructed from the training set containing 2000 samples. The next question is how accurate the compressed model is. Fig 3.14d demonstrates that even with only three quantized vectors per cluster, the model accuracy can be as high as 0.976. With a few tens of quantized vectors, the non-linear shape of the cluster is well-represented and the accuracy can be as high as 0.994. The non-linear decision boundary of the kNN-VQ model is shown as the dotted line (Fig 3.14d), while the linear decision boundary of a two-layer linear neural network with 100 hidden units is shown in black (Fig 3.14a). Notably, adding non-linear units into the hidden layer in the multilayer neural network can also yield similar accuracy as kNN-VQ. However, kNN-VQ is much easier to implement in an FPGA, and it works as a "white box" compared to the black box neural network model.

To apply the same kNN-VQ to real data, we want to automatically set the number of quantized vectors for any cluster. Here the fixed maximum total number (the quota) of quantized vectors was set to 200 for each group. The initial allocation of the total of 200 quantized vectors was in proportion to the spike count of each cluster, including the first background "noise" cluster[29]. Because the first cluster (noise) usually contains more spikes than the rest of the clusters, the number of the quantized vectors allocated to it is also much higher. Once the VQ is done, a cross-validation accuracy test is applied to each cluster. The clusters which fail a preset minimum desirable accuracy[30] obtain five more VQ quota from

[29]During the spike sorting, we always put the noise into the first cluster for every group.

[30]This should be decided by the experimentalist. For reference, we always use threshold larger than 0.94.

the first cluster (noise) and the next iteration of VQ construction and evaluation starts. This automatic VQ quota allocation usually finishes in just one or a few iterations, and when it returns, the minimum preset accuracy for knn-VQ classification is guaranteed.

A limitation of the automatic reconstruction of VQ occurs when two clusters are not separable; it is possible that no matter how we balance the number of quantized vectors among different clusters, a minimum accuracy such as 0.94 can never be met. But then we will know the spike sorting of this group is inadequate and we can choose not to use this group or re-sort and reject the problematic clusters (in Spiketag; re-sort only when time is permitted in an online experiment). In practice, this scenario (configuring an inadequately sorted group to the FPGA NSP) is quite rare. An experienced spike sorter would produce clusters that are either linearly or non-linearly separable. Fig. 3.15 illustrates two typical examples of VQ clusters automatically constructed from the spike-sorted clusters, with a VQ quota of 200 (i.e., the total number of quantized vectors per group).

In order to make the FPGA implementation easier, the VQ quota for each group is fixed. On completion of automatic VQ quota allocation, the VQ cluster with its VQ label would be downloaded into the FPGA for real-time inference. The real-time inference goes as follows. When a feature packet arrives, according to its group#, the knn-VQ FPGA classifier will first fetch the group's VQ clusters and labels. Then the feature vector would be compared with each quantized vector in each of the VQ clusters. The inferred spike identity is determined by the label of its nearest quantized vector. To determine the nearest neighbour only requires calculating the Euclidean distance between two 4D vectors 200 times. The current FPGA implementation finishes this nearest neighbour calculation within 500 nanoseconds.

With the latency of both the spike transformation and classification being 500 nanoseconds, our NSP can identify a spike packet as soon as it is extracted. This latency is much shorter than the acquisition interval. Each identified spike feature packet (spike-id packet) contains the spike's electrode group number. On PC, the group numbers in spike-id packets can be used for real-time visualization (in Spiketag) of the NSP spike inference in many feature spaces of those configured groups (see an example of a multi-group real-time spike inference at https://www.youtube.com/watch?v=EItzDSH_un0). This visualization can be made as soon as the VQ vectors, the final part of a spike-sorted model, are downloaded into the FPGA. It is a fast and convenient way to check the overall quality of the spike-sorted model and the state of the FPGA NSP.

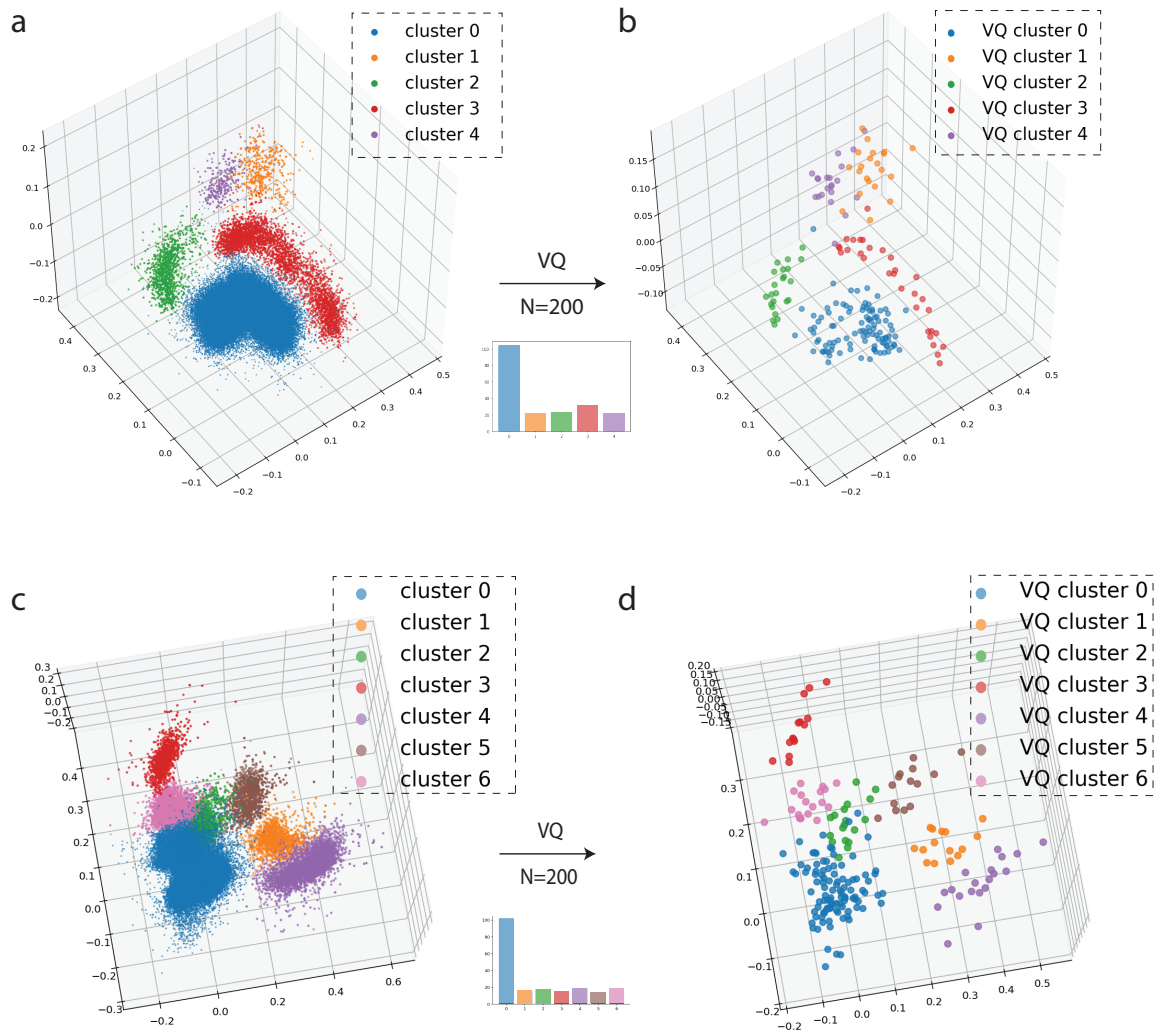


Fig. 3.15 **Examples of kNN-VQ spike classification on both acute and chronic data:** (a) The clustering result of an individual group; cluster 0 is noise. Data recorded from cortex in an acute preparation. Cluster 3 exhibits a typical non-linear shape presumably as result of a combination of neuronal bursting and continuous amplitude drift (estimated by visually checking how this cluster evolves through time). (b) The VQ compressed clusters, consisting of 200 quantized vectors, depicts the shape of each cluster and captures the non-linear boundary between different classes. The histogram between the original cluster (a) and VQ model (b) shows the distribution of the number of quantized vectors used to describe each cluster. This VQ model gives rise to a minimum 0.97 accuracy rate among all clusters. (c) The clustering result of an individual group; cluster 0 is noise. Data recorded from the hippocampus in a chronically implanted animal. (d) The VQ compressed clusters consist of 200 quantized vectors, depicting the shape of each cluster and capturing the boundary between different classes. The histogram between the original cluster (c) and VQ model (d) shows the distribution of the number of quantized vectors used to describe each cluster. This VQ model gives rise to a minimum 0.94 accuracy rate among all clusters.

3.3.4 The NSP digital protocol

So far, I have introduced the NSP interface for acquiring and moving the data, the NSP workflow and signal processing pipeline for converting the raw data into the spike identity, and the algorithmic detail of each FPGA NSP module. These would be sufficient for a software simulation of the hardware function. However, for the actual hardware (where the data is constantly moving clock cycle by clock cycle) to work properly, we have to ensure that the correct data appears in the correct place at the correct time. Digital protocols were implemented as finite state machines (FSMs) to ensure that each circuit/module receives and produces data with nanoseconds precision.

3.3.4.1 Clocks and clock domains

The FPGA NSP modules are driven by two different clock frequencies, therefore can be divided into two clock-domains[31]. While the data acquisition of the SPI IP is at 25000 samples/sec (25000 Hz), the data processing and PCIe IP is governed by a 250 MHz clock. These two clocks are asynchronous, meaning a Clock-Domain-Crossing (CDC) design is required. Furthermore, because the INTAN SPI acquisition protocol uses 16 bits to form a data sample, a higher clock rate than 25000 Hz is required to operate the SPI IP to sample bit by bit and then bind 16 bits together as a single data sample. In my design, a 250 MHz source clock (PCIE_CLK) generated from the PCIe IP was fed into a clock generator[32] to provide a 70 MHz clock (SPI_CLK) for the acquisition circuits in the SPI IP and feedforward a 250 MHz clock (NSP_CLK) to the NSP algorithmic modules for data processing (Fig. 3.16a blue). When crossing clock domains, a dual-clock asynchronous FIFO was used as a buffer to connect two clock domains and ensure that no data is lost when transferring the data from one clock domain to another (Fig. 3.16a: across the clock boundary). Because the read clock frequency (250 MHz) is higher than the write clock frequency (70 MHz) of the asynchronous FIFO, most of the time this FIFO is empty.

[31] A clock domain is a portion of the circuitry driven by a single clock.

[32] A Phase-Locked Loop (PLL) was used here, which is commonly used in FPGA design as a clock generator.

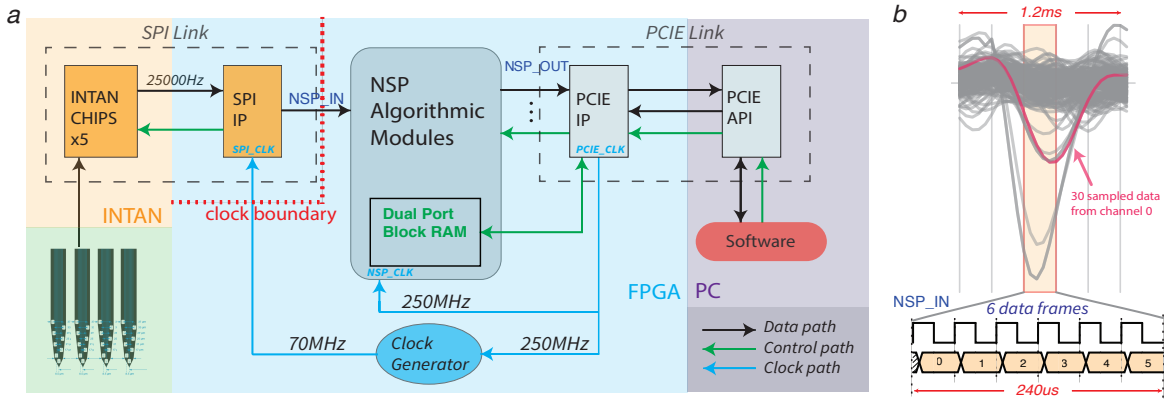


Fig. 3.16 **The NSP clock domains:** (a) The high-level schematic of clock path, data path and control path. The clock path (blue arrows): The FPGA generates three clocks from the PLL in the PCIe-IP: 70 MHz SPK_CLK, 250 MHz NSP_CLK and 250MHz PCIE_CLK. These three clocks serve the sampling, processing, and transmission to the PC, respectively. The data pathway (black arrows): The extracellular voltage signals are amplified and converted from analog signals into digital signals via 5 32-channel INTAN chips. These 5 INTAN chips link to the SPI IP inside the FPGA and move the raw data into the FPGA at 25000 Hz per sample (NSP_IN signal). These data cross the clock domain (the red dashed line) and enter into the NSP Algorithmic Modules. After the NSP signal processing pipeline, the processed data (NSP_OUT) are transmitted at 250 MHz to the PC. (b) The data sampling is frame-based. A segment of 160 channels of extracellular voltage signals (grey), sampled frame by frame, is shown in both analog (above: extracellular waveforms) and digital (below: NSP_IN signal plotted as an orange) form. One waveform from a single channel is highlighted (red).

3.3.4.2 The NSP acquisition protocol

The NSP acquisition protocol was developed on top of the INTAN SPI sampling protocol. It aims to organize the data acquisition of five INTAN RHD2132 chips (160 channels total). The protocol goes as follows. A 70 MHz clock (SPI_CLK) is generated from the FPGA to drive five INTAN RHD2132 chips ($chip0, 1, 2, 3, 4$) through a customized SPI IP[33]. Each INTAN chip initiates the sampling protocol separated at an interval of 14.286 ns (a single SPI_CLK cycle)[34]. Notably, each chip uses the same sampling protocol (Fig. 3.17b), which takes 80 SPI_CLK cycles to sample 16 bits from one channel of a chip. These 16 bits constitute a single data sample of the recorded extracellular waveform from a specific channel among the 160 channels. Every 40 µs, one frame of data (i.e., 160 data samples from 160 channels) are sampled from the five chips in the order illustrated in Fig. 3.17a.

[33] Aarón Cuevas López from Open-Ephys (https://open-ephys.org/) provided valuable support walking me through the Open-Ephys SPI acquisition module written in Verilog. The SPI IP in our NSP was developed on top of their work which was originally developed by INTAN Inc.

[34] In other words, $Chip0$ starts the sampling protocol first, then 14.286 ns later $Chip1$ starts the sampling protocol, then $chip2, 3, 4$ starts one after another.

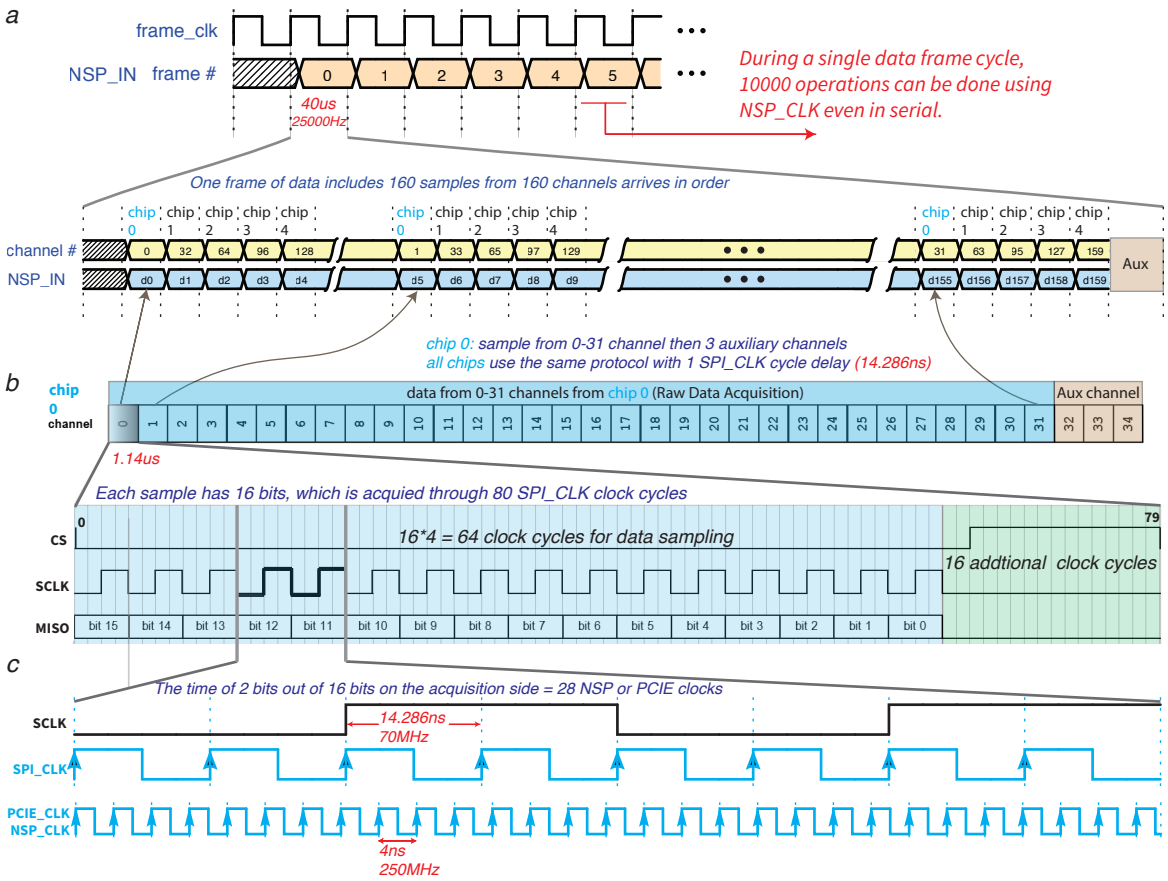


Fig. 3.17 **The NSP acquisition protocol:** (a) The acquisition order of 160 data samples (blue: d0, d1 ... d159), from five INTAN chips, in a single data frame (40 µs). Each chip contains 32 data channels and 3 auxiliary (Aux) channels. The FPGA SPI protocol first samples the 1st channel from *chip*0 to *chip*4 (yellow) and then the 2nd channel from *chip*0 to *chip*4 up to the 31st channel from *chip*0 to *chip*4. (b) Illustration of how a single chip (*chip*0) packages a 16 bit ADC sample into a complete data sample. Each chip has 35 sampling cycles (i.e., 32 data channels + 3 auxiliary channels); sampling from a single channel takes 1.14 µs, in which 16 bits are sampled over 80 SPI_CLK cycles (1.14 µs = 80 * 14.286 ns); sampling from all 35 channels of a chip costs 2800 (35 * 80) SPI_CLK, which is 40 µs (2800 * 14.286 ns), which is why the sampling rate is 25000 Hz. Every chip uses the same sampling protocol except that they are separated by one SPI_CLK cycle (14.286 ns). To make the acquisition work, the protocol signals (CS, SCLK and MOSI) are generated by a FSM and are fed into the INTAN chips. The returned bit stream (MISO) packages every 16 bits as a single data sample. (c) The examples of three clocks (PCIE_CLK, NSP_CLK, SPI_CLK) during a period of acquiring two bits (bit 11,12 of the MISO signal).

The discrepancy between the slow sampling speed (40 µs per frame; 25000 Hz) and the fast processing speed (4 ns interval) is the core reason that the FPGA is a powerful solution for online neural signal processing. Up to 10000 operations can be scheduled during the

acquisition of a single frame of data. Scheduling more operations on each 4 ns processing cycle can produce even faster computation speed. Furthermore, the slow sampling and fast computation are conducted simultaneously in the same chip, introducing no communication delay between these two processes. Coupled with the pipelined algorithmic modules (every module starts processing as soon as they receive data from their upstream module), the data is processed in real-time much faster than the data enters the FPGA.

Figure 3.17 illustrates the detail of the acquisition protocol of the *NSP_IN* signal in Fig. 3.16, which is based on the 70 MHz clock domain. In order for the NSP to process the raw data and intermediate data, and in order for the NSP to send the final result out of the FPGA, a processing and output protocol was designed and developed.

3.3.4.3 The NSP processing and output protocols

Once the NSP_IN signal enters the NSP Algorithmic Modules, it will go through three signal processing stages, in which each stage will process and output a signal to both the next stage and out of the FPGA via PCIe (Fig. 3.18a). In order for the NSP to process the raw data and intermediate data correctly, and in order for the NSP to send the final result out of the FPGA, a few protocols were designed to schedule the digital content of the intermediate and final output signals over clock cycles. Here I describe these protocols as follows.

The stage 1 intermediate output is the denoised MUA (Fig. 3.18c,d). This signal was designed to be sent to the next stage for every five samples during the acquisition of a single data frame (Fig. 3.18b,c), such that the computation of the downstream circuit can be carried out concurrently with the acquisition. The same signal was also designed to be output to the PC (stored in *mua.bin*). However, we do need the PC to process during the acquisition of a single frame of data, so the digital protocol was designed to send a complete data frame every 160 samples (Fig. 3.18b,d).

The stage 2 intermediate output is the spike packet (Fig. 3.18e). Each packet contains 128-bit wide numbers that are scheduled in 21 consecutive clock cycles. Every 32 bits among a 128-bit number encode an individual sample in a spike waveform, extracted from one of the four channels, as illustrated in Fig. 3.18e. The first two clock cycles are used to transmit the index of the packet, and the clock cycles after are used to transmit the spike waveforms of four adjacent electrodes. A complete packet, once extracted, is transmitted to both downstream circuit and to PC (stored in *spk.bin*) simultaneously.

The stage 3 final output is the identified feature packet with spike id (spike-id packet) (Fig. 3.18f). Each packet contains 32-bit width numbers that are scheduled in 7 consecutive clock cycles. The first two clock cycles are used to transmit the index of the packet, and the clock cycles after are used to transmit a 4-D feature vector of that spike event appended by

the inferred spike id. A complete packet, once classified, is transmitted to the PC (stored in *fet_clu.bin*).

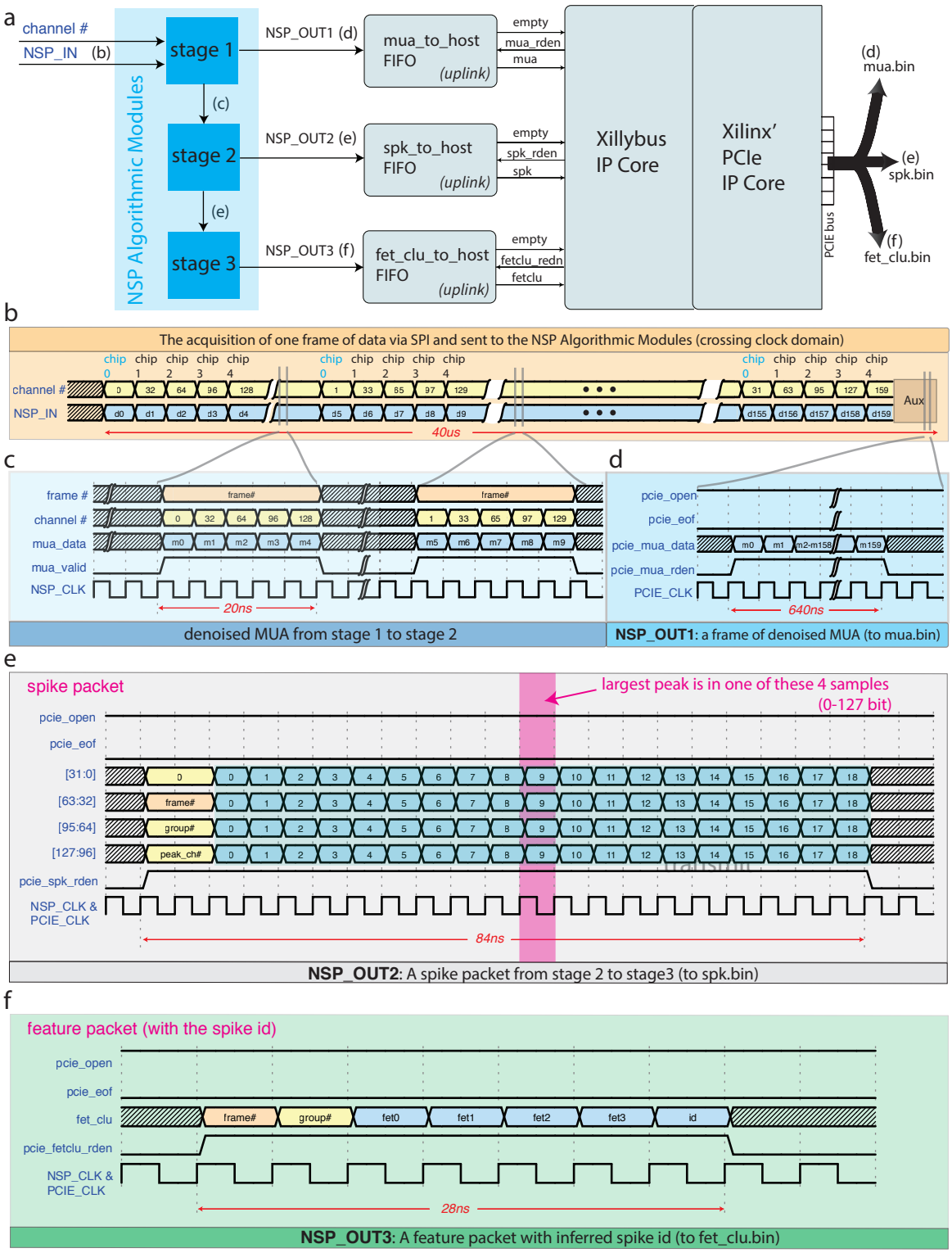


Fig. 3.18 The NSP processing and output protocols of the three NSP signal processing stages: (a) The key set of digital signals and their positions in the simplified schematic: three FIFOs bridge the NSP Algorithmic modules and PCIe. There is one input (NSP_IN) and three outputs (NSP_OUT1,2,3). (b) The acquisition protocol of a complete frame (copied from Fig. 3.17). (c) The intermediate signal from the stage 1 to the stage 2, i.e. the denoised MUA which is intermittently sent to the next stage for spike packet extraction. Each transmission takes 20 ns, i.e. send every 5 MUA samples rather than waiting for the whole frame to complete, hence more computations can be done during a single frame acquisition. (d) NSP_OUT1, one frame of the denoised MUA, which contains the same content as (c). The only difference is that the NSP_OUT1 contains a full buffered frame, which takes 640 ns (160*4 ns) to transmit. (e) NSP_OUT2 is the spike packet. Every data sample in the spike packet contains 128 bits, wherein each set of 32 bits represents a data sample collected from one channel of a 4-channel group. The first two clock cycles are scheduled for the index and the rest of the 19 clock cycles are scheduled for the 19*4 data samples. The peak sample of the spike event is always in the 11th (2 index + 9 data) clock cycle highlighted in the red column. The full spike packet is transmitted to both the next stage and the PC. (f) is the NSP_OUT3, which is the feature packet with the spike id. This is the final stage output therefore is only sent to the PC. The first two clock cycles are scheduled for the frame# and group#, the 3rd to 6th clocks are scheduled for the 4D feature vector. The 7th clock cycle is used to transmit the spike id, the final result of the full NSP processing pipeline. The signals ending with '_open', '_eof' and '_rden' are part of the FIFO interface, while other signals ending with '_valid', '_data' are part of the AXI4-stream interface (see Chapter 3.3.1.2). These signals either represent or actively control the state of the FIFO or AXI4-stream and are immediately useful for FPGA engineers to perform testing and debugging.

The processing and output protocols described above were implemented using an AXI4-Stream interface directly between the up- and downstream circuits or indirectly by inserting a FIFO between the up- and downstream circuits.

3.3.5 Summary

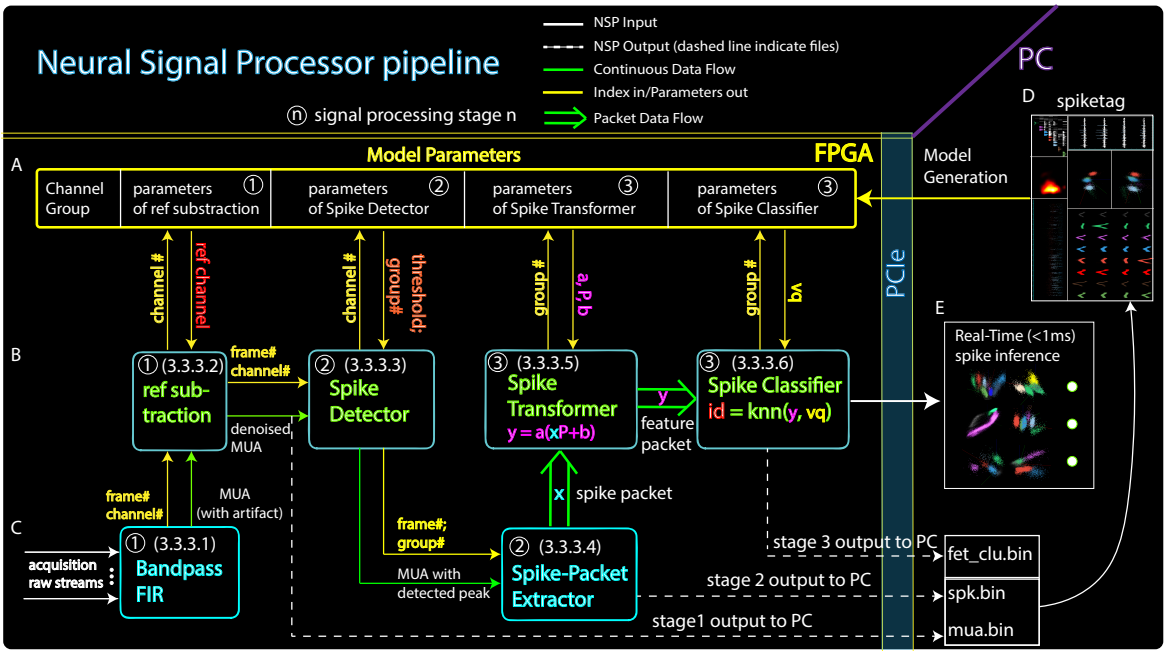


Fig. 3.19 **The NSP schematic (section number of each module is shown in the box):** (A) The model parameters (their signal processing stage numbers are shown in circles). (B) The algorithmic modules (in green) interface with model parameters (in yellow) by directly accessing several dual-port-RAMs. (C) Two modules (Bandpass FIR, Spike-Packet Extractor) that use fixed parameters. (D) The Spiketag software as the model generator (Chapter 2). (E) The final output of the NSP. Once at stage 3, the NSP outputs the feature packet with spike id (spike-id packet) for every spike event within 1 ms, which can be visualized as a feature vector coloured by its spike id in the feature space of each group.

To summarize (Fig. 3.19), an FPGA NSP was designed and implemented to acquire 160 channels of raw neural signals, from either tetrode arrays or silicon probes, using five INTAN chips and to output the real-time spike trains of a population of single-units.

Three stages of signal processing pipeline were designed and implemented for the NSP real-time spike inference. Each stage includes specific algorithmic modules and requires specific model parameters to operate. The methods to construct the model parameters for each module were introduced in this Chapter. The first stage algorithmic modules consist of a bandpass filter and a reference subtraction module; it outputs the multichannel denoised MUA. The multichannel denoised MUA is sent to both the PC and the FPGA NSP second stage algorithmic modules. The second stage involves a spike detector and a spike packet extractor. The output of the second stage is the spike packet. Each spike packet is sent to both the PC and the final stage FPGA processing. In the final stage, the spike packet is first

transformed into a feature packet in a scaled PCA space and then classified with a kNN-VQ algorithm. The final output is sent to the PC for further population decoding or to trigger real-time feedback[35]. The actual hardware implementation of the overall system is included in Appendix A.0.1, while the resource utilization inside the FPGA is included in Appendix A.0.2.

The total latency of the pipeline is around one millisecond which breaks downs into 0.84 milliseconds for filtering, 40 microseconds for reference subtraction, 80 microseconds for spike detection and 1 microsecond for transformation and classification. Digital protocols were designed and implemented to ensure the precise data movement between modules (acquisition, processing and output) at the nanosecond level, which in turn regulates the pipelined operations of up/downstream circuits inside the FPGA. The end-to-end latency from the raw analog neural signal to the final spike identity also involves the latency of the signal amplification, digitization, multiplexing, which is handled by the INTAN chip (not included in this Chapter). The result from tests of the end-to-end latency and accuracy of the system will be introduced in the next Chapter.

[35] If the real-time feedback is conditioned on the spike timing of a specific single-unit rather than the output of a popualation decoder, a spike-triggered TTL signal in the FPGA can be directly used to activate the real-time feedback without involving the decoder.

Chapter 4

Results

In the previous chapters, I described the design and development of the system in terms of its technological detail. In this chapter, I will present the test results concerning both the individual module and the end-to-end (from the raw analog neural signal to the final spike identity) performance of the system. Each following section has a specific test configuration and goal.

4.1 FPGA algorithmic module test

Before chaining every FPGA NSP algorithmic module together to form a hardware real-time signal processing pipeline, each hardware algorithmic module needs to be tested. The test is constructed by feeding the test input data (all extracted from real data) into the FPGA, then looping back the output to the PC and comparing it with the result computed by the same algorithm implemented in software. For example, to test an FPGA digital filter, artificial input signals and a software-generated ground-truth output are used. A multi-channel signal (input) is written into the implemented FPGA digital filter through the PCIe interface. The output of the digital filter is read back by the PC, again through PCIe, and then compared to the target ground-truth output generated by the software simulation. This PC-FPGA-PC test configuration is implemented by a Xillybus PCIe module (Xillybus, 2010), and was used for testing every FPGA algorithmic module, as illustrated in Fig. 4.1.

The goal here is to validate that the FPGA modules generate the same results as the software simulation given the same input. This test was carried out without any acquisition system, and each FPGA module under test received 32 channels of input from the PC[1]. The tests show the hardware implementation performs as accurately as its software counterpart

[1]The modules that pass the test were upscaled to support 160 channels input.

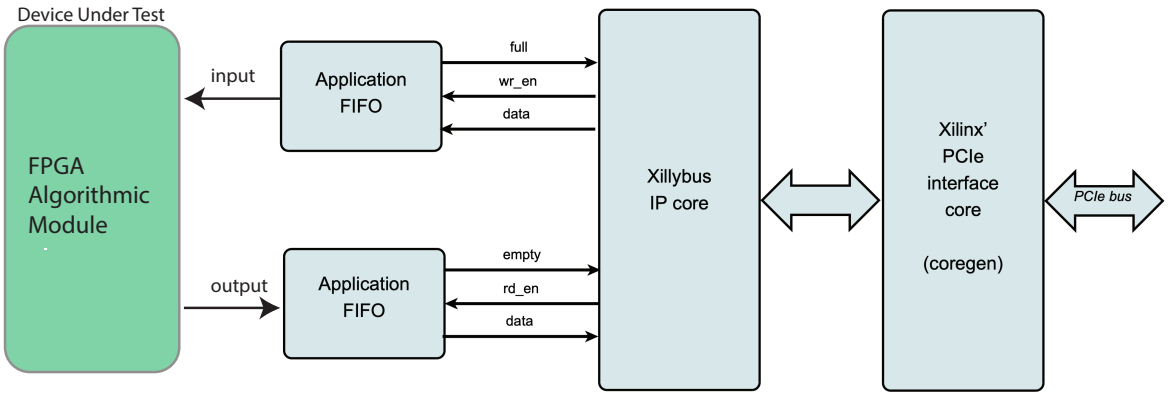


Fig. 4.1 **The testing configuration for individual FPGA algorithmic modules:** The test signal is fed into the FPGA through the Xillybus PCIe bus, and the computed result is read out from the same interface. Two FIFOs are used to interface the PCIe core and the FPGA Algorithmic Module (the green block) under test. The customized FPGA Algorithmic module only needs to interact with FIFOs. All the element blocks in this figure are implemented in the FPGA Fabric. After the FPGA output from the Algorithmic module is read back by the PC through the PCIe bus, the error between the output and the desired output is computed in the PC.

using the same algorithm, while the tiny measurable error is less than the quantization error of each sample (5 test results are listed[2] in the Table 4.1). The difference between the number of bits used to encode a number in the FPGA and the number of bits used to encode the same number in the PC makes such discrepancies inevitable. For example, the output from the FPGA filter has an error of less than 2e-6 for each sample. This tiny difference is much less than the noise floor in the neural data, thus negligible. Another example is the spike transformer which uses 8 bits to encode each transformed feature, so the minimum resolution of each feature sample is 0.0078. An error below that resolution is possible. This quantization error is not critical, given its dynamic range from -1 to +1, and only leads to less than a 0.6% classification error rate among all tested spikes. However, the benefit is enormous because a single 32-bit number can encode a 4-dimensional feature. It only occupies 1/4 of the memory bandwidth compared to using 32 bits for each transformed feature, and the operation on a 4-dimensional feature sample can be finished within a single clock cycle. The errors reported in table .4.1 are the difference between the hardware (NSP) generated output and the software simulation generated output.

The abbreviations for test input and output signals in the Table 4.1 are listed here:

1. RAW: Continuous Raw data.

[2]The reference subtraction module is not listed here because it is the only module added after the overall system was built. Its functionality was tested later in the end-to-end test configuration.

Table 4.1 Performance Test on the individual Algorithmic Module

Algorithmic module	Input	Output	Width×Depth	Error
Band pass FIR filter	RAW	MUA	32bits×N	< 2e-6
SPK detector	MUA	MUAP	32bits×N	0.0
SPK packet extractor	MUAP	Spike packet	128bits×21	< 2e-6
SPK transformer	Spike packet	Feature packet	32bits×6	< 0.0078
SPK classifier	Feature packet	Spike-id packet	32bits×7	< 0.6%

2. MUA: Continuous Multi-unit activity.

3. MUAP: Continuous Multi-unit activity encoded in 33-bit numbers, in which the last bit of each sample indicates whether it is a peak or not.

4. SPK: Spike.

5. Spike packet: a digital packet that contains four spike waveforms from four channels in an electrode group.

6. Feature packet: a digital packet that contains spike features from four channels in an electrode group.

7. Spike-id packet: final output packet that contains both spike features and spike-id.

For the FIR filter and SPK detector, the input and output have the same length (their depth is N samples where N is the length of the input). In contrast, for the SPK packet extractor, transformer, and classifier, the output depth is fixed according to the size of a single packet. NSP digital protocols (Chapter 3.3.4.3) are used in the validation. For example, the size of output from the SPK packet extractor was 21 data points describing the index and the content of extracted spike waveforms. In a spike packet, the first two 128-bit data points contain the timestamps (when) and the electrode group number (where) this spike origin. The remaining 19 128-bit data points describe four spike waveforms from four channels. Each 128-bit data point encodes four samples (32-bits wide) from four different channels in a group. Passing all these tests marked the end of the first phase (design phase) of the project. The next phase (integration phase) is to link these FPGA algorithmic modules together, along with the FPGA acquisition module and test the system's end-to-end single-unit assignment accuracy.

4.2 End-to-end single-unit assignment accuracy

The accuracy of individual FPGA algorithmic modules does not guarantee the accurate assignment of single-units. The latter requires testing the integrated system that pipelines all the tested FPGA modules in the previous section, into an end-to-end scenario, from the raw data acquisition end to the spike-id output end.

The end-to-end test configuration is very different from the Algorithmic Module test, where acquisition is not required. I will introduce the test configuration first before examining the test data and the test results.

4.2.1 End-to-end test configuration

In order to test the accuracy and latency of the real-time classified spike output of the overall system, it is necessary to test the full end-to-end data link. The end-to-end test requires the system to perform data acquisition at one end, perform real-time data processing simultaneously, and assign a label to every detected spike at the outpput end. As shown in the previous chapters, this full real-time data processing pipeline includes five INTAN chips (which perform analog signal amplification, digitization, multiplexing for 160 channels), the FPGA digital acquisition module (digital signal acquisition for 160 channels), and the FPGA NSP (filtering, denoising, spike detection, spike packaging, spike transformation and feature classification for 160 channels). All these instruments, from raw analog input to producing the final digital spike identity output, cascaded together constitute the device under test (DUT) (Fig. 4.2 b)

A rigorous test for the DUT requires multi-channel extracellular signals as input. However, recording from the real animal for repeated testing is expensive, and it will not be compatible with a ground-truth test. In order to generate 'real' raw analog input that contains ground-truth extracellular signals, a neural signal generator (NSG) was constructed to replay multi-channel data (Fig. 4.2 a), where extracellular spikes are output at the tens to hundreds of $\pm\mu V$ level as in the real neural recording.

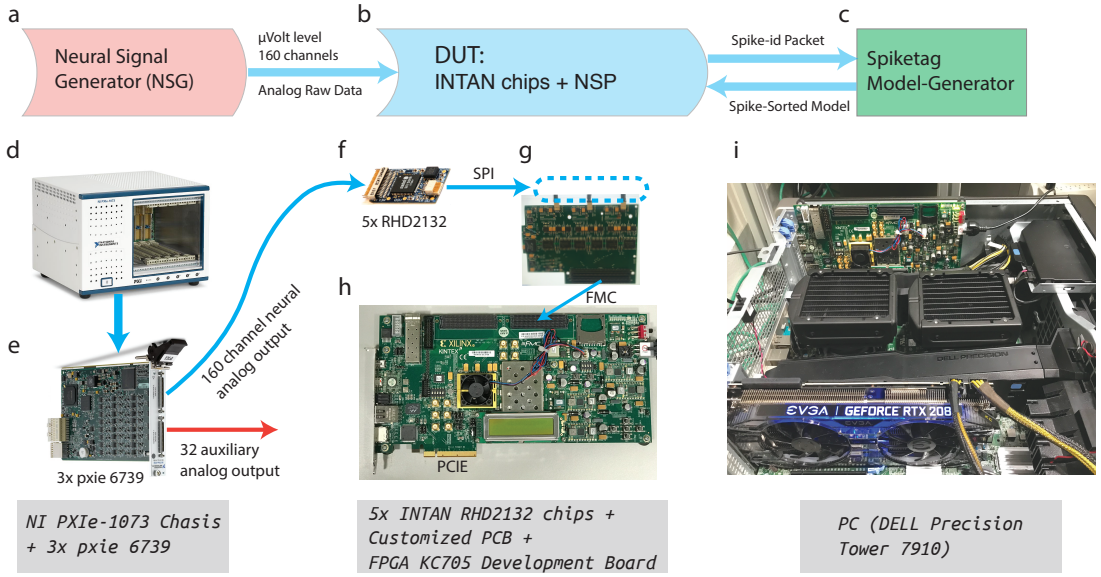


Fig. 4.2 End-to-end test configuration: (a) A neural signal generator (NSG) was used to load and replay high-channel-count recorded neural data. It output 192-channel analog signals at 25000 Hz, in which 160 channels are fed to the device under test (DUT). The amplitude of the signals fed into the DUT are attenuated to the same level as the original neural signals. (b) The DUT is composed of 5 INTAN RHD2132 amplifier chips, a customized PCB and a KC705 development board. The FPGA on the KC705 contains the FPGA NSP, with the chained and pipelined algorithmic modules. Five INTAN RHD2132 chips receive and amplify 160 channels of analog signal, multiplex and convert it into digital form and send the digitized data into three SPI cables. The communication between INTAN chips and FPGA is routed via a customized PCB using a 500-pin FPGA Mezzanine Card (FMC) connector, a high pin-count industry-standard. The FPGA NSP generates clock and control signals to manage the acquisition of five INTAN RHD2132 chips; processes the raw digital data during the acquisition sampling period (40 microseconds), and assigns the spike identity to each detected spike using the model parameters downloaded from the PC. (c) Spiketag, a highly interactive spike sorting pipeline is used to generate a spike inference model from the recorded data. Usually a few tens of minutes of recording is sufficient to build a spike inference model. After the model is downloaded into the NSP, Spiketag was used to compare the real-time spike-id packets produced from the NSP and the ground-truth labels. (d) National Instruments (NI) PXIe-1073 Chassis, a 5 slot PXI backplane, that takes binary data from the connected computer, and feeds digital data into its resident analog output cards. (e) Three NI PXIe 6739 analog output cards, each of which can output 64 channels of analog signal. (f) Five INTAN RHD2132 chips. (g) The customized PCB cards that route SPI digital signals into the FPGA. (h) The KC705 development Board. The NSP is in the KC705 FPGA. The KC705 development Board connects to the PC through the PCIe slot. (i) The PC, a DELL Precision Tower 7910, has 128GB RAM, a 48 core Xeon CPU, and a NVIDIA GEFORCE RTX 2080 GPU. The computer upper PCIe slot is connected to the KC705 FPGA board.

The NSG consists of one National Instrument (NI) PXIe-1073 chassis and 3 NI PXIe 6739 64-channel analog output boards. Here I used the NSG to convert a computer-stored

digital file into a multi-channel analog output. Three NI PXIe 6739 cards are plugged into the PXIe-1073 chassis, and each of them can generate up to 64-channels of analog output, totaling 192 channels of output[3]. A stored binary file that contains up to 192 channels of either real or synthesized neural data is loaded and replayed at 25000 Hz via the NSG. 160 channels among 192 NSG analog outputs are then attenuated[4] and fed into five INTAN chips in the same manner as in the actual recording system. In order to quantify the error of the DUT, the real-time assigned spike identity was then transmitted to the PC and compared with the ground-truth results.

With this test configuration, the accuracy is ready to be quantified by comparing the offline spike-sorted spike identity and the NSP produced spike identity. The Spiketag software is part of the DUT. To make the test rigorous, ground-truth data is required, in which intracellular/juxtacellular spikes are simultaneously recorded with extracellular spikes. In this section, I call the cell that generates intracellular/juxtacellular spikes the ground-truth cell and the extracellular spikes generated by that cell the ground-truth spikes. Our system was designed to be compatible with both tetrodes and silicon probes to operate in various brain regions. In order to test that, two such ground-truth datasets were used to test the spike inference accuracy of the system; the first dataset mimics the data collected from a tetrode (Hunt et al., 2019) while the second dataset is collected from a silicon probe (Neto et al., 2016). These two datasets were chosen because they represent the raw data collected from two different recording devices (tetrode and silicon probe) and they were recorded from two different brain regions (hippocampus and motor cortex). Comparison is conducted between the ground-truth, the FPGA NSP-produced result, and the offline sorting results produced by Spiketag, Kilosort (Pachitariu et al., 2016), and JRClust (Jun et al., 2017a)[5]. Both ground truth datasets contain simultaneous intracellular (or juxtacellular) recording and extracellular recording. These two datasets were merged into a single data file that contains the neural signal in five 4-electrode groups (the rest of the 160 channels were made to be Gaussian noise). This file is then replayed by the NSG (Fig. 4.2a), and the system's accuracy in identifying two different ground-truth neurons was tested.

[3] 160 data channels + 32 auxiliary analog outputs. 160 data channels were connected to five INTAN chips, while the auxiliary analog outputs were used as reference signals to test the end-to-end latency of the device. This will be introduced in the section on the latency test.

[4] to $\pm 5mV$ corresponding to the full 16-bit digital range. That is, a digital value of 2^{15} produces 5 mV analog output while 2^{-15} produces -5 mV analog output.

[5] These are currently two of the most popular spike-sorting packages for large-scale extracellular neural recording.

4.2.2 Test result - dataset 1

The first ground-truth data set was collected using a Pipette-integrated microelectrodes device called the Patch-Tritrode (Hunt et al., 2019). This is an enhanced intracellular glass patch pipette with three extracellular electrodes attached outside the glass pipette, each around 25-30 microns away from the pipette tip (Fig. 4.3a, b, c).

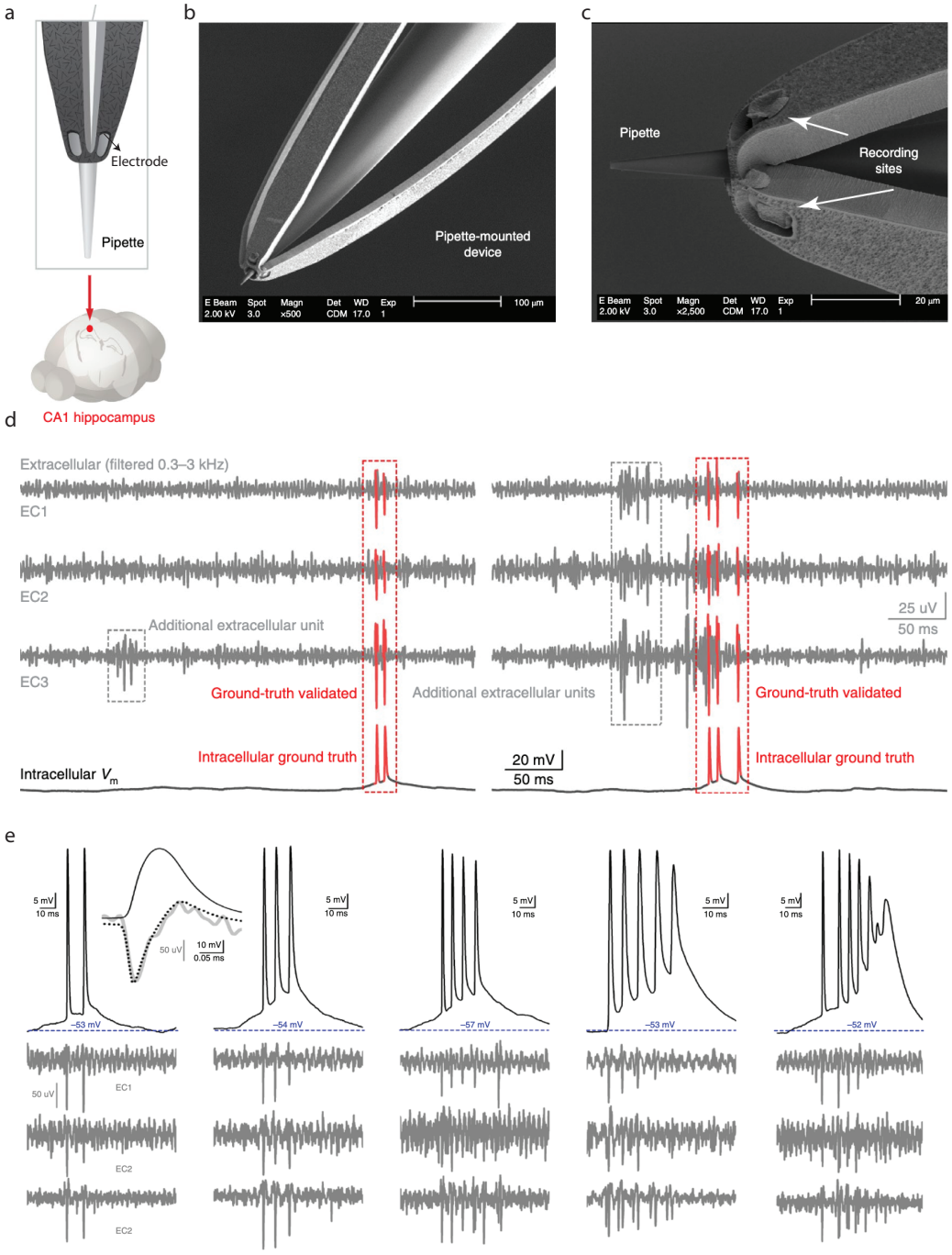


Fig. 4.3 The Patch-Tritrode ground-truth dataset: "(a) The Patch-Tritrode allows the placement of three extracellular sites near to the patch pipette tip, thus enabling simultaneous in vivo intracellular whole-cell and extracellular recordings from the same cell. This data was collected from hippocampal area CA1. (b) Low-magnification scanning electron micrograph of the pipette-mounted Patch-Tritrode device. (c) High-magnification scanning electron micrograph of pipette-mounted Patch-Tritrode device illustrating the 3D geometry of the extracellular recording sites relative to the pipette tip. (d) Representative traces of a single experiment from the extracellular recording sites (EC1-3) of the Patch-Tritrode where the ground-truth-validated spikes (corresponding to the intracellularly recorded unit) are indicated. Note that additional extracellular units are present in the extracellular recording. (e) Representative example traces from a single multimodal recording of bursts containing 2, 3, 4, 5 and >5 spikes (left to right) where the intracellular trace is shown above in black and extracellular traces are shown below for EC1 (top), EC2 (middle) and EC3 (bottom). Inset, representative example of an individual spike (solid black line), the first derivative of the intracellular waveform (dotted line) and the corresponding superimposed extracellular waveform shown in grey." Figure adapted from Hunt et al. (2019)

This dataset contains 3 extracellular channels (Fig. 4.3d) and was recorded in hippocampal area CA1 (Fig. 4.3a). It contains much bursting activity (Fig. 4.3e). Hence it is ideal for testing whether Spiketag and the NSP can deliver decent spike sorting and inference performance on bursty neurons.

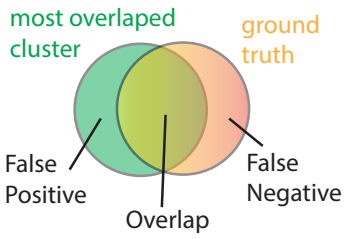
The first two stages of output of the NSP were used to generate the spike-sorted model in Spiketag. Then the model was used by the NSP to produce final output (spike-id packet) in real-time. The hit rate and the precision (see Fig. 4.4b for definition) of the cluster that overlaps the most with the ground-truth spikes are used to measure the spike sorting and the spike inference performance (Fig. 4.4a). The test results show good performance for the NSP real-time spike inference as well as all three offine spike sorting packages (Fig. 4.4b).

The test results (Fig. 4.4) give rise to several implications. First, the full data link functionality is correct. What could be the alternative explanation for the above 90% accuracy for a ground-truth neuron? After all, any real mistake in the long signal processing pipeline would be likely lead to a poor result. Second, although a bursting neuron is generally thought to be difficult to sort[6], the system can handle it well, as did the other modern spike-sorting packages. Third, our NSP can accurately identify a sparse cluster, given this ground-truth neuron contributed only 731 (7.8%) spikes out of 9309 total detected spikes. Fourth, the spikes that have medium peak amplitude (slightly larger than $50\mu V$) (Fig. 4.3 e: the bursty extracellular waveforms) can be sorted and classified with beyond 90% accuracy. This is

[6]The extracellular spikes during a burst often exhibit an increase in spike width, as well as a decrease in spike amplitude. Such waveform distortion is a major reason for the incorrect spike sorting of a bursting neuron.

quite surprising to many manual spike sorters, because clusters within this range are often removed from the final sorted units (see the waveform and features in 4.4 c).

a



b

hitrate = overlap/(overlap + false negative)
precision = overlap/(overlap + false positive)

Source	hitrate	precision
Spiketag (FPGA NSP)	92% (93%)	90% (87%)
JRClust	88%	92%
Kilosort1	96%	83%

c

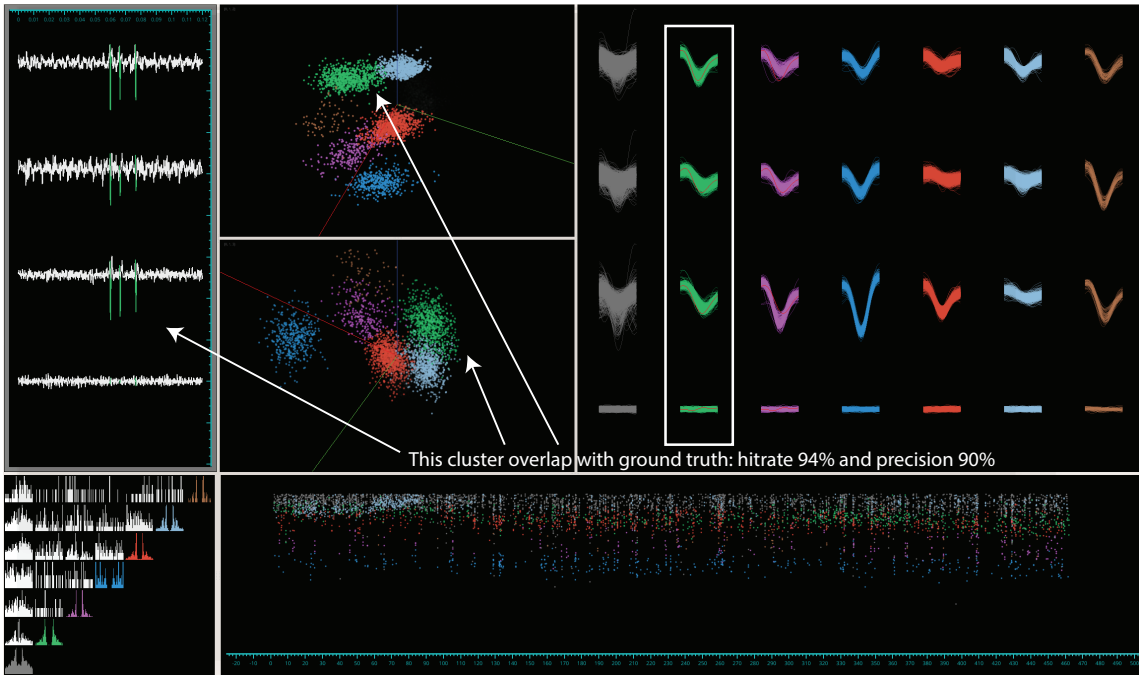


Fig. 4.4 Accuracy of single-unit inference on the Patch-Tritrode ground-truth data: (a) The Venn diagram illustrates that the False Positives (FP), Overlap, and False Negatives (FN) calculated by the overlap between the cluster with the most overlap with the ground truth spikes and the full set of ground truth spikes. (b) Performance: The Spiketag uses a short recording to generate a model for real-time spike inference, and the most overlapped cluster has a 92% hitrate and the 90% precision. Independent sorting performed by other sorting packages shows that JRClust's hitrate is 88% and its precision is 92%. Kilosort1's hitrate is 96% and its precision is 83%. Moreover, the VQ model built for the NSP final output (the NSP spike-id packet) results in a 93% hitrate and 87% precision. (c) The Spiketag screenshot shows the green cluster is the most overlapped cluster. Note: It is not the highest SNR cluster among all clusters, which makes it a good test example for ground truth evaluation of sorting algorithms. The purple and blue clusters both contain larger peaks than the green cluster. In practice, the medium range SNR units are often thrown away by spike sorters.

Due to short duration of the ground-truth recording (∼7 mins) and low number of ground-truth spikes (731), I did not perform cross-validation on this test. However, cross-validation is not necessary here for two reasons. First, the major source of discrepancy between the spike sorted result and the real-time spike inference result is the lossy compression of the VQ classification model. If the VQ model built from the validation set is different from the test set, then we can conclude waveform drift happened, presumably due to non-stable recording. Solving spike inference for a non-stable neural recording is not part of my goal in this thesis. The NSP only works under the assumption that the recording and VQ models are stable from the time the model was built (training set) to the time the model was used (test set)[7].

Therefore, using the ground-truth validation, we can conclude that at least for the tetrode like multi-electrode, the performance of hardware single-unit assignment is as accurate as offline spike sorting (Fig. 4.4 b) for the hitrate and precision in a stable recording. The next step is to show the system would also work with a silicon probe. For that, we need silicon probe ground-truth data.

4.2.3 Test result - dataset 2

The second dataset was collected by Neto et al., (Neto et al., 2016), in which the authors performed simultaneous recording from both a customized high-density silicon probe and a patch pipette in primary motor (M1) cortex (Fig. 4.5a,c,d). For this test, I created 4 ground truth groups in the probe, each containing 4 electrodes (Fig. 4.5e). These four groups cover an area (Fig. 4.5b: colored electrodes) spanning a 60 μm by 50 μm area on the probe; beyond this area the extracellular spikes cannot be distinguished from noise. The ground-truth extracellular spikes (Fig. 4.5d: coloured extracellular waveforms) are colored to indicate their electrode source (Fig. 4.5b,d).

[7] In contrast, cross-validation might be necessary when using the NSP-produced population spike trains for real-time behavior decoding. The reason for that is, even if the neural recording and VQ models are stable as assumed, the representation of neurons might or the state of the network can drift or change due to physiological reasons, which would be captured by the cross-validation.

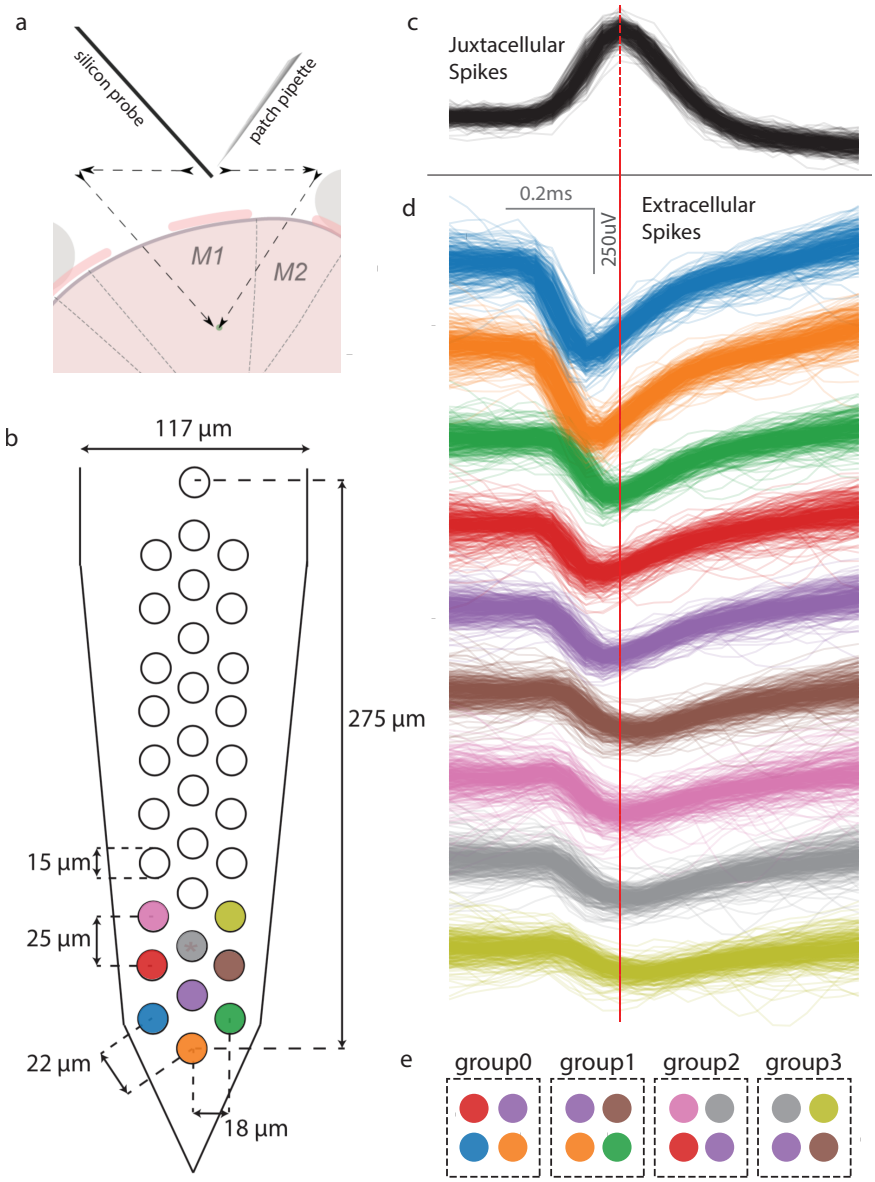


Fig. 4.5 The Patch-Silicon probe ground truth dataset: (a) From (Neto et al., 2016):"schematic of a coronal view of the craniotomy and durotomies with both probes positioned at the calibration point. The distance between durotomies, such that the probe tips meet at deep layers in cortex, was around 2 mm. The black arrows represent the motion path for both electrodes entering the brain." (b) "extracellular dense polytrode array with a span of 275 μm along the shank axis." The colored electrodes are selected to form 4 groups that contain sortable ground-truth spikes. (c) The juxtacellular spike waveforms. (d) The simultaneous ground-truth extracellular spikes waveforms (colours represent electrodes/channels). (e) The artificially created groups. Each group contains 4 electrodes (colours represent electrodes/channels). Figure adapted from (Neto et al., 2016).

To illustrate the shape of clustered units in each group, the screenshot of the sorted result in the Spiketag is shown, where the green cluster is the cluster that most overlapped with the ground-truth unit (Fig. 4.6). The same ground-truth unit is surrounded by different other units in the different groups.

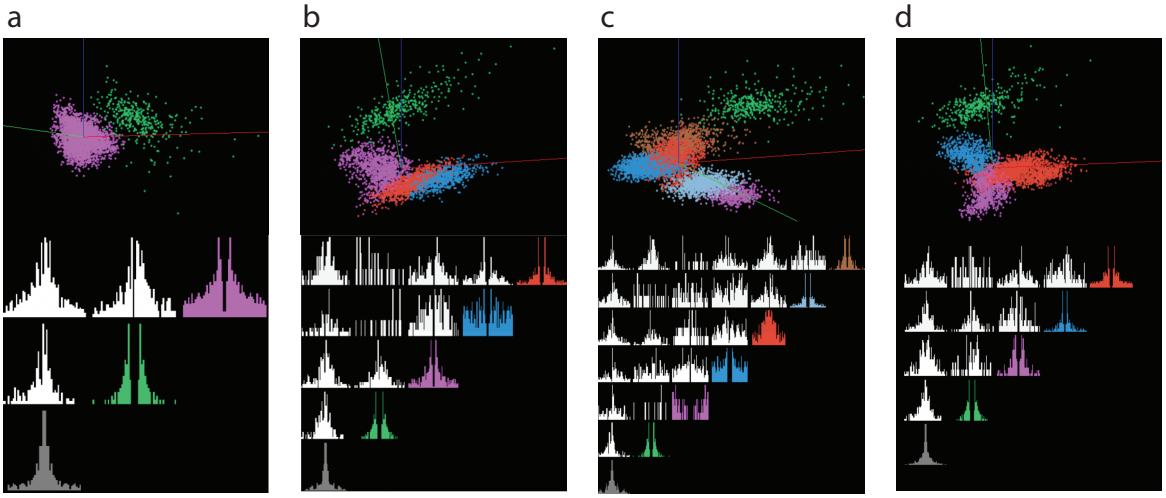


Fig. 4.6 Spike sorted result in the Spiketag for four created groups (groups 0-3 from left to right). The green cluster is the ground-truth unit.

To prove that the system works with high-density probes, we need to prove the ground-truth data is sortable and classifiable at least in one 4-electrode group under my test configuration. The validation of both the spike sorted result (software) generated by the Spiketag and the FPGA NSP (hardware) single-unit assignment result reveals that the ground-truth unit is assigned with high accuracy in all four groups (groups are shown in Fig. 4.5e and results are shown in Fig. 4.6 and Table 4.2):

Table 4.2 Accuracy of single-unit inference on the Patch-Silicon probe ground truth data

#Group	Hitrate Spiketag (FPGA NSP)	Precision Spiketag (FPGA NSP)
group0	99.7% (99.7%)	99.4% (97.1%)
group1	97.7% (98.2%)	97.1% (95.3%)
group2	98.5% (96.4%)	98.5% (98.8%)
group3	96.2% (97.1%)	96.2% (93.2%)

In summary, once the peak of the extracellular spikes go near or below the noise level, the spike sorting accuracy drops quickly to less than a random guess, or even nearly zero (because spikes cannot be detected). Hence in this dataset, only four sortable groups can be

created (Fig. 4.5b,d,e). Interestingly, spike sorting, for all these groups, generates a nearly perfect sorted model (Table 4.2). TA thorough analysis of spike sorting performance itself, e.g., the underlying factors, parameters and hyperparameters that cause an ideal or poor spike-sorted model, requires more ground-truth data in the future, which is not in the scope of this thesis.

From this test I validated several aspects of the tool. First, the real-time NSP, including the FPGA acquisition system and the FPGA algorithmic modules integrated together, produced the designated functionality. Second, the hardware single-unit assignment is as reliable as offline spike sorting. Third, real-time single-unit assignment using 4-electrode grouping performs well with a multichannel silicon probe. This is an expected result because all the algorithmic modules are designed to ensure that the hardware generates nearly the same output as its software counterpart. In other words, if a spike sorted model can predict single-units well in a group, the real-time NSP employing the same model will work equally well. However, if the spike sorting result is not accurate, there is no reason that the real-time NSP could generate better results.

Dataset-1 and dataset-2 contain five electrode groups. I put these five groups into a single file for the test. The file is replayed by the NSG and the ground-truth spikes from each group had been identified simultaneously. This test demonstrates the system's accuracy in resolving the identity of a population of single-units.

4.3 End-to-end single-unit assignment latency

Low-latency is another major engineering objective of this thesis. Algorithmic modules inside the FPGA have been tuned to minimize latency at each processing node. In addition, the FPGA modules have been pipelined to minimize end-to-end latency. Testing their final performance on the full data pipeline in an end-to-end manner is critical.

The full real-time data processing pipeline consists of 5 INTAN amplification chips (which provide amplification, digitization, and multiplexing of 160 analog channels), the FPGA digital acquisition module (for digital signal acquisition), and FPGA algorithmic modules (for filtering, denoising, spike detection, spike packaging, spike transformation and feature classification). As in the end-to-end accuracy test, the DUT includes both INTAN chips and our FPGA NSP (Fig. 4.7 b). Before the test was conducted, a spike-sorted model was created from the Spiketag (Fig. 4.7 d).

In the latency test, the NSG (Fig. 4.7 b) sent not only 160 channels of raw analog input to 5 INTAN chips, but also a TTL signal on channel 161 indicating the precise ground-truth

spike timing[8]. This ground-truth TTL was used for the latency measurement and was fed into a logic analyzer running at 50 MHz (Fig. 4.7 a). The same logic analyzer also received the spike-triggered TTL from the FPGA so that the ground-truth TTL timing and the spike-triggered TTL timing can be compared objectively and precisely using a single clock source (Fig. 4.7 a).

Four artificial ground-truth single-units were synthesized on channels 157-160 (Fig. 4.7 e) at 25000 Hz sampling rate. These artificial units were firing at 25 Hz. Each spike causes a ground-truth TTL pulse. The 0-1 transition of the TTL occurs at the peak sample of each spike waveform, while the 1-0 transition occurs when the spike is finished (10 samples after the peak (Fig. 4.7 h)). One of the four synthesized units had been selected to trigger the spike-triggered TTL on the FPGA (Fig. 4.7 e: highlight in blue). Over 30 minutes, 11250 spike-triggered TTLs had been generated by the DUT, which corresponds to a quarter of the ground-truth TTLs (because one out of four units are selected to be the trigger unit (Fig. 4.7e)). These ground-truth TTLs were compared to the spike-triggered TTLs for calculating the latency.

The reason I used synthesized single-units spike waveforms to test latency is that the FPGA NSP could generate 100% accuracy in identifying these four synthesized neurons. Thus I can test latency faithfully without being confounded by spikes that were not accurately reported. The latency, defined from the completion of the spike waveform (10 samples after the peak) to its correct id assignment (Fig. 4.7 h), is consistently less than 1ms for all spikes from the trigger unit (Fig. 4.7 i). No drifting of latency arises. The variation of the timing is less than 40 μs, a sampling interval (Fig. 4.7 j). Also, changing the trigger neuron to another neuron presented in channels 157-160 did not alter the results. The latency for each neuron in the population is the same.

The 40 μs latency variation (Fig. 4.7 j) is highly likely produced by the INTAN chip or the NSG analog output. The FPGA NSP is designed to have a 4 ns precision[9], so if any bit involved in the FPGA algorithmic pipeline fails to move to the designated location in the chip within the required time scale, the computation would be likely to fail. The timing performance and computational correctness are significantly overlapping issue for the FPGA. But this is not as critical for the analog device in the signal chain, in which slight timing

[8]The ground-truth spike timing pulse was encoded in channel 161 in the prepared data file and was output as a TTL signal from the NSG.

[9]As introduced before, the FPGA is a programmable algorithmic integratd circuit that contains a massive number of registers, wires and flip-flops. By programming it, these physical elements were connected to produce a real digital circuit (with no CPU). The state of these elements (the voltage) are updated through digital clocks inside the FPGA. In my design, the algorithmic modules co-use a 250MHz clock, such that the temporal precision is 4 ns.

variations in converting from digital to analog (in the NSG) and vice versa (in the INTAN chips) are allowed.

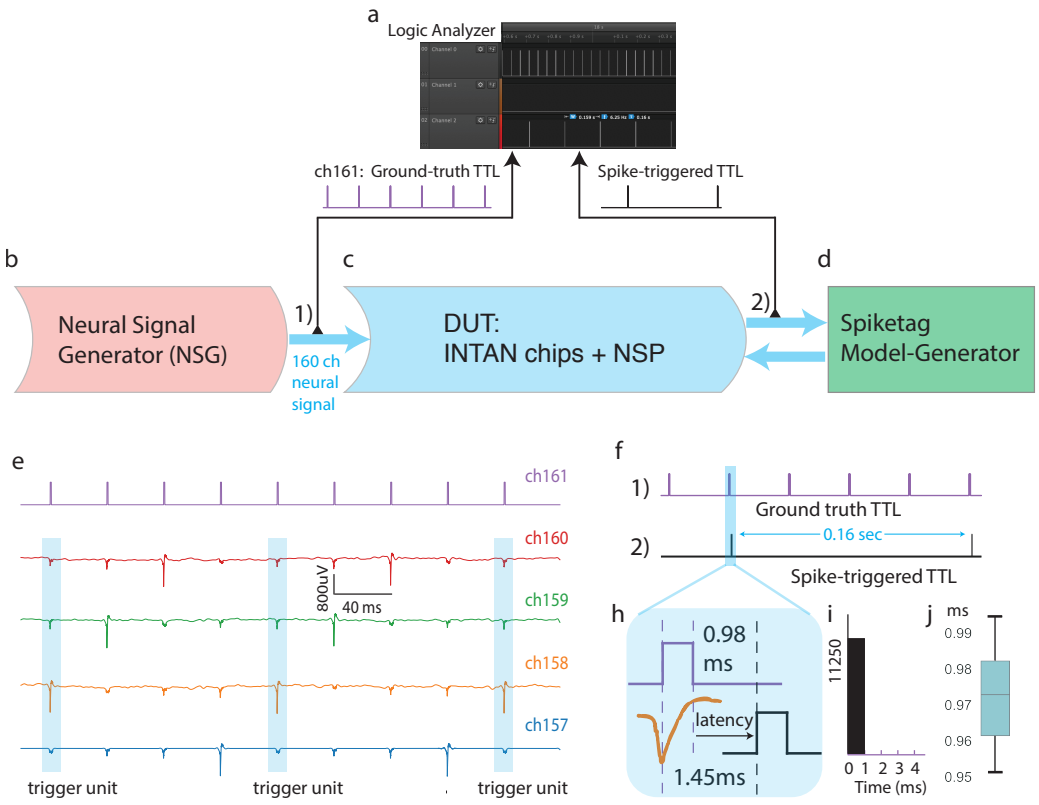


Fig. 4.7 The latency test configuration and results: (a) A Saleae logic analyzer receives both ground-truth TTL and the spike-triggered TTL. (b-d) The NSG, the device under test (DUT, which including the INTAN chips and the FPGA NSP) and Spiketag (same as section 4.2). Here the NSG outputs the channel 161 (node number 1) analog signal (TTL) directly to the logic analyze (a), and the NSP-generated spike-triggered TTL (node number 2) is output to the same logic analyzer (a). (e) The four synthesized waveforms of single-units with the ground-truth TTL that indicates each spike. (f) Examples of the ground-truth TTL and the spike-triggered TTL. This shows that the spike-triggered TTL immediately follows the ground-truth TTL every 0.16 seconds, which is the firing interval of the trigger neuron. (h) An expanded view of the highlighted (marked with blue) two TTLs, the 0-1 transition of the ground-truth TTL (the first vertical dotted line) indicates the peak sample of the largest spike waveform of all four channels, the 1-0 transition comes 10 samples after the peak (the second vertical dotted line), indicating the end of the spike waveform. The spike-triggered TTL (black) indicating this neuron was identified to be the trigger-unit. Its transition from 0-1 reveals a 1.45 ms latency from the spike peak and 0.98 ms latency from spike completion to its correct inference. (i) The histogram of the latency from the spike completion in its analog form to the digital spike-id output. (j) The boxplot of the latency shows a tiny variation ($< 40\ \mu s$).

As illustrated by this test, when any extracellular spike waveform, in its small analog signal form, comes to the completion within 1 ms, its identity will be accurately decided and reported by the FPGA NSP as a TTL signal (the spike-triggered TTL in Fig. 4.7). Although the low-latency spike-triggered TTL might enable some closed-loop stimulation experiments (see section 1.3.3), in the current FPGA implementation, only one trigger neuron can be set by users (using the Python APIs) to enable the spike-triggered TTL at a time. In the future version, there will be many TTL outputs reporting in parallel the activity for all the sorted neurons.

Even without the on-chip TTL output, the real-time identified spike-id packets were sent to PC via PCIe low-latency interface. The population spike trains were constructed by reading out the content of these spike-id packets and were used for real-time population decoding. In order to validate the performance of the NSP-based population decoding, the real-time decoded animal positions from a hippocampal silicon probes recording were compared to the ground-truth animal positions. I will present this test in the next section.

4.4 End-to-end real-time population decoding of hippocampal CA1 activity during movement

The final test is NSP-based real-time population decoding, in which the population activity from rat dorsal hippocampus CA1 was used to estimate the position of an animal. The test dataset was recorded from a freely moving rat during random foraging in an $160cm \times 160cm$ 2D maze (Fig. 4.8 a) by Brian Lustig in Albert Lee's lab. Food pallets were thrown into the maze randomly to encourage movement by the animal. Four identical Neuronexus 64-channel silicon probes (Fig. 4.9 a, c) were implanted and were gradually positioned into both left and right dorsal hippocampal CA1 of a Long-Evans rat before the recording. The recording is 36 minutes long, during which time the trajectory of the rat was recorded by video and later reconstructed as the ground-truth trajectory (Fig. 4.8 b). The trajectory data and electrophysiology data were synchronized through a blinking LED fed to both the video and the neural recording system.

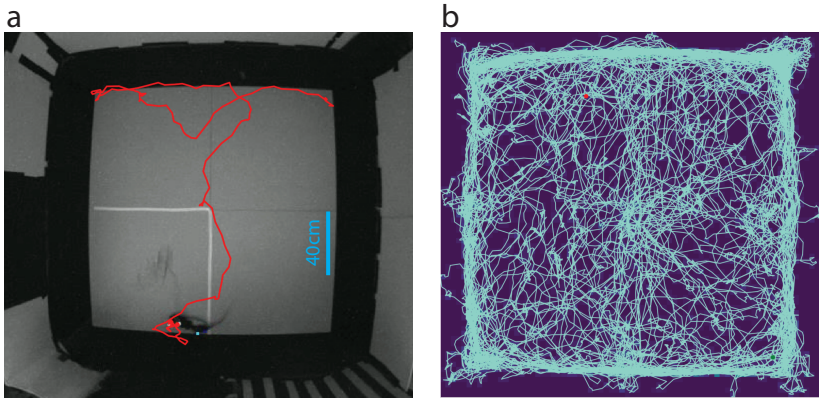


Fig. 4.8 The freely moving rat trajectory during silicon probe recording: (a) One frame of the video recording on a freely moving rat. The trajectory tracking was performed using Deeplabcut (Mathis et al., 2018). The red indicating the rat trajectory of the last 18 seconds from the current frame. (b) The overall trajectory during a 36 minutes silicon probe recording.

This data was then used to test the performance of real-time population decoding. Similar to the previous test configuration (Fig. 4.2), 160 channels[10] of the raw data were replayed in their analog form by the NSG, spikes were identified in real-time by the NSP based on the spike-sorted model, then were transmitted to the computer through PCIe. Spike sorting was performed group by group (Fig. 4.9b shows the group configuration).

[10]Four 64-channel silicon probes have 256 channels. Among them, 40 4-electrode groups (160 channels) have been used in this test.

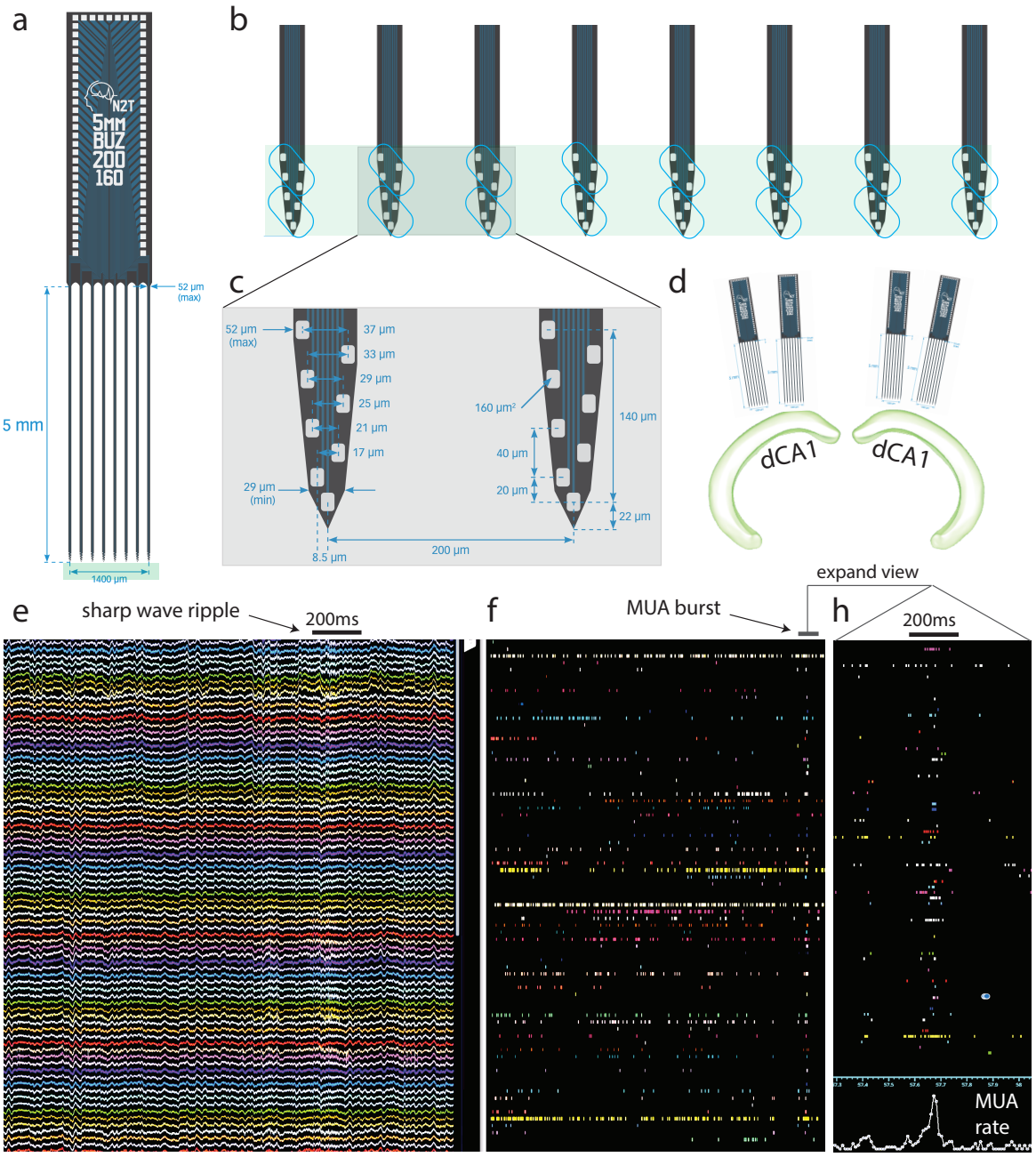


Fig. 4.9 The silicon probe electrodes and the recorded dataset in dorsal hippocampal CA1 used to test real-time population decoding: (a) A 5mm long, 8-shank Neuronexus silicon probe (Buzaki-64 probe). (b) The group configuration of a single probe: four electrodes form a group is surrounded by a blue rectangle. Each shank has 8 electrodes and two groups, and each probe has 16 groups (64 channels). (c) The electrode geometry of two shanks (among 8 shanks) of one probe. (d) Four probes (256 channels) were implanted into rat dorsal hippocampal area CA1. (e) A screenshot (an acquisition software GUI adapted from the Open-Ephys GUI) of the real-time data acquisition of 160 channels of the raw extracellular signals. The duration of a sharp-wave-ripple event is highlighted with a black bar. (f) A screenshot (a Spiketag spike raster GUI) of the real-time rendering of spike trains of 76 neurons that are simultaneously recorded and identified by the FPGA NSP (1 ms latency for each spike; and **see movie at https://youtu.be/th9_POUjwjg?t=8**). A multi-unit burst, corresponding to a SWR replay event in (f), is highlighted with a black bar and expanded in time in (h).

An important additional step in this test is to quantify the spatial information that each single-unit carries. Sortable neurons were found in 18 groups (among 40), among which eight groups were in the left hippocampal CA1 and ten groups were in the right hippocampal CA1. The firing pattern of many neurons in CA1 is well-known to be spatially tuned. These cells are known as place cells (see Chapter 1 for more introduction on the place cell). Two place cells' firing locations are plotted (Fig 4.10 a,b) along with their place fields (Fig. 4.10 a,b: in the left is the firing location of each spike from that neuron, the right is this neuron's place field; the maximum firing rates of these two neurons are 5.79 Hz and 4.80 Hz respectively). The place field is calculated by correlating the firing activity of individual neurons with the trajectory of the rat. First, the animal trajectory during navigation is interpolated and smoothed. Then the full map is binned with $2 \times 2cm$ squares, and an occupancy map (in matrix form) is calculated as the time that the animal spent in each spatial bin. After that, a firing map (a matrix of the same size as the occupancy matrix) is calculated, for each single-unit, to represent how many spikes the neuron fired in each spatial bin. A place field of a neuron is calculated by dividing its firing map by the occupancy map, followed by a 2D convolution using a two dimensional Gaussian kernel (standard deviation of 5 cm). This creates a smoothed firing rate map for that neuron. It is well known that the hippocampal dynamics vary greatly with or without the animal moving. To account for this effect, when calculating place fields, those spikes (and occupancy) occurring when the rat's speed is less than 10 cm s^{-1} are excluded. In summary, the place field was computed as a spatial firing rate map $f(x)$ where x is the spatial bin and $f(x)$ is the spike rate in the bin x. In addition, each neuron's information rate I (bits per second) was calculated as follows (Skaggs et al., 1993):

$$I = \int_x \frac{f(x)}{\lambda} \log_2 \frac{f(x)}{\lambda} p(x) dx$$

Where λ is the mean firing rate of the neuron and $p(x)$ is the occupancy probability that the rat is present in the xth bin. A typical place cell only exhibits high firing rate in a small portion of the environment. Therefore, each spike of these cells exhibits a high information rate (bits per spike) according to the above equation. The neurons that have spike information rates less than 0.25 bits/spike were excluded. In total, 76 single-units, with spatial information rate larger than 0.25 bits/spike, were used for the real-time decoding (Fig. 4.10). Their corresponding spike-sorted model was downloaded into the FPGA for the real-time decoding.

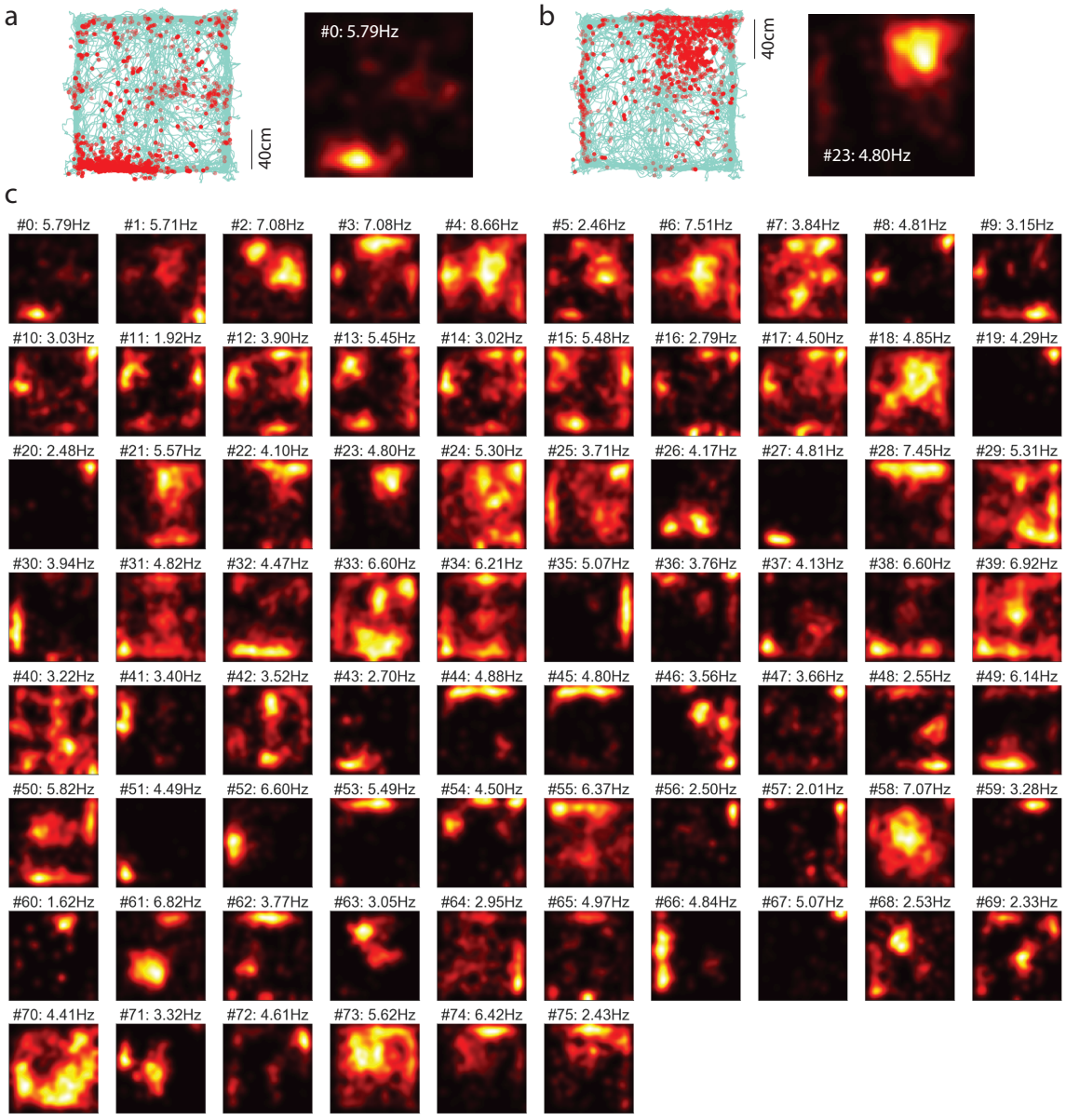


Fig. 4.10 Place fields of 76 simultaneously recorded single-units used for real-time decoding: Place fields of 76 simultaneously recorded single-units used for real-time decoding: (a) One example of a place cell. This cell only fires spikes (red dots) at one corner of the maze, left. Its calculated place field, right. The place field is a spatial firing rate map where each spatial bin is encoded by a firing rate in (Hz). The colormap of the field plot (the heatmap) spans the minimum firing rate (black) to the maximum firing rate (white). The cell number and its maximum firing rate are printed above the field plot. (b) Another example of a place cell. (c) In total, 76 cells that carry spatial information were used for real-time position decoding. Many of them possess multiple peaks of firing rate, resembling those from Pfeiffer and Foster (see Supplementary Fig.3 in (Pfeiffer and Foster, 2013)) except that interneurons are excluded from the analysis. In this dataset, the interneurons exhibit a less than 0.25 bits/spike information rate.

Before the real-time decoding, a simple visual inspection of the real-time spike trains of these 76 neurons was performed[11], in which the spike trains sent from the FPGA NSP were rendered in real-time as a raster plot along with the acquisition of the raw data[12] (Fig. 4.9 e).

The population spike trains from these 76 neurons, as the NSP's low-latency output, were first binned in real-time and then used for the online position decoding. Both real-time binning (a software FIFO) and decoder (a software Bayesian decoder) were implemented in Python code. Here, a widely adopted naive Bayesian algorithm (Zhang et al., 1998a) (equations 35,36) is used. This algorithm has two assumptions. First, the firing activities of an individual place cell are subject to Poisson statistics. Second, different neurons are statistically independent. Although greatly simplified, it has been proven to be highly accurate in decoding the animal position during animal running with a sufficient number of neurons in the hippocampus (Pfeiffer and Foster, 2013; Zhang et al., 1998a). The algorithm works as follows: First, the whole 2D maze was binned into $2cm \times 2cm$ spatial bins. At any given time t, the posterior probability $P(x_t|n_t)$ of being at position bin x, given $n_{i,t}$ spikes from a set of neurons indexed by i in the decoding time window τ is:

$$P(x_t|n_t) = C \cdot P(x) \cdot \left(\prod_{i=1}^{N} f_i(x)^{n_{i,t}} \right) \cdot \exp\left(-\tau \cdot \sum_{i=1}^{N} f_i(x) \right)$$

Here, C is a normalizing factor to make sure the conditional distribution $P(x_t|n_t)$ adds up to one[13]. It is assumed the animal would equally likely visit any place in the maze, hence the prior $P(x)$ is uniform and not time-dependent. Other variables that are not time-dependent include: $f_i(x)$ the average firing rate of cell i at position bin x, N the total number of place cells and the decoding window τ_t. At any time t, the decoded position $\widehat{x_t}$ is the most probable position bin given the spike count vector n_t:

$$\widehat{x_t} = \arg\max_{x_t} P(x_t|n_t)$$

In decoding the animal's position during running, the spike count vector n_t was binned each 50 ms, with a 250 ms decoding window. That is, each 50 ms a new vector n_t is formed to decode the animals' position, of which the ith element $n_{i,t}$ is the spike count of the ith neuron in the past 250 ms. A well-studied issue in one-step Bayesian decoding (see (Zhang et al., 1998a) Fig 3) is erratic jumping of the decoded trajectory, due to the non-linear max

[11] Such real-time visualzation conveys abundant information about the status of the tool as well as the underlying brain activities, helping experimentists and tool builders to make decisions during the experiment or the testing.

[12] See https://youtu.be/th9_POUjwjg?t=8

[13] See (Zhang et al., 1998a) equation 36.

operation and the occasional period with low number of spiking. Zhang et al (Zhang et al., 1998a) used a 2-step probabilistic decoding[14] scheme that was used as a continuity constraint to solve this issue. Here I use a simple moving average of the last 30 decoded positions to gain a similar smoothing effect. I found using the moving average of the last 20-40 decoded positions (50 ms update rate) leads to accurate and smooth decoded animal positions. In Zhang et al., it has been shown that the position decoding is most accurate for some animals if the last 1-2 seconds of CA1 spiking activities are used in Bayesian 2-step reconstruction (see Fig. 6A in Zhang et al., 1998a). Tampuu et al., (2019) further demonstrate that, when using CA1 spiking activity to decode animal position, a 1-2 second window is the optimal time window for both the Bayesian and the LSTM (long short-term memory) decoder. In my naive Bayesian decoding with 50 ms decoding update rate, a 1-2 second window corresponds to the last 20-40 decoded positions. By using the moving average of the last 30 decoded positions, I not only obtain a smoothed and accurate estimation of the animal's current position (using the spiking activities from the last 1.5 s + 250 ms)[15] but also maintain a fast update rate (every 50 ms). A fast decoding update rate can be useful in BMI applications, because it can be translated into an actuator or virtual reality command and fed back to the animal at a high rate. The real-time decoded trajectory was then compared to both off-line decoding and the true trajectory (Fig. 4.11). Cross-validation was employed so that the spike sorting and place field analysis were performed on the training set (the first 70% of the data) and both online and offline decoding were performed on the test set (the remaining 30% of the data). Again, the recording when the rat's running speed is less than 10 cm s^{-1} was excluded from the test. The full test set contains a 320 second long true trajectory as the ground-truth trajectory.

Although both the online and the offline decoding adopt the same decoding algorithm described above (and this is always true in this chapter), the online decoding result can deviate from the offline decoding result if inaccurate real-time spike inference occurs in population level. Therefore, my first goal is to quantify the discrepancy between the offline (no NSP involved) and online (using NSP-produced produced real-time spike trains) decoding.

[14]Bayesian 2-step reconstruction: The decoded probability of the animal's current position not only depends on the current neural activities but also the animal's position in the last step.

[15]The decoding performance not only depends on how long the past neural activities are used but also depends on how many neurons are used. The effect of the number of CA1 neurons in the position decoding is shown in Fig. 4.14

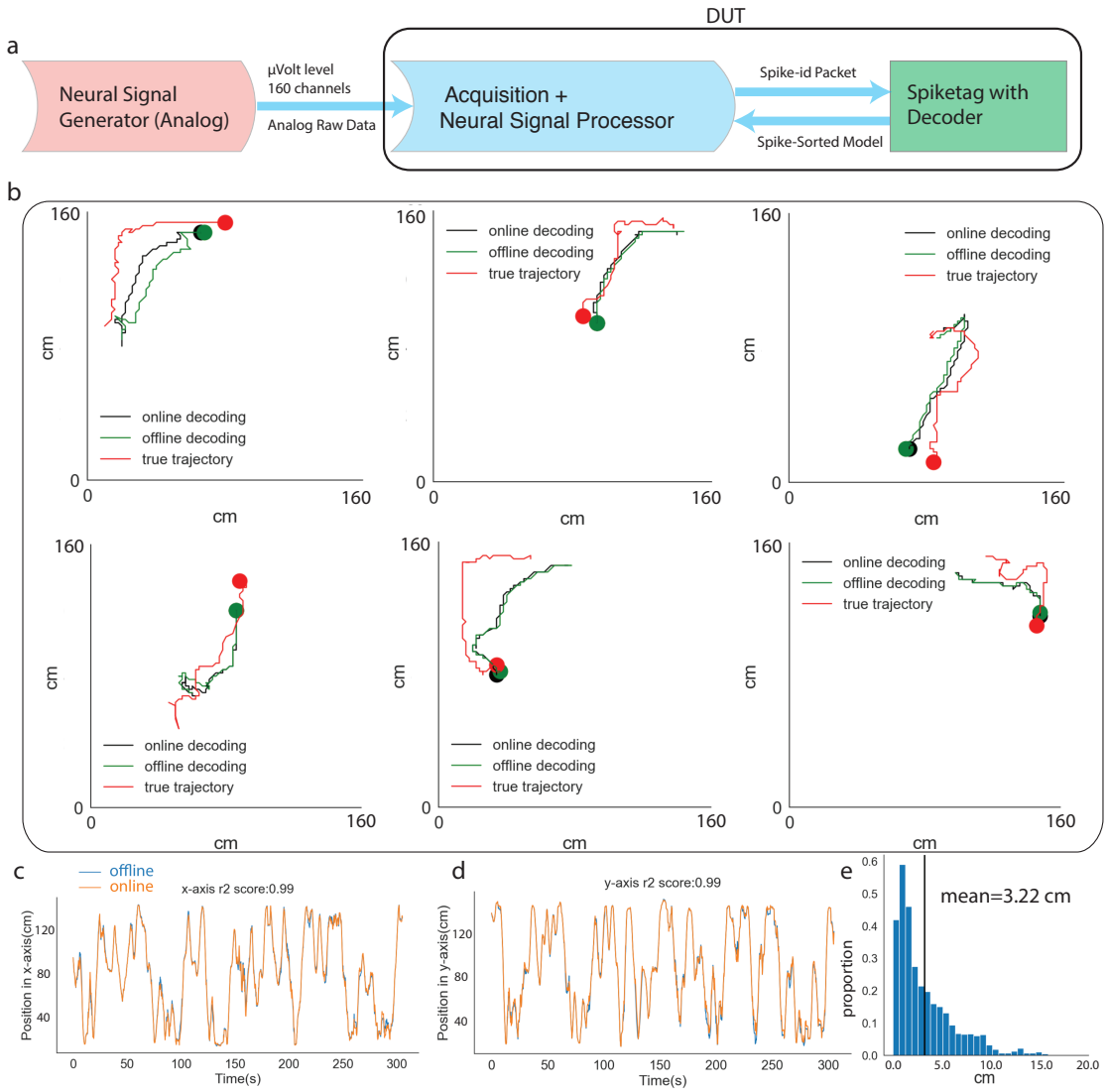


Fig. 4.11 Online population decoding of animal trajectory versus offline decoding during animal movement: (a) Test configuration: here the DUT includes the NSP and the real-time decoder. (b) Six episodes of test results: the current (round marker) and past four seconds (line) of the true, online and offline estimated animal positions are denoted in red, black and green respectively. (c) The overlayed trace of the offline and online decoded trajectories along the x-axis for the full test set. (d) The overlayed trace of the offline and online decoded trajectories along the y-axis for the full test set. The R-square scores between the offline and online decoded x and y position are both 0.99. (e) The histogram of the distance between online and offline decoded trajectories. The mean distance between the offline and online decoded trajectories is $3.22cm \pm 3.0cm$.

The test results shows very little difference between the offline and online decoded trajectory (Fig. 4.11 b,c,d,e). The maximum distance, among all estimated positions, is less than 20 cm (Fig. 4.11 e), which is roughly the length of the animal itself. The mean distance is $3.22cm \pm 3.0cm$ between the offline and online decoded trajectories. Both offline and online decoded trajectories were highly correlated with true trajectories (see the examples in Fig. 4.11 b; will be further analyzed later). The close overlap between offline and online decoding is expected, given the validated accuracy of the NSP in previous sections. From this, I conclude that the basic engineering aim of the NSP-based population decoding is reached, that the NSP generates accurate enough population spike trains that the real-time decoder produces almost the same result as the offline decoder.

In order to examine the possibility of applying the real-time position decoding in a closed-loop hippocampal BMI experiment, I further quantify the statistical difference between online/offline decoded trajectories (same) and true trajectory.

Here I first quantify the decoding error statistics in x-axis and y-axis position independently (as in Zhang et al., 1998) during running, using the R-square value. The R-square score is a commonly used metric in the BMI works (Lebedev and Nicolelis, 2017) and machine learning based decoders (Glaser et al., 2017) to quantify the performance of regression from one variable to another. Here I use it to quantify the fit between the decoded trajectory and the true trajectory. The test shows that the R-square score is 0.88 in the x-axis and 0.91 in the y-axis (Fig. 4.12 a,b). This can be seen as a very high score (see Glaser et al. 2017 Fig. 4 where both Naive Bayesian and LSTM decoder achieve an R-square value of less than 0.75 in a similar 2D environment). Accordingly, the 90th percentile error distribution in both x and y axes is less than $27.44cm$ and $23.92cm$, respectively[16], while the mean error of x,y axis is $13.04cm$ and $11.92cm$, respectively.

[16]To compare, the average length of a Long-Evans rat is 20 cm.

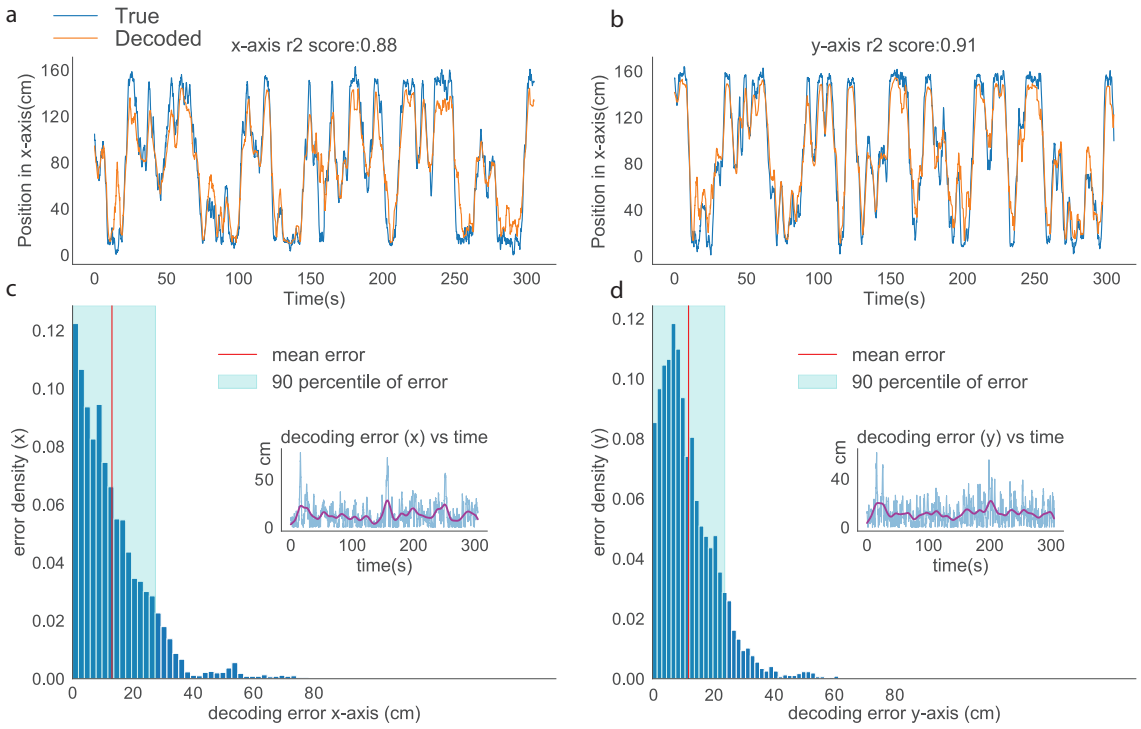


Fig. 4.12 Decoding error in the x and y axis of the 2D maze during animal movement: (a) The true and decoded x positions in the full 320 second long test set. The R-square score between the true and decoded x positions is 0.88. (b) The true and decoded y positions in the full 320 second long test set (during movement). The R-square score between the true and decoded y positions is 0.91. (c) The distribution of the decoding error in the x-axis. The mean and 90th percentile error are 13.04*cm* and 27.44*cm*, respectively. The inner box shows that the decoding error in the x-axis is stationary over time. The blue trace is the real-time decoding error while the purple trace is the smoothed error. (d) The distribution of the decoding error in the y-axis. The mean and 90th percentile error are 11.92*cm* and 23.92*cm*, respectively. The inner box shows that the decoding error in the y-axis is stationary over time. The blue trace is the real-time decoding error while the purple trace is the smoothed error.

Fig 4.12 shows the low decoding error in both the x-axis and y-axis of the 2D maze. However, this does not reveal whether the error has a covariance structure between x and y positions. To see how the error distributes around the animal, I quantified the spatial distribution of the decoding error relative to the animal's true position. The relative decoded position values (the decoded position subtracted by the animal's true position) have been spatially binned, followed by a kernel density estimation. The result shows that the spatial decoding error is concentrated around the animal (Fig. 4.13 a: the 2D distribution in red. The circle at the center is the estimated size of the rat).

To compare this performance with chance level decoding, I shuffled the 'spike-place field' mapping (using random permutation) and then ran the real-time decoding again. The chance level decoded position is no longer near the rat's position (Fig. 4.13 a: the 2D distribution in blue). The Euclidean decoding error (measured as the distance between the real-time decoded 2D position and the actual 2D position), using shuffled data, is significantly larger than both online and offline decoding error. Meanwhile, the mean distance between online and offline decoded positions is as low as $3.22cm \pm 3.0cm$ (Fig. 4.13 b and Fig. 4.11 e).

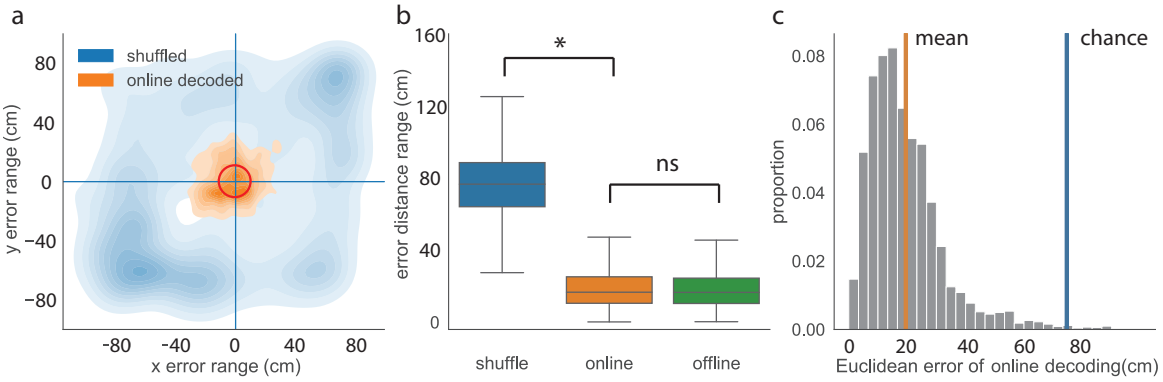


Fig. 4.13 Decoding error map and comparison to the chance level during animal movement: (a) The density maps of the decoded position relative to the actual animal position. The orange density map represents the real-time decoded position relative to the animal's true location (origin; the red circle represents the size of the animal); the blue density map represents the shuffled (chance level) real-time decoded position. The shuffling is implemented through random permutation of the relationship between the spike identity and its place field. The density map is calculated with kernel density estimation, which was performed using the Python package seaborn's "kdeplot" method with default parameters, a smooth Gaussian window with Scott bandwidth. 20 contour levels are used for visualization. (b) Comparison of the Euclidean distance between the 2D decoded position and the true 2D location for the shuffle, online and offline decoded groups. The shuffle group's Euclidean error is significantly higher than the other two groups. (c) The distribution of the Euclidean error in real-time decoding. The mean error ($19.60cm \pm 13.2cm$) is marked with an orange bar, while the chance level (shuffle group) is marked with a blue bar.

In the end, the Euclidean error in real-time decoding resulted in a mean error of $19.60cm \pm 13.2cm$ in a $160cm \times 160cm$ maze, while the chance level decoding error is around $76cm$ (Fig. 4.13 c). In comparison, Frey et al. 2019 has demonstrated that using wideband neural activities along with deep learning could result in a better decoding performance than single-units alone (see Frey et al. 2019 Fig. 1), in which the Euclidean mean error of decoding is $14.2cm \pm 12.9cm$ in a $120cm \times 175cm$ maze. A fair comparison would require more

information about the number of recording channel, the occupation map, and the total spatial information rate of the place cells etc. For the mean Euclidean error, this result is consistent and comparable to previously published offline decoding results. (Also see Tampuu et al. 2019 in which the best-reported decoding mean error is 12.50cm for the RNN decoder and is above 15cm for the Bayesian decoder.)

It is common, in BMI studies, to report the neuronal dropping curve (NDC), which describes the performance of real-time decoding as a function of neuronal ensemble size. The NDC curve provides an estimate of when the desired BMI application could fail from an insufficient number of neurons in a specific brain region (see Lebedev and Nicolelis, 2017: Fig.11). Here I report how the performance (both R^2 and mean Euclidean error) changes with the number of neurons randomly picked for the decoding (Fig. 4.14). The NDC graph is constructed by calculating R^2 and mean Euclidean error for all the units, then excluding one of the units from the population and calculating R^2 and mean Euclidean error again, and so on until only one neuron is left. This NDC construction procedure was repeated fifty times, from which 1500 bootstraps were created to calculate the 99% confidence level of the mean NDC curve (Fig. 4.14), for both R^2 and mean Euclidean error. The R^2 score is the most standard metric adopted in the BMI studies (Lebedev and Nicolelis, 2017), while the mean Euclidean error is more often reported in the studies that involve hippocampal population decoding. From the test result I observe the tight inversely proportional relationship between the two in the NDC curve of decoding the animal's current position during movement (Fig. 4.14).

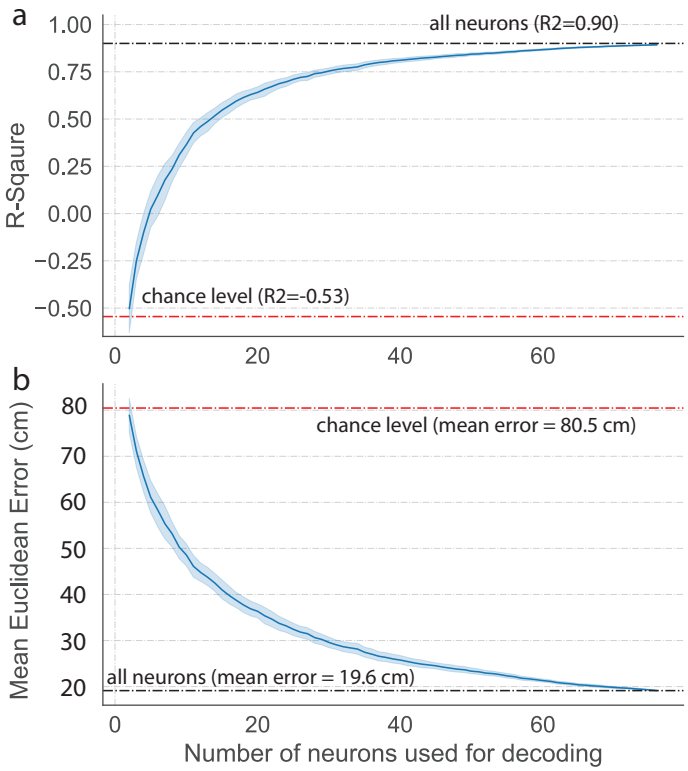


Fig. 4.14 Neuron dropping curve during animal movement: (a) The neuron dropping curve using the R-Square value. The R-Square value increases as more neurons are used for decoding. The blue band shows the 99% confidence level (1500 bootstraps from 50 runs; each run has a different random neuron removal order) of the mean neuron dropping curve. (b) The neuron dropping curve using the mean Euclidean decoding error. The mean Euclidean decoding error drops as more neurons are used for decoding. The blue band shows the 99% confidence level (1500 bootstraps from 50 runs; each run has a different random neuron removal order) of the mean neuron dropping curve.

This section only quantifies the decoding performance while the animal moves. The decoding result is updated each 50 ms. This large decoding interval does not benefit from the low-latency spike assignment capability. In the next section, a much faster decoding rate (5 ms interval) was adopted to test the decoding performance for replay events, a type of hippocampal neuron activity that is only generated when the animal is still (see chapter 1). The low-latency spike assignment capability is useful when the real-time decoding interval is as low as 5 ms (or even lower), because all the units inside the decoding window have to be identified before decoding. The faster the spike assignment can be finished in the FPGA NSP, the larger the amount of time is left for the decoding algorithm itself. To decode the 2D content of replay events, a short decoding window ($\sim$ 25 ms) is usually used, during

which each neuron can only fire a few spikes. In this case, small spike inference error could lead to a big decoding discrepancy between the offline and online setting (although the decoding algorithm is the same). Therefore, decoding awake hippocampal replay events and quantify the discrepancy between the online and offline decoded 2D trajectories will not only validate the speed but also further validate the accuracy of our system in real-time population decoding.

4.5 End-to-end real-time population decoding of awake hippocampal replay events

It is well established that rodent hippocampal dynamics changes when the animal stops moving. One of the most synchronous forms of hippocampal activity, known as the awake replay event, appears only when the animal is stationary (Davidson et al., 2009; Diba and Buzsáki, 2007; Gupta et al., 2010; Karlsson and Frank, 2009; Pfeiffer and Foster, 2013). During awake replay, many neurons tend to fire together and this synchronous population firing lasts for hundreds of milliseconds, during which the overall population firing rate surges, accompanied by a sharp-wave ripple (SWR, see awake replay and SWR in chapter 1). A threshold crossing of overall spike count has been used to identify awake replay events (Pfeiffer and Foster, 2013, 2015). During an awake replay event, it has been shown that a short (25 ms) and rapidly (3-5 ms) updated window generates a sequence of estimated positions that looks like a trajectory even when the animal is still. Some of these decoded trajectories predict the animal's immediate future behavior (Pfeiffer and Foster, 2013, 2015). These studies imply the possible role of awake replay events in planning, separate from the previously established role in memory consolidation (see chapter 1) (Buzsáki, 1989, 2015). However, the ability to decode the forward replay content in real time to study the causal role of replay events is still lacking. Hence the purpose of this section is to examine such a possibility by quantifying the difference between offline and online decoding, during hippocampal replay.

The decoder was trained on the validation set as in the previous section (first 70% of the full recording while the speed was larger than 10 cm/s) and was tested on the 653 second long test set (last 30% of the full recording without speed cutoff)[17] in an offline and online manner. The test configuration is the same as in previous sections (see Fig. 4.11), except the test data here includes replay events that happen when the animal is not moving. The same

[17] In contrast with the 320 second long test set during animal movement (in the previous section), in this test, all the periods, including those in which the animal was immobile, are included. Here I confirmed that the awake replay events (MUA population bursts) only occurred while the animal was immobile.

Bayesian decoding algorithm was adopted, with a 25ms decoding window advancing in 5 ms intervals.

In this test, I found similar awake replay events as reported by Pfeiffer and Foster that predict the animal's future trajectory. A selected example is shown in Fig. 4.15. This replay event lasts for 350ms, within which two MUA bursts are found (Fig. 4.15 b). The first MUA burst lasts for 200ms, during which both offline and online decoded posteriors (Fig. 4.15 d and e: the heatmap) form a trajectory which tightly predicts the animal's future behavior[18] (Fig. 4.15 d and e: the white line). In this example, the online decoded content (Fig. 4.15 e) is very similar to the offline decoded content (Fig. 4.15 d). Two more examples are shown in Fig. 4.16 and Fig.4.17.

[18]Interestingly, the second MUA burst encodes a trajectory that does not correlate with future behavior.

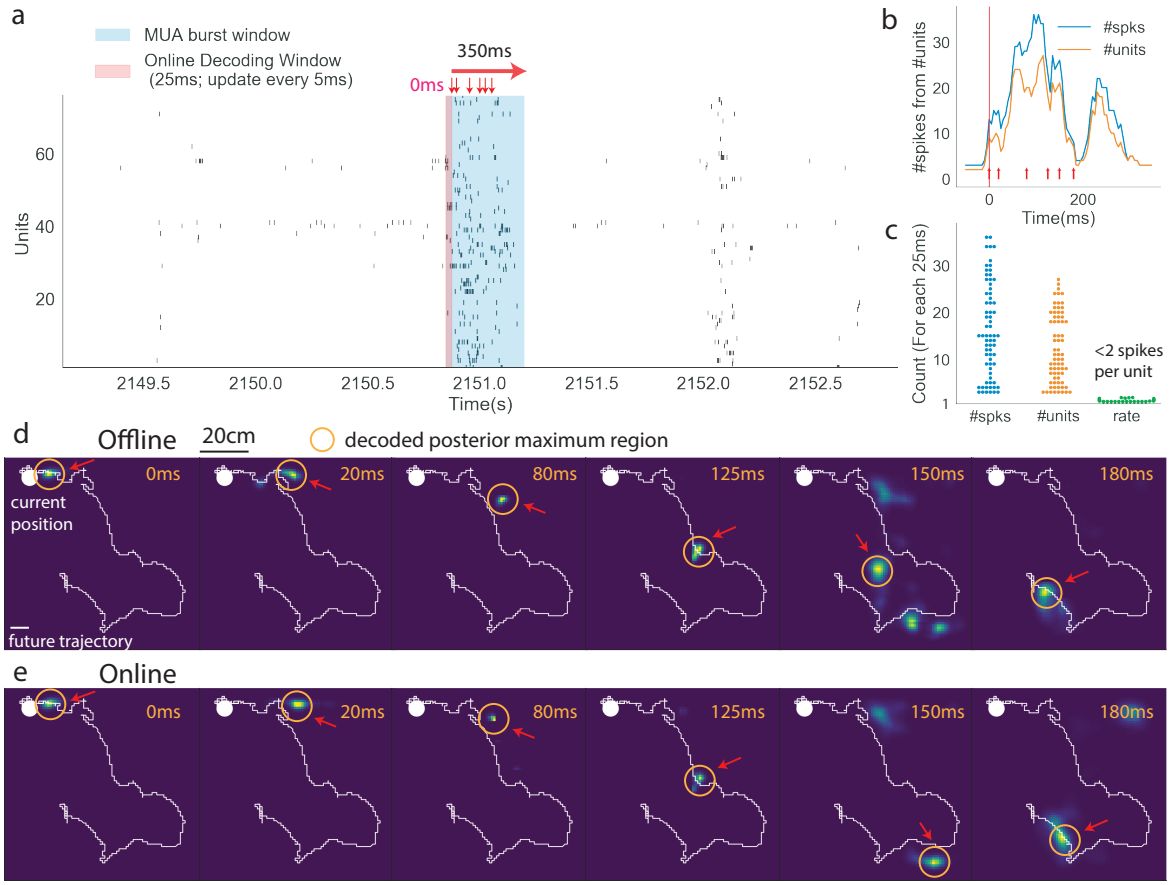


Fig. 4.15 Example 1 of real-time decoding of the awake hippocampal replay content: (a) The unit raster plot contains one identified replay MUA burst. A 350 ms long replay event is highlighted with a blue semi-transparent rectangle (MUA burst window). A 25 ms long decoding window is marked with a red semi-transparent rectangle (Online Decoding window). The start of the replay event is marked by the first red arrow with text (0 ms). The six red arrows indicate the window in which the decoded position jumps from one location to another (see d and e). Decoding was updated each 5 ms, meaning each 25 ms decoding window overlaps with the previous window by 20 ms. (b) The total multi-unit firing rate (number of spikes) in each decoding window is plotted in blue, while the total number of neurons that fire is plotted in orange. The end of the first decoding window is marked by a vertical line, and six red arrows indicate the end of the other decoding windows in which the decoded position clearly jumps from one location to another (same as a, and see d and e). The number of total spikes and the total number of firing neurons is highly correlated during the replay event. (c) During each 25ms window, each neuron (green dot) only fires a few spikes (rate <2 spikes/window), but many units (orange dot, the number of units in a window) contribute to the overall spike count (blue dot, the number of spikes in a window). (d) Six snapshots of the offline decoding, each one corresponding to a red arrow in (a) and (b). The white dot is the current position of the rat. During the replay event, the rat was still. The white line shows the immediate future trajectory after the rat restarts movement. The heat map (Viridis colourmap) depicts the decoded posterior at the time (in the upper-right corner). The maximum region of the posterior is highlighted by a circle pointed to by a red arrow. (e) Same as in (d) except that the snapshots display the real-time decoded posterior. Similar to the offline decoding, the sequence of posteriors predicts the animal's future trajectory.

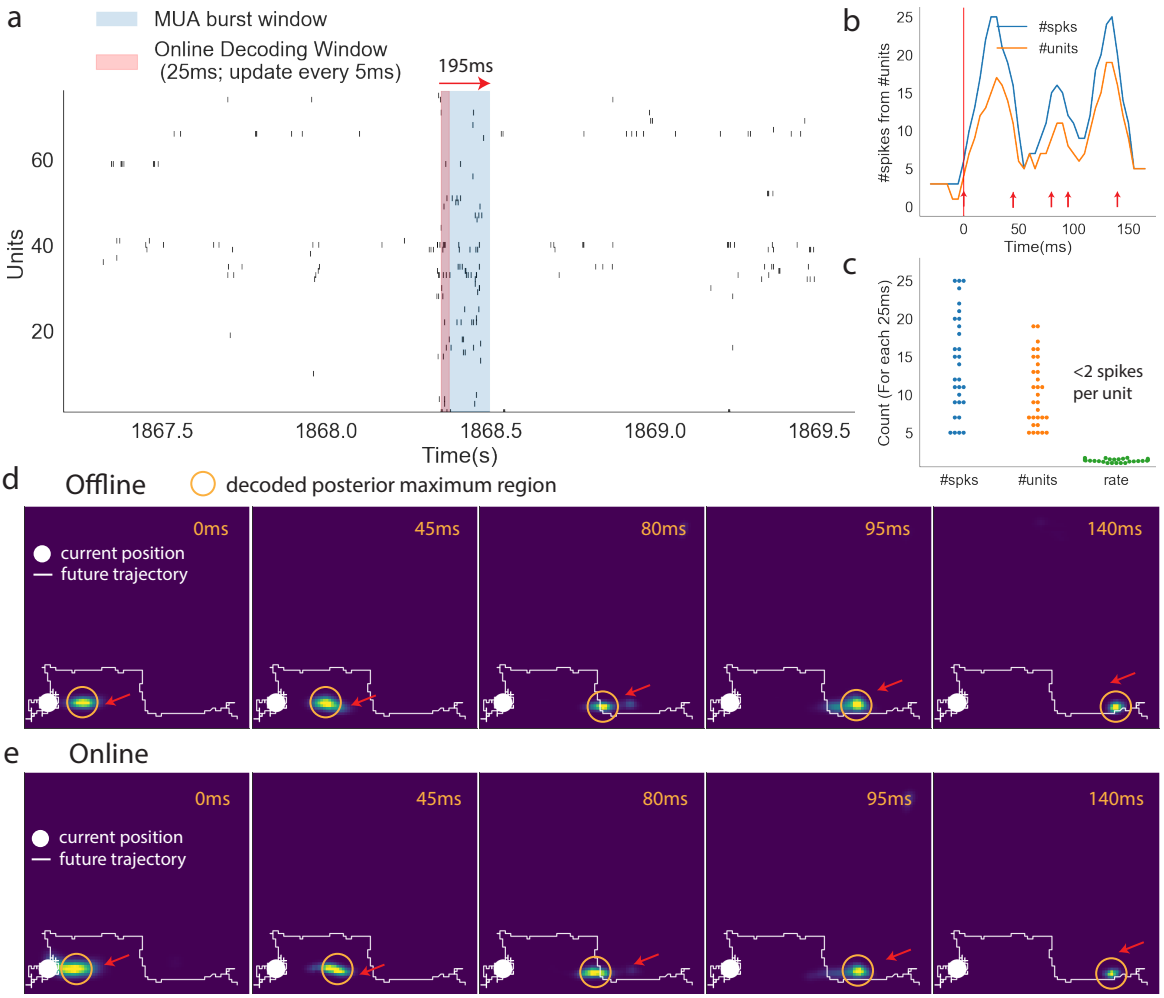


Fig. 4.16 Example 2 of real-time decoding of the awake hippocampal replay content: (a) The unit raster plot contains one identified replay MUA burst. A 195 ms long replay event is highlighted with a blue semi-transparent rectangle (MUA burst window). A 25 ms long decoding window is marked with a red semi-transparent rectangle (Online Decoding window). (b) The total multi-unit firing rate (number of spikes) in each decoding window is plotted in blue, while the total number of neurons that fire is plotted in orange. The end of the first decoding window is marked by a vertical line, and five red arrows indicate the end of the other decoding windows in which the decoded position clearly jumps from one location to another (see d and e). The number of total spikes and the total number of firing neurons are highly correlated during the replay event. (c) During each 25 ms window, each neuron (green dot) only fires a few spikes (rate < 2 spikes/window), but many units (orange dot, the number of units in a window) contribute to the overall spike count (blue dot, the number of spikes in a window) in a short window. (d) Five snapshots of the offline decoding, each one corresponding to a red arrow in (b). The white dot is the current position of the rat. During the replay event, the rat was still. The white line shows the immediate future trajectory after the rat restarts movement. The heat map (Viridis colourmap) depicts the decoded posterior at the time (in the upper-right corner). The maximum region of the posterior is highlighted by a circle pointed to by a red arrow. (e) Same as in (d) except that the snapshots display the real-time decoded posterior. Similar to the offline decoding, the sequence of online posteriors predicts the animal's future trajectory.

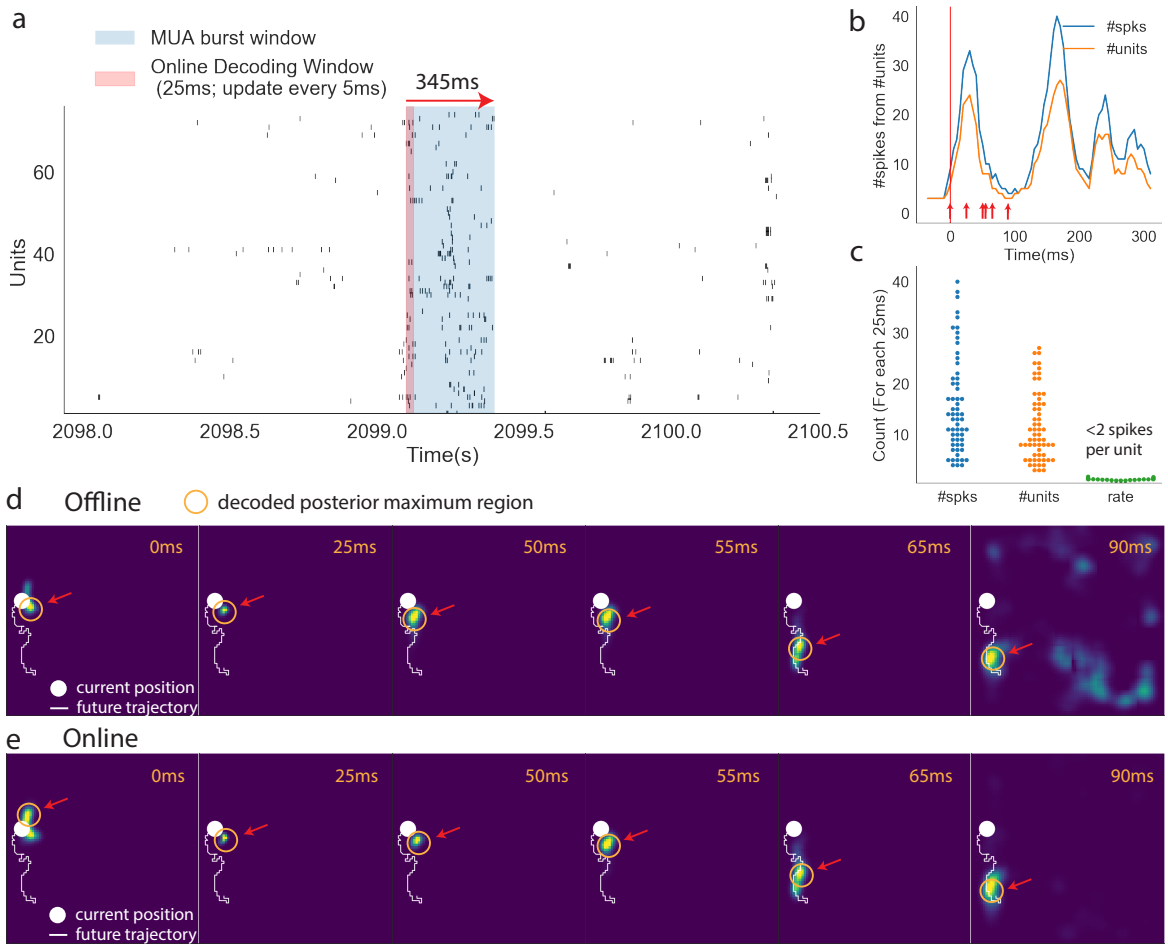


Fig. 4.17 Example 3 of real-time decoding of the awake hippocampal replay content: (a) The unit raster plot contains one identified replay MUA burst. A 345 ms long replay event is highlighted with a blue semi-transparent rectangle (MUA burst window). A 25 ms long decoding window is marked with a red semi-transparent rectangle (Online Decoding window). (b) The total multi-unit firing rate (number of spikes) in each decoding window is plotted in blue, while the total number of neurons that fire is plotted in orange. The end of the first decoding window is marked by a vertical line, and six red arrows indicate the end of the other decoding windows in which the decoded position changed from one location to another (see d and e). The number of total spikes and the total number of firing neurons are highly correlated during the replay event. (c) During each 25 ms window, each neuron (green dot) only fires a few spikes (rate < 2 spikes/window), but many units (orange dot, the number of units in a window) contribute to the overall spike count (blue dot, the number of spikes in a window) in a short window. (d) Six snapshots of the offline decoding, each one corresponding to a red arrow in (b). The white dot is the current position of the rat. During the replay event, the rat was still. The white line shows the immediate future trajectory after the rat restarts movement. The heat map (Viridis colourmap) depicts the decoded posterior at the time (in the upper-right corner). The maximum region of the posterior is highlighted by a circle pointed to by a red arrow. (e) Same as in (d) except that the snapshots display the real-time decoded posterior. Similar to the offline decoding, the sequence of online posteriors predicts the animal's future trajectory.

These identified examples illustrate the agreement between the online and offline decoded replay trajectory content using the rapidly updating (5ms) short decoding window (25ms). Also, these examples of replay content are highly correlated with the animal's future movement trajectory. However, there are more replay events, maybe other types of replay events, that do not predict future behavior. To reliably quantify the difference between the offline and online decoder during various types of replay events, we need to collect more samples of different types of replay events for statistical analysis. Regardless of the replay content, all replay events are population bursts during which the overall spike counts increase dramatically. Importantly, the dramatically increased number of spikes were generated from different neurons (Fig. 4.15 4.16 4.17 b,c). Almost all neurons only contribute less than two spikes in each 25ms decoding window (Fig. 4.15 c: the green dots). In the following test, additional replay events were automatically identified by a simple criterion, within a single 25 ms decoding window at least five spikes from at least five different neurons (see Fig. 4.15 4.16 4.17 b) were detected. This simple criterion reflects a hallmark of replay events, that is, not only does the overall spike count sharply increase, but also many neurons participate simultaneously. To rule out the possibility that the test result can be biased by using only one fixed parameter, various parameters have been used. The mean distance between the online and offline decoder has been measured as a function of n, where n means at least n spikes from at least n neurons. The test result shows that the difference between online and offline decoders is consistently less than 10 cm, while the parameter n changes from 1 to 30 (Fig. 4.18 a). For $n = 5$, that is, for all decoding windows that contain at least five spikes from at least five neurons, the mean distance between offline and online decoders is 7.1 cm (Fig. 4.18 a) while the median difference is 4.18 cm (Fig. 4.18 b).

In addition to accuracy, the low-latency response to the decoding result is also important. Since the decoded content to trigger the feedback is dependent on the specific question, here I only quantify how rapidly the FPGA can respond to the end of every decoding result. The latency is tested using the PC-FPGA memory interface. Once the criterion is reached (at least five spikes from at least five neurons), a random number is written into the FPGA's memory[19] and then immediately read back to the PC via the PCIe interface. The time for the writing operation was measured as the one-way latency, while the time for both the writing and reading back operation was measured as the round-trip latency. The read-back data were used to validate the success of the writing operation. The test result shows the mean one-way latency is $93\mu s$ and the mean round-trip latency is $175\mu s$ (Fig. 4.18 c). This latency is

[19]The data written to the FPGA memory can be connected to a vector of TTL signals to further trigger the patterned external stimulation within a few nanoseconds of latency (4 nanoseconds for a 250MHz clock), but this FPGA memory mapped stimulation has not been integrated into the current version of the FPGA implementation.

consistent with the well-known PCIe transmission latency (Preußer and Spallek, 2014)) and is sufficiently low for the possible feedback triggered by any 5 ms online decoding update.

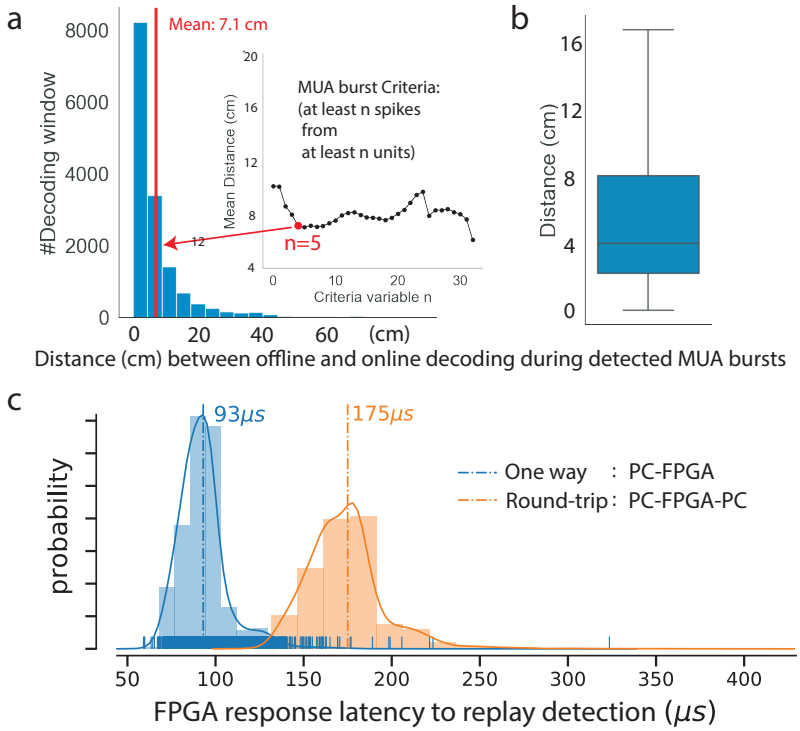


Fig. 4.18 Accuracy and latency of real-time decoding during population bursts: (a) The histogram of the distance between the online and offline decoded positions of all decoding windows (a sliding 25 ms window with 5 ms intervals) during population bursts ($\geq$5 spikes from $\geq$5 neurons). All of the decoding windows during the test set were used to calculate the posterior. The maximum of the posterior was considered as the decoded position, but only the decoding windows that satisfy the population bursts condition ($\geq$5 spikes from $\geq$5 neurons) were used to compare the distance between the offline and online decoded positions. Among these decoding windows, the mean Euclidean distance between offline and online decoders is 7.1 cm. The inner box curve (black line) shows that as parameter n changes for the population bursts detection ($\geq$n spikes from $\geq$n neurons, $n > 0$), the mean decoding difference remains low, suggesting that the online decoding is not sensitive to the parameter n. (b) The boxplot shows the 25th, median and 75th percentile of the distance between offline and online decoders over all of the decoding windows for $n = 5$ ($n = 5$ is marked with a red dot in (b)). (c) The latency distribution illustrates how rapidly the FPGA responds to a possible triggering command during replay events. Each tick (on top of the x-axis) is an FPGA memory write event triggered by an MUA burst detection condition ($\geq$5 spikes from $\geq$5 neurons). Both one-way (PC to FPGA) and round-trip (PC to FPGA to PC) latencies for this MUA burst detection condition were measured. The mean one-way PC to FPGA latency is 93μs while the mean round-trip latency is 175μs.

Chapter 5

Discussion

5.1 Single-units in BMI experiments

In previous chapters, I presented the design and development of a system (FPGA NSP + Spiketag software) that can assign identities to a population of single-units in multi-electrode recordings, at single-spike resolution with low-latency. In the previous chapter, I validated the low-latency and accurate NSP spike inference performance using ground-truth data as well as accurate NSP-based real-time decoding using data recorded from rat hippocampus CA1. In the first chapter to justify our motivation, I suggested various types of experimental applications that would be made possible with this tool . However, there are a few technical concerns that remain to be discussed in this chapter in greater detail. The first concern is whether accurate spike sorting for single-units is always necessary in BMI experiments. When and how can we bypass spike sorting, and when is accurate spike sorting necessary or desirable?

5.1.1 Can we bypass spike sorting in BMI experiments?

Based on the inter-electrode distance, we had broadly categorized multi-electrode devices into two classes. In class-1 devices, such as tetrode arrays or silicon probes, each spike can be detected by at least one electrode in a local electrode group. In class-2 devices, such as microwire arrays, each spike can be detected by at most one electrode. Microwire array technology is not ideal for isolating single-units because of its large inter-electrode distance, but it remains the most popular recording method in BMI experiments where spike sorting is often neglected. Therefore, previous engineering work in real-time spike assignment was mainly based on microwire arrays (see review Lebedev and Nicolelis, 2017). Accordingly, the real-time spike assignment in previous studies either used simple threshold crossing

(Carmena et al., 2003; Christie et al., 2014; Fraser et al., 2009; Golub et al., 2018; Hochberg et al., 2012; Sadtler et al., 2014; Santhanam et al., 2006; Schwarz et al., 2014; Sussillo et al., 2016) or single-unit inference on each of the independent electrodes in implanted microwire arrays (Athalye et al., 2018; Chapin et al., 1999; Ganguly et al., 2011; Gulati et al., 2017, 2014; Koralek et al., 2012; Neely et al., 2018; Wessberg et al., 2000). In either case, the output unit labels do not necessarily originate from true single-units (Gray et al., 1995; Harris et al., 2000; Henze et al., 2000). Meanwhile, previous studies also suggested that accurate single-units isolation via careful spike sorting might not be necessary for the population-level data analysis and BMI decoding. Two methods to bypass spike sorting in real-time spike assignment emerged from previous studies. Here I further discuss the strengths and limitations of both methods.

5.1.1.1 Bypass spike sorting by using multi-unit threshold crossings for real-time decoding

Carmena et al., (2003) first demonstrated that macaque monkeys can learn to control the firing rates of multi-units to reach and grasp virtual objects. As the authors put it in their landmark paper (Carmena et al., 2003), "this observation is essential because it eliminates the need to develop elaborate real-time spike-sorting algorithms, a major technological challenge, for the design of a future cortical neuroprosthesis for clinical applications".

Bypassing spike sorting in this way brought great convenience to the BMI experiments, which might be one of the main reasons why microwire arrays are still predominantly used in even state-of-the-art motor cortical BMI studies (Gilja et al., 2015; Golub et al., 2018; Hennig et al., 2020; Pandarinath et al., 2017; Sadtler et al., 2014), in which the activity of each 'neural unit' is defined as "the number of threshold crossings recorded by an electrode in non-overlapping 45 ms bins" (Golub et al., 2018; Hennig et al., 2020; Sadtler et al., 2014), and the population vector of 'neural units' is fed to various decoders to produce BMI behavior.

In both BMI decoding and population-level analysis, dimensionality reduction[1] (Cunningham and Yu, 2014; Saxena and Cunningham, 2019) is commonly used as a critical step to find "a neural manifold that captures the prominent patterns of co-modulation" among a large number of recorded 'neural units' (Golub et al., 2018; Hennig et al., 2020; Sadtler et al., 2014). The neural population dynamics in the low-dimensional manifold (also named the 'neural trajectory') is viewed as reflecting the underlying computation that causes the behavior

[1] e.g., principal component analysis (PCA), Gaussian-process factor analysis (GPFA) (Yu et al., 2009), demixed PCA (dPCA) (Kobak et al., 2016), tensor components analysis (TCA) (Williams et al., 2018), latent factor analysis via dynamical systems (LFADS) (Pandarinath et al., 2018)

(Saxena and Cunningham, 2019; Trautmann et al., 2019). Trautmann et al., (2019) further demonstrated that spike sorting is not required in such population dynamics estimation, in which the authors used "Neuropixels probes in motor cortex of nonhuman primates and reanalyzed data from three previous studies and found that neural dynamics and scientific conclusions are quite similar using multiunit threshold crossings rather than sorted neurons."

Nonetheless, a prominent limitation of multi-unit threshold crossings is that it is not generally applicable in population decoding or estimating the low-dimensional neural trajectory. To bypass spike sorting using multi-unit threshold crossings, the representations of neurons around an electrode have to be similar, or at least not completely different (Trautmann et al., 2019). For Trautmann et al., 2019, the single-units around an electrode tend to be redundant in information encoding in their firing rate, hence pooling multi-units from anatomically neighbouring single-units using threshold crossings would not lead to significant information loss. This is because the representations of neurons in primate neocortex are indeed similar across anatomically neighbouring neurons. The columnar structure in primate neocortex (Georgopoulos et al., 2007; Hatsopoulos, 2010) makes it an ideal brain region to apply multi-unit threshold crossings. In other brain regions or other species, this premise might not hold. For example, in the rodent hippocampus, anatomically neighbouring place cells recorded from a single tetrode can encode completely different places (see O'Keefe et al. (1998) Figure 1). In this case, pooling multi-units from multiple anatomically neighbouring single-units would lead to considerable information loss. There is not a single study, to our knowledge, that has shown that multi-unit threshold crossings can be used to decode location as accurately as when using the single-units in hippocampus with the same number of electrodes.

5.1.1.2 Bypass spike sorting by using waveform features for real-time decoding

Another method known as 'clusterless' (or unsorted) decoding has been developed, in which neither multi-units nor single-units are used. The"spike sorting step is bypassed by creating a direct mapping between spike waveform features and stimulus" (Deng et al., 2015; Kloosterman et al., 2013). In these studies, the authors adopt the Bayesian framework for decoding. Here I reinterpret this type of clusterless decoding as having a resemblance to both multi-unit and single-unit decoding. It classifies each spike, according to its waveform features, into another type of unit; here I call it a 'feature-unit'. To decide which 'feature-unit' to assign a spike to, clusterless decoding divides a feature space into a grid of nonoverlapping regions, in which each spike is contained in one and only one region. The 'feature-unit' label is assigned according to the grid region it belongs to. Since each grid region has its own

firing rate, the tuning curve of each grid region in the feature space can be calculated by relating the average firing rate of the grid region to the behavioral variables[2].

By splitting the feature space into a large number of grid regions, the real-time assigned spike 'feature-unit' (i.e., which grid region) can be directly used to decode the animal location without knowing the source neuron. According to the authors (Deng et al., 2016), "currently, real-time spike-sorting algorithms tend not to be sufficiently accurate to allow for closed-loop interventions". Therefore, clusterless decoding represents an important advance for online population decoding without spike sorting, and recent studies have efficiently implemented clusterless decoding for real-time hippocampal replay content classification (Ciliberti and Kloosterman, 2017; Deng et al., 2016; Hu et al., 2018).

Clusterless decoding is almost as convenient as multi-unit decoding because both automatic clustering and manual curation are bypassed. Unlike multi-unit threshold decoding, it does need to extract spike waveforms from adjacent electrodes and transform the waveforms into feature vectors (Fig .5.1a). The feature extraction can be implemented by a PCA transformation, and a simple arithmetic algorithm can decide in which grid region the feature vector is embedded. Clusterless decoding is compatible with tetrode arrays and silicon probes since the feature spaces can be generated from non-overlapping electrode groups. Therefore, it can be used to decode behavioral variables in brain regions that do not have columnar structure and in which neurons are densely packed in a small volume such as rodent hippocampus. This renders clusterless decoding more generally applicable than the multi-unit decoding.

Importantly, can the 'unit' sources of spikes be arbitrarily defined for the purpose of decoding? In order to decode behavioral variables, only a few assumptions are required. First, each spike can be traced to a unique 'unit' source (i.e., an electrode for a multi-unit, a putative neuron for a single-unit, a grid feature region for a feature-unit). Second, each 'unit' has a firing rate. Third, the firing rate of each 'unit' is potentially correlated with the behavioral variables. Critically, it is the behavioral variables that give rise to the firing rate of each 'unit', regardless how the 'unit' is defined. Therefore, we should be able decode the probability of a behavioral variable given observed firing rates of arbitrarily defined 'units'. It is possible that in the future other studies will find other ways of defining a 'unit' source of each spike, other than multi-unit or feature-unit, that increase the accuracy or specificity of the decoding, without knowing the actual single-unit source of each spike.

[2]Calculating the place field of a feature-unit is equivalent to calculating the place field of a single-unit (see section 4.4): a single-unit place field is calculated according to a spatial firing map of spikes produced by a putative neuron; a feature-unit place field is calculated according to a spatial firing map of spikes of which the features are confined in a specific grid region in the feature space.

5.1.1.3 Unit model landscape: a unified perspective

The multi-unit, single-unit and feature-unit are special cases in building the unit model for real-time spike inference and population decoding. A generic unit model landscape is shown in Fig. 5.1, in which the x-axis is the number of units and y-axis is the number of correctly assigned single-units. The multi-unit model and feature-unit model are two extreme examples in the unit model landscape. In the multi-unit model, each multi-unit contains more than one single-unit. Therefore, the total number of units in the model and the number of accurate single-units are both low. A spike sorting procedure starting with a multi-unit model is to split each multi-unit into multiple single-units. In the feature-unit model, each feature-unit can be part of a single-unit given a feature space is over-split. Therefore, the total number of units in the feature-unit model is much higher than the number of accurately assigned single-units. A spike sorting procedure beginning with an over-split feature space (i.e., over-gridded feature regions) is to merge multiple tiny grid regions into a single-unit cluster. Our NSP readily contains all necessary modules for both multi-unit and feature-unit inference (Fig. 5.1a), and the type of model loaded into the NSP determines whether it outputs multi-units, feature-units or single-units. A single-unit model requires more time to build, but its units correspond to accurate single-units (Fig. 5.1b). Next, I will discuss what are the unique advantages of using accurate single-unit models in the BMI experiments with some examples.

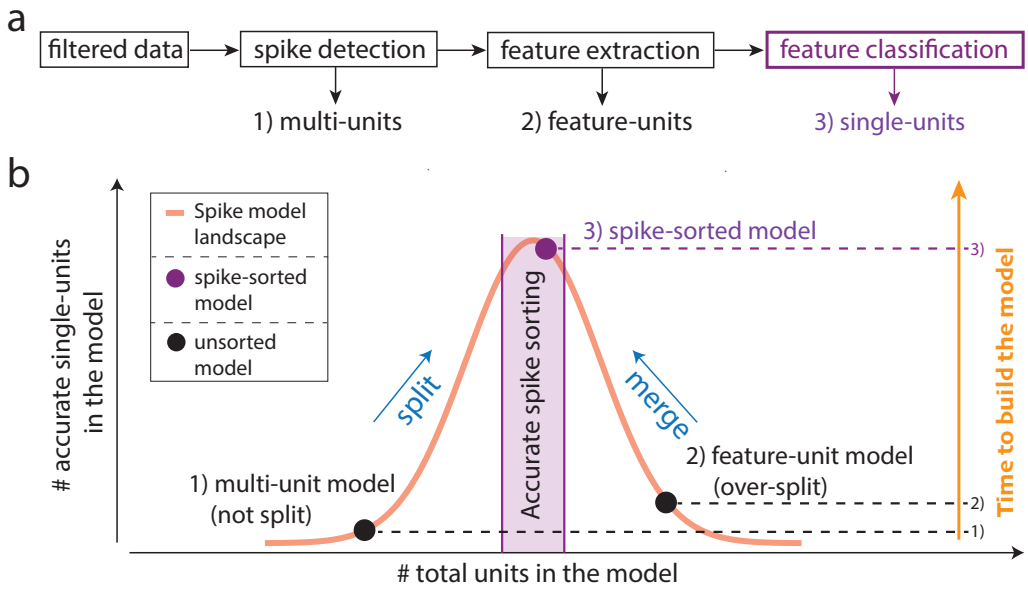


Fig. 5.1 Unit model landscape: (a) A simplified schematic view of the necessary modules to produce multi-units, feature-units and single-units in real time. Our single-unit inference NSP contains all the necessary modules for both multi-unit and feature-unit inference. (b) Unit model landscape (the curve). Spike detection (threshold crossing) produces multi-units, spike sorting produces single-units, and an unsupervised spike feature extraction followed by a simple arithmetic rule (to decide in which grid region the feature vector is) produces feature-units. The time used to build a single-unit model is much longer than to build a multi-unit model or feature-unit model (orange axis). The multi-units can be split (blue arrow), or the feature-units can be merged (blue arrow) to obtain an accurate single-unit model (purple dot). Both are commonly used as strategies of automatic clustering and manual curation in spike sorting. Accurate spike sorting is the middle part (purple region) of the unit model landscape as the number of total units in the model approximates the number of accurate single-units.

5.1.1.4 What are the unique advantages of using single-units in BMI experiments

In the multi-unit and feature-unit models, each 'unit' has a behavior-dependent firing rate (either in BMI or natural behavior) regardless of the nature of the 'unit' itself. Both multi-unit and feature-unit models can work for population decoding using a population rate vector produced by a large number of 'units'. However, such online decoding can only employ the rate coding scheme. The single-unit model, on the other hand, reflects the genuine source of each spike, the neuron. Therefore, importantly, only the single-unit model can be used to identify temporal codes (see section 1.2.2) such as bursting or specific relative timing relationships among several neurons. Fig. 5.2 illustrates the unique ability of single-units inference to employ temporal codes in putative BMI tasks (also see Sadtler et al., 2014: Fig 1 for comparison). Whether an animal can learn to control various forms of predefined temporal codes (e.g., in Fig. 5.2b) is unknown, therefore worth testing in actual BMI experiments.

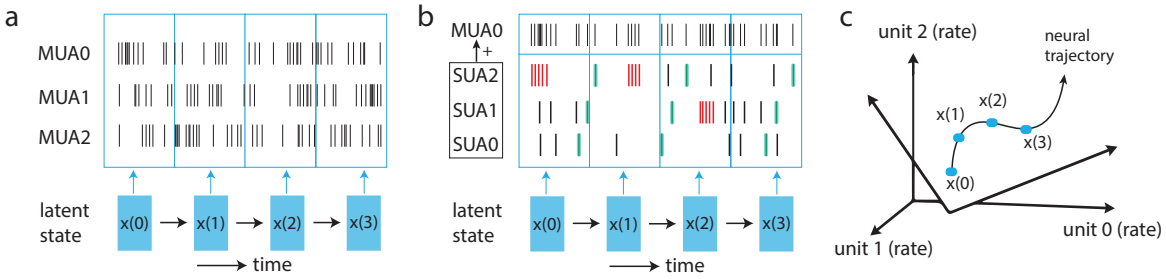


Fig. 5.2 Only single-units based BMI can employ temporal codes: (a) Three multi-unit (MUA) spike trains (black) produced by the latent neural states during a putative BMI trial. Each latent state corresponds to a predefined time bin (blue) for decoding. (b) The MUA0 consists of three single-units (SUA). Two temporal patterns are shown (red: burst; green: an ordered sequence). Both can be used to test whether the animal can control such temporal patterns rather than the latent neural states. (c) Both SUAs and MUAs can be used to extract the underlying neural state transitions (the neural trajectory) in a low-dimensional neural manifold (2D shown) using population dimensionality reduction (intrinsic neural manifold in Sadtler et al., 2014). Neither the temporal codes in (b) nor the cell-type specificity of each single-unit in (b) are present in the MUA and the neural trajectory in (c).

Even in the rate coding regime, there are neuroscience-related questions that require tracking of the activity level of multiple single-units in BMI experiments. For example, to investigate the mechanism of credit assignment in BMI learning[3], we need to check how BMI learning affects each individual neuron distinctly from the others (Gulati et al., 2017, 2014). In these studies, the authors found that, during sleep, credit assignment treats the task-related single-units and the task-unrelated single-units differently[4]. Why can these studies not use multi-units and draw the same conclusions? Because credit assignment, by definition, is a process whereby a network decides how much 'credit' or 'blame' a given neuron or synapse should receive. The network can adapt, during and after learning, to achieve a well-defined goal (i.e., a cost function that output a scalar value). BMI experimental paradigm is particularly effective to investigate the credit assignment problem in animal brains because the performance of a BMI behavior can be defined as a cost function, and how each neuron causally contributes to this cost function is artificially and arbitrarily defined. Therefore, we can track the learning curve and study how each neuron changes its activity pattern to produce a better final performance.

Other questions in BMI studies concerned with cell-type specificity also require single-units. Different cell-types of single-units are routinely classified optogenetically or using

[3]On the other hand , not all questions regarding the BMI learning requires single-units. For example, if the question is to investigate the forms of large-scale fluctuations of population activity aligned with learning (Hennig et al., 2020), then multi-unit might be sufficient.

[4]Unlike the method developed in this thesis, microwire arrays were used in these studies (Gulati et al., 2017, 2014). The single-unit activities were real-time inferred channel by channel.

extracellular waveforms (Barthó et al., 2004; Buzsáki, 2004; Ebbesen et al., 2016; Jia et al., 2018; Li et al., 2019, 2015; Luo et al., 2018; Trouche et al., 2019). How are single-units of specific cell-types engaged in BMI tasks and how do they contribute to BMI learning? For example, in a conventional BMI task in which an animal controls its M1 population activities to move a cursor to a goal location on a 2D screen (Golub et al., 2018; Hennig et al., 2020; Sadtler et al., 2014), how much of the population activity fluctuations are contributed by pyramidal neurons, how much are contributed by GABAergic neurons and how much by modulatory neurons? What if we ask animals to only control the pyramidal neurons or the GABAergic neurons, would they be able to learn it? The spikes produced by pyramidal neurons and GABAergic neurons trigger opposite effects on downstream neurons (EPSPs versus IPSPs). In addition, subtypes of GABAergic neurons exhibit distinct connectivity motifs by targeting specific domains of the pyramidal neurons in neocortex and hippocampus. They also respond to neuromodulation differently, suggesting they are recruited differentially in computations that are crucial for regulating information processing in neural circuits. Despite the possible different nature of spikes produced by different types of neuron, they have generally been transformed the same way by dimensionality reduction to the same intrinsic neural manifold without distinction (Golub et al., 2018; Hennig et al., 2020; Sadtler et al., 2014). The authors reasoned in their landmark BMI paper: the "intrinsic manifold presumably reflects constraints imposed by the underlying neural circuitry" (Sadtler et al., 2014). Applying a decoder to allow the animal control of different types of neuron with different rules can reveal further "constraints imposed by the underlying neural circuitry".

Last but not least, another type of experiment in which single-units inference can be uniquely useful is a hippocampal BMI experiment to investigate cognitive control over both rate and temporal codes. Many well established rate (e.g., place cells) and temporal (e.g., phase precession) (see section 1.2.1 and 1.2.2) codes that relate to the cognitive map have been found in hippocampus, providing a rich repertoire of highly abstract representations for testing cognitive control over these codes. To illustrate, here I design a novel hippocampal BMI task in which an animal is required to focus on controlling place cell activities without moving its body to a particular place. This task provides a window to investigate the possible constraints underlying neural circuits for spatial cognition. Specifically, our NSP is connected with a VR system (Aronov and Tank, 2014) in which an animal is surrounded by a 360° screen and stands on the top of a spherical treadmill with a body harness system. The animal can use the harness system to rotate its body and use the spherical treadmill to move forward in a 2D virtual environment. In the first phase (natural navigation), an animal navigates in a 2D maze by turning and moving as in natural navigation, from which many place cells can be recorded (Aronov and Tank, 2014). In the second phase (a novel mental navigation

task), the VR stops responding to the body movement, and a decoder is designed to output the real-time estimation of the 'mental' location using the place fields obtained in the first phase. The 'mental spatial trajectory' connected by discretely decoded positions can be further smoothed using a moving average filter. In a single trial, the animal can only control the decoder to navigate, following the smoothed mental trajectory, to a visible or hidden goal location. The reward is delivered once the goal is reached[5]. With many trials across days, we should be able to determine if the animal can use this decoder[6] to perform a goal-directed 2D BMI mental navigation task in a virtual environment. Can they learn to perform the task? Can they immediately perform the task without learning? Do they run even though the running does not contribute to the navigation in the second phase? How do different types of neurons contribute to the computation required by the task? What is the timing and phase relationship among neurons during mental navigation? Do animals control the timing of the spikes in this task? Indeed, some of these questions might be answered using clusterless decoding. However, a single-unit BMI can be used to construct a more specific decoder that employs single neuron cell-type specificity or relative timing among place cells. With our NSP to infer a population of single-unit identities with single spike resolution, many possible rules relating to the real source of each spike (single neuron) can be designed for answering specific questions in this experimental paradigm.

In conclusion, while multi-units and feature-units can be used to decode behavioral variables under the assumption of the rate coding scheme, a population of single-units are necessary for many questions that require single-unit specificity in BMI experiments. The single-unit specificity includes both spike timing specificity and cell-type specificity. The examples of related questions include, can an animal control predefined temporal codes, how credit assignment treats single neurons differently during learning and sleep, if and how can animals mentally navigate their cognitive map by manipulating activities of their place cells with single-units specificity, etc. Based on the validation results presented in the previous chapter, our NSP is readily capable of facilitating these experiments because it can (1) assign single-unit identity accurately to each spike in real-time and (2) feed real-time spike trains to a predefined decoder to accurately decode behavioral variables.

[5]It is common to use a water tube to deliver water rewards in VR. Recently, optogenetic stimulation provides another way to deliver reward in BMI tasks (Athalye et al., 2018).

[6]The key to this experiment is to try various predefined decoders because we do not know which hippocampal neural code is under the animal's control when body movement does not directly contribute to the navigation.

5.2 Activity-dependent closed-loop perturbation experiments

BMI experimental paradigms are highly effective for studying credit assignment and cognitive control because the direct causal relationship between the arbitrarily defined the neural codes and the produced behavior[7]. Our real-time NSP can be particularly useful since it broadens the spectrum of real-time detectable neural codes from the rate coding regime to the temporal coding regime, as well as providing information about single-unit specificity. In addition, our NSP can also trigger feedback at single-spike resolution with low latency. As discussed in the first chapter, this capability was driven by the need to study inter-area communication and micro-circuit processing. Both types of studies can be facilitated by low-latency and precisely timed closed-loop perturbation. With the motivation of developing our NSP for spiking activity-dependent closed-loop perturbation experiments justified in the first chapter, here I will further discuss the possible experiments, feedback signals and latencies with examples.

The synchrony between two brain areas has long been suggested to be critical for brain computation (Fries, 2009; Singer, 1993; Singer and Gray, 1995). The transient synchrony between two brain regions has been found to be associated with perception (Singer and Gray, 1995), visual attention (Fries, 2009), spatial memory retrieval (Benchenane et al., 2010), and self-correction of incorrect behavior (Yamamoto et al., 2014).While all of these studies reveal behavioral correlates, Yamamoto et al., have performed the pathway-specific neural perturbation to show that transient MEC-CA1 synchrony specifically and causally contributes to a spatial working memory task. Here, I will discuss this experiment and propose a way to apply our NSP to an activity-specific closed-loop perturbation experiment.

Using a delayed nonmatching-to-place (DNMP) T-Maze task, Yamamoto et al., demonstrated that the transient synchrony between MEC and CA1 that tends to occur at the T-junction[8] decision point can be delayed, and that such a delay tightly precedes "oops success trials, in which animals momentarily entered the wrong arm but quickly corrected their behavior by reversing their run direction" (see Yamamoto et al. (2014): Graphical Abstract and Figure 4; the occurrences of the transient synchrony are around "200 ms prior to the directional reversal"). This MEC-CA1 transient synchrony is characterized by a highly synchronized high gamma band (65–140 Hz) oscillation in both regions, which are accompanied with spiking activities that first appear in MEC and are immediately followed by spiking activities in CA1 (see Yamamoto et al. (2014): Figure 5D). Furthermore, the authors optogenetically reduced the direct inputs from MEC to CA1 while the animal was

[7]Such causal relationship between the neural activities and the final behavior is hard to obtain in non-BMI experimental paradigm.

[8]T(Y)-junction is a region in a T(Y)-Maze where an animal needs to decide to go to the left or right arm.

at the T Maze junction and showed that both MEC-CA1 transient synchrony events and behavioral performance were significantly reduced (see Yamamoto et al. (2014): Figure 6F). Both the behavioral correlation and perturbation effect are highly significant, suggesting the MEC-CA1 transient synchrony events contribute to the successful execution of spatial working memory. However, the photo inhibition (500 ms light pulses) in this experiment was conditioned on behavior (animal approaching the T-junction), while the transient synchrony only lasts for a few tens of milliseconds. As shown in the study (see Yamamoto et al. (2014): Figure 5C), the total recorded firing rate in CA1 was reduced to nearly zero for a duration of 500 ms.

Therefore, one question is, during the long inhibition (500 ms), what communication (i.e., what neural code is used) between MEC and CA1 contributes to the memory retrieval and causes the oops moment. Do all suppressed activities in CA1 contribute equally to the underlying computation? Or do only a few spikes from a few key neurons that resonate with specific MEC inputs contribute? If so, how many spikes from how many neurons are sufficient to replicate the same behavioral deficit? Are these putative critical activities bursts or single spikes? What are the representations and what are the cell-types of these putative key neurons for inter-region communication? Does a specific temporal pattern of spikes matter? As analyzed in the first chapter, to investigate these questions requires low-latency activity-dependent perturbation. The current fastest ChR-assisted photoinhibition can trigger GABAergic neurons to fire in 1.1 ± 0.2 ms (Li et al., 2019), rapidly inhibiting a local circuit. Conditioning the ChR-assisted photoinhibition of CA1 (downstream) on the hypothesized neural code from MEC (upstream), without changing other parts of the same experiment (Yamamoto et al., 2014), might help to examine the exact neural code that subserves communication between MEC and CA1 during the transient synchrony and is specifically associated with the behavior deficit.

Notably, ChR-assisted photoinhibition is rapid but not pathway-specific. So applying ChR-assisted photoinhibition can reduce the ability of CA1 to integrate inputs from other regions as well (secondary effect). To minimize this secondary effect, it is necessary to reduce the feedback latency as much as possible such that a precisely timed and ultrashort perturbation window can be employed (Otchy et al., 2015). Different hypotheses might require different timing of perturbation depending on the axonal delay from the upstream to the downstream region, and the integration time for target neurons of a specific cell-type[9].

[9]The transmission latency from neurons in an upstream region to neurons in a downstream region can be precisely measured using an optogenetic-assisted collision test with cell-type specificity (Li et al., 2015; Luo et al., 2018), in which antidromically activated action potentials triggered by axonal photostimulation collide with spontaneous somatic action potentials.

Our NSP might be useful for experimentalists to search for the optimal timing of perturbation in various conditions, as it provides deterministic timing with only 1 ms latency for all spikes.

5.3 Limitations and future development

There are two types of limitations to this thesis. The first are the intrinsic limitations that cannot be overcome by future work. The second are the limitations due to my current version of implementation that can be overcome by future development. Here I will first discuss the intrinsic limitations of this study.

5.3.1 Intrinsic limitations

First, my work is rooted in multi-electrode recording technology that uses the extracellular voltage waveforms collected near the neuron, especially extracellular spike waveforms. This prevents us from acquiring the physiological processes that occur in the dendrites. However, the effects caused by spikes produced by the upstream neurons on the downstream neurons first activate the dendrites. Dendritic processing is both regenerative and receptor-dependent. For example, "synchronous activation of 10–50 neighbouring glutamatergic synapses triggers a local dendritic regenerative potential, NMDA spike/plateau, which is characterized by significant local amplitude (40–50 mV) and an extraordinary duration (up to several hundred milliseconds)." (Antic et al., 2010). The dendritic spikes are potentially an important part of the neural code, and unfortunately cannot generally be captured by current multi-electrode techniques (but see Moore et al., 2017).

Second, my work employs spike sorting for isolating single-units in the model construction. Spike sorting has limitations that have not yet been overcome. The major limitation of spike sorting is the lack of large-scale ground-truth data for building quality metrics to identify the best pipelines and algorithms. There are numerous offline spike sorting packages adopting a variety of strategies to isolate single-units in large-scale extracellular recording. However, without a large ground-truth dataset that contains recordings from many brain regions, it is hard to determine which ones are truly good and why. Spiketag software package seeks to minimize the impact of this limitation by maximizing model flexibility, in which experimentalists can generate arbitrary single-unit models that they consider as optimal for their specific experiment. However, this step can take considerable human interaction (i.e., visual inspection, splitting and merging, etc), and the result is not guaranteed to be absolutely accurate.

Third, my work optimizes the processing speed for real-time spike inference using an FPGA. To modify the components in the FPGA NSP or update the algorithm can be time-consuming and requires special expertise. A neuroscientist is unlikely to modify and test the FPGA circuits for their specific experimental needs. However, customization is ubiquitous in the experimental world. Due to the difficulty in programming an FPGA, the possible scope for adapting my work by user customization is mainly limited to Spiketag.

Our NSP was developed based on multi-electrode recording, spike-sorting procedure and the FPGA. Hence I consider these limitations intrinsic because they are inherited from multi-electrode recording, spike-sorting and FPGA programming respectively. These limitations might be overcome in the future by scientists or engineers who specifically work on building mechanically flexible and electronically ultra-sensitive multi-electrodes, fully automatic and accurate spike-sorting pipelines and easy-access FPGA programming and testing. However, none of these is under the scope of my work.

5.3.2 Solvable limitations and future work

The solvable limitations are due to my current version of the implementation. Our future work will be mainly solving these limitations.

5.3.2.1 Limitations in the current development

The most obvious limitation of our current version of implementation is that it only supports 160 channels of recording with INTAN chips as the amplifier, and it only supports 4-electrode groups. Therefore, the recording devices can be up to 40 tetrodes or 160 channels on silicon probes, and these recording devices must connect to five INTAN RHD2132 chips. Since the acquisition circuits were fixed in my current implementation in the FPGA, there is little flexibility on the data acquisition.

However, this 160 channel FPGA implementation can be wrapped as an individual processing unit, and be replicated several copies of them to support more channels. My current NSP was implemented in a XC7K325T-2FFG900C FPGA chip (28 nm lithography process), which was introduced in 2010. The recently developed advanced ultrascale+ FPGA chips already contain more than ten times the computational resources and can support a much larger implementation of a scaled-up NSP.

Another strategy to scale up the number of channels is to integrate the NSP with Neuropixels probes which has an entirely different data acquisition scheme. This requires us to completely rewrite the FPGA acquisition circuits and will take time to both develop and test. However, there are multiple benefits to integration with Neuropixels probes. First, implanting

many INTAN chips on the skull of the animal would no longer be needed, minimizing total weight. Second, the NSP will be able to simultaneously access multiple brain regions with a much larger number of neurons.

There are other limitations in our algorithmic modules that could be overcome by updating the algorithm. For example, currently, only the extracellular spike waveforms with negative peak can be detected. However, positive spike waveforms can occasionally occur. In addition, the LFP signal is currently not processed inside our NSP. Our current NSP can be used to compute the relative timing among single-units. However, many forms of temporal code require the phase information for each spike, requiring the LFP to be filtered into multiple frequency bands (e.g., theta, gamma.) in real-time. With this information the phase of each spike with respect to specific LFP oscillations can be inserted into each NSP spike-id packet.

5.3.2.2 Future work for technology deployment

Finally, my work in this thesis has not demonstrated a successful application on behaving animals in the lab. Instead, as described in chapter 4, during the development and validation, a multi-channel NSG was used as an artificial brain that generates pre-recorded raw analog signals to verify the accuracy and latency of the real-time NSP for single-unit inference. The result presented in chapter 4 validates that our system can infer a population of spiking activities at low-latency with single-unit and single-spike resolution, and the resulting real-time spike trains can be used to decode behavioral variables as accurately as an offline decoder. The gap from technology development to a deployable platform for a real experiment can be considerable. For example, it took seven years from the early development of the clusterless decoding (Kloosterman et al., 2013) to the first experiment that used clusterless decoding for hippocampal sleep replay content based perturbation (Gridchyn et al., 2020). Despite adopting state-of-the-art technology, the resolution of real-time decoded content in that study was still poor (it only distinguished two different environments rather than specific trajectories from the hippocampal replay events). It might take a longer time for future studies to apply perturbation conditioned on the single-trial real-time decoded trajectories within awake hippocampal replays.

Much of the difficulty in deploying the technology into the lab resides in the software. The software for experimentalists to operate has to be flexible and interactable enough to cope with many unexpected scenarios in actual online experiments where behaving animals are involved. Our Spiketag software was developed to be highly interactive with real-time 3D data visualization at all intermediate steps. It will speed up future development needed for the specific experiments suggested in the first and last chapter. For example, the predefined temporal code detector is required for the experiments to test the causal role of the relative

timing pattern of a population of single-units in inter-region communication. Such a decoder can be developed in Spiketag, and the feedback can be delivered through the low-latency PCIe interface using our APIs. In addition, integrating our system with various feedback tools, such as optogenetics, with careful calibration, requires more future work for low-latency neural code dependent perturbation. Except for perturbation, our tool also opens the possibility for BMI experiments using a population of neurons in the hippocampus (i.e., the hippocampal prosthesis) with single-unit single-spike resolution. Tetrode arrays and silicon probes are widely used in hippocampal recording but have not been widely adopted in BMI experiments (Fetz, 2007; Lebedev and Nicolelis, 2017; Nicolelis and Lebedev, 2009). This thesis fills that technological gap. With some future work to connect our real-time NSP-based decoder to VR or other external devices, many questions listed in previous sections will be open to exploration. Finally, although our system was designed for single-unit inference, it can also be used in those experiments where single-units are not required. The users can easily generate a unit model, using Spiketag, that contains multi-units or feature-units, as shown in Fig. 5.1. Decoding without single-units can be convenient for some experiments, but its accuracy with our tool requires some future validation. In conclusion, some immediate future work is required to deploy our system into specific spike-based real-time closed-loop experiments. When finished, our system can be widely applied to advance our understanding on several aspects of neural coding.

References

Abeles, M. (1982). *Local Cortical Circuits: An Electrophysiological Study*. Springer, Berlin.

Abeles, M. (1991). *Corticonics: Neural Circuits of the Cerebral Cortex*. Cambridge University Press, New York.

Ackerman, M. (2012). *Towards Theoretical Foundations of Clustering*. PhD thesis, University of Waterloo.

Adrian, E. D. (1926). The impulses produced by sensory nerve endings: Part I. *The Journal of Physiology*, 61(1):49–72.

Adrian, E. D. (1928). *The Basis of Sensation*. W W Norton & Co, New York, NY, US.

Adrian, E. D. and Zotterman, Y. (1926a). The impulses produced by sensory nerve-endings: Part II. *The Journal of Physiology*, 61(2):151–171.

Adrian, E. D. and Zotterman, Y. (1926b). The impulses produced by sensory nerve endings: Part III. *The Journal of Physiology*, 61(4):465–483.

Ahmad, S. U. and Antoniou, A. (2007). A multiobjective genetic algorithm for asymmetric fir filters. In *2007 IEEE International Symposium on Signal Processing and Information Technology*, pages 525–530.

Ambrose, R. E., Pfeiffer, B. E., and Foster, D. J. (2016). Reverse replay of hippocampal place cells is uniquely modulated by changing reward. *Neuron*, 91(5):1124–1136.

Angelini, M., Santucci, G., Schumann, H., and Schulz, H.-J. (2018). A review and characterization of progressive visual analytics. *Informatics*, 5(3):31.

Antic, S. D., Zhou, W.-L., Moore, A. R., Short, S. M., and Ikonomu, K. D. (2010). The decade of the dendritic NMDA spike. *Journal of Neuroscience Research*, 88(14):2991–3001.

Aronov, D. and Tank, D. W. (2014). Engagement of neural circuits underlying 2D spatial navigation in a rodent virtual reality system. *Neuron*, 84(2):442–456.

Ashe, J. and Georgopoulos, A. P. (1994). Movement parameters and neural activity in motor cortex and area 5. *Cerebral Cortex*, 4(6):590–600.

Athalye, V. R., Santos, F. J., Carmena, J. M., and Costa, R. M. (2018). Evidence for a neural law of effect. *Science*, 359(6379):1024–1029.

Attneave, F. (1954). Some informational aspects of visual perception. *Psychological Review*, 61(3):183–193.

Baldwin, H. A., Frenk, S., and Lettvin, J. Y. (1965). Glass-coated tungsten microelectrodes. *Science*, 148(3676):1462–1464.

Barlow, H. B. (1953). Summation and inhibition in the frog's retina. *The Journal of Physiology*, 119(1):69–88.

Barlow, H. B. (1961). Possible principles underlying the transformations of sensory messages. In Rosenblith, W. A., editor, *Sensory Communication*, pages 216–234. The MIT Press.

Barna, J. S., Arezzo, J. C., and Vaughan, H. G. (1981). A new multielectrode array for the simultaneous recording of field potentials and unit activity. *Electroencephalography and Clinical Neurophysiology*, 52(5):494–496.

Barthó, P., Hirase, H., Monconduit, L., Zugaro, M., Harris, K. D., and Buzsáki, G. (2004). Characterization of neocortical principal cells and interneurons by network interactions and extracellular features. *Journal of Neurophysiology*, 92(1):600–608.

Basu, J., Srinivas, K. V., Cheung, S. K., Taniguchi, H., Huang, Z. J., and Siegelbaum, S. A. (2013). A cortico-hippocampal learning rule shapes inhibitory microcircuit activity to enhance hippocampal information flow. *Neuron*, 79(6):1208–1221.

Benchenane, K., Peyrache, A., Khamassi, M., Tierney, P. L., Gioanni, Y., Battaglia, F. P., and Wiener, S. I. (2010). Coherent theta oscillations and reorganization of spike timing in the hippocampal- prefrontal network upon learning. *Neuron*, 66(6):921–936.

Berchin, G. (2007). Precise filter design. *IEEE Signal Processing Magazine*, 24(1):137–139.

Bernstein, J. (1868). Ueber den zeitlichen verlauf der negativen schwankung des nervenstroms. *Archiv für die gesamte Physiologie des Menschen und der Tiere*, 1(1):173–207.

Bi, G.-q. and Poo, M.-m. (1998). Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type. *Journal of Neuroscience*, 18(24):10464–10472.

Bliss, T. V. P. and Lømo, T. (1973). Long-lasting potentiation of synaptic transmission in the dentate area of the anaesthetized rabbit following stimulation of the perforant path. *The Journal of Physiology*, 232(2):331–356.

Bollt, E. M., Sun, J., and Runge, J. (2018). Introduction to focus issue: Causation inference and information flow in dynamical systems: Theory and applications. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 28(7):075201.

Boyden, E. S., Zhang, F., Bamberg, E., Nagel, G., and Deisseroth, K. (2005). Millisecond-timescale, genetically targeted optical control of neural activity. *Nature Neuroscience*, 8(9):1263–1268.

Branco, T., Clark, B. A., and Häusser, M. (2010). Dendritic discrimination of temporal input sequences in cortical neurons. *Science*, 329(5999):1671–1675.

Brette, R. (2015). Philosophy of the spike: Rate-based vs. spike-based theories of the brain. *Frontiers in Systems Neuroscience*, 9.

Buzsáki, G. (1989). Two-stage model of memory trace formation: A role for "noisy" brain states. *Neuroscience*, 31(3):551–570.

Buzsáki, G. (2004). Large-scale recording of neuronal ensembles. *Nature Neuroscience*, 7(5):446–451.

Buzsáki, G. (2015). Hippocampal sharp wave-ripple: A cognitive biomarker for episodic memory and planning. *Hippocampus*, 25(10):1073–1188.

Buzsáki, G., Anastassiou, C. A., and Koch, C. (2012). The origin of extracellular fields and currents — EEG, ECoG, LFP and spikes. *Nature Reviews Neuroscience*, 13(6):407–420.

Buzsáki, G., Lai-Wo S., L., and Vanderwolf, C. H. (1983). Cellular bases of hippocampal EEG in the behaving rat. *Brain Research Reviews*, 6(2):139–171.

Buzsáki, G., Penttonen, M., Nádasdy, Z., and Bragin, A. (1996). Pattern and inhibition-dependent invasion of pyramidal cell dendrites by fast spikes in the hippocampus in vivo. *Proceedings of the National Academy of Sciences of the United States of America*, 93(18):9921–9925.

Buzsáki, G., Stark, E., Berényi, A., Khodagholy, D., Kipke, D. R., Yoon, E., and Wise, K. D. (2015). Tools for probing local circuits: High-density silicon probes combined with optogenetics. *Neuron*, 86(1):92–105.

Buzsáki, G. and Wang, X.-J. (2012). Mechanisms of gamma oscillations. *Annual Review of Neuroscience*, 35(1):203–225.

Campello, R. J. G. B., Moulavi, D., Zimek, A., and Sander, J. (2015). Hierarchical density estimates for data clustering, visualization, and outlier detection. *ACM Transactions on Knowledge Discovery from Data*, 10(1):5:1–5:51.

Cardin, J. A., Carlén, M., Meletis, K., Knoblich, U., Zhang, F., Deisseroth, K., Tsai, L.-H., and Moore, C. I. (2009). Driving fast-spiking cells induces gamma rhythm and controls sensory responses. *Nature*, 459(7247):663–667.

Carmena, J. M., Lebedev, M. A., Crist, R. E., O'Doherty, J. E., Santucci, D. M., Dimitrov, D. F., Patil, P. G., Henriquez, C. S., and Nicolelis, M. A. L. (2003). Learning to control a brain-machine interface for reaching and grasping by primates. *PLOS Biology*, 1(2):e42.

Carr, C. E. (1993). Processing of temporal information in the brain. *Annual Review of Neuroscience*, 16(1):223–243.

Cei, A., Girardeau, G., Drieu, C., Kanbi, K. E., and Zugaro, M. (2014). Reversed theta sequences of hippocampal cell assemblies during backward travel. *Nature Neuroscience*, 17(5):719–724.

Chapin, J. K., Moxon, K. A., Markowitz, R. S., and Nicolelis, M. A. L. (1999). Real-time control of a robot arm using simultaneously recorded neurons in the motor cortex. *Nature Neuroscience*, 2(7):664–670.

Cho, J.-H., Bayazitov, I. T., Meloni, E. G., Myers, K. M., Carlezon, W. A., Zakharenko, S. S., and Bolshakov, V. Y. (2012). Coactivation of thalamic and cortical pathways induces input timing–dependent plasticity in amygdala. *Nature Neuroscience*, 15(1):113–122.

Christie, B. P., Tat, D. M., Irwin, Z. T., Gilja, V., Nuyujukian, P., Foster, J. D., Ryu, S. I., Shenoy, K. V., Thompson, D. E., and Chestek, C. A. (2014). Comparison of spike sorting and thresholding of voltage waveforms for intracortical brain–machine interface performance. *Journal of Neural Engineering*, 12(1):016009.

Chung, J. E., Magland, J. F., Barnett, A. H., Tolosa, V. M., Tooker, A. C., Lee, K. Y., Shah, K. G., Felix, S. H., Frank, L. M., and Greengard, L. F. (2017). A fully automated approach to spike sorting. *Neuron*, 95(6):1381–1394.e6.

Ciliberti, D. and Kloosterman, F. (2017). Falcon: a highly flexible open-source software for closed-loop neuroscience. *Journal of Neural Engineering*, 14(4):045004.

Cobb, S. R., Buhl, E. H., Halasy, K., Paulsen, O., and Somogyi, P. (1995). Synchronization of neuronal activity in hippocampus by individual GABAergic interneurons. *Nature*, 378(6552):75–78.

Colgin, L. L. (2016). Rhythms of the hippocampal network. *Nature Reviews Neuroscience*, 17(4):239–249.

Collura, T. F. (1993). History and evolution of electroencephalographic instruments and techniques. *Journal of Clinical Neurophysiology*, 10(4):476.

Constantinidis, C., Williams, G. V., and Goldman-Rakic, P. S. (2002). A role for inhibition in shaping the temporal flow of information in prefrontal cortex. *Nature Neuroscience*, 5(2):175–180.

Cunningham, J. P. and Yu, B. M. (2014). Dimensionality reduction for large-scale neural recordings. *Nature Neuroscience*, 17(11):1500–1509.

Dan, Y. and Poo, M.-m. (2004). Spike timing-dependent plasticity of neural circuits. *Neuron*, 44(1):23–30.

Davidson, T. J., Kloosterman, F., and Wilson, M. A. (2009). Hippocampal replay of extended experience. *Neuron*, 63(4):497–507.

Davies, C. H., Starkey, S. J., Pozza, M. F., and Collingridge, G. L. (1991). $GABA_B$ autoreceptors regulate the induction of LTP. *Nature*, 349(6310):609–611.

de Lavilléon, G., Lacroix, M. M., Rondi-Reig, L., and Benchenane, K. (2015). Explicit memory creation during sleep demonstrates a causal role of place cells in navigation. *Nature Neuroscience*, 18(4):493–495.

deCharms, R. C. and Zador, A. (2000). Neural representation and the cortical code. *Annual Review of Neuroscience*, 23(1):613–647.

Deisseroth, K. (2014). Circuit dynamics of adaptive and maladaptive behaviour. *Nature*, 505(7483):309–317.

Deng, X., Liu, D. F., Karlsson, M. P., Frank, L. M., and Eden, U. T. (2016). Rapid classification of hippocampal replay content for real-time applications. *Journal of Neurophysiology*, 116(5):2221–2235.

Deng, X., Liu, D. F., Kay, K., Frank, L. M., and Eden, U. T. (2015). Clusterless decoding of position from multiunit activity using a marked point process filter. *Neural Computation*, 27(7):1438–1460.

Diba, K. and Buzsáki, G. (2007). Forward and reverse hippocampal place-cell sequences during ripples. *Nature Neuroscience*, 10(10):1241–1242.

Diesmann, M., Gewaltig, M.-O., and Aertsen, A. (1999). Stable propagation of synchronous spiking in cortical neural networks. *Nature*, 402(6761):529–533.

Diggelmann, R., Fiscella, M., Hierlemann, A., and Franke, F. (2018). Automatic spike sorting for high-density microelectrode arrays. *Journal of Neurophysiology*, 120(6):3155–3171.

Donoghue, J. P. (2008). Bridging the brain to the world: A perspective on neural interface systems. *Neuron*, 60(3):511–521.

Dragas, J., Jäckel, D., Franke, F., and Hierlemann, A. (2014). High-throughput hardware for real-time spike overlap decomposition in multi-electrode neuronal recording systems. In *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 658–661.

Dragas, J., Jäckel, D., Hierlemann, A., and Franke, F. (2015). Complexity optimization and high-throughput low-latency hardware implementation of a multi-electrode spike-sorting algorithm. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 23(2):149–158.

Dudman, J. T., Tsay, D., and Siegelbaum, S. A. (2007). A role for synaptic inputs at distal dendrites: Instructive signals for hippocampal long-term plasticity. *Neuron*, 56(5):866–879.

Ebbesen, C. L., Reifenstein, E. T., Tang, Q., Burgalossi, A., Ray, S., Schreiber, S., Kempter, R., and Brecht, M. (2016). Cell type-specific differences in spike timing and spike shape in the rat parasubiculum and superficial medial entorhinal cortex. *Cell Reports*, 16(4):1005–1015.

Ego-Stengel, V. and Wilson, M. A. (2010). Disruption of ripple-associated hippocampal activity during rest impairs spatial learning in the rat. *Hippocampus*, 20(1):1–10.

Einevoll, G. T., Franke, F., Hagen, E., Pouzat, C., and Harris, K. D. (2012). Towards reliable spike-train recordings from thousands of neurons with multielectrodes. *Current Opinion in Neurobiology*, 22(1):11–17.

Einevoll, G. T., Kayser, C., Logothetis, N. K., and Panzeri, S. (2013). Modelling and analysis of local field potentials for studying the function of cortical circuits. *Nature Reviews Neuroscience*, 14(11):770–785.

Ekanadham, C., Tranchina, D., and Simoncelli, E. P. (2014). A unified framework and method for automatic neural spike identification. *Journal of Neuroscience Methods*, 222:47–55.

English, D. F., McKenzie, S., Evans, T., Kim, K., Yoon, E., and Buzsáki, G. (2017). Pyramidal cell-interneuron circuit architecture and dynamics in hippocampal networks. *Neuron*, 96(2):505–520.e7.

Evarts, E. V. (1966). Pyramidal tract activity associated with a conditioned hand movement in the monkey. *Journal of Neurophysiology*, 29(6):1011–1027.

Evarts, E. V. (1968a). Relation of pyramidal tract activity to force exerted during voluntary movement. *Journal of Neurophysiology*, 31(1):14–27.

Evarts, E. V. (1968b). A technique for recording activity of subcortical neurons in moving animals. *Electroencephalography and Clinical Neurophysiology*, 24(1):83–86.

Faraday, M. (1832). Experimental researches in electricity. *Philosophical Transactions of the Royal Society of London*, 122:125–162.

Fee, M. S., Mitra, P. P., and Kleinfeld, D. (1996). Automatic sorting of multiple unit neuronal signals in the presence of anisotropic and non-Gaussian variability. *Journal of Neuroscience Methods*, 69(2):175–188.

Fekete, J.-D., Chen, Q., Feng, Y., and Renault, J. (2019). Practical use cases for progressive visual analytics. In *DSIA 2019 - 4th Workshop on Data Systems for Interactive Analysis*, Vancouver, Canada.

Feng, T., Silva, D., and Foster, D. J. (2015). Dissociation between the experience-dependent development of hippocampal theta sequences and single-trial phase precession. *Journal of Neuroscience*, 35(12):4890–4902.

Fernández-Ruiz, A., Oliva, A., Oliveira, E. F. d., Rocha-Almeida, F., Tingley, D., and Buzsáki, G. (2019). Long-duration hippocampal sharp wave ripples improve memory. *Science*, 364(6445):1082–1086.

Ferster, D. and Spruston, N. (1995). Cracking the neuronal code. *Science*, 270(5237):756–757.

Fetz, E. E. (1969). Operant conditioning of cortical unit activity. *Science*, 163(3870):955–958.

Fetz, E. E. (1997). Temporal coding in neural populations? *Science*, 278(5345):1901–1902.

Fetz, E. E. (2007). Volitional control of neural activity: implications for brain–computer interfaces. *The Journal of Physiology*, 579(3):571–579.

Foster, D. J. (2017). Replay comes of age. *Annual Review of Neuroscience*, 40(1):581–602.

Foster, D. J. and Wilson, M. A. (2006). Reverse replay of behavioural sequences in hippocampal place cells during the awake state. *Nature*, 440(7084):680–683.

Foster, D. J. and Wilson, M. A. (2007). Hippocampal theta sequences. *Hippocampus*, 17(11):1093–1099.

Fraser, G. W., Chase, S. M., Whitford, A., and Schwartz, A. B. (2009). Control of a brain-computer interface without spike sorting. *Journal of Neural Engineering*, 6(5):055004.

Frey, M., Tanni, S., Perrodin, C., O'Leary, A., Nau, M., Kelly, J., Banino, A., Doeller, C. F., and Barry, C. (2019). Deepinsight: a general framework for interpreting wide-band neural activity. preprint, Neuroscience.

Fries, P. (2009). Neuronal gamma-band synchronization as a fundamental process in cortical computation. *Annual Review of Neuroscience*, 32(1):209–224.

Froemke, R. C. and Dan, Y. (2002). Spike-timing-dependent synaptic modification induced by natural spike trains. *Nature*, 416(6879):433–438.

Fujisawa, S., Amarasingham, A., Harrison, M. T., and Buzsáki, G. (2008). Behavior-dependent short-term assembly dynamics in the medial prefrontal cortex. *Nature Neuroscience*, 11(7):823–833.

Galvani, L. (1791). *De viribus electricitatis in motu musculari commentarius [Italian]*. bologna accademia delle scienze.

Ganguly, K., Dimitrov, D. F., Wallis, J. D., and Carmena, J. M. (2011). Reversible large-scale modification of cortical networks during neuroprosthetic control. *Nature Neuroscience*, 14(5):662–667.

Georgopoulos, A. P. (1994). Population activity in the control of movement. *International Review of Neurobiology*, 37:103–119; discussion 121–123.

Georgopoulos, A. P., Kalaska, J. F., Caminiti, R., and Massey, J. T. (1982). On the relations between the direction of two-dimensional arm movements and cell discharge in primate motor cortex. *Journal of Neuroscience*, 2(11):1527–1537.

Georgopoulos, A. P., Merchant, H., Naselaris, T., and Amirikian, B. (2007). Mapping of the preferred direction in the motor cortex. *Proceedings of the National Academy of Sciences*, 104(26):11068–11072.

Georgopoulos, A. P., Schwartz, A. B., and Kettner, R. E. (1986). Neuronal population coding of movement direction. *Science*, 233(4771):1416–1419.

Gerstein, G. L. and Clark, W. A. (1964). Simultaneous Studies of Firing Patterns in Several Neurons. *Science*, 143(3612):1325–1327.

Gerstner, W., Kempter, R., van Hemmen, J. L., and Wagner, H. (1996). A neuronal learning rule for sub-millisecond temporal coding. *Nature*, 383(6595):76–78.

Gerstner, W., Kistler, W. M., Naud, R., and Paninski, L. (2014). *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge University Press.

Gilja, V., Pandarinath, C., Blabe, C. H., Nuyujukian, P., Simeral, J. D., Sarma, A. A., Sorice, B. L., Perge, J. A., Jarosiewicz, B., Hochberg, L. R., Shenoy, K. V., and Henderson, J. M. (2015). Clinical translation of a high-performance neural prosthesis. *Nature Medicine*, 21(10):1142–1145.

Girardeau, G., Benchenane, K., Wiener, S. I., Buzsáki, G., and Zugaro, M. B. (2009). Selective suppression of hippocampal ripples impairs spatial memory. *Nature Neuroscience*, 12(10):1222–1223.

Glaser, J. I., Benjamin, A. S., Chowdhury, R. H., Perich, M. G., Miller, L. E., and Kording, K. P. (2017). Machine learning for neural decoding. *arXiv:1708.00909*.

Golub, M. D., Sadtler, P. T., Oby, E. R., Quick, K. M., Ryu, S. I., Tyler-Kabara, E. C., Batista, A. P., Chase, S. M., and Yu, B. M. (2018). Learning by neural reassociation. *Nature Neuroscience*, 21(4):607–616.

Gray, C. M., Maldonado, P. E., Wilson, M., and McNaughton, B. (1995). Tetrodes markedly improve the reliability and yield of multiple single-unit isolation from multi-unit recordings in cat striate cortex. *Journal of Neuroscience Methods*, 63(1-2):43–54.

Green, A. M. and Kalaska, J. F. (2011). Learning to move machines with the mind. *Trends in Neurosciences*, 34(2):61–75.

Green, J. D. (1958). A simple microelectrode for recording from the central nervous system. *Nature*, 182(4640):962–962.

Gridchyn, I., Schoenenberger, P., O'Neill, J., and Csicsvari, J. (2020). Assembly-specific disruption of hippocampal replay leads to selective memory deficit. *Neuron*, 106(2):291–300.e6.

Gulati, T., Guo, L., Ramanathan, D. S., Bodepudi, A., and Ganguly, K. (2017). Neural reactivations during sleep determine network credit assignment. *Nature Neuroscience*, 20(9):1277–1284.

Gulati, T., Ramanathan, D. S., Wong, C. C., and Ganguly, K. (2014). Reactivation of emergent task-related ensembles during slow-wave sleep after neuroprosthetic learning. *Nature Neuroscience*, 17(8):1107–1113.

Gupta, A. S., van der Meer, M. A. A., Touretzky, D. S., and Redish, A. D. (2010). Hippocampal replay is not a simple function of experience. *Neuron*, 65(5):695–705.

Hafting, T., Fyhn, M., Molden, S., Moser, M.-B., and Moser, E. I. (2005). Microstructure of a spatial map in the entorhinal cortex. *Nature*, 436(7052):801–806.

Harris, K. D. (2014). Sleep replay meets brain–machine interface. *Nature Neuroscience*, 17(8):1019–1021.

Harris, K. D., Henze, D. A., Csicsvari, J., Hirase, H., and Buzsáki, G. (2000). Accuracy of tetrode spike separation as determined by simultaneous intracellular and extracellular measurements. *Journal of Neurophysiology*, 84(1):401–414.

Hartline, H. K. (1938). Th response of single optic nerve fibers of the vertebrate eye to illumination of the retina. *American Journal of Physiology-Legacy Content*, 121(2):400–415.

Hartline, H. K. (1940). The receptive fields of optic nerve fibers. *American Journal of Physiology-Legacy Content*, 130(4):690–699.

Hasselmo, M. E., Bodelón, C., and Wyble, B. P. (2002). A proposed function for hippocampal theta rhythm: Separate phases of encoding and retrieval enhance reversal of prior learning. *Neural Computation*, 14(4):793–817.

Hatsopoulos, N. G. (2010). Columnar organization in the motor cortex. *Cortex*, 46(2):270–271.

Hebb, D. O. (1949). *The Organization of Behavior: A Neuropsychological Theory*. Wiley, New York.

Hennig, J. A., Oby, E. R., Golub, M. D., Bahureksa, L. A., Sadtler, P. T., Quick, K. M., Ryu, S. I., Tyler-Kabara, E. C., Batista, A. P., Chase, S. M., and Yu, B. M. (2020). Learning is shaped by abrupt changes in neural engagement. *bioRxiv doi: https://doi.org/10.1101/2020.05.24.112714*.

Henze, D. A., Borhegyi, Z., Csicsvari, J., Mamiya, A., Harris, K. D., and Buzsáki, G. (2000). Intracellular features predicted by extracellular recordings in the hippocampus in vivo. *Journal of Neurophysiology*, 84(1):390–400.

Hirabayashi, T. and Miyashita, Y. (2005). Dynamically modulated spike correlation in monkey inferior temporal cortex depending on the feature configuration within a whole object. *Journal of Neuroscience*, 25(44):10299–10307.

Hochberg, L. R., Bacher, D., Jarosiewicz, B., Masse, N. Y., Simeral, J. D., Vogel, J., Haddadin, S., Liu, J., Cash, S. S., van der Smagt, P., and Donoghue, J. P. (2012). Reach and grasp by people with tetraplegia using a neurally controlled robotic arm. *Nature*, 485(7398):372–375.

Hochberg, L. R., Serruya, M. D., Friehs, G. M., Mukand, J. A., Saleh, M., Caplan, A. H., Branner, A., Chen, D., Penn, R. D., and Donoghue, J. P. (2006). Neuronal ensemble control of prosthetic devices by a human with tetraplegia. *Nature*, 442(7099):164–171.

Hodgkin, A. L. and Huxley, A. F. (1939). Action Potentials Recorded from Inside a Nerve Fibre. *Nature*, 144(3651):710–711.

Hodgkin, A. L. and Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, 117(4):500–544.

Hong, G. and Lieber, C. M. (2019). Novel electrode technologies for neural recordings. *Nature Reviews Neuroscience*, 20(6):330–345.

Hopfield, J. J. (1995). Pattern recognition computation using action potential timing for stimulus representation. *Nature*, 376(6535):33–36.

Hu, S., Ciliberti, D., Grosmark, A. D., Michon, F., Ji, D., Penagos, H., Buzsáki, G., Wilson, M. A., Kloosterman, F., and Chen, Z. (2018). Real-time readout of large-scale unsorted neural ensemble place codes. *Cell Reports*, 25(10):2635–2642.e5.

Hubel, D. H. (1957). Tungsten microelectrode for recording from single units. *Science*, 125(3247):549–550.

Hubel, D. H. (1959). Single unit activity in striate cortex of unrestrained cats. *The Journal of Physiology*, 147(2):226–2382.

Hubel, D. H. and Wiesel, T. N. (1959). Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology*, 148(3):574–591.

Hubel, D. H. and Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 160(1):106–1542.

Huber, D., Petreanu, L., Ghitani, N., Ranade, S., Hromádka, T., Mainen, Z., and Svoboda, K. (2008). Sparse optical microstimulation in barrel cortex drives learned behaviour in freely moving mice. *Nature*, 451(7174):61–64.

Humphrey, D. R., Schmidt, E. M., and Thompson, W. D. (1970). Predicting measures of motor performance from multiple cortical spike trains. *Science*, 170(3959):758–762.

Hung, C. P., Kreiman, G., Poggio, T., and DiCarlo, J. J. (2005). Fast readout of object identity from macaque inferior temporal cortex. *Science*, 310(5749):863–866.

Hunt, D. L., Lai, C., Smith, R. D., Lee, A. K., Harris, T. D., and Barbic, M. (2019). Multimodal in vivo brain electrophysiology with integrated glass microelectrodes. *Nature Biomedical Engineering*, 3(9):741–753.

Huxter, J., Burgess, N., and O'Keefe, J. (2003). Independent rate and temporal coding in hippocampal pyramidal cells. *Nature*, 425(6960):828–832.

Izhikevich, E. M. (2006). Polychronization: Computation with spikes. *Neural Computation*, 18(2):245–282.

Jacobs, A. L., Fridman, G., Douglas, R. M., Alam, N. M., Latham, P. E., Prusky, G. T., and Nirenberg, S. (2009). Ruling out and ruling in neural codes. *Proceedings of the National Academy of Sciences*, 106(14):5936–5941.

Jadhav, S. P., Rothschild, G., Roumis, D. K., and Frank, L. M. (2016). Coordinated excitation and inhibition of prefrontal ensembles during awake hippocampal sharp-wave ripple events. *Neuron*, 90(1):113–127.

Jaramillo, J. and Kempter, R. (2017). Phase precession: a neural code underlying episodic memory? *Current Opinion in Neurobiology*, 43:130–138.

Jazayeri, M. and Afraz, A. (2017). Navigating the Neural Space in Search of the Neural Code. *Neuron*, 93(5):1003–1014.

Jensen, O. and Lisman, J. E. (2000). Position reconstruction from an ensemble of hippocampal place cells: Contribution of theta phase coding. *Journal of Neurophysiology*, 83(5):2602–2609.

Jia, X., Siegle, J., Bennett, C., Gale, S., Denman, D. R., Koch, C., and Olsen, S. R. (2018). High-density extracellular probes reveal dendritic backpropagation and facilitate neuron classification. *Journal of Neurophysiology*, 121(5):1831–1847.

Jun, J. J., Mitelut, C., Lai, C., Gratiy, S. L., Anastassiou, C. A., and Harris, T. D. (2017a). Real-time spike sorting platform for high-density extracellular probes with ground-truth validation and drift correction. *bioRxiv doi: https://doi.org/10.1101/101030*.

Jun, J. J., Steinmetz, N. A., Siegle, J. H., Denman, D. J., Bauza, M., Barbarits, B., Lee, A. K., Anastassiou, C. A., Andrei, A., Aydın, Ç., Barbic, M., Blanche, T. J., Bonin, V., Couto, J., Dutta, B., Gratiy, S. L., Gutnisky, D. A., Häusser, M., Karsh, B., Ledochowitsch, P., Lopez, C. M., Mitelut, C., Musa, S., Okun, M., Pachitariu, M., Putzeys, J., Rich, P. D., Rossant, C., Sun, W.-l., Svoboda, K., Carandini, M., Harris, K. D., Koch, C., O'Keefe, J., and Harris, T. D. (2017b). Fully integrated silicon probes for high-density recording of neural activity. *Nature*, 551(7679):232–236.

Kadir, S. N., Goodman, D. F. M., and Harris, K. D. (2014). High-dimensional cluster analysis with the masked em algorithm. *Neural Computation*, 26(11):2379–2394.

Karlsson, M. P. and Frank, L. M. (2009). Awake replay of remote experiences in the hippocampus. *Nature Neuroscience*, 12(7):913–918.

Kayser, C., Montemurro, M. A., Logothetis, N. K., and Panzeri, S. (2009). Spike-phase coding boosts and stabilizes information carried by spatial and temporal spike patterns. *Neuron*, 61(4):597–608.

Keehn, D. G. (1966). An iterative spike separation technique. *IEEE Transactions on Biomedical Engineering*, BME-13(1):19–28.

Kim, S.-J., Manyam, S. C., Warren, D. J., and Normann, R. A. (2006). Electrophysiological mapping of cat primary auditory cortex with multielectrode arrays. *Annals of Biomedical Engineering*, 34(2):300–309.

Kim, T.-i., McCall, J. G., Jung, Y. H., Huang, X., Siuda, E. R., Li, Y., Song, J., Song, Y. M., Pao, H. A., Kim, R.-H., Lu, C., Lee, S. D., Song, I.-S., Shin, G., Al-Hasani, R., Kim, S., Tan, M. P., Huang, Y., Omenetto, F. G., Rogers, J. A., and Bruchas, M. R. (2013). Injectable, cellular-scale optoelectronics with applications for wireless optogenetics. *Science*, 340(6129):211–216.

Kleinberg, J. M. (2003). An impossibility theorem for clustering. In Becker, S., Thrun, S., and Obermayer, K., editors, *Advances in Neural Information Processing Systems 15*, pages 463–470. MIT Press.

Kloosterman, F., Layton, S. P., Chen, Z., and Wilson, M. A. (2013). Bayesian decoding using unsorted spikes in the rat hippocampus. *Journal of Neurophysiology*, 111(1):217–227.

Kobak, D., Brendel, W., Constantinidis, C., Feierstein, C. E., Kepecs, A., Mainen, Z. F., Qi, X.-L., Romo, R., Uchida, N., and Machens, C. K. (2016). Demixed principal component analysis of neural population data. *eLife*, 5:e10989.

König, P., Engel, A. K., and Singer, W. (1996). Integrator or coincidence detector? The role of the cortical neuron revisited. *Trends in Neurosciences*, 19(4):130–137.

Koralek, A. C., Costa, R. M., and Carmena, J. M. (2013). Temporally precise cell-specific coherence develops in corticostriatal networks during learning. *Neuron*, 79(5):865–872.

Koralek, A. C., Jin, X., Long Ii, J. D., Costa, R. M., and Carmena, J. M. (2012). Corticostriatal plasticity is necessary for learning intentional neuroprosthetic skills. *Nature*, 483(7389):331–335.

Larson, J. and Munkácsy, E. (2015). Theta-burst LTP. *Brain Research*, 1621:38–50.

Larson, J., Wong, D., and Lynch, G. (1986). Patterned stimulation at the theta frequency is optimal for the induction of hippocampal long-term potentiation. *Brain Research*, 368(2):347–350.

Lebedev, M. A. and Nicolelis, M. A. L. (2017). Brain-machine interfaces: From basic science to neuroprostheses and neurorehabilitation. *Physiological Reviews*, 97(2):767–837.

Lee, A. K. and Wilson, M. A. (2002). Memory of sequential experience in the hippocampus during slow wave sleep. *Neuron*, 36(6):1183–1194.

Lee, J. H., Carlson, D. E., Shokri Razaghi, H., Yao, W., Goetz, G. A., Hagen, E., Batty, E., Chichilnisky, E., Einevoll, G. T., and Paninski, L. (2017). YASS: Yet Another Spike Sorter. In *Advances in Neural Information Processing Systems 30*, pages 4002–4012.

Leroy, F., Brann, D. H., Meira, T., and Siegelbaum, S. A. (2017). Input-timing-dependent plasticity in the hippocampal ca2 region and its potential role in social memory. *Neuron*, 95(5):1089–1102.e5.

Lewicki, M. S. (1998). A review of methods for spike sorting: the detection and classification of neural action potentials. *Network: Computation in Neural Systems*, 9(4):R53–R78.

Li, N., Chen, S., Guo, Z. V., Chen, H., Huo, Y., Inagaki, H. K., Chen, G., Davis, C., Hansel, D., Guo, C., and Svoboda, K. (2019). Spatiotemporal constraints on optogenetic inactivation in cortical circuits. *eLife*, 8:e48622.

Li, N., Chen, T.-W., Guo, Z. V., Gerfen, C. R., and Svoboda, K. (2015). A motor cortex circuit for motor planning and movement. *Nature*, 519(7541):51–56.

Lippmann, G. (1873). Relation entre les phénomènes électriques et capillaries. *C . R.. Séances Acad. Sci., Ser. C*, 76:1407–1408.

Lisman, J. (2005). The theta/gamma discrete phase code occuring during the hippocampal phase precession may be a more general brain coding scheme. *Hippocampus*, 15(7):913–922.

Lisman, J. E. (1997). Bursts as a unit of neural information: making unreliable synapses reliable. *Trends in Neurosciences*, 20(1):38–43.

Lisman, J. E. and Jensen, O. (2013). The theta-gamma neural code. *Neuron*, 77(6):1002–1016.

Liu, X., Ramirez, S., Pang, P. T., Puryear, C. B., Govindarajan, A., Deisseroth, K., and Tonegawa, S. (2012). Optogenetic stimulation of a hippocampal engram activates fear memory recall. *Nature*, 484(7394):381–385.

London, M., Roth, A., Beeren, L., Häusser, M., and Latham, P. E. (2010). Sensitivity to perturbations in vivo implies high noise and suggests rate coding in cortex. *Nature*, 466(7302):123–127.

Luan, S., Williams, I., Maslik, M., Liu, Y., Carvalho, F. D., Jackson, A., Quiroga, R. Q., and Constandinou, T. G. (2018). Compact standalone platform for neural recording with real-time spike sorting and data logging. *Journal of Neural Engineering*, 15(4):046014.

Lucas, K. (1912). On a mechanical method of correcting photographic records obtained from the capillary electrometer. *The Journal of Physiology*, 44(3):225–242.

Luo, L., Callaway, E. M., and Svoboda, K. (2018). Genetic dissection of neural circuits: A decade of progress. *Neuron*, 98(2):256–281.

Magee, J. C. and Johnston, D. (1997). A synaptically controlled, associative signal for hebbian plasticity in hippocampal neurons. *Science*, 275(5297):209–213.

Mainen, Z. F. and Sejnowski, T. J. (1995). Reliability of spike timing in neocortical neurons. *Science (New York, N.Y.)*, 268(5216):1503–1506.

Maingret, N., Girardeau, G., Todorova, R., Goutierre, M., and Zugaro, M. (2016). Hippocampo-cortical coupling mediates memory consolidation during sleep. *Nature Neuroscience*, 19(7):959–964.

Marey, E.-J. (1876). Des variations électriques des muscles et du cœur en particulier étudiées au moyen de l'électromètre de M. Lippmann. *CR Acad Sci.*, 82:975–7.

Marinescu, I. E., Lawlor, P. N., and Kording, K. P. (2018). Quasi-experimental causality in neuroscience and behavioural research. *Nature Human Behaviour*, 2(12):891–898.

Markram, H., Lübke, J., Frotscher, M., and Sakmann, B. (1997). Regulation of synaptic efficacy by coincidence of postsynaptic aps and epsps. *Science*, 275(5297):213–215.

Marr, D. (1969). A theory of cerebellar cortex. *The Journal of Physiology*, 202(2):437–4701.

Mathis, A., Mamidanna, P., Cury, K. M., Abe, T., Murthy, V. N., Mathis, M. W., and Bethge, M. (2018). Deeplabcut: markerless pose estimation of user-defined body parts with deep learning. *Nature Neuroscience*, 21(9):1281–1289.

McClelland, J. L., McNaughton, B. L., and O'Reilly, R. C. (1995). Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionist models of learning and memory. *Psychological Review*, 102(3):419–457.

McNaughton, B. L., O'Keefe, J., and Barnes, C. A. (1983). The stereotrode: A new technique for simultaneous isolation of several single units in the central nervous system from multiple unit records. *Journal of Neuroscience Methods*, 8(4):391–397.

Mehta, M. R., Lee, A. K., and Wilson, M. A. (2002). Role of experience and oscillations in transforming a rate code into a temporal code. *Nature*, 417(6890):741–746.

Mel, B. W. (1999). Why have dendrites? A computational perspective. In Häusser, G. S. S. M., editor, *Dendrites*, chapter 16. Oxford University Press.

Moore, J. J., Ravassard, P. M., Ho, D., Acharya, L., Kees, A. L., Vuong, C., and Mehta, M. R. (2017). Dynamics of cortical dendritic membrane potential and spikes in freely behaving rats. *Science*, 355(6331).

Morris, R. G. M., Garrud, P., Rawlins, J. N. P., and O'Keefe, J. (1982). Place navigation impaired in rats with hippocampal lesions. *Nature*, 297(5868):681–683.

Moxon, K. A. and Foffani, G. (2015). Brain-machine interfaces beyond neuroprosthetics. *Neuron*, 86(1):55–67.

Müller, J., Bakkum, D. J., and Hierlemann, A. (2013). Sub-millisecond closed-loop feedback stimulation between arbitrary sets of individual neurons. *Frontiers in Neural Circuits*, 6.

Musk, E. and Neuralink (2019). An integrated brain-machine interface platform with thousands of channels. *Journal of Medical Internet Research*, 21(10):e16194.

Navajas, J., Barsakcioglu, D. Y., Eftekhar, A., Jackson, A., Constandinou, T. G., and Quian Quiroga, R. (2014). Minimum requirements for accurate and efficient real-time on-chip spike sorting. *Journal of Neuroscience Methods*, 230:51–64.

Neely, R. M., Koralek, A. C., Athalye, V. R., Costa, R. M., and Carmena, J. M. (2018). Volitional modulation of primary visual cortex activity requires the basal ganglia. *Neuron*, 97(6):1356–1368.e4.

Neher, E. and Sakmann, B. (1976). Single-channel currents recorded from membrane of denervated frog muscle fibres. *Nature*, 260(5554):799–802.

Neto, J. P., Lopes, G., Frazão, J., Nogueira, J., Lacerda, P., Baião, P., Aarts, A., Andrei, A., Musa, S., Fortunato, E., Barquinha, P., and Kampff, A. R. (2016). Validating silicon polytrodes with paired juxtacellular recordings: method and dataset. *Journal of Neurophysiology*, 116(2):892–903.

Nicolelis, M. A., Lin, R. C., Woodward, D. J., and Chapin, J. K. (1993). Dynamic and distributed properties of many-neuron ensembles in the ventral posterior medial thalamus of awake rats. *Proceedings of the National Academy of Sciences*, 90(6):2212–2216.

Nicolelis, M. A. L. and Lebedev, M. A. (2009). Principles of neural ensemble physiology underlying the operation of brain–machine interfaces. *Nature Reviews Neuroscience*, 10(7):530–540.

Nobili, L. (1825). Ueber einen neuen Galvanometer. *J.Chem.Phys., Nuremburg*, 45:249–254.

Nordhausen, C. T., Maynard, E. M., and Normann, R. A. (1996). Single unit recording capabilities of a 100 microelectrode array. *Brain Research*, 726(1-2):129–140.

O'Connor, D. H., Hires, S. A., Guo, Z. V., Li, N., Yu, J., Sun, Q.-Q., Huber, D., and Svoboda, K. (2013). Neural coding during active somatosensation revealed using illusory touch. *Nature Neuroscience*, 16(7):958–965.

O'Keefe, J. (1976). Place units in the hippocampus of the freely moving rat. *Experimental Neurology*, 51(1):78–109.

O'Keefe, J. and Bouma, H. (1969). Complex sensory properties of certain amygdala units in the freely moving cat. *Experimental Neurology*, 23(3):384–398.

O'Keefe, J., Burgess, N., Donnett, J. G., Jeffery, K. J., and Maguire, E. A. (1998). Place cells, navigational accuracy, and the human hippocampus. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 353(1373):1333–1340.

O'Keefe, J. and Dostrovsky, J. (1971). The hippocampus as a spatial map. Preliminary evidence from unit activity in the freely-moving rat. *Brain Research*, 34(1):171–175.

O'Keefe, J. and Nadel, L. (1978). *The Hippocampus as a Cognitive Map*. Oxford: Clarendon Press.

O'Keefe, J., Nadel, L., Keightley, S., and Kill, D. (1975). Fornix lesions selectively abolish place learning in the rat. *Experimental Neurology*, 48(1):152–166.

O'Keefe, J. and Recce, M. L. (1993). Phase relationship between hippocampal place units and the EEG theta rhythm. *Hippocampus*, 3(3):317–330.

Olds, J. (1965). Operant conditioning of single unit responses. *Proc. 23rd Congr. Physiological Sciences Excerpta Med. Int. Congr. Ser.*, (no. 87):372–80.

Orsborn, A. L. and Pesaran, B. (2017). Parsing learning in networks using brain–machine interfaces. *Current Opinion in Neurobiology*, 46:76–83.

Otchy, T. M., Wolff, S. B. E., Rhee, J. Y., Pehlevan, C., Kawai, R., Kempf, A., Gobes, S. M. H., and Ölveczky, B. P. (2015). Acute off-target effects of neural circuit manipulations. *Nature*, 528(7582):358–363.

Pachitariu, M., Steinmetz, N. A., Kadir, S. N., Carandini, M., and Harris, K. D. (2016). Fast and accurate spike sorting of high-channel count probes with KiloSort. In *Advances in Neural Information Processing Systems 29*, pages 4448–4456.

Pandarinath, C., Nuyujukian, P., Blabe, C. H., Sorice, B. L., Saab, J., Willett, F. R., Hochberg, L. R., Shenoy, K. V., and Henderson, J. M. (2017). High performance communication by people with paralysis using an intracortical brain-computer interface. *eLife*, 6:e18554.

Pandarinath, C., O'Shea, D. J., Collins, J., Jozefowicz, R., Stavisky, S. D., Kao, J. C., Trautmann, E. M., Kaufman, M. T., Ryu, S. I., Hochberg, L. R., Henderson, J. M., Shenoy, K. V., Abbott, L. F., and Sussillo, D. (2018). Inferring single-trial neural population dynamics using sequential auto-encoders. *Nature Methods*, 15(10):805–815.

Panzeri, S., Harvey, C. D., Piasini, E., Latham, P. E., and Fellin, T. (2017). Cracking the neural code for sensory perception by combining statistics, intervention, and behavior. *Neuron*, 93(3):491–507.

Panzeri, S., Ince, R. A. A., Diamond, M. E., and Kayser, C. (2014). Reading spike timing without a clock: intrinsic decoding of spike trains. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 369(1637):20120467.

Park, J., Kim, G., and Jung, S.-D. (2017). A 128-channel fpga-based real-time spike-sorting bidirectional closed-loop neural interface system. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 25(12):2227–2238.

Penfield, W. and Boldrey, E. (1937). Somatic motor and sensory representation in the cerebral cortex of man as studied by electrical stimulation. *Brain: A Journal of Neurology*, 60:389–443.

Perkel, D. H. and Bullock, T. H. (1968). Neural coding. *Neurosciences Research Program Bulletin*, 6(3):221–348.

Pfeiffer, B. E. and Foster, D. J. (2013). Hippocampal place-cell sequences depict future paths to remembered goals. *Nature*, 497(7447):74–79.

Pfeiffer, B. E. and Foster, D. J. (2015). Autoassociative dynamics in the generation of sequences of hippocampal place cells. *Science*, 349(6244):180–183.

Piccolino, M. (1997). Luigi Galvani and animal electricity: two centuries after the foundation of electrophysiology. *Trends in Neurosciences*, 20(10):443–448.

Pillow, J. W., Shlens, J., Chichilnisky, E. J., and Simoncelli, E. P. (2013). A model-based spike sorting algorithm for removing correlation artifacts in multi-neuron recordings. *PLOS ONE*, 8(5):e62123.

Place, R., Farovik, A., Brockmann, M., and Eichenbaum, H. (2016). Bidirectional prefrontal-hippocampal interactions support context-guided memory. *Nature Neuroscience*, 19(8):992–994.

Pouille, F. and Scanziani, M. (2001). Enforcement of temporal fidelity in pyramidal cells by somatic feed-forward inhibition. *Science*, 293(5532):1159–1163.

Preußer, T. B. and Spallek, R. G. (2014). Ready PCIe data streaming solutions for FPGAs. In *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–4.

Quiroga, R. Q., Nadasdy, Z., and Ben-Shaul, Y. (2004). Unsupervised spike detection and sorting with wavelets and superparamagnetic clustering. *Neural Computation*, 16(8):1661–1687.

Ranck, J. B. J. (1984). Head direction cells in the deep layer of dorsal presubiculum in freely moving rats. *Society of Neuroscience Abstract*, 10:599.

Reece, M. and O'Keefe, J. (1989). The tetrode : a new technique for multi-unit extracellular recording. *Soc. Neurosci. Abstr.*, 15:1250.

Rey, H. G., Pedreira, C., and Quian Quiroga, R. (2015). Past, present and future of spike sorting techniques. *Brain Research Bulletin*, 119:106–117.

Richards, B. A., Lillicrap, T. P., Beaudoin, P., Bengio, Y., Bogacz, R., Christensen, A., Clopath, C., Costa, R. P., de Berker, A., Ganguli, S., Gillon, C. J., Hafner, D., Kepecs, A., Kriegeskorte, N., Latham, P., Lindsay, G. W., Miller, K. D., Naud, R., Pack, C. C., Poirazi, P., Roelfsema, P., Sacramento, J., Saxe, A., Scellier, B., Schapiro, A. C., Senn, W., Wayne, G., Yamins, D., Zenke, F., Zylberberg, J., Therien, D., and Kording, K. P. (2019). A deep learning framework for neuroscience. *Nature Neuroscience*, 22(11):1761–1770.

Rickgauer, J. P., Deisseroth, K., and Tank, D. W. (2014). Simultaneous cellular-resolution optical perturbation and imaging of place cell firing fields. *Nature Neuroscience*, 17(12):1816–1824.

Rieke, F., Warland, D., Steveninck, R. d. R. v., and Bialek, W. (1999). *Spikes: exploring the neural code*. Computational neuroscience series. The MIT Press, Cambridge, Massachusetts.

Rodriguez, A. and Laio, A. (2014). Clustering by fast search and find of density peaks. *Science*, 344(6191):1492–1496.

Romo, R., Hernández, A., Zainos, A., and Salinas, E. (1998). Somatosensory discrimination based on cortical microstimulation. *Nature*, 392(6674):387–390.

Rossant, C. and Harris, K. D. (2013). Hardware-accelerated interactive data visualization for neuroscience in Python. *Frontiers in Neuroinformatics*, 7.

Rossant, C., Kadir, S. N., Goodman, D. F. M., Schulman, J., Hunter, M. L. D., Saleem, A. B., Grosmark, A., Belluscio, M., Denfield, G. H., Ecker, A. S., Tolias, A. S., Solomon, S., Buzsáki, G., Carandini, M., and Harris, K. D. (2016). Spike sorting for large, dense electrode arrays. *Nature Neuroscience*, 19(4):634–641.

Rothschild, G., Eban, E., and Frank, L. M. (2017). A cortical–hippocampal–cortical loop of information processing during memory consolidation. *Nature Neuroscience*, 20(2):251–259.

Roux, L., Hu, B., Eichler, R., Stark, E., and Buzsáki, G. (2017). Sharp wave ripples during learning stabilize the hippocampal spatial map. *Nature Neuroscience*, 20(6):845–853.

Royer, S., Zemelman, B. V., Losonczy, A., Kim, J., Chance, F., Magee, J. C., and Buzsáki, G. (2012). Control of timing, rate and bursts of hippocampal place cells by dendritic and somatic inhibition. *Nature Neuroscience*, 15(5):769–775.

Sadtler, P. T., Quick, K. M., Golub, M. D., Chase, S. M., Ryu, S. I., Tyler-Kabara, E. C., Yu, B. M., and Batista, A. P. (2014). Neural constraints on learning. *Nature*, 512(7515):423–426.

Sakellaridi, S., Christopoulos, V. N., Aflalo, T., Pejsa, K. W., Rosario, E. R., Ouellette, D., Pouratian, N., and Andersen, R. A. (2019). Intrinsic variable learning for brain-machine interface control by human anterior intraparietal cortex. *Neuron*, 102(3):694–705.e3.

Salinas, E. and Sejnowski, T. J. (2001). Correlated neuronal activity and the flow of neural information. *Nature Reviews Neuroscience*, 2(8):539–550.

Salzman, C. D., Britten, K. H., and Newsome, W. T. (1990). Cortical microstimulation influences perceptual judgements of motion direction. *Nature*, 346(6280):174–177.

Santhanam, G., Ryu, S. I., Yu, B. M., Afshar, A., and Shenoy, K. V. (2006). A high-performance brain–computer interface. *Nature*, 442(7099):195–198.

Saxena, S. and Cunningham, J. P. (2019). Towards the neural population doctrine. *Current Opinion in Neurobiology*, 55:103–111.

Schuetze, S. M. (1983). The discovery of the action potential. *Trends in Neurosciences*, 6:164–168.

Schwartz, A. B. (1993). Motor cortical activity during drawing movements: population representation during sinusoid tracing. *Journal of Neurophysiology*, 70(1):28–36.

Schwartz, A. B. (1994). Direct cortical representation of drawing. *Science*, 265(5171):540–542.

Schwartz, A. B. (2004). Cortical neural prosthetics. *Annual Review of Neuroscience*, 27(1):487–507.

Schwartz, A. B. and Moran, D. W. (1999). Motor cortical activity during drawing movements: Population representation during lemniscate tracing. *Journal of Neurophysiology*, 82(5):2705–2718.

Schwarz, D. A., Lebedev, M. A., Hanson, T. L., Dimitrov, D. F., Lehew, G., Meloy, J., Rajangam, S., Subramanian, V., Ifft, P. J., Li, Z., Ramakrishnan, A., Tate, A., Zhuang, K. Z., and Nicolelis, M. A. L. (2014). Chronic, wireless recordings of large-scale brain activity in freely moving rhesus monkeys. *Nature Methods*, 11(6):670–676.

Segundo, J. P., Moore, G. P., Stensaas, L. J., and Bullock, T. H. (1963). Sensitivity of neurones in aplysia to temporal pattern of arriving impulses. *The Journal of Experimental Biology*, 40:643–667.

Seidenbecher, T., Laxmi, T. R., Stork, O., and Pape, H.-C. (2003). Amygdalar and hippocampal theta rhythm synchronization during fear memory retrieval. *Science*, 301(5634):846–850.

Sejnowski, T. J. and Paulsen, O. (2006). Network oscillations: Emerging computational principles. *Journal of Neuroscience*, 26(6):1673–1676.

Serruya, M. D., Hatsopoulos, N. G., Paninski, L., Fellows, M. R., and Donoghue, J. P. (2002). Instant neural control of a movement signal. *Nature*, 416(6877):141–142.

Seyfarth, E.-A. (2006). Julius Bernstein (1839–1917): pioneer neurobiologist and biophysicist. *Biological Cybernetics*, 94(1):2–8.

Shadlen, M. N. and Movshon, J. A. (1999). Synchrony unbound: a critical evaluation of the temporal binding hypothesis. *Neuron*, 24(1):67–77, 111–125.

Shadlen, M. N. and Newsome, W. T. (1995). Is there a signal in the noise? *Current Opinion in Neurobiology*, 5(2):248–250.

Shadlen, M. N. and Newsome, W. T. (1998). The variable discharge of cortical neurons: Implications for connectivity, computation, and information coding. *Journal of Neuroscience*, 18(10):3870–3896.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423.

Sherrington, C. (1906). *The Integrative Action of the Nervous System*. Cambridge University Press Archive.

Shoham, S., Fellows, M. R., and Normann, R. A. (2003). Robust, automatic spike sorting using mixtures of multivariate t-distributions. *Journal of Neuroscience Methods*, 127(2):111–122.

Siegle, J. H. and Wilson, M. A. (2014). Enhancement of encoding and retrieval functions through theta phase-specific manipulation of hippocampus. *eLife*, 3:e03061.

Singer, W. (1993). Synchronization of cortical activity and its putative role in information processing and learning. *Annual Review of Physiology*, 55:349–374.

Singer, W. (1999). Neuronal synchrony: A versatile code for the definition of relations? *Neuron*, 24(1):49–65.

Singer, W. and Gray, C. M. (1995). Visual feature integration and the temporal correlation hypothesis. *Annual Review of Neuroscience*, 18:555–586.

Sirota, A., Montgomery, S., Fujisawa, S., Isomura, Y., Zugaro, M., and Buzsáki, G. (2008). Entrainment of neocortical neurons and gamma oscillations by the hippocampal theta rhythm. *Neuron*, 60(4):683–697.

Sjöström, P. J., Turrigiano, G. G., and Nelson, S. B. (2001). Rate, timing, and cooperativity jointly determine cortical synaptic plasticity. *Neuron*, 32(6):1149–1164.

Skaggs, W. E. and McNaughton, B. L. (1996). Replay of neuronal firing sequences in rat hippocampus during sleep following spatial experience. *Science*, 271(5257):1870–1873.

Skaggs, W. E., McNaughton, B. L., and Gothard, K. M. (1993). An information-theoretic approach to deciphering the hippocampal code. In Hanson, S. J., Cowan, J. D., and Giles, C. L., editors, *Advances in Neural Information Processing Systems 5*, pages 1030–1037. Morgan-Kaufmann.

Skaggs, W. E., McNaughton, B. L., Wilson, M. A., and Barnes, C. A. (1996). Theta phase precession in hippocampal neuronal populations and the compression of temporal sequences. *Hippocampus*, 6(2):149–172.

Softky, W. R. (1995). Simple codes versus efficient codes. *Current Opinion in Neurobiology*, 5(2):239–247.

Sohal, V. S., Zhang, F., Yizhar, O., and Deisseroth, K. (2009). Parvalbumin neurons and gamma rhythms enhance cortical circuit performance. *Nature*, 459(7247):698–702.

Spruston, N. and Cang, J. (2010). Timing isn't everything. *Nature Neuroscience*, 13(3):277–279.

Stark, E., Koos, T., and Buzsáki, G. (2012). Diode probes for spatiotemporal optical control of multiple neurons in freely moving animals. *Journal of Neurophysiology*, 108(1):349–363.

Steinmetz, N. A., Koch, C., Harris, K. D., and Carandini, M. (2018). Challenges and opportunities for large-scale electrophysiology with Neuropixels probes. *Current Opinion in Neurobiology*, 50:92–100.

Stevens, C. F. and Zador, A. (1995). Neural Coding: The enigma of the brain. *Current Biology*, 5(12):1370–1371.

Stolper, C. D., Perer, A., and Gotz, D. (2014). Progressive visual analytics: User-driven visual exploration of in-progress analytics. *IEEE Transactions on Visualization and Computer Graphics*.

Stuart, G. J. and Spruston, N. (2015). Dendritic integration: 60 years of progress. *Nature Neuroscience*, 18(12):1713–1721.

Sussillo, D., Stavisky, S. D., Kao, J. C., Ryu, S. I., and Shenoy, K. V. (2016). Making brain–machine interfaces robust to future neural variability. *Nature Communications*, 7(1):13749.

Suvrathan, A. (2019). Beyond STDP—towards diverse and functionally relevant plasticity rules. *Current Opinion in Neurobiology*, 54:12–19.

Swadlow, H. A. (1985). Physiological properties of individual cerebral axons studied in vivo for as long as one year. *Journal of Neurophysiology*, 54(5):1346–1362.

Swadlow, H. A. (1988). Efferent neurons and suspected interneurons in binocular visual cortex of the awake rabbit: receptive fields and binocular properties. *Journal of Neurophysiology*, 59(4):1162–1187.

Swadlow, H. A. (1992). Monitoring the excitability of neocortical efferent neurons to direct activation by extracellular current pulses. *Journal of Neurophysiology*, 68(2):605–619.

Takahashi, H., Suzurikawa, J., Nakao, M., Mase, F., and Kaga, K. (2005). Easy-to-prepare assembly array of tungsten microelectrodes. *IEEE Transactions on Biomedical Engineering*, 52(5):952–956.

Tampuu, A., Matiisen, T., Ólafsdóttir, H. F., Barry, C., and Vicente, R. (2019). Efficient neural decoding of self-location with a deep recurrent network. *PLOS Computational Biology*, 15(2):e1006822.

Taylor, D. M., Tillery, S. I. H., and Schwartz, A. B. (2002). Direct cortical control of 3D neuroprosthetic devices. *Science*, 296(5574):1829–1832.

Theunissen, F. and Miller, J. P. (1995). Temporal encoding in nervous systems: A rigorous definition. *Journal of Computational Neuroscience*, 2(2):149–162.

Thorpe, S. (1990). Spike arrival times: A highly efficient coding scheme for neural networks. *Parallel Processing in Neural Systems and Computers*.

Thorpe, S., Delorme, A., and Van Rullen, R. (2001). Spike-based strategies for rapid processing. *Neural Networks*, 14(6):715–725.

Tolman, E. C. (1948). Cognitive maps in rats and men. *Psychological Review*, 55(4):189–208.

Trautmann, E. M., Stavisky, S. D., Lahiri, S., Ames, K. C., Kaufman, M. T., O'Shea, D. J., Vyas, S., Sun, X., Ryu, S. I., Ganguli, S., and Shenoy, K. V. (2019). Accurate estimation of neural population dynamics without spike sorting. *Neuron*, 103(2):292–308.e4.

Trimberger, S. M. (2015). Three ages of FPGAs: A retrospective on the first thirty years of FPGA technology. *Proceedings of the IEEE*, 103(3):318–331.

Trouche, S., Koren, V., Doig, N. M., Ellender, T. J., El-Gaby, M., Lopes-dos Santos, V., Reeve, H. M., Perestenko, P. V., Garas, F. N., Magill, P. J., Sharott, A., and Dupret, D. (2019). A hippocampus-accumbens tripartite neuronal motif guides appetitive memory in space. *Cell*, 176(6):1393–1406.e16.

Tulving, E. (2002). Episodic memory: From mind to brain. *Annual Review of Psychology*, 53(1):1–25.

van de Ven, G. M., Trouche, S., McNamara, C. G., Allen, K., and Dupret, D. (2016). Hippocampal offline reactivation consolidates recently formed cell assembly patterns during sharp wave-ripples. *Neuron*, 92(5):968–974.

VanRullen, R., Guyonneau, R., and Thorpe, S. J. (2005). Spike times make sense. *Trends in Neurosciences*, 28(1):1–4.

Velliste, M., Perel, S., Spalding, M. C., Whitford, A. S., and Schwartz, A. B. (2008). Cortical control of a prosthetic arm for self-feeding. *Nature*, 453(7198):1098–1101.

von der Malsburg, C. (1981). The correlation theory of brain function. Technical report, Max Planck Institute for Biophysical Chemistry, Germany.

von der Malsburg, C. (1999). The what and why of binding: The modeler's perspective. *Neuron*, 24(1):95–104.

Wang, P. K., Pun, S. H., Chen, C. H., McCullagh, E. A., Klug, A., Li, A., Vai, M. I., Mak, P. U., and Lei, T. C. (2019). Low-latency single channel real-time neural spike sorting system based on template matching. *PLOS ONE*, 14(11):e0225138.

Wehr, M., Pezaris, J. S., and Sahani, M. (1999). Simultaneous paired intracellular and tetrode recordings for evaluating the performance of spike sorting algorithms. *Neurocomputing*, 26-27:1061–1068.

Wessberg, J., Stambaugh, C. R., Kralik, J. D., Beck, P. D., Laubach, M., Chapin, J. K., Kim, J., Biggs, S. J., Srinivasan, M. A., and Nicolelis, M. A. L. (2000). Real-time prediction of hand trajectory by ensembles of cortical neurons in primates. *Nature*, 408(6810):361–365.

Williams, A. H., Kim, T. H., Wang, F., Vyas, S., Ryu, S. I., Shenoy, K. V., Schnitzer, M., Kolda, T. G., and Ganguli, S. (2018). Unsupervised discovery of demixed, low-dimensional neural dynamics across multiple timescales through tensor component analysis. *Neuron*, 98(6):1099–1115.e8.

Wilson, M. A. and McNaughton, B. L. (1993). Dynamics of the hippocampal ensemble code for space. *Science*, 261(5124):1055–1058.

Xilinx (2016). *7 Series FPGAs Configurable Logic Block User Guide (UG474)*. Xilinx, Inc., https://www.xilinx.com/, 1.8 edition.

Xillybus (2010). *An FPGA IP core for easy DMA over PCIe with Windows and Linux*. Xillybus Ltd, Xillybus.

Yael, D. and Bar-Gad, I. (2017). Filter based phase distortions in extracellular spikes. *PloS One*, 12(3):e0174790.

Yamamoto, J., Suh, J., Takeuchi, D., and Tonegawa, S. (2014). Successful execution of working memory linked to synchronized high-frequency gamma oscillations. *Cell*, 157(4):845–857.

Yger, P., Spampinato, G. L., Esposito, E., Lefebvre, B., Deny, S., Gardella, C., Stimberg, M., Jetter, F., Zeck, G., Picaud, S., Duebel, J., and Marre, O. (2018). A spike sorting toolbox for up to thousands of electrodes validated with ground truth recordings in vitro and in vivo. *eLife*, 7:e34518.

Yu, B. M., Cunningham, J. P., Santhanam, G., Ryu, S. I., Shenoy, K. V., and Sahani, M. (2009). Gaussian-process factor analysis for low-dimensional single-trial analysis of neural population activity. *Journal of Neurophysiology*, 102(1):614–635.

Zhang, F., Wang, L.-P., Brauner, M., Liewald, J. F., Kay, K., Watzke, N., Wood, P. G., Bamberg, E., Nagel, G., Gottschalk, A., and Deisseroth, K. (2007). Multimodal fast optical interrogation of neural circuitry. *Nature*, 446(7136):633–639.

Zhang, K., Ginzburg, I., McNaughton, B. L., and Sejnowski, T. J. (1998a). Interpreting neuronal population activity by reconstruction: Unified framework with application to hippocampal place cells. *Journal of Neurophysiology*, 79(2):1017–1044.

Zhang, L. I., Tao, H. W., Holt, C. E., Harris, W. A., and Poo, M.-m. (1998b). A critical window for cooperation and competition among developing retinotectal synapses. *Nature*, 395(6697):37–44.

Appendix A

NSP hardware implementation

Here we present the actual implementation of the FPGA NSP from the implemented hardware perspective.

A.0.1 The KC705 evaluation kit with FMC-SPI board

The NSP is implemented in the XC7K325T-2FFG900C FPGA Chip using the Xilinx Vivado Design Suite on the Kintex-7 FPGA KC-705 development board. A custom FPGA Mezzanine Card (FMC connector) based PCB (Fig. A.1a, designed by Brian Barbarits at Janelia) interfaces 5 INTAN RHD2132 chips with the Xilinx KC705 FPGA development board (Fig. A.1b). The KC705 FPGA development board is installed physically on the PC via the PCIe slot.

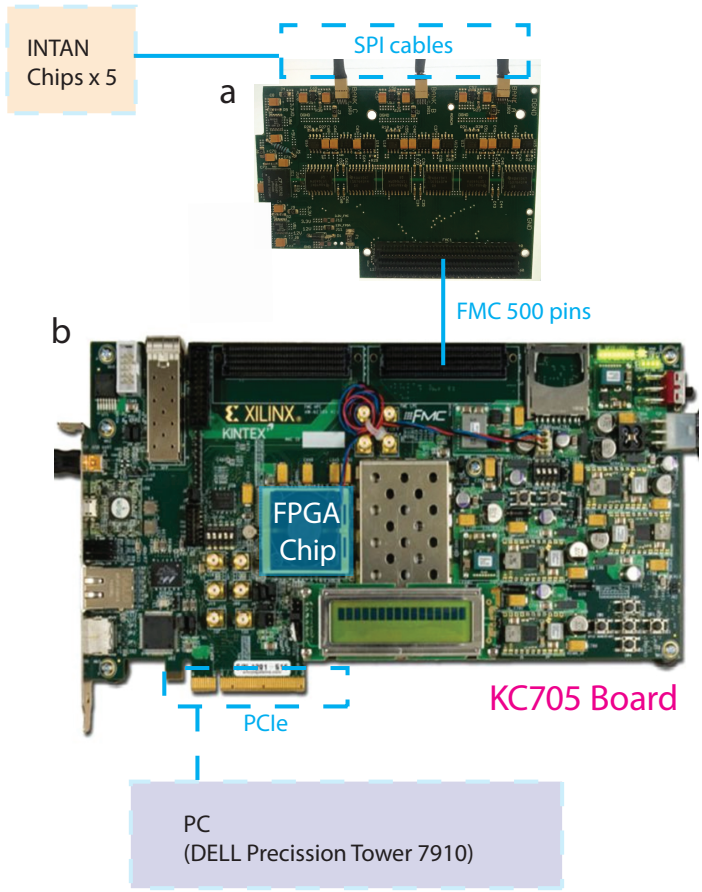


Fig. A.1 **The Hardware Setup (the blue lines indicate physical connections):** (a) A custom FMC-based PCB that interfaces 5 Intan RHD2132 chips with the Xilinx KC705 FPGA development board. (b) The Xilinx KC705 FPGA development board. It has a PCIe Express connector to plug into the PC. The Kintex-7 XC7K325T-2FFG900C Chip is used to develop the FPGA model-based NSP.

The model-based FPGA NSP described in the previous sections is built inside the FPGA Chip (in Fig. A.1b). In the rest of this section, the implementation of the NSP will be presented.

A.0.2 The implemented circuits inside the FPGA

The NSP uses physical digital circuit elements to perform data acquisition, data processing and data transmission. The utilized resources of the implemented digital circuits inside the XC7K325T-2FFG900C FPGA Chip are listed here (Fig. A.2):

Utilization – Post-Implementation

Resource	Utilization	Available	Utilization %
LUT	70542	203800	34.61
LUTRAM	17343	64000	27.10
FF	52888	407600	12.98
BRAM	307.50	445	69.10
DSP	33	840	3.93
IO	27	500	5.40
GT	8	16	50.00
BUFG	9	32	28.13
MMCM	1	10	10.00
PLL	1	10	10.00
PCIe	1	1	100.00

Fig. A.2 **The model-based NSP's utilization of resources in the XC7K325T-2FFG900C FPGA** These resources has been introdeced in the begining of this chapter. The most used physical resource in the implementation of NSP is the BRAM (69% has been used) which is used to store model parameters.

As shown here, the BRAM has been used up to 69%. The limited on-chip memory is certainly one of the major constraints of FPGA design for low-latency application. If the channel count would need to scale up in the future, another FPGA chip with a bigger on-chip memory is required. Also, how to further prune and compress the model is going to be one of the important objectives for future work.

To illustrate how the physical resources wired up together to allow the NSP implementation, the overall usage of the chip (Fig. A.2a) and a zoom-in view in which 10 CLBs are highlighted (Fig. A.2b). The resources and programmed connections inside those 10 CLBs (Fig. A.2c), and the internal view of 1 CLB, where the LUT, Gates/MUX and FF are, was shown (Fig. A.2d).

To illustrate the final NSP circuits with placed and routed physical resources, the FPGA modules and their associated resources is highlighted with different color code in Fig. A.4.

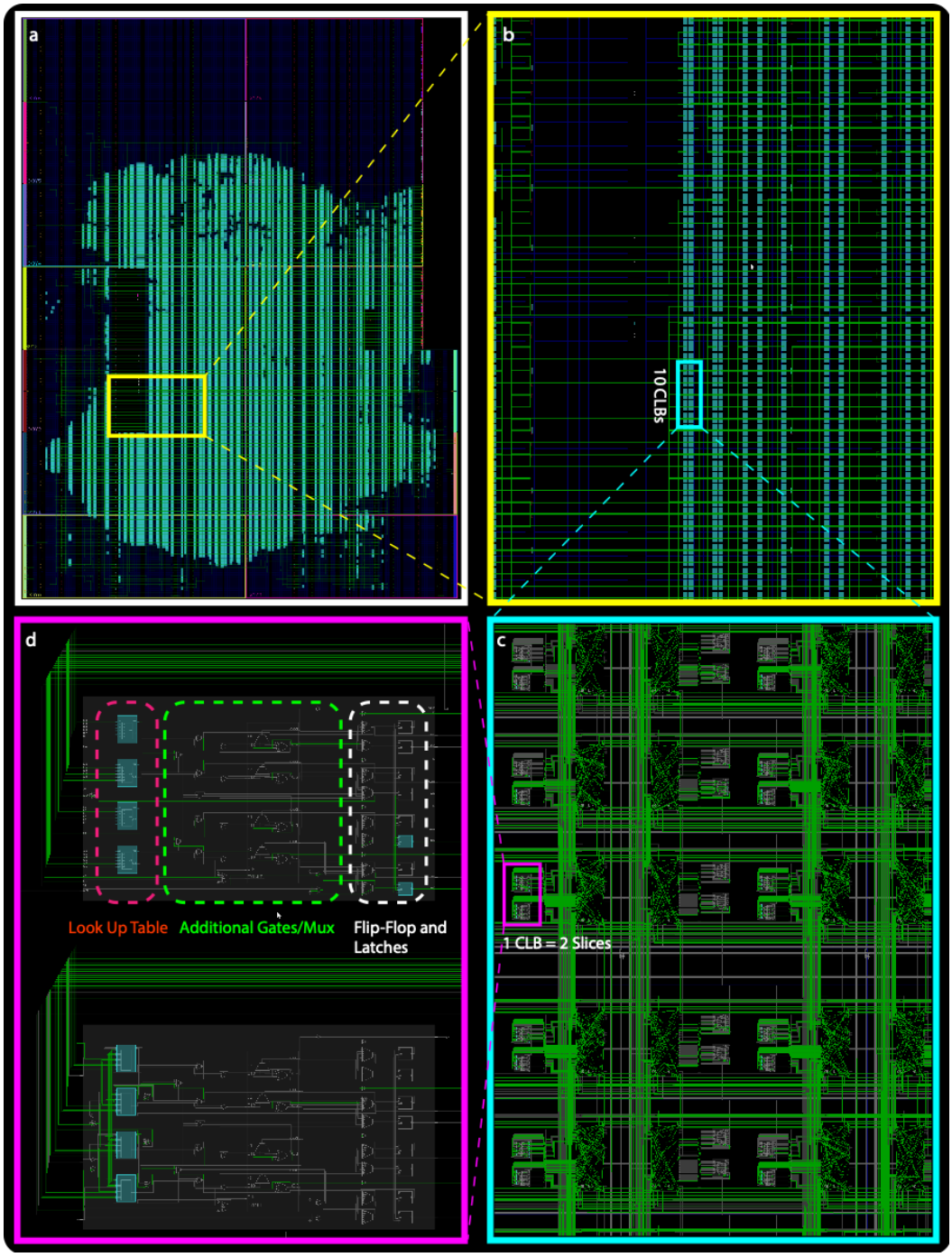


Fig. A.3 **The wired resources in the form of digital circuits in the XC7K325T-2FFG900C FPGA** lines/blocks marked with green indicating programmed connections between physical resources such as look-up-tables (LUT), gates, Mux, flip-flops (FFs) and DSPs. (a) The complete NSP circuit inside the FPGA is highlighted with color; the black part is the unused part of the chip. (b) Part of the total circuit from part of one clock region. (c) 10 configurable logic blocks (CLBs) with 1 CLB highlighted. (d) 2 slices in 1 CLB with highlighted physical resources.

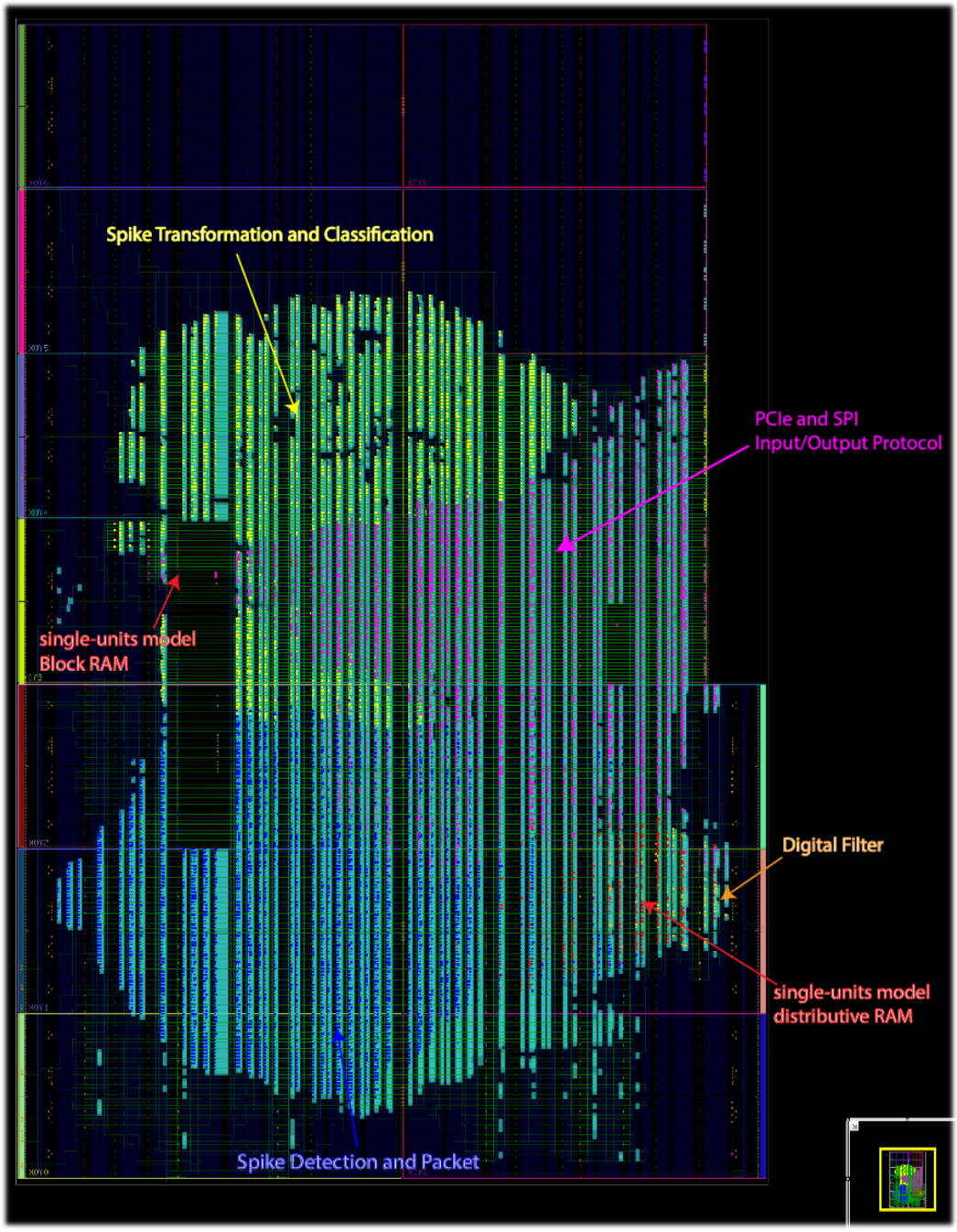


Fig. A.4 **The final NSP circuits in FPGA:** The modules in the NSP are distributed circuits and are spatially clustered (resources from different modules are marked with different colours) resulting from Xilinx place and route tools. Adjacent modules in the NSP signal processing pipeline are physically close to each other.

Appendix B

PCIe communication channels

B.0.1 Xillybus Linux devices for NSP data interface

Device 0:	——— /dev/xillybus_write_32
Specification:	Downstream (host to FPGA): Data width: 32 bits DMA buffers: 1024 x 256 kB = 256 MB
Function:	/dev/xillybus_write_32 writes binary data directly to the NSP algorithmic module (bypassing the SPI) for the development and verification of individual modules independently of the acquisition module.
Device 1:	——— /dev/xillybus_mua_32
Specification:	Upstream (FPGA to host): Data width: 32 bits DMA buffers: 1024 x 128 kB = 128 MB
Function:	/dev/xillybus_mua_32 receives MUA data. Once transmission starts it will receive one frame of data (channels 0 to 159)
Device 2:	——— /dev/xillybus_spk_32
Specification:	Upstream (FPGA to host): Data width: 32 bits DMA buffers: 256 x 128 kB = 32 MB
Function:	/dev/xillybus_spk_32 receives spike packet data. Once transmission starts it will receive one complete packet.

Device 3:	——— /dev/xillybus_fet_clu_32
Specification:	Upstream (FPGA to host): Data width: 32 bits DMA buffers: 128 x 128 kB = 16 MB Seekable: No
Function:	/dev/xillybus_fet_clu_32 receives the NSP final output, identified spike feature packets: (frame#, group#, spike feature vector, spike id)

B.0.2 Xillybus Linux devices for NSP memory interface

Device 4:	——— /dev/xillybus_bmi_model_32
Specification:	Upstream (FPGA to host): Data width: 32 bits DMA buffers: 4 x 8 kB = 32 kB Seekable: Yes, with 16 address bits Downstream (host to FPGA): Data width: 32 bits DMA buffers: 4 x 8 kB = 32 kB Seekable: Yes, with 16 address bits
Function:	/dev/xillybus_model_32 writes and reads from a DPRAM that stores the model parameters for all three stages of NSP signal processing.
Device 5:	——— /dev/xillybus_control_regs_16
Specification:	Upstream (FPGA to host): Data width: 16 bits DMA buffers: 4 x 4 kB = 16 kB Seekable: Yes, with 5 address bits Downstream (host to FPGA): Data width: 16 bits DMA buffers: 4 x 4 kB = 16 kB Seekable: Yes, with 5 address bits
Function:	/dev/xillybus_control_regs_16 generates control signals, e.g. start/stop NSP acquisition and other acquisition related parameters.