Brain-Machine Interface Neural Firing Rate Estimation

Jingchao Zeng
MSc(Hons)
Supervisor: Prof. Timothy G. Constandinou

A Thesis submitted in fulfilment of requirements for the degree of
Master of Science
Applied Machien Learning
of Imperial College London

Department of Electrical and Electronic Engineering
Imperial College London
September 2, 2021

Abstract

With the development of wireless brain-machine interface (BMI) applications, neural signals with multiple channels are collected. The multi-unit activity (MUA) is a common signal for BMI decoding (Neuralink, BrainGate). Usually, the high-sampling MUAs are detected and transmitted wirelessly. The received spikes are then estimated by the neural firing rate method and used for decoding on PC. Due to high transmitted rates in this off-implant approach, it is highly desirable to estimate the firing rate on-implant, saving communication bandwidth.

To meet this requirement, we have developed a hardware-friendly method, known as penalised inter-spike interval (PISI), to estimate the instantaneous frequency of spikes. At the first stage, PISI is compared with other state-of-the-art firing rate methods on the synthetic data. On the real-world data, MUA spikes are firstly extracted and estimated by the PISI algorithm. Both asynchronous and synchronous ways are presented to organise the estimates, remedying the bandwidth requirement. Then, a novel normalisation function is applied to data, improving the entropy of Huffman encoding. Finally, the proposed method is implemented and optimised in Xilinx Zedboard for power and resource measuring.

Implementing Huffman encoding on 96-channel normalised estimates can reduce the data rate by nearly 36000 times and improves the entropy by four times. The proposed algorithm achieves around 0.72 correlation coefficient (CC) and 0.62 on the prediction of 2-dimensional position and velocity. After fixed-point optimisation, the real-time hardware implementation of the PISI algorithm consumes 21 mW and takes 400 clock cycles to operate on average.

Acknowledgment

Firstly, I want to show my special gratitude to my supervisor Prof. Timothy Constandinou. You assigned me two excellent PhDs, Zheng Zhang and Savolainen Oscar. Their helpful advice and guidance pushed me further on the development of methodology in both software and hardware implementation.

I also liked to thank my best friend, Dennison Lau, for our active discussion in hardware optimisation. Finally, I would like to thank my parent and girlfriend for their continuous support during the year.

Contents

List of Figures

List of Tables

Abbreviations

BMI: Brian-Machine Interface

PISI: Penalised Interspike Interval

PFG: Penalised FrequencyGram

EEG: Electroencephalography

AP: Action Potential

LFP: Local Field Potential

SNR: Signal-to-noise Ratio

MEG: Magnetoencephalograph

MUA: Multi-Unit Activity

SUA: Single-Unit Activity

MISE: Mean Integrated Squared Error

GKS: Guassian Kernel Smoother

BAKs: Bayesian Adaptive Kernel Smoother

ISIs: Interspike Intervals

KF: Kalman Filter

WCF: Wiener Cascade Filter

RNN: Recurrent Neural Network

LSTM: Long Short-term Memory

GRU: Gated Recurrent Unit

CC: Pearson's Correlation Coefficient

RMSE: Root Mean-squared Error

IP: Intellectual Property

FF: Flip Flop

LUT: Look-up Table

BRAM: Bidirectional Random Access Memory

Chapter 1

Introduction

1.1 Motivation and objectives

Nowadays, many people suffer from neurological diseases, including stroke, Parkinson disease and spinal cord injury. Intracortical BMI systems are promised to restore the lost motor functions by translating the neural signals into control commands that guide neuroprosthesis devices [2]. Early clinical research has successfully demonstrated BMIs can be applied to control computer cursor [3, 4] and robotic arms in both human and nonhuman primate [5]. However, some challenges remain for current wireless intracortical BMIs research, including the high data bandwidth, resource and power limitation, and poor decoding accuracy.

The previous research focuses on off-implant methods, where high sampling broadband signals are processed for spike detection and then transmitted directly to PC for firing rate estimation. This approach is not suggested in the small-size implantable device since the enormous data rates always lead to a fair amount of power consumption. The histogram method, binning, is actively used in many BMI hardware systems, but it can only indicates temporal spike dynamics [6]. Therefore, it is highly desirable to have a hardware-friendly way to estimate the firing rate, targeting these existing problems.

Facing these requirements, a Penalised Interspike-Interval (PISI) method is proposed to characterise instantaneous spike dynamics. The proposed algorithm is first im-

plemented in MATLAB for performance testing and hyperparameter tuning on synthetic and real-world data. Then, a normalisation function is developed to improve the coding efficiency of the Huffman coding scheme. Furthermore, both synchronous and asynchronous approaches are designed and investigated to reduce the data rate of transmission. The classical neural decoders evaluate the performance of rate estimation.

Finally, the PISI and binning approaches are implemented and optimised in a Xilinx Zedboard using fixed-point representation. The results of resource utilisation, power consumption, throughput performance and processing speed are given for comparison.

1.2 Thesis Structure

The thesis is organised as follow: Chapter 2 provides background and literature review from previous works. Chapter 3 discusses the methodology for PISI implementation and testing in both software and hardware domain, and Chapter 4 presents the results and discussion on the performance of PISI and binning. Finally, Chapter 5 summarises the outcome of the whole project and lists some future work.

Chapter 2

Literature Review

2.1 Background

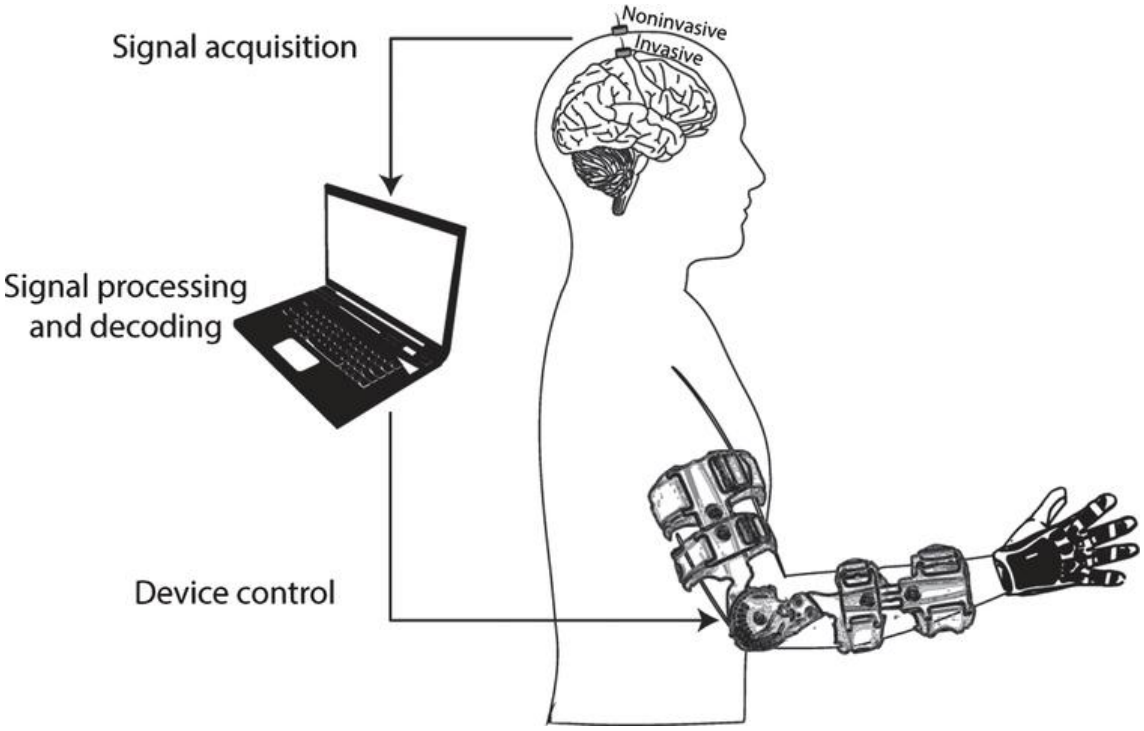

Figure 2.1: The demo for current BMI application in arm movement recovery [1]

Figure 2.1 demonstrates the typical process from recording the neural signals to command the prosthetic devices, which restores arm kinematics. There are two types

of recording techniques, either invasive and non-invasive. Patients can wear non-invasive headsets to record Electroencephalography (EEG) or Magnetoencephalography (MEG) signals to track neuron activities. However, the signal-to-noise ratio (SNR) is generally small compared with the invasive implantation [7]. For the invasive micro-electrode recording, action potential (AP) and local field potential (LFP) are commonly obtained, conveying more information and high SNR.

According to the research in [8], AP locates between 250 and 6000 Hz whereas LFP locates below 250 Hz, both of which are extracted by digital filters. For AP, a spike detection with a defined threshold is required to extract a sequence of spikes, known as spike train. Typically, the detected spike train is transmitted wirelessly to the mobile platform. Then, the frequency of spikes, known as firing rates, is then estimated and collected as inputs to train the decoder. Finally, the decoder is employed to predict velocity or position in real-time, which controls the mechanical movement of the device.

As discussed, the direct transmission of spike trains leads to considerably high data rates. In the Figure 2.2, the proposed system can be applied to facilitate a low data rate.

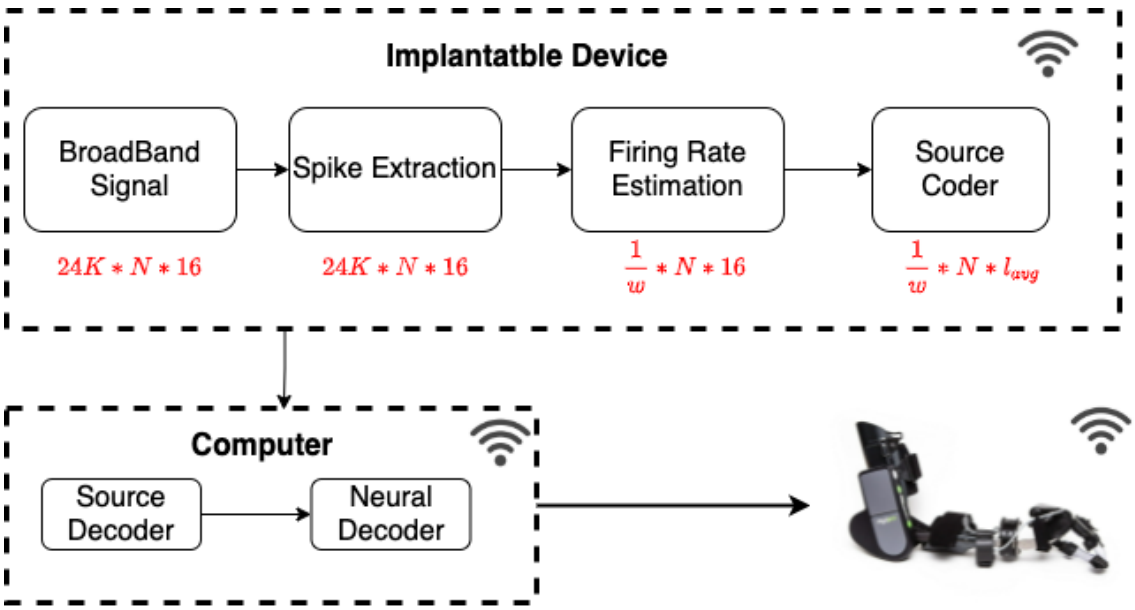

Figure 2.2: Block diagram of the proposed wireless BMI system. Red texts represent the data rate required for each block. The arrow demonstrate the order of flow. N: number of channels. l_{avg}: average codeword length

Assumed the signal is sampled at 24 kHz and quantised with 16 bits/sample, the initial data rate for spike extraction is supposed to be 24k*16 = 384k bits/s/channel, which consumes a fair amount of transmission power. Depending on the window size, estimating the firing rate in hardware can compress the sampling rate by more than thousands of times. Furthermore, the data compression coding schemes are applied to reduce the bit width before transmission. The firing rate approach and source coder implementation should be low complexity in the targeted implantable device with limited power and area budgets. Finally, a source decoder is applied to convert codewords back to firing rates, followed by neural decoding.

2.2 Multi-Unit Activity Spike

Single unit activity (SUA) and multi-unit activity (MUA), are measured from AP, which corresponds to the single unit or several units of neuron [7]. Although SUA can provides high decoding performance, several research suggest that the the spike sorting procedure is computationally heavy, which consumes lots of power in the real-time application [9,10]. Besides, the decoding performance using SUA will degrade in the long term due to unstable electrodes and degradation of insulation properties [10, 11]. The alternative approach using MUA not only removes the spike sorting procedure but also maintain a similar performance as SUA in a long term [12]. Therefore, using MUA from neural population offers computational simplicity and compromised decoding performance.

2.3 Firing Rate Estimation

Firing rate, known as rate coding, represents the frequency at which a neuron fires spikes, commonly related to specific behaviour tasks. Usually, the firing rate is typically estimated offline by averaging the number of repeated trials to obtain the temporal spike dynamics [6, 13, 14]. It is usually unavailable for real-time BMI applications, which requires the firing rate to be estimated accurately on a single trial [15].

Due to its inherent stochastic nature, the spike response may differ even though

the environment setting remains the same all the time [6, 15]. In practice, there are three existing categories of firing rate measurement to target this problem. The most frequently used method for rate estimation is the time histogram [16], know as binning. For this method, it counts the number of spikes in the defined bin size Δ and divides the Δ to obtain the coarse representation of firing rate [13],

$$\lambda(t) = \frac{k_i}{\Delta} \tag{2.1}$$

where k_i denotes the number of spikes in the i^{th} bin. Since it is straightforward to implement, it may not describe rate fluctuation accurately due to the inaccurate bin size [13]. Shimazaki et al. [17] proposed a method to optimise the bin size based on the mean integrated squared error (MISE) between estimated and underlying rate. Later, Endres et.al [18] developed a Bayesian Binning method to vary Δ, which can capture rapid change in the firing rate.

Another popular method is kernel smoothing, which convolves every data points s_i locally with the designed kernel function K_w in bandwidth w [19],

$$\lambda(t) = \sum_{i=1}^{N} K_w(t - s_i) \tag{2.2}$$

The choice of the valid kernel must be nonnegative, finite bandwidth and symmetric [14]. Gaussian kernel smoother (GKS) has infinite supports to obtain a smooth estimate of rate function, providing good approximation than rectangular kernel one [13, 14]. Unlike GKS with fixed w, Bayesian adaptive kernel smoother (BAKs) employs a Bayesian framework to adaptively modify w, further improving the estimation [6].

One of the oldest methods, known as Frequencygram, investigate the reciprocal value of interspike intervals (ISIs) to determine the instantaneous firing rate between neighbouring spikes [20]. The key difference with the previous two methods is that estimated firing rates are typically higher [13], which leads to inaccurate estimation.

2.4 Data Compression Mechanism

The Huffman encoding is one type of lossless compression scheme, which uses variable-length codeword to represent each symbol based on their occurrence frequency [21]. Each codeword length is designed to be close to Shannon's entropy [22]. In practice, the non-adaptive Huffman encoding has been effectively applied to different intracortical neural signals to boost the compression ratio [23].

The delta encoding computes the difference between sequence values. With this approach, the variance of values is reduced if the neighbour data are highly correlated, resulting in less bit usage for the same data [24]. The Huffman encoding can then compress the delta-sampling data to obtain a higher compression ratio [25].

2.5 Neural decoding

To map the estimated firing rate to the target behaviour output (kinematics etc.), there are many types of decoders under research [26, 27], such as linear vector algorithm, Bayesian algorithm and machine learning (ML) decoder. Under certain conditions for the linear vector algorithm, the cosine of reach direction is related to the firing rate. But it often leads to poor performance compared to closed-loop Bayesian decoders, such as Kalman filter (KF) [28] and Wiener filter (WF) [29] due to additional Guassian assumption that is not present in linear vector algorithm. Several nonlinear Bayesian decoders, such as unscented Kalman filter (UKF) [30], and wiener cascade filter (WCF) are investigated to map the nonlinear neuronal activities to decoded variables with slight improvement. Until recent years, the advance of recurrent neural network (RNN), such as Long short-term memory (LSTM) and Gated recurrent unit (GRU), boosts the decoding performance to the next level but suffers from longer training time and more parameters for tuning [26, 27, 31].

Chapter 3

Methodology

3.1 Overview

This section describes synthetic and real-world datasets that are given and applied in the following testing stages. Then, the concept for a PISI method is developed and tested using MATLAB on the provided datasets. After all testing, the algorithm is migrated onto the Zedboard with the fixed-point optimisation. The throughput optimisation is investigated to increase the overall throughput performance. Finally, the effectiveness of the algorithm is evaluated by comparing it with the popular binning method.

3.2 Experiment Data

Synthetic Data

Since the firing rates are unknown in the real-world neural signal, it is essential to have models to generate the synthetic spike train, which benchmarks firing rate estimation methods' performance with respect to the underlying rate functions. According to a study in [13], a spike train can be modelled as a point process that describes localised events, referred to Poisson Process. However, the Poisson process fails to model history-dependent properties (refractory period and bursting) that often existed in the real-world data [6]. Therefore, we used datasets provided by Ahmadi [6], which can model a non-Poisson

process to capture these properties.

Real-world Data

Raw neural signals recorded from a primate subject monkey (Indy) are used [32], which performs a point-to-point task, that is, to reach random targets on an 8x8 square grid by moving a cursor. The 16-bit digitised data were recorded by a Utah array (offering 96-channels) with a sampling rate of 24414.0625 Hz, which are pre-amplified and filtered by a 4^{th} order low-pass filter at 7.5 kHz with a roll-off of 24 dB per octave. The corresponding behaviour outputs (x and y cursor position) sampled at 250 Hz are also given. The total recording duration lasts around 390 seconds.

3.3 Penalised Interspike-Interval (PISI)

Generally, the firing rate may depend on the history of MUA spikes. To deal with time-dependent changes, the instantaneous firing rates $\lambda(t)$ are calculated by interspike intervals (ISIs) x_i, that is, the time difference between adjacent MUA spikes s_i and s_{i-1} in (3.1). It is known as the Frequencygram method shown in (3.2).

$$x_i = s_i - s_{i-1}, \qquad s_{i-1} < t < s_i \tag{3.1}$$

$$\lambda(t) = \frac{1}{x_i} \tag{3.2}$$

The intuition of this method comes from the relationship between ISIs and the frequency of spikes. If neighbouring spikes are close together, the neurons should fire frequently. The simplicity of this algorithm can easily capture the abrupt changes of spike activities but suffers from the variability of ISIs, where small changes will lead to inaccurate rate estimation.

To target this problem, a penalised version of the Frequencygram method (PFG) is proposed, aiming to smooth the effects of variation in the ISIs and maintain the sensitivity to the change of spike density. The penalised condition is to check the difference between

the current firing rate λ_i and the previous one λ_{i-1}. If significant changes are detected ($\geq$ predefined threshold T), λ_i is set to almost equal to λ_{i-1} with a slight increase (declare p $\approx$ 0). The algorithm is demonstrated below:

Algorithm 1 PFG

Require: s_i, s_{i-1}, λ_{i-1}

Ensure:

$x_i \leftarrow s_i$ - s_{i-1}

$\lambda_i \leftarrow 1/x_i$

if λ_i - $\lambda_{i-1} \geq$ T **then**

$\lambda_i \leftarrow \lambda_i * p + \lambda_{i-1} * (1 - p)$

end if

$\lambda_{i-1} \leftarrow \lambda_i$

$s_{i-1} \leftarrow s_i$

In the Figure 3.1, the $\lambda(t)$ estimated from the Frequencygram method are inaccurate and large in the high-density area, which is mitigated by the penalty condition. In the meantime, most nutritional information, such as waveform and overall trend, are successfully maintained. The PFG method highlights $\lambda(t)$ in the crowded spike areas according to its correlated relationship. In contrast, the original FrequencyGram method may estimate high $\lambda(t)$ in the sparse region due to tiny gaps between two spikes. It can be concluded that with this additional penalty condition, the estimated $\lambda(t)$ are more valuable and accurate.

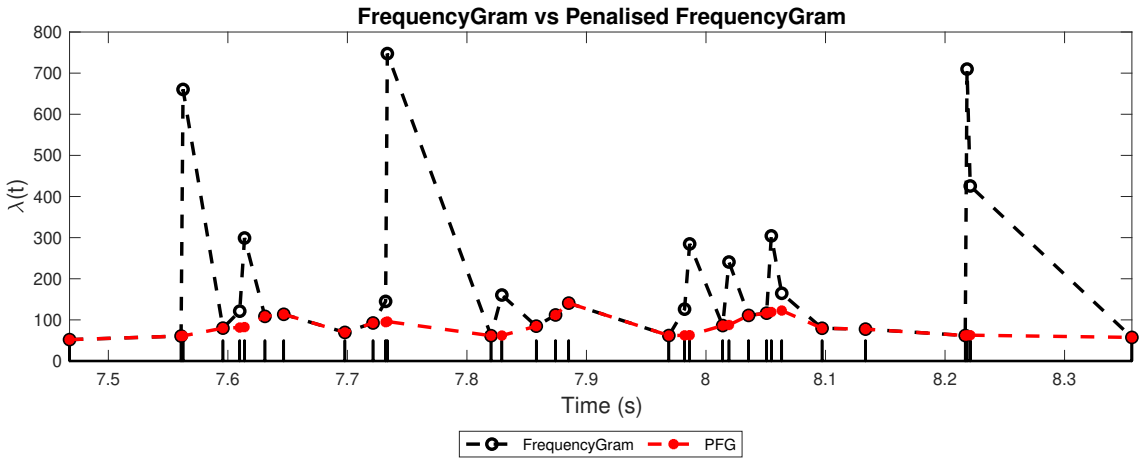

Figure 3.1: Comparison between PFG and Frequencygram method for 18s spike train (black raster). The magnitude of spike train is enlarged by 50 times and the other firing rates are leveled up (start by 50) correspondingly for visualisation.

In the Equation 3.2, the x_i and $\lambda(t)$ contain same information. This suggests that using ISIs can represent the instantaneous frequency of spikes. The favourable reason for using ISIs is the absence of division operation for every sample, resulting in fewer hardware resources, such as DSP. Therefore, we can modify the PFG algorithm and work with ISIs directly for neural decoding. All multiplication in the conditional statement can be optimised in the fixed-point representation. With this context, a PISI algorithm is presented below:

Algorithm 2 PISI

Require: s_i, s_{i-1}, x_{i-1}

Ensure:

$x_i \leftarrow s_i$ - s_{i-1}

if x_{i-1} - $x_i \geq T * x_i * x_{i-1}$ **then**

$x_i \leftarrow x_{i-1} * (1-p)$ - $x_i * p$

end if

$x_{i-1} \leftarrow x_i$

$s_{i-1} \leftarrow s_i$

3.4 MATLAB Implementation

The section describes the floating-point operation of the proposed method in MATLAB. It covers two testing stages in detail, which is given in Figure 3.2.

Firstly, the proposed PISI is compared with the other state-of-the-art firing rate methods on synthetic datasets. Besides, two hyperparameters T and p, are tuned to reduce the estimation error, where the optimal values are configured in the real-world stage.

The BMI implementation flow is simulated on the broadband neural signal, from the spike detection to PISI estimation. Given the fixed window w, the estimated ISIs are organised in either synchronous or asynchronous ways. Two coding schemes, delta and Huffman, are then applied to compress the data information. A normalisation function is proposed to reduce the dynamic range of estimates before Huffman encoding. The classical decoders are applied to evaluate the decoding performance for encoded ISIs with normalisation, original ISIs and firing rate from the binning.

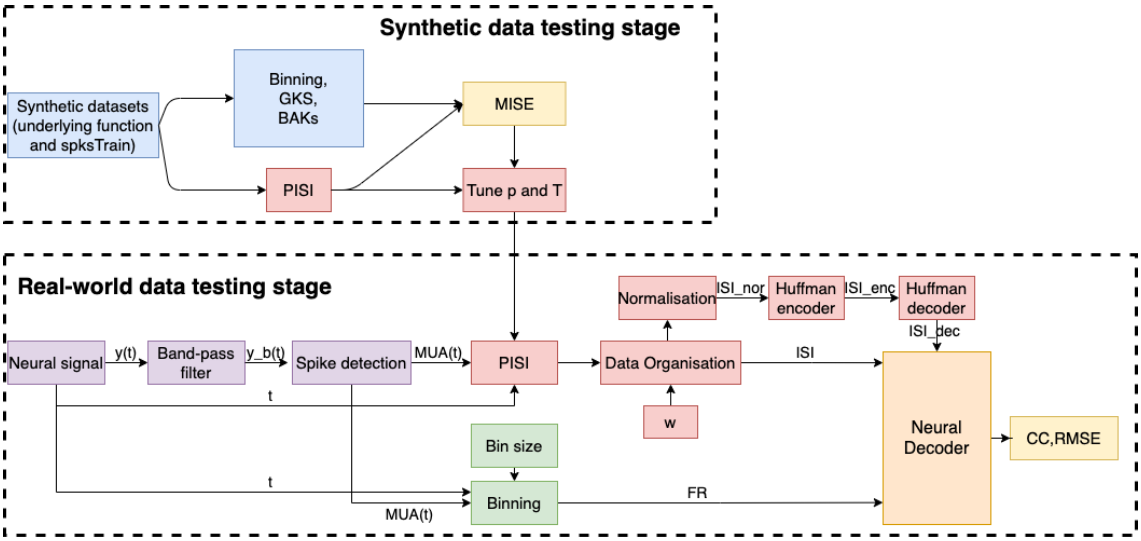

Figure 3.2: Block diagram of synthetic and real-world data testing stage. Pink box: our works. Blue box: inputs and other firing rate methods. Purple box: spike train extraction; Green box: binning method. Upper yellow box: MISE errors w.r.t ground-truth values. Below yellow box: decoding accuracy (CC,RMSE)

3.4.1 Synthetic Data Testing

In Ahmadi's synthetic datasets, the spike trains are generated from non-Gaussian Models with four types of rates function, including chirp, sine and sawtooth wave and Guassian-damped sine wave. Each functions have their own characteristics related to spike behaviour [6]:

- chirp: a continuous process with changing frequency
- sine: a continuous process with oscillatory frequency
- sawtooth: a discontinuous process with sudden rate changes
- Guassian-damped sine wave: a continuous process with various intensity and frequency

We investigated the binning, GKS, and BAKs methods on a single-trial spike train for comparison. The firing rates of PISI are obtained from the reciprocal of estimated ISIs. The kernel-based methods, like GKS and BAKs, convolve each data points with

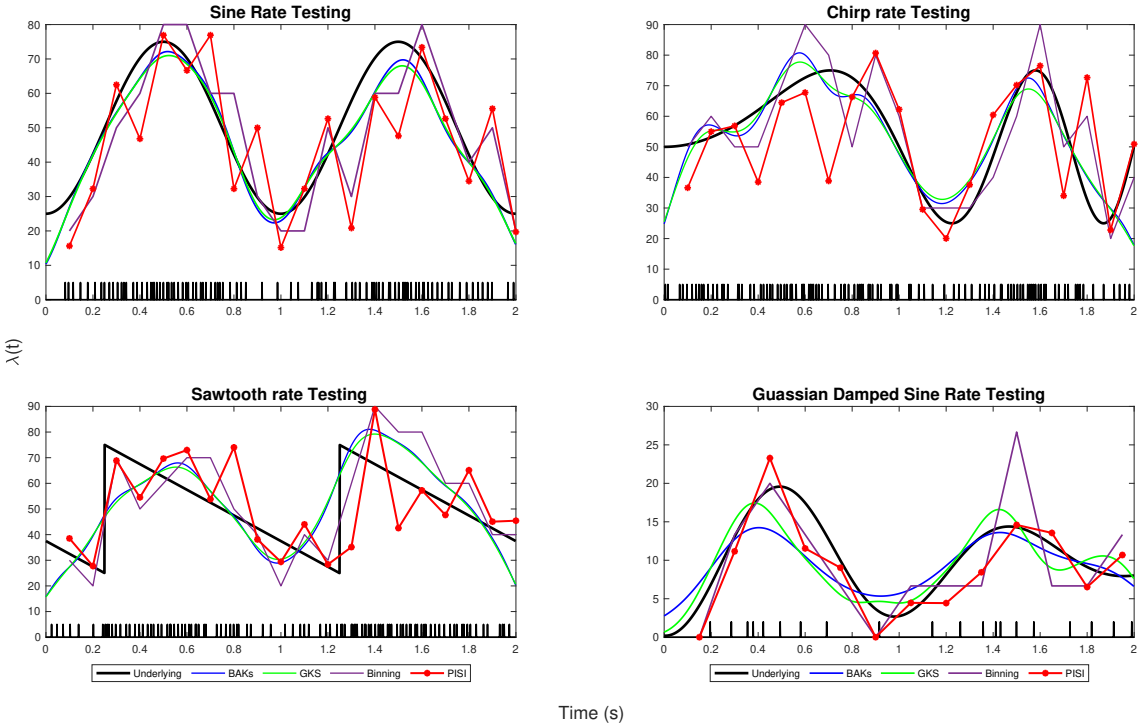

Figure 3.3: Comparison of firing rate estimates across all rate functions on the black raster of spike train. Black line indicates the underlying function. Both bandwidth of kernel methods and window size of PISI and binning are set to 100 ms.

a kernel function that describes the spike distribution smoothly. However, the trade-off comes from heavy computation required on the exponent, square root, and many division operations, which is unlikely to implement on low-power hardware. On the contrary, the PISI and binning only require addition/subtraction and multiplication/division with fixed values can be optimised later by bit-shifting. Unlike kernel smoothing methods, the sampling rate of PISI and binning methods depends on the window size, offering the sparse representation of spike behaviour. Therefore, the PISI and binning methods' primary goal should aim to capture different types of spike activity with compromised performance.

Figure 3.3 demonstrates the estimation performance across all the methods. Overall, the kernel-based methods can smoothly capture most features of the underlying rate functions, while the other two methods can approximate the rate function with fluctuation. In the Gaussian damped and sawtooth wave, the binning method performs slightly poor than the PISI method. However, If the spike events are continuously active (chirp and

sine wave), the PISI estimates fluctuate around the underlying signal due to the penalised implementation.

As discussed, the PISI method has two penalty hyperparameters T, and p. 100-trial estimates for sine function are averaged to get the trend of effects in Figure 3.4. As T increases from 16 to 64, the penalised condition is hardly satisfied for some spike intervals, overestimating the firing rate. Similarly, if p is set high, it reduces penalised effects on measured firing rate, where the higher magnitude and jagged waveform are observed. The impact of T and p are correlated, where the best combination will be investigated later.

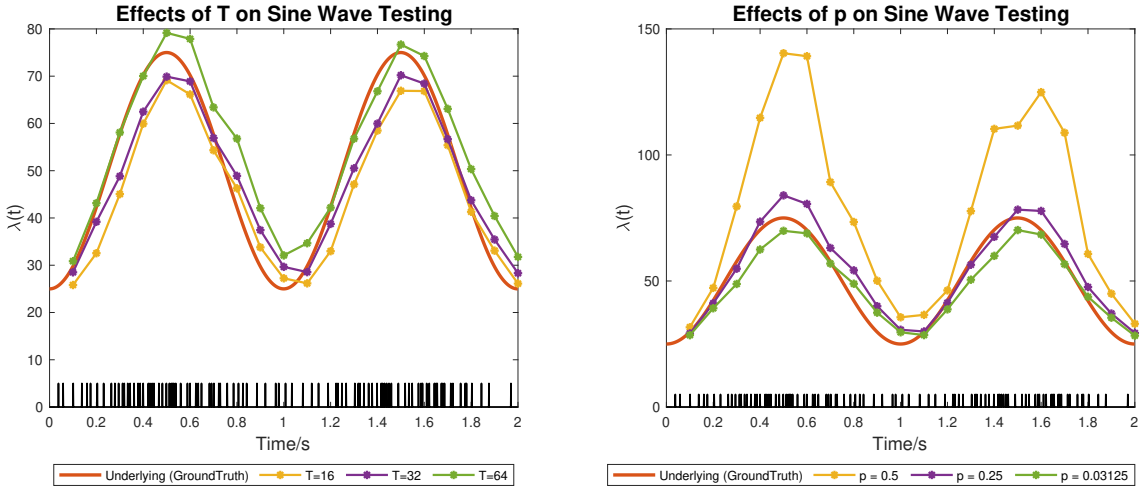

Figure 3.4: Effects of T and p on the firing rates of PISI method on sine wave

3.4.2 Real-world Data Testing

Spike Detection

Unlike given actual spike in the synthetic datasets, we need to filter raw broadband signal, define a threshold and finally detect spikes for signal crossing the threshold. As discussed, most MUA spikes locate between 250 and 6000 Hz. A 4^{th} order Butterworth bandpass filter is applied to remove low-frequency background signal and high-frequency noise. The unwanted transient response of the filter is removed by discarding first 500 samples.

The primary and straightforward way to define a threshold is based on the standard

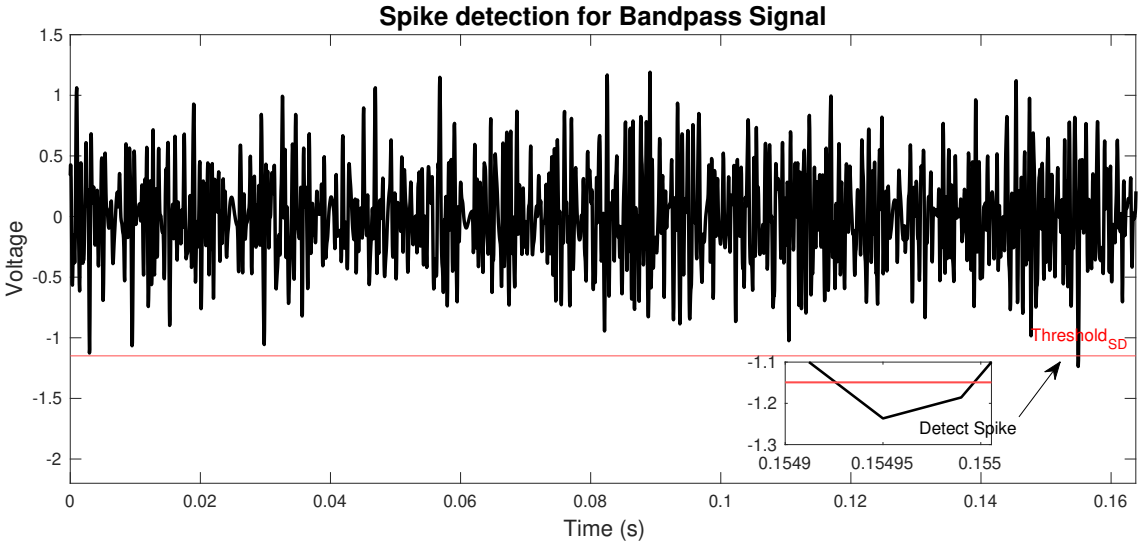

Figure 3.5: Spike detection for band-pass signal with threshold definition (k = 3.6). Zoom-in spike waveform is given in the small plot.

deviation (std) of filtered signal (y) shown in Equation 3.4. The value k can be selected from 2 to 4 according to empirical testing.

$$std = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}|y_i - \frac{1}{N}\sum_{i=1}^{N}y_i|^2} \tag{3.3}$$

$$T = -k * std \tag{3.4}$$

Figure 3.5 illustrates all stages for spike detection on the band-pass data. Given the threshold, the spike detection windows are swiped across every input sample to detect

Algorithm 3 Spike Detection

Require: $detect \leftarrow false$

Ensure:

if threshold crossing and false detect **then**
  $s_i \leftarrow 1$
  $detect \leftarrow true$
else if threshold crossing and true detect **then**
  $s_i \leftarrow 0$
else if threshold not crossing **then**
  $s_i \leftarrow 0$
  $detect \leftarrow false$
end if

the points crossing the negative threshold. The algorithm is described above. Using a boolean variable *detect* enables to mark one spike until crossing the threshold next time.

Firing Rate Estimation

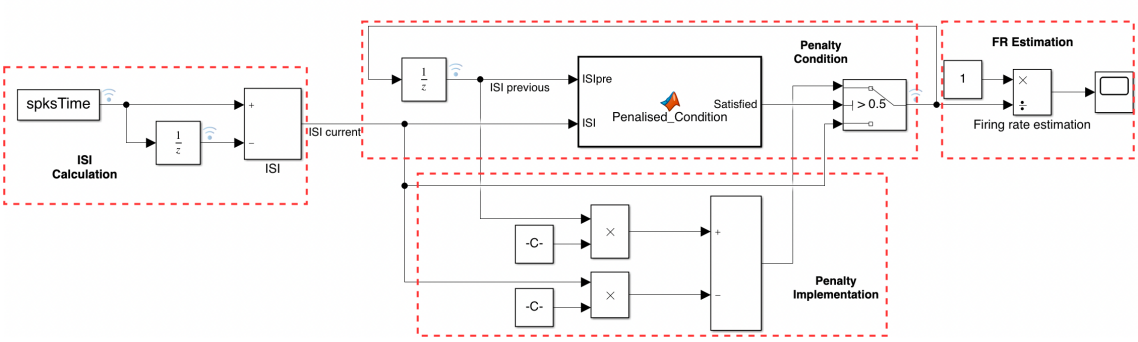

Figure 3.6: Block diagram of the PISI model in Simulink

As shown in Figure 3.6, the real-time PISI algorithm is modelled in Simulink and include blocks: ISI calculation, Penalised Condition, Penalty Implementation and Firing rate estimation. To measure current ISI, the input spike time is provided and delayed. Similarly, the previous ISI output is also delayed along with the current ISI for the penalty condition. Inside the condition block, their differences can be measured, which configures *Satisfied* to 1 or 0. This result is then passed to a digital switch that controls if selecting penalised values from Penalty Implementation. The new ISI output is finally fetched back to the Penalised Condition block as the previous ISI in the next time. In the meantime, the firing rate is calculated for visualisation in Scope.

Both fixed-point and floating-point operations are tested on 20-second spike time. The fixed-point implementation can effectively reduce the cost of hardware resources for storage and power, but it will also degrade the signal accuracy. To maintain this accuracy, the best way is to consider the possible range for different variables, and give enough bits for representation shown in Table 3.1.

All variables are configured by 16 bits with different integer points. For the spike time, the number of integer bits depends on the maximum time. Since all ISIs are smaller

Table 3.1: Bit configuration for fixed-point operation in Simulink.$< a, b >$: a represents word length and b represents integer bits

Variable	Configuration
Spike Time	$< 16, 6 >$
ISI	$< 16, 1 >$
Firing Rate	$< 16, 12 >$

than one, 15 fractional bits and one integer bits are declared. Finally, the bit configuration of the firing rate depends mainly on the ISIs. Surprisingly in Figure 3.7, the firing rate measured from the fixed-point operation can maintain most of information from the floating-pointing operation. Overall, the total bits can be reduced from 64 bits (double) to only 16 bits without sacrificing too much accuracy.

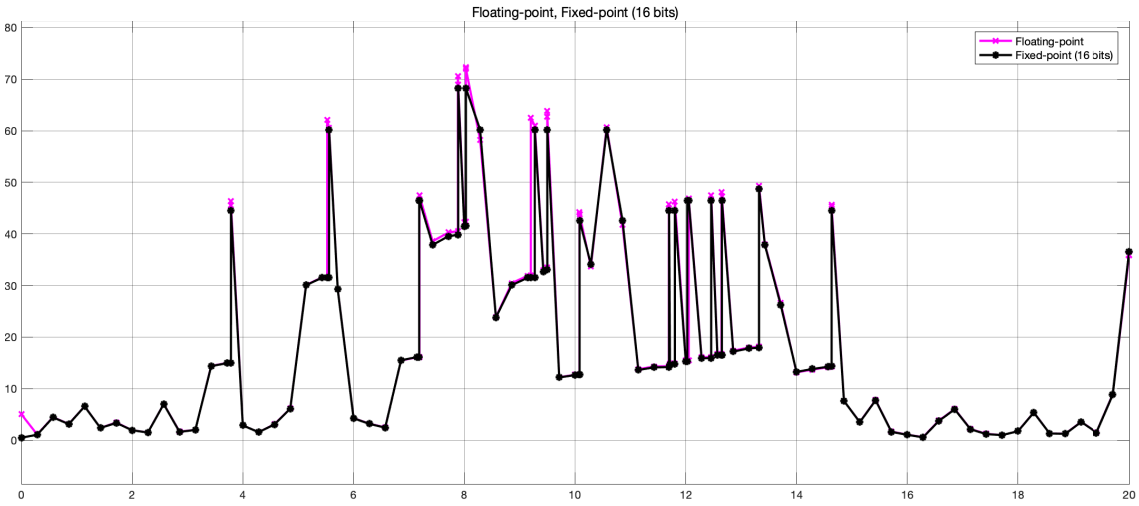

Figure 3.7: Estimated Firing rate from 16-bit fixed point and floating point

Data Organisation

Neural Decoders require ISIs with a fixed sampling rate (f_s) to decode the behaviour outputs. As shown in the Figure 3.8, a fixed window size (w) is defined for sending N-channel ISIs out. If no spikes are detected in the window, ISIs are set to 1, guaranteeing infrequent spike activities. Otherwise, the last ISIs (spike interval between the last two spikes) are retrieved periodically.

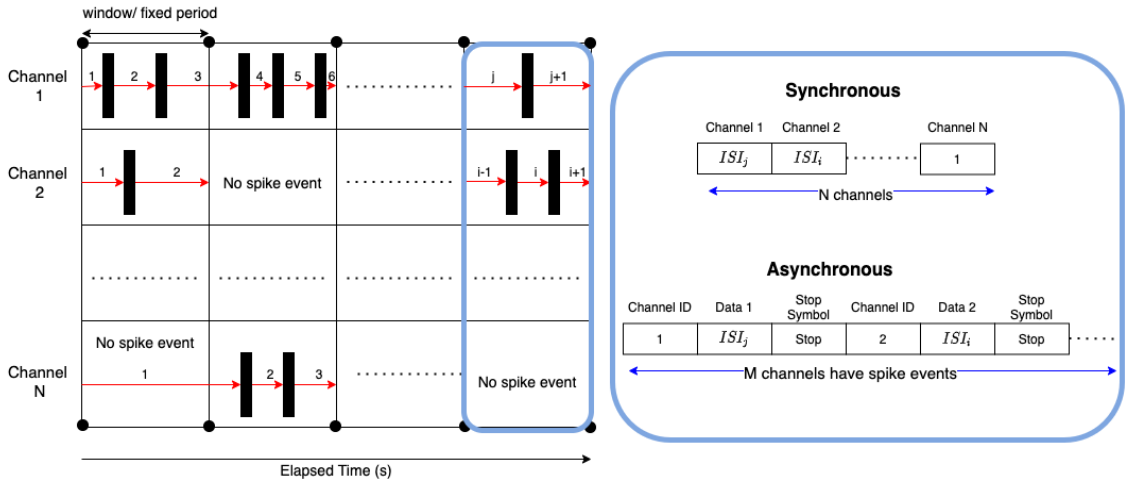

Figure 3.8: The demo for the data transmission and organisation. Black block: spike train. Red arrow with number above: The current number of ISI between two spikes. the ISI values of final window (blue encircled area) are organised in both synchronous and asynchronous way

In Figure 3.8, the organisation of sending data out plays a crucial role in the reduction of data bandwidth. In the synchronous approach, N-channel data are sent simultaneously in each window, where the channel IDs are inherited inside. The data rate (d_{syn}) of synchronous transmission is shown below:

$$f_s = \frac{1}{w} \tag{3.5}$$

$$d_{syn} = f_s * N * l_{syn} \tag{3.6}$$

where l_{syn} represents average codeword length obtained from the pre-trained Huffman dictionary discussed later.

If the spike events are rare most of the time, the asynchronous organisation can further reduce the data rates. Only channels with spike events are transmitted, whereas the channels without spikes are ignored. But in this case, the channel IDs are not encoded implicitly on the data where the decoder does not know the origin of received data. Therefore, a friendly organisation, including channel ID, data, and stop symbol, is presented, in which the stop symbol indicates the transition between each active channel. In this way, it has more average bits on each channel than synchronous one, but fewer channels

are sent. The probability of the stop symbol will always be one third while that of data symbols vary with w and typical data. The stop symbol can be configured to the same or different values for each channel. Usually, it should be better to set the stop codeword fixed, which reduces the number of symbols for higher coding efficiency. The data rate (d_{asyn}) are listed by the following equations :

$$p_{act} = \frac{\sum_{j=1}^{N} p_j}{N} \tag{3.7}$$

$$M = p_{act} * N \tag{3.8}$$

$$d_{asyn} = f_s * M * l_{asyn} \tag{3.9}$$

where p_{act} indicates the average probability of all channels have spike events and M represents the approximate number of channels are active.

The choice of the synchronous or asynchronous organisation depends on the window size and the characteristic of datasets. In tiny windows, it is more likely to have less active channels (small p_{act}), where the asynchronous compression is suitable and vice versa. But in the meantime, the data rate might boost due to the higher sampling rate.

Data Normalisation and Compression

Both lossless Huffman and delta coding schemes are employed on estimated ISIs, aiming to reduce the bit width of transmitted data. The delta coder is implemented by *diff* function, followed by the Huffman coding to compress the bit usage further.

It is not suggestive to use dynamic ranges of ISIs directly as inputs for Huffman encoding, resulting in worse performance (more bits required in codeword). So the values should be pre-processed to reduce the range of variation. Based on empirical testing, the firing rate between 10 and 100 Hz has nutritional information related to behaviour change.

Therefore, a normalisation function is proposed on ISIs between 0.1 and 0.01, corresponding to 10 and 100 Hz firing rates, respectively. The flow of the normalisation process is demonstrated in Figure 3.9. Given an ISI output, a spike event is determined. If no event occurs (ISI == 1), the ISI value is unchanged. Inspired by the uniform quantisation,

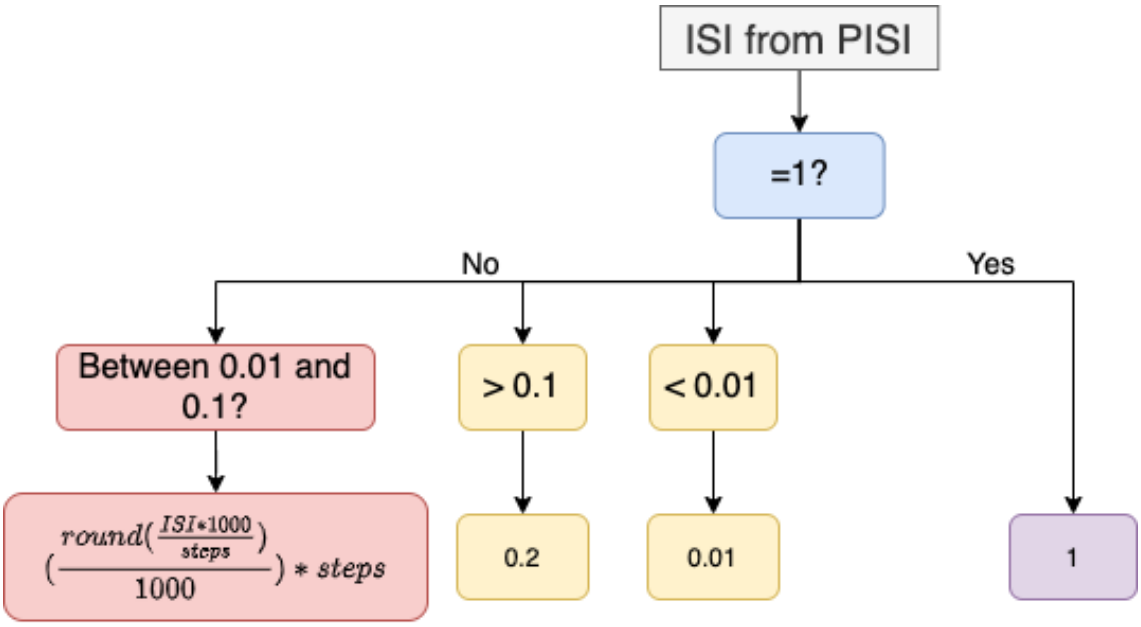

Figure 3.9: The flow of normalisation process. Purple: 1 for no spike events; Yellow: fixed values for two special cases (significantly small and large); Red: Normalised region

when values are inside the range, ISIs are normalised by rounding the division between ISIs and constant step interval and then multiplying the interval. Multiplying an additional 1000 can keep three fractional numbers of ISIs, and dividing by 1000 after rounding can convert values back. Otherwise, the fixed numbers 0.2 (ISI $>$ 0.1) and 0.01 (ISI $<$ 0.01) are configured, which represents 5 Hz and 100 Hz in firing rate, respectively. These fixed numbers can be tuned later to get better decoding performance. In this study, the step interval is set to eight to guarantee 12 different outputs inside the normalisation range and three unique outputs, 15 in total.

The training datasets are prepared to train the Huffman dictionary and evaluate the coding efficiency of both delta and Huffman encoding. Firstly, ISIs are estimated and then normalised. The histogram approach with normalised probability is used to calculate the probability distribution of 15 outputs based on their occurrence frequency. Then, given outputs and their corresponding probability, the *huffmandict* function is applied to generate the dictionary tree with optimised binary codewords. Using the pre-trained Huffman dictionary, normalised ISIs are encoded and decoded back for neural decoding.

Neural Decoding

Three non-casual decoding algorithms are benchmarked on the PISI normalised estimates, including Wiener filter (WF), Wiener Cascade filter (WCF), Kalman filter (KF). The behaviour outputs (x and y recorded cursor coordinate) are initially sampled at 250 Hz. Using an anti-aliasing FIR low-pass filter on the data, it is resampled to the f_s to match the sampling rate of estimated ISIs.

$$p(z) = p_1 z^i + p_2 z^{i-1} + .. + p_n z \tag{3.10}$$

In this study, all parameters of decoders are trained and estimated by training datasets. According to Wu's study, it adds l_2 regularisation to the state transition and parameters of the KF [28]. An optimal past tap of the WF is determined under the assumption of known stationary signal and additive noise. WCF is the nonlinear version of WF to target the nonlinear dynamic neural system, where an i^{th} order polynomial is fitted on the WF predicted outputs (z) shown in Equation 3.10.

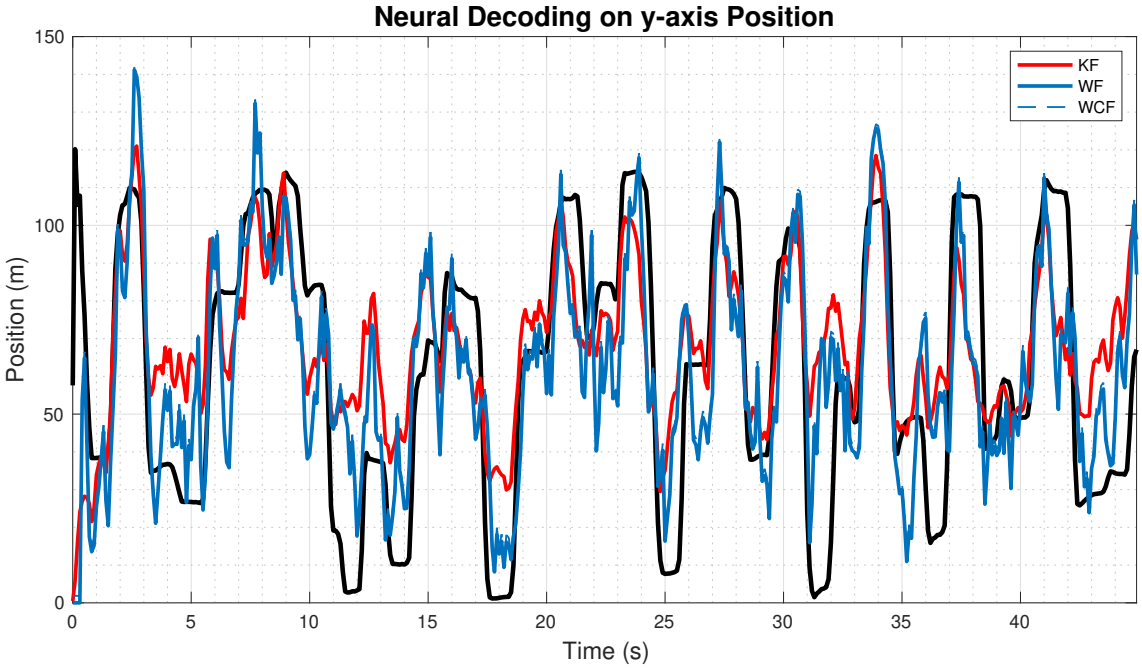

Figure 3.10: The prediction of y-axis coordinates from three types of decoder. Black line indicates the recorded groundtruth y-axis cursor coordinates

In Figure 3.10, y-axis cursor coordinates are predicted from the tested ISIs data. All decoders can successfully predict the change of ground-truth coordination with initial delays. Compared with WF, KF fails to capture the dynamic change of magnitudes. The polynomial function implemented in WCF is not significant on the WF outputs, where the overlapped waveform is observed.

3.5 Hardware Implementation

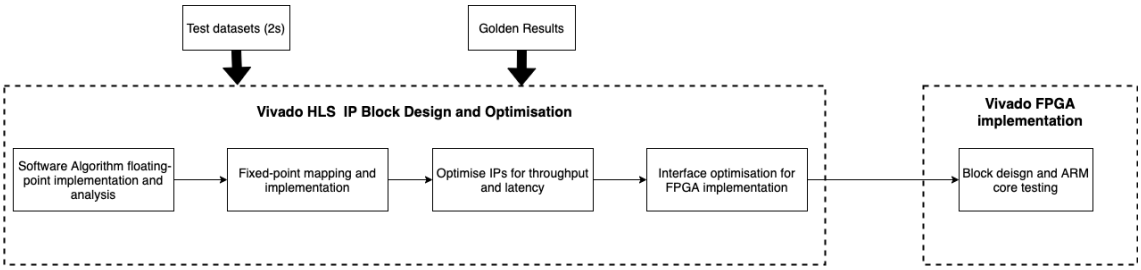

Figure 3.11: The flow of hardware design and implementation in Xilinx Zedboard

This section firstly creates custom IP blocks for the PISI and binning algorithm in Vivado_HLS high-level synthesis tool. By speeding up the testing process, only a 2-second spike train with 96 channels and corresponding golden outputs generated from MATLAB are provided to test the performance of each function. Then, all variables are mapped to the fixed-point representation to reduce latency and resources. Interface and throughput optimisation, are investigated to communicate with the Xilinx ARM core and improve processing speed. Finally, a board design is created and synthesised to interface the custom block with the Zynq7 processing ARM core in Vivado and programmed in Xilinx SDK tool. The details of binning can be seen in Appendix B.

3.5.1 High-level Synthesis

Floating-point Implementation

A test bench is written in the floating-point operation to fetch inputs, receive outputs, and compare the estimated outputs with golden results. The *SpksTime* (t) is the elapsed time

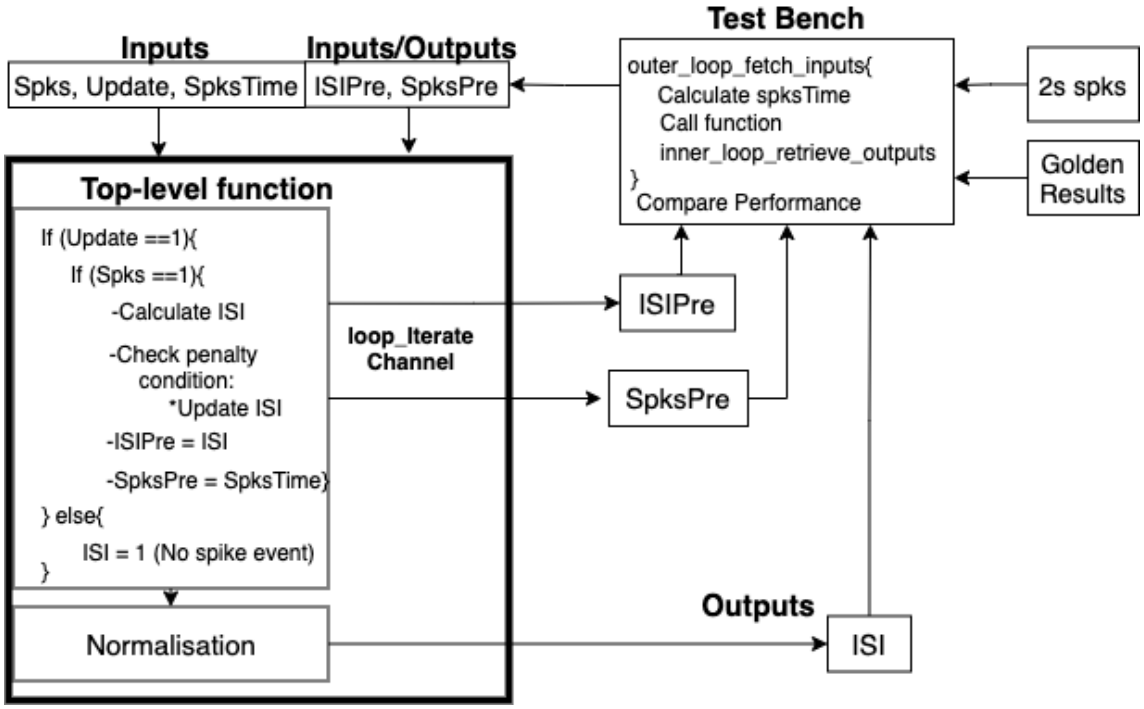

Figure 3.12: The high-level interface between test bench and PISI function in Vivado HLS

calculated from Equation 3.11. The sampling rate of the spike train is 24k, and samples are incremented in the for loop. Based on *Spks*, the *Update* signal indicates if a spike event occurs ($Spks == 1$ for detected spike event) in each fixed window. In the beginning, both *ISIPre* and *SpksPre* are initialised to 0. Then, all inputs and previous outputs (*ISIPre* and *SpksPre*) are passed to the top-level PISI function.

$$t = t_{initial} + \frac{samples}{samples\_rate} \tag{3.11}$$

Inside the function, the occurrence of spike events is detected in the current period from *Update* signal. If *Update* is equal to 0, *ISI* is set to 1 to indicate no spike event. Otherwise, the new *ISI* is calculated and penalised if satisfying the penalty condition, and *ISIPre* and *SpksPre* are then updated. By dealing with multiple spikes in the same period, *ISI* and previous outputs are kept updated for each coming spike. All *ISI* outputs are normalised before sending back to the test bench. Since no dependence exists between

each channel, the array of inputs are iterated in a for loop, named *loop_InterateChannel*, to generate multiple-channel ISIs, where the size depends on the number of channels (96 in this case).

In the test bench, multiple-channel ISIs are retrieved in every fixed period, and *Update* signal of each channel is set to 0. Finally, the retrieved outputs are compared with the floating-point golden results in terms of precision accuracy.

Top-level Data Interface

AXI-Lite provides point-to-point bidirectional communication interface between a IP core and Xilinx ARM processor core. Besides, it supports multiple address ranges for the array but the limitation is that it only supports 32-bit integer in the data transfer bus [33]. This will have a negative impact on the overall performance since six 32-bit integer arrays are required in the top level function. All fractional points of *ISI* are lost in the meantime.

To deal with this issue, *SpksTime* is firstly multiplied by 1000 to preserve three

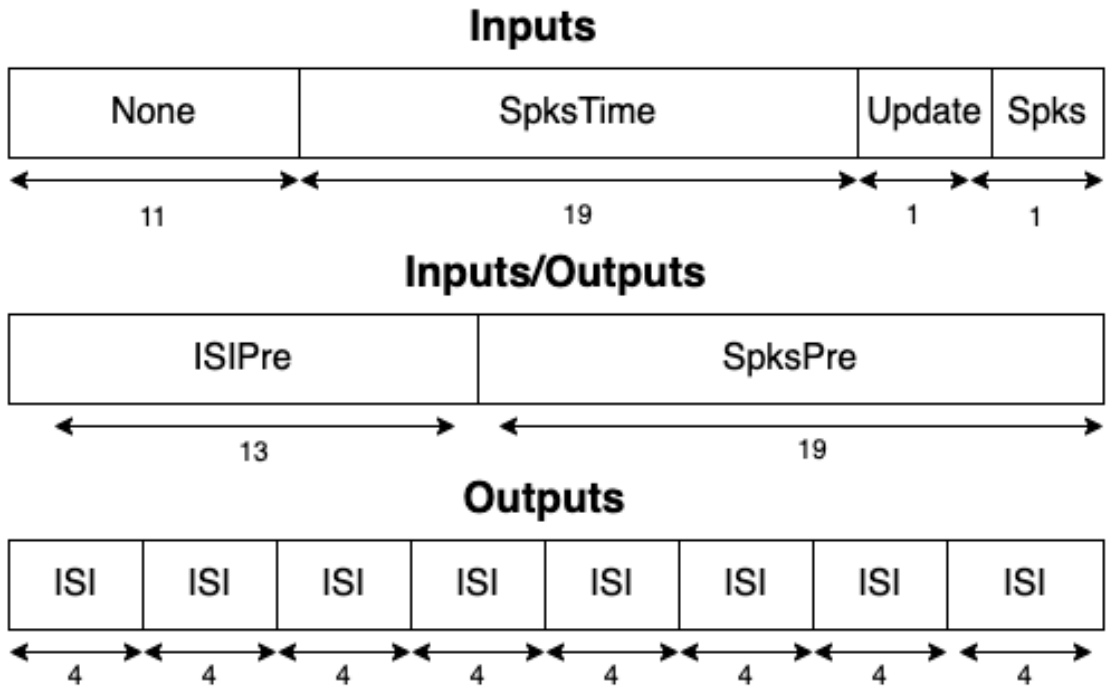

Figure 3.13: Method for packing fixed-point variables in the top-level arguments. Number under arrow represents bit utilisation. None represents empty/spare area

decimal points of *ISI*. Furthermore, additional optimisation is implemented to improve the communication speed and resource utilisation: bit packing described in Figure 3.13. The bit configuration of each variable will be discussed in detail in the next section.

The bit packing process includes two stages, packing and unpacking. In the beginning, three variables (Update, Spike, SpksTime) are packed into the 32-bit *Inputs* by left shifting and masking into non-overlapping bit location before transmitting to the PISI function. In the PISI hardware function, *Inputs* is unpacked into three corresponding fixed-point variables by masking first and then right shifting data out. The proper fixed-point typecasting (ap_uint for unsigned fixed integer, ap_ufixed for unsigned fixed integer with fraction points) are applied to these variables. For *Inputs/Outputs* variable, if a new spike is detected, the previous ISI and spksTime are unpacked and then updated, finally followed by packing them back into *Inputs/Outputs*. As shown in Table 3.2, after normalisation with step interval eight, 15 distinct values can be represented by the values between 1 and 15 using four bits. Therefore, 8-channel ISIs are packed into a 32-bit *Outputs*, which reduces the array size to N/8.

Table 3.2: The conversion of normalised values

Normalised ISI	Value
0.01 $\geq$ISI$\leq$ 0.1	1 - 12
ISI $<$ 0.01	13
ISI $>$ 0.1	14
ISI == 1	15

It can be seen that the bit-packing can reduce from originally six arrays to only three 32-bit arrays by utilising all the bits effectively. It avoids wasting extra resources for interface and minimise the transmission cost between processor and IP core by transferring fewer inputs and smaller array lengths of *Outputs* in the AXI-Lite bus.

Fixed-point Optimisation

All variables from floating-point implementation are mapped to the correct fixed-point equivalent to prevent overflow and preserve precision. Any multiplication with fixed values

represented by the power of 2 can be simplified by the bit shifting. <Word length, Integer> is used to represent the bit configuration. As discussed, the datasets last around 6 mins, thereby choosing $< 19, 0 >$ ($2^{19} * 0.001 = 542s$) to represent *spksTime*. Both *Update* and *spks* signal are configured by $< 1, 1 >$ to represent 1 and 0. In our datasets, most ISIs are smaller than 1 second, which uses $< 12, 10 >$ ($2^{10} * 0.001 = 1.024s$) to represent. The *SpksPre* and *ISIPre* are in the same configuration as *SpksTime* and *ISI*. In this study, p (0.03125) and T (32) are chosen.

Referring to the PISI in Algorithm 2, both ISI and ISIPre are divided by 1000 in the conditional statement due to previous multiplication in *spksTime*. By multiplying 1000 in 3.12, the multiplication with p ($2^{-5} = 0.03125$) and $\frac{T}{1000}$ ($\frac{32}{1000} \approx 0.03125$) can be approximately replaced by right shifting (3.13,3.14,3.15). The rounding in the normalisation function is substituted by right shifting in 3.16, which produces the 4-bit values between 1 and 15.

$$\frac{ISIPre}{1000} - \frac{ISI}{1000} >= \frac{ISI}{1000} * \frac{ISIPre}{1000} * T \tag{3.12}$$

$$ISIPre - ISI >= ISIPre * ISI * \frac{T}{1000} \tag{3.13}$$

$$ISI * ISIPre * \frac{T}{1000} = (ISI * ISIPre) >> 5 \tag{3.14}$$

$$ISIPre * (1 - p) - ISI * p = ISIPre - ISIPre >> 5 - ISI >> 5 \tag{3.15}$$

$$round(\frac{ISI}{8}) = ISI >> 3 \tag{3.16}$$

In Equation 3.14, one DSP48E unit, supported by FPGA, is utilised for the multiplication between 12-bit ISI and ISIPre. One solution to reduce this extra DSP is to use smaller bit width (< 8 bits) in the penalty condition, where two additional variables are declared. These two variables are measured by dividing both ISI and ISIPre by 32 (>>5) and then typecasting to $< 7, 5 >$. It is crucial to notice that the right shifting by 5 in 3.14 is cancelled by multiplying 32 (<<5) due to new variables. With this approach, 7-bit variables are multiplied in the condition statement without affecting the performance.

The fixed-point version of the proposed algorithm reduces FPGA resources, consumes lower power and provides higher throughput (less latency). Some precision is lost

since only three fractional bits of ISI are preserved, but it still maintains almost the same performance as the floating-point operation. However, overflow may happen if ISIs are larger than 1, but normalisation will solve this problem by setting them to constant values.

Throughput Optimisation

The PISI function exists no independence between channels. Therefore, by using the pipeline approach, a parallel structure can be implemented to replace the originally sequential hardware structure. To fully utilise the benefits of pipeline, cycling partition the arrays in smaller size and unrolling the for loop according to the size of partitioned array are investigated.

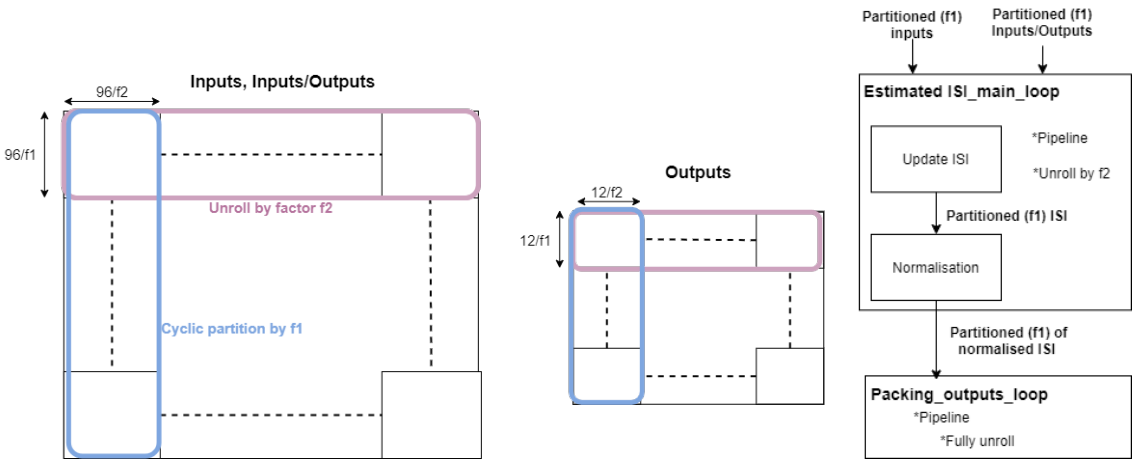

Figure 3.14: The demonstration of unrolling and cyclic partitioning process in three types of variables, where f1,f2 represents cyclic factor and unroll factor respectively. Array size is 96 for *Inputs* and *Inputs/Outputs* and 12 (96/8) for *Outputs*. The throughput optimisation flow can be seen in the right hand side.

Figure 3.14 shows the flow for throughput optimisation. According to an initial analysis from the synthesis report, most of the latency came from the **Estimated_ISI_main_loop**. By maximising the resource utilisation, the variable arrays are partitioned in a cyclic approach with factor f1, which reuses the array with size $\frac{Array\ Size}{f1}$ at each clock cycle. The unroll factor f2 should be the same as the f1, ensuring all resources are applied for parallel execution since unrolling the loop enables partitioned channels to operate simultaneously. Then, the main loop generates partitioned *ISI* array

to the normalisation function, where normalised outputs are partitioned (f1 = 8) as inputs to **Packing_outputs_loop**. Inside this loop, it is fully unrolled to pack eight channels in parallel. Finally, a pipeline with initial interval one is applied to both loops, which processes new inputs every clock cycle.

It is always a trade-off between latency and power consumption in the throughput optimisation. By increasing the throughput, many power-hungry resources are needed for the parallel architecture. Therefore, depending on the requirement of specific BMI applications, the right balance is found between processing speed and resource utilisation by setting factors f1 and f2.

3.5.2 FPGA Testing

Vivado Block Diagram

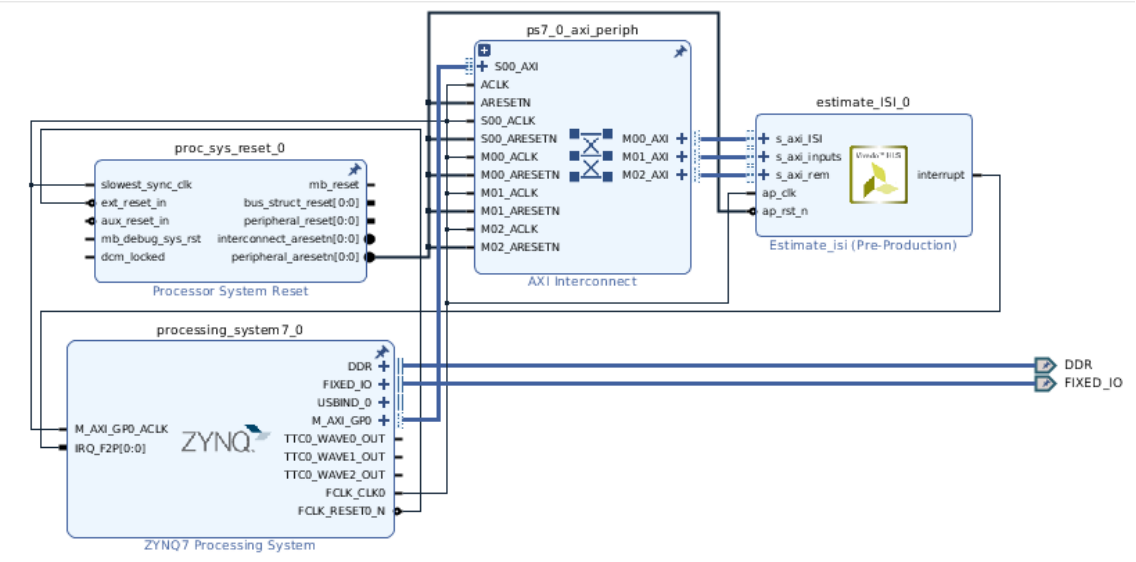

Figure 3.15: Vivado block design of custom hardware IP and processing system

As shown in Figure 3.15, the custom IP - EstimateISI is being controlled by the Zynq7 processing system, providing two ARM A9 processor cores. The AXI Interconnect and Processor System Reset are configured to interface and control the AXI-Lite communication system. The interrupt signal is enabled and connected between IP and CPU to

manage interrupted events: 1) Indicate the Ready signal for retrieving output from IP block; 2) Free the CPU to perform other tasks while the IP block is running. The clock frequency of the CPU is set to 100 MHz at default.

The block design is then verified, synthesised and implemented. Several reports, like timing violation, power consumption, and area utilisation, are analysed and discussed. The bitstream of the tested board is generated for arm core programming and debugging.

Programming and Debugging

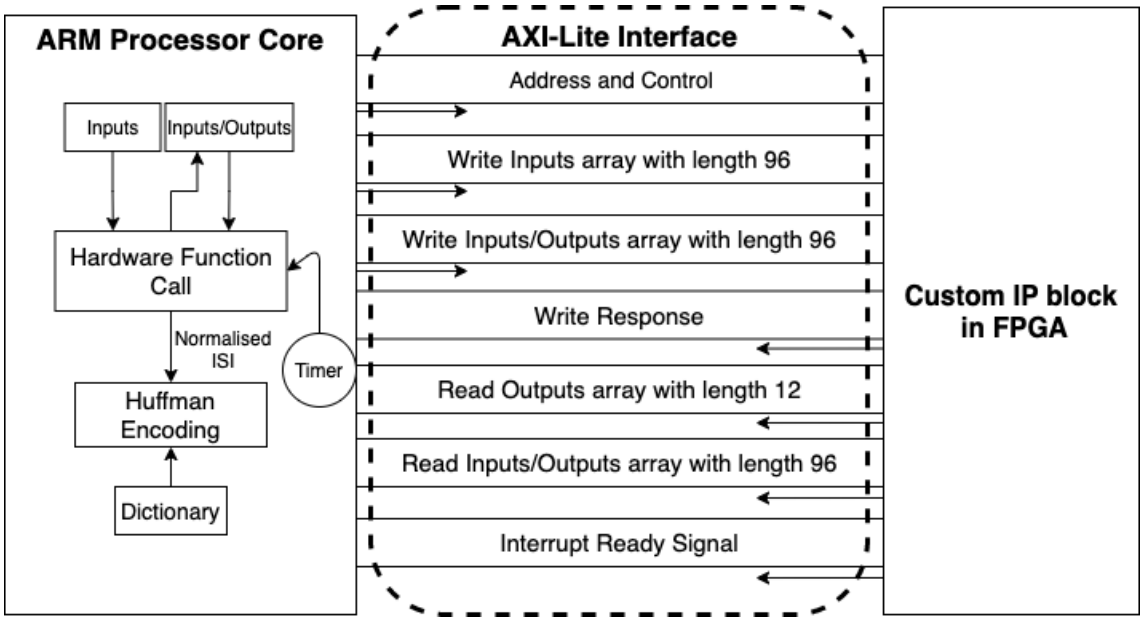

Figure 3.16: The top-level interface between Arm Processor Core and Custom IP block in Xilinx Zedboard

Two types of custom IP blocks (PISI function) are tested, including sequential and parallel architecture. Figure 3.16 demonstrates the top-level flow between Arm core (Master) and IP core (Slave). Using bit-packing, 96-channel spike train, spike time and update signals are packed to *Inputs*. The previous spike time and ISI are initialised to 0 in the beginning for *Inputs/Outputs*. Then, these arrays are passed to the custom IP block by multiple hardware function calls, and both *Outputs* and *Inputs/Outputs* are waited for the Ready signal to retrieve. This updated *Inputs/Outputs* are fetched back to the

IP core in the next run. Then, in each fixed period, *Outputs* is unpacked sequentially to obtain the 96-channel normalised ISIs. However, the outputs are required to retrieve in an interleaving way for the parallel architecture, which depends on how the arrays are partitioned.

The pre-trained Huffman dictionary finally encodes these multi-channel outputs to reduce the bit lengths using the switch statement. Besides, the processing speed of both architectures is measured by the provided timer function. The multi-channel results, corresponding encoded codewords, and processing speed of hardware function are printed and debugged in the UART port.

Chapter 4

Results and Discussion

In this chapter, the proposed algorithm has been evaluated to demonstrate its low power, small area utilisation, lower data rates, and compromised decoding performance. Each phase of the software implementation on MATLAB are analysed and discussed. In the hardware synthesis and FPGA testing, the overall hardware implementation of binning and PISI are compared in detail.

4.1 Software-based Evaluation

4.1.1 Performance Testing on Synthetic Data

The choice of parameters P and T are firstly investigated that yield the most accurate estimation on the synthetic data. The mean integrated squared error (MISE) is a standard metric to evaluate the goodness-of-fit of density estimation. It is measured by the difference between the ground-truth firing rate and the estimated one listed in 4.1.

$$MISE = \Delta t \sum \mathbb{E}[\hat{\lambda}(t) - \lambda(t)]^2 \tag{4.1}$$

Each rate function, including sine, sawtooth, Gaussian-damped sine, and chirp, along with 100 repetitions of spike trains are generated from the non-Gaussian model. Also, different frequencies and intensity of spike activities are investigated.

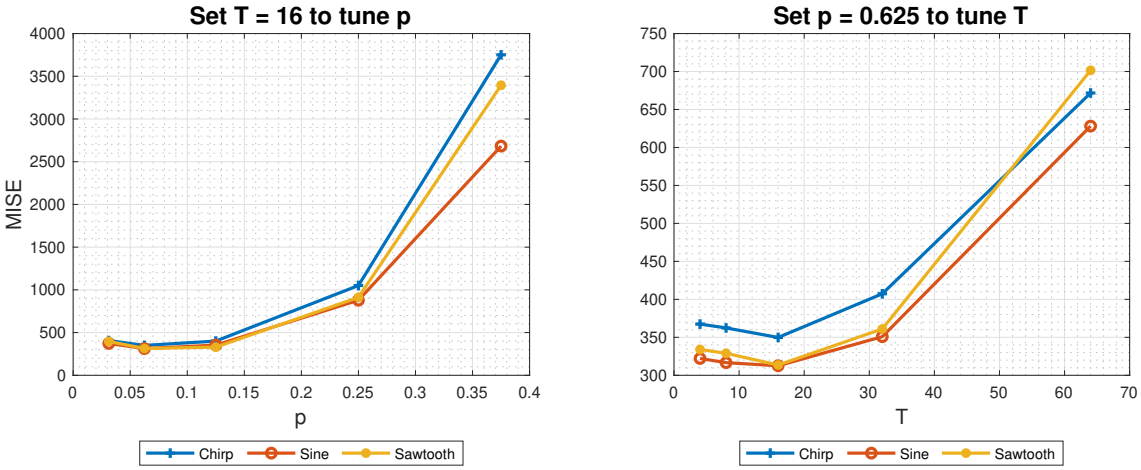

Figure 4.1: Average MISE comparison under different values of p (left) and T (right).

The hyperparameter tuning is performed on three types of rate function with medium intensity and frequency. The choice of parameter values should be the power of 2, which can be optimised later by bit-shifting in hardware. Figure 4.1 illustrates the average MISE from the 100 repetitions of each rate function. Since there are two parameters, we firstly set T to 16 and vary p between 0.03125 and 0.375. As p increases to 0.25, the estimation becomes unreliable, which reduces the penalised effects on the significant firing rates. The optimal p (0.0625) is found on the elbow of the curve. It is then utilised to tune T between 4 and 64, where a rapid increase of MISE is observed on the growth of

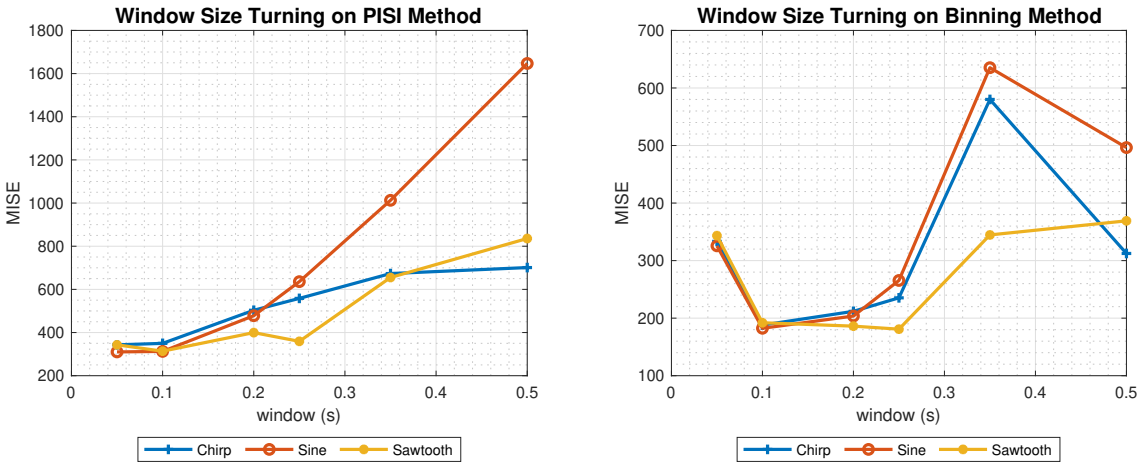

Figure 4.2: MISE comparison under window sizes on PISI (left) and Binning (right)

T. If the threshold is larger than 32, it will be hard to implement penalty on the certain firing rates. To conclude, it is safe to set p = 0.625 and T = 16 to guarantee the smallest MISE.

The proposed method is evaluated and compared with established methods, including GKS, BAKs and binning. Since binning and PISI depends on the window size, its relationship with average MISE are shown in Figure 4.2, where the window of 0.1 s produces the smallest MISE. To measure MISE, the wave functions are resampled according to the window size.

The spike activity may vary differently in real-world data, either on number (intensity) or on temporal fluctuation (frequency). Therefore, all the algorithms are applied on four rate functions with two types of intensity (low, high) and frequency (low, high) as shown in Figure 4.3. On average, the kernel-based methods outperform PISI and bin-

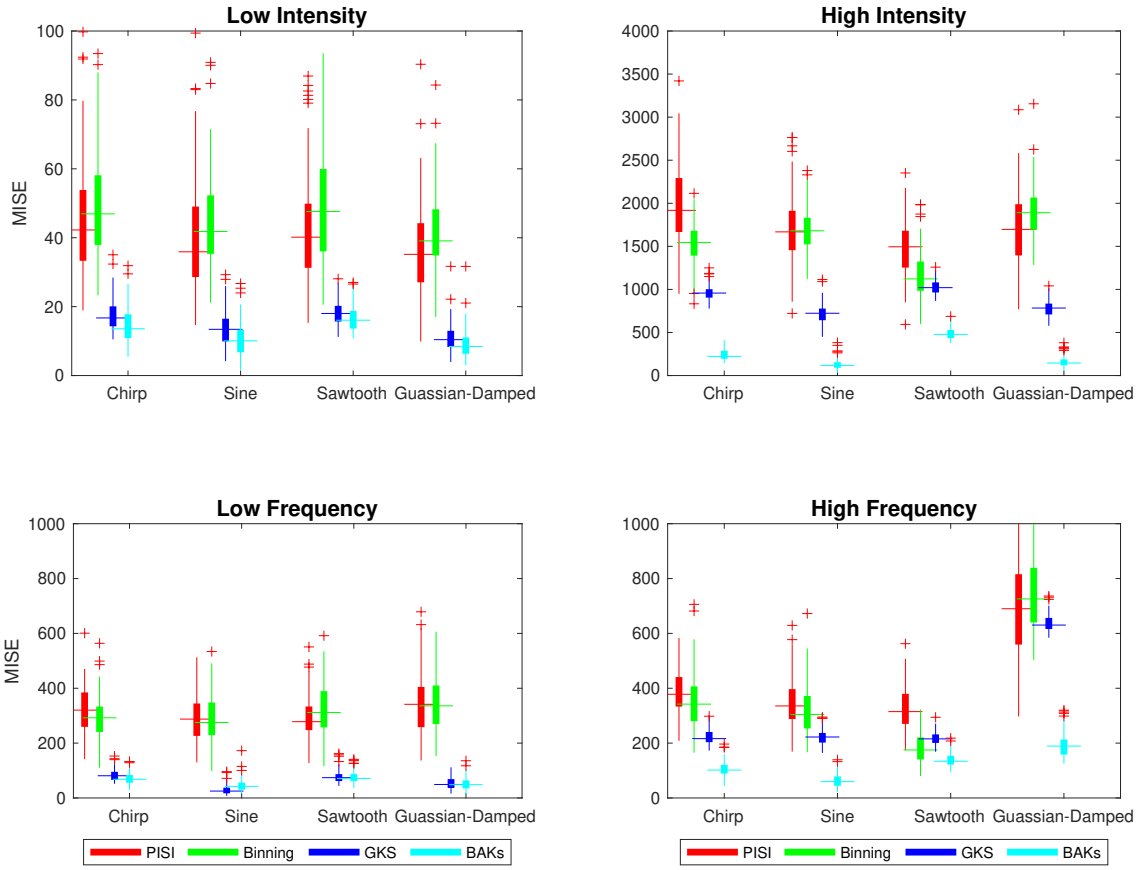

Figure 4.3: MISE comparison under different values of intensity and frequency

ning methods in all scenarios, especially in rapidly changing spike activities. Significant performance improvements are achieved by the algorithm complexity and high data rates.

On the contrary, the performances vary considerably in the single-trial PISI and binning estimates due to their compressed data representation. Compared with the traditional binning method, PISI performs well in the low intensity and frequency while slightly underperforming when the number of spikes and temporal fluctuation of activities grows. It suggests that PISI is suitable for estimating continuous and discontinuous rate function with inactive spike dynamics.

4.1.2 Data Compression

300-second datasets (96-channel spikes) are used to train both delta and Huffman coder. The effectiveness of normalisation is investigated in the Huffman coding. The performance metrics, such as average codeword length (l_{avg}) and entropy (H) are utilised to assess the efficiency of coding with the following formulas:

$$H(x) = -\sum_{j=1}^{n} p(x_j) log(p(x_j)) \tag{4.2}$$

$$l_{avg} = \sum_{j=1}^{n} p_j l_j \tag{4.3}$$

The p_j, l_j, x_j correspond to probability of codeword, length for each codeword and input symbol j respectively. Higher l_{avg} and H indicates ineffective coding performance.

Delta Encoding Scheme

The delta coder, the differences between sequential values (ISIs in this case), is implemented and analysed. In the beginning, the probability distribution of inputs is calculated and plotted using *hist* function. As seen in Figure 4.4, the PISI estimated and values of delta-sampling span across all data points with minimal probability. The most frequent probability of the symbol drop by half after the implementation of the delta coder. In Table 4.1, when implementing delta-sampling on ISIs, it leads to worse coding performance (higher H) since the difference of less correlated data produces more unique values and

decreases the probability of the most frequent value. When training by Huffman encoding, the delta coder produces longer l_{avg}.

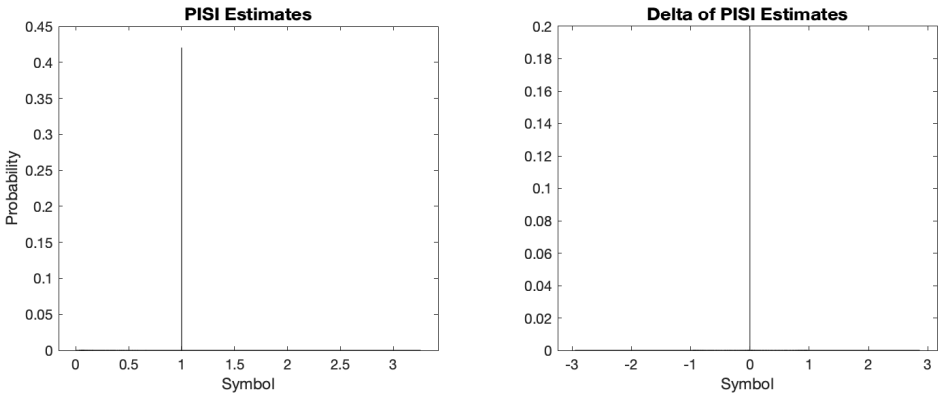

Figure 4.4: The histogram of ISIs and delta ISIs. The bin widths are set to 0.0001 to obtain the probability distribution of all unique values.

Huffman Encoding Scheme

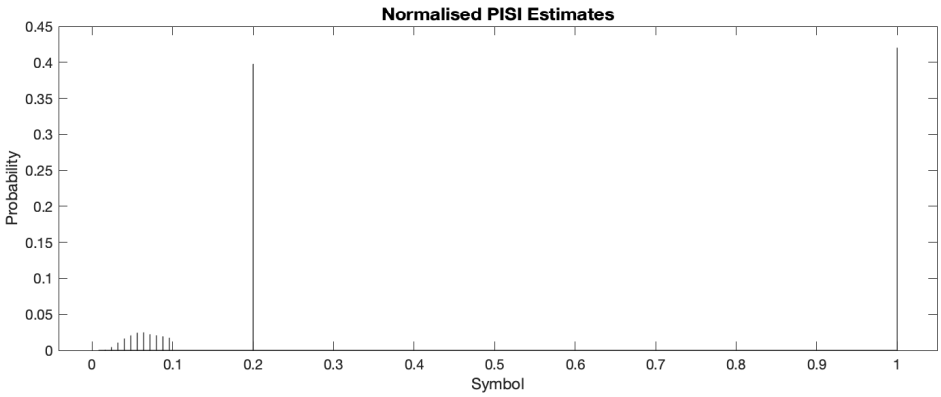

Figure 4.5: The histogram of normalised ISIs

The datasets are used to pre-train the Huffman dictionary, where a codeword represents each symbol. If applying ISIs directly, the l_{avg} is comparatively high (8.24). The critical solution is to reduce the dynamic range to only 15 unique symbols by normalisation. It can be observed in Figure 4.5 that all data points between 0.01 and 0.1 are normalised to the small region with apparent probabilities. The values outside the ranges

Table 4.1: Performance analysis for coding efficiency

Type	Number of Unique Values	H	l_{avg}
Delta-sampling	20339	8.249	11.29
ISIs	10378	5.687	8.24
Normalised ISIs	15	1.45	2.18

are set to fixed values. Finally, the Huffman dictionary is trained by inputting 15 symbols and their corresponding probabilities. After normalisation, it improves the coding efficiency by around four times on both H and l_{avg}.

Figure 4.6 demonstrates the effects of normalisation on the reciprocal of PISI estimates (firing rate), noticing that the rates less than 10 Hz are set to 5 Hz. The interesting range of firing rate between 10 and 100 Hz are nearly overlapped with original estimates. Overall, the normalisation not only improves the coding performance but also preserves the detail of estimates.

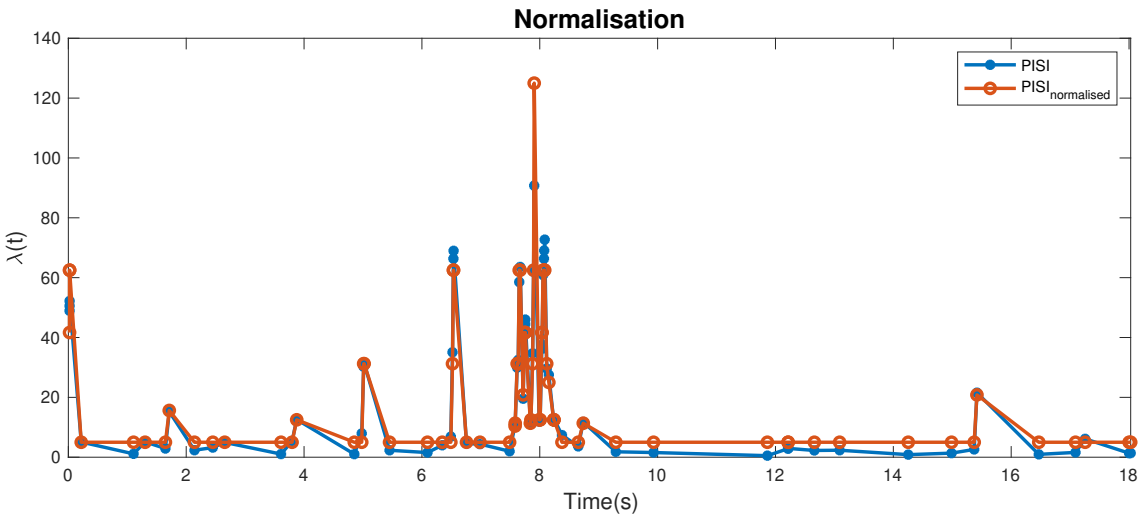

Figure 4.6: Normalisation of firing rate from the reciprocal of estimated ISIs.

4.1.3 Synchronous and Asynchronous Organisation

The data rate is used to evaluate the two organisation schemes. The Huffman dictionary codes both schemes for data compression. As discussed, the FPGA returns the 4-bit

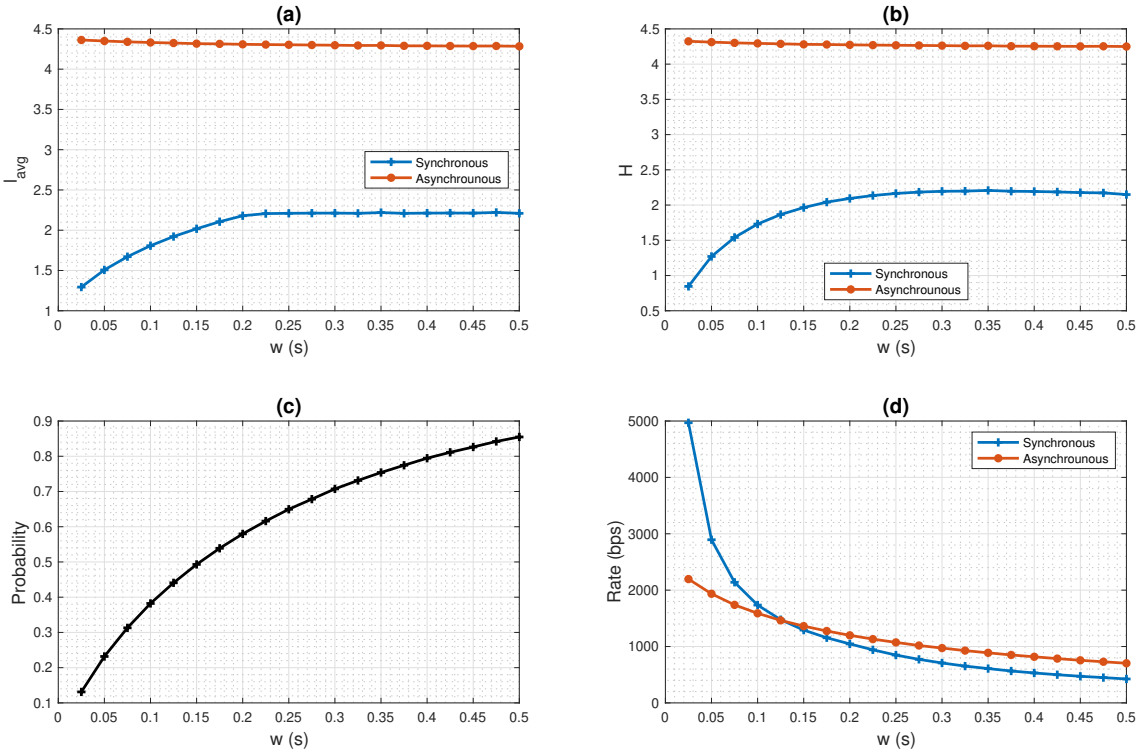

Figure 4.7: The effect of window size on the performance of synchronous and asynchronous data management. (a): Average codeword length. (b): Entropy. (c): Mean Probability of Channels have Spike Events (p_{act}). (d): Data Rate with unit bps (bits per second)

normalised values to represent 15 distinct ISIs. Therefore, in the asynchronous dictionary training, the symbols between 1 and 15 are overlapped to represent both data and channel ID. The number of symbols is channel size + stop symbol (97 in this case), where the stop symbol is fixed to 97 uniquely.

The window size (w) is varied between 0.025 and 0.5 to evaluate the corresponding performance, as seen in Figure 4.7. As the window size increases, it is less likely to have channels with spike events, where the logarithmic growth of p_{act} is observed. In this case, both the entropy and l_{avg} of synchronous compression increase slightly and stabilise until setting the window to 0.2 seconds. It suggests that further increases of p_{act} has fewer impacts on the coding efficiency. On the other hand, the coding performance of asynchronous compression remains unchanged. The reason is that three symbols are always needed, including channel id, data and stop, in each active channel, where the

probability of the most frequent stop symbol is always one third, and that of other symbols vary slightly on w.

Overall, the data bandwidth reduces exponentially with w. Interestingly, asynchronous compression outperforms the synchronous one in a tiny window (w $\leq$ 0.125), while the synchronous one starts to perform slightly better in a larger window. The original data rate from spike detection (24k*96*16 $\approx$ 36800 kbps) are reduced to only 1 or 2 kbps after data organisation and compression.

4.1.4 Neural Decoding on Real-world Data

The decoding performance indicates how the frequency of spikes maps to the behaviour outputs. Similar to the research study in [2], the performance is compared by using two commonly used metrics: 1) root-mean-squared error (RMSE) 2) Pearson's correlation coefficient (CC). RMSE represents the average magnitude of decoding error, and CC assesses the linear correlation between predicted and actual kinematics.

$$RMSE = \sqrt{\sum_{j=1}^{N} \frac{(\hat{y}_j - y_j)^2}{N}} \tag{4.4}$$

$$CC = \frac{\sum_{j=1}^{N}(y_j - \bar{y}_j)(\hat{y}_j - \bar{\hat{y}}_j)}{\sqrt{\sum_{j=1}^{N}(y_j - \bar{y}_j)^2}\sqrt{\sum_{j=1}^{N}(\hat{y}_j - \bar{\hat{y}}_j)^2}} \tag{4.5}$$

where y_j and $\hat{y}_j$ denote the decoded and predicted position at samples j respectively and $\bar{y}_j$ and $\bar{\hat{y}}_j$ represent the corresponding mean.

The datasets contain 390 seconds spike train and two-dimensional behaviour outputs (x and y cursor position). We divided 80% data for parameter training (300 s) and

Table 4.2: Hyperparameter search for PISI and decoders

Type	Hyperparameter	Search Range	Optimal Parameter
PISI	w	{0.025,...,0.5}	0.2
KF	α	{0,..,1}	0
WF/WCF	t	{1,..,15}	6
WCF	d	{1,..,6}	2

20% for performance testing (90 s). At first, the window size (w) of PISI is investigated. Since the different decoders have multiple hyper-parameters, such as regularisation (α), past taps (t), and polynomial degrees (d), we split 30% train data for hyperparameter tuning using 5-fold cross-validation approach. The optimal parameters are found using both CC and MSE. The search range is selected according to empirical testing. In Table 4.2, it suggests that no regularisation is required on the KF due to the model simplicity, and d should be small to map linear relationship between PISI estimates and decoded outputs. The best window size is 0.2 seconds for the PISI method.

As discussed, both firing rate (reciprocal of ISI) and ISI can indicate the frequency of spikes so that ISIs are used directly as inputs in PISI. The binning, ISIs and normalised ISIs are selected for the performance comparison. The RMSE and CC are averaged across two state dimensions (x and y direction), and the velocity is measured from the first differentiation of the position outputs [2].

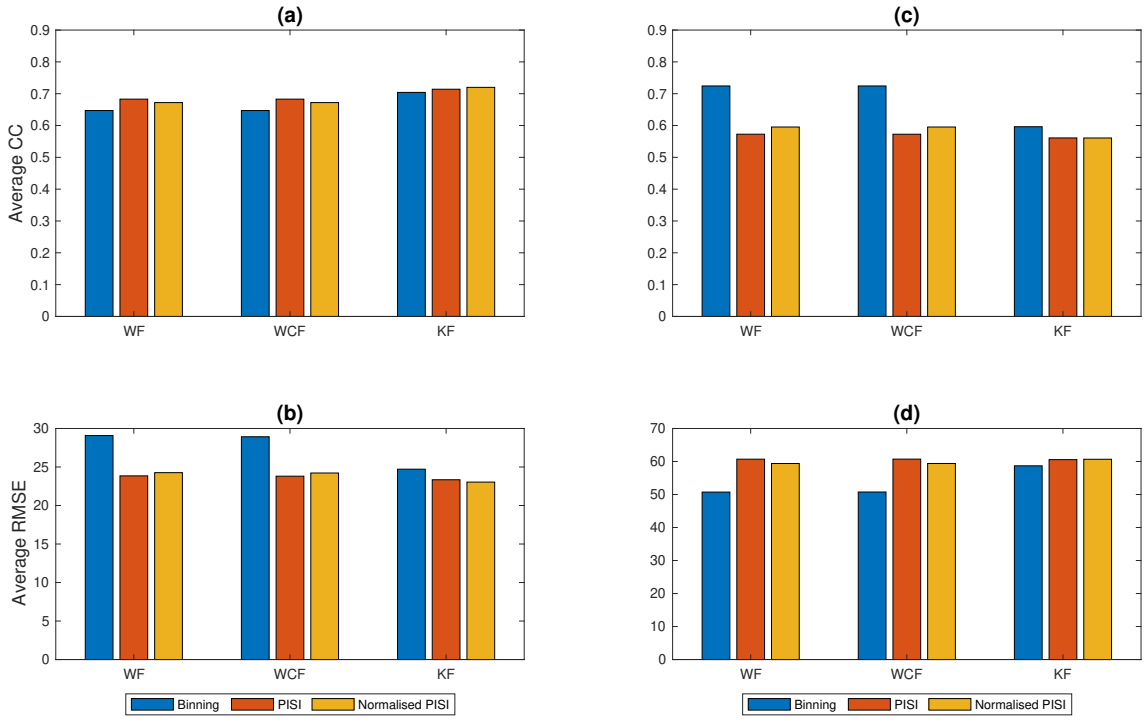

Figure 4.8: Decoding performance comparison across different decoders driven by 3 types of inputs: 1) FR estimated from binning 2) ISIs from PISI 3) normalised ISIs. (a)-(b): Average CC and RMSE according to decoded position. (c)-(d) Average CC and RMSE according to decoded velocity.

Figure 4.8 demonstrates the average CC and RMSE on both position and velocity. KF yields the highest average decoding performance on position outputs (RMSE $\approx$ 25 and CC $\approx$ 0.7), while WCF delivers the highest on velocity outputs (RMSE $\approx$ 55 and CC $\approx$ 0.63). It suggests that decoding performance drops when using the velocity to train the decoders. In the choice of inputs, ISIs and normalised ISIs achieve slightly better performance than the binning on the prediction of position, while the binning generally perform better on velocity data. Surprisingly, using the normalised ISIs maintains similar performance as original ISIs on both velocity and position. Overall, CC in all cases are round 0.65 and RMSE are around 40, indicating good decoding performance using both binning and PISI methods.

4.2 Hardware-based Evaluation

Both binning and the proposed algorithm are implemented and compared in Xilinx Zedboard. The fixed-point optimisation, throughput optimisation, power consumption and processing speed are tested. For fast testing, the 2-second spike (96-channel) datasets and corresponding golden outputs are given. Table 4.3 demonstrate final hardware performance for both binning and PISI. All results are discussed and analysed in detail.

Table 4.3: Fixed-point PISI and binning hardware performance

Type	Latency	BRAM_18K	FF	LUT	Power(mW)	Speed(us)
PISI	400	6	436	664	21	62.88
BIN	620	7	429	699	23	64

4.2.1 Fixed-point Optimisation

As discussed, the floating-point operation occupies many resources on costly arithmetics, resulting in high power consumption and latency. Many complex implementations are optimised by the bit shifting and dynamic bit lengths in the fixed-point conversion, while the precision of results is also maintained. Table 4.4 presents all corresponding results.

In floating-point, the normalisation rounds ISIs to the nearest integers. In contrast,

Table 4.4: Estimated resource, performance and precision of floating-point and fixed-point operation. A-B latency represents the min (A) and max (B)

Type	Performance	Resource				Precision
	Latency	BRAM_18K	DSP48E	FF	LUT	Accuracy
PISI Float	289-3745	11	14	2345	4758	100
PISI Fix	350-446	6	0	436	664	98.75
BIN Float	385-4129	6	0	4179	5742	100
BIN Fix	555-675	7	0	429	699	100

the rounding is replaced by the bit-shifting in the fixed-point, where some precision (1.25%) are lost. On the other hand, binning preserves all the accuracy.

The throughput performance is varied due to the conditional statement. The minimum latency of floating-point binning (four clock cycles) comes from checking two if conditions to update the count and firing rate in the single channel. In comparison, it takes three cycles in PISI to check the condition statement from the update signal and sets ISI to 1 to indicate no detection of spikes.

The worst-case scenarios are observed if consecutive spikes are detected. Except for load and store operation, PISI requires one clock cycle to calculate ISI, 15 cycles to implement penalty and ten cycles to normalise ISIs. On the contrary, binning needs five cycles to increment counters, and 35 cycles to divide the count values for firing rate estimation and reset counters. When migrating to the fixed-point representation, extra latency are required by the bit-packing process, but the worst-case scenarios are improved by almost eight times. The table demonstrates the final latency for 96 channels.

All arrays of top-level arguments are stored in the BRAMs for data transmission, where two BRAMs are required for a single array. As discussed, the bit-packing process in PISI can reduce the array size to three, resulting in fewer BRAMs, while it reduce only one array for the binning. One extra BRAM is utilised in binning to execute multiple read and write operations on the increment of counters. Due to internal optimisation by Vivado HLS, it is surprising that Extra LUT and FFs replace DSPs in the floating-point binning method, whereas PISI uses 14 DSPs and less number of FFs and LUTs. The fixed point optimisation significantly reduces DSP, FF and LUTs. Like multiplication and division, all

complex operations are optimised away by shifting bits left and right, occupying very few resources. The lower bit widths remove most instances and registers to operate functions and store variables. No DSPs are required on both PISI and binning after optimisation.

Overall, after fixed-point optimisation, the resource utilisation of binning and PISI is almost the same, but the PISI method has higher throughput performance.

4.2.2 Throughput Optimisation

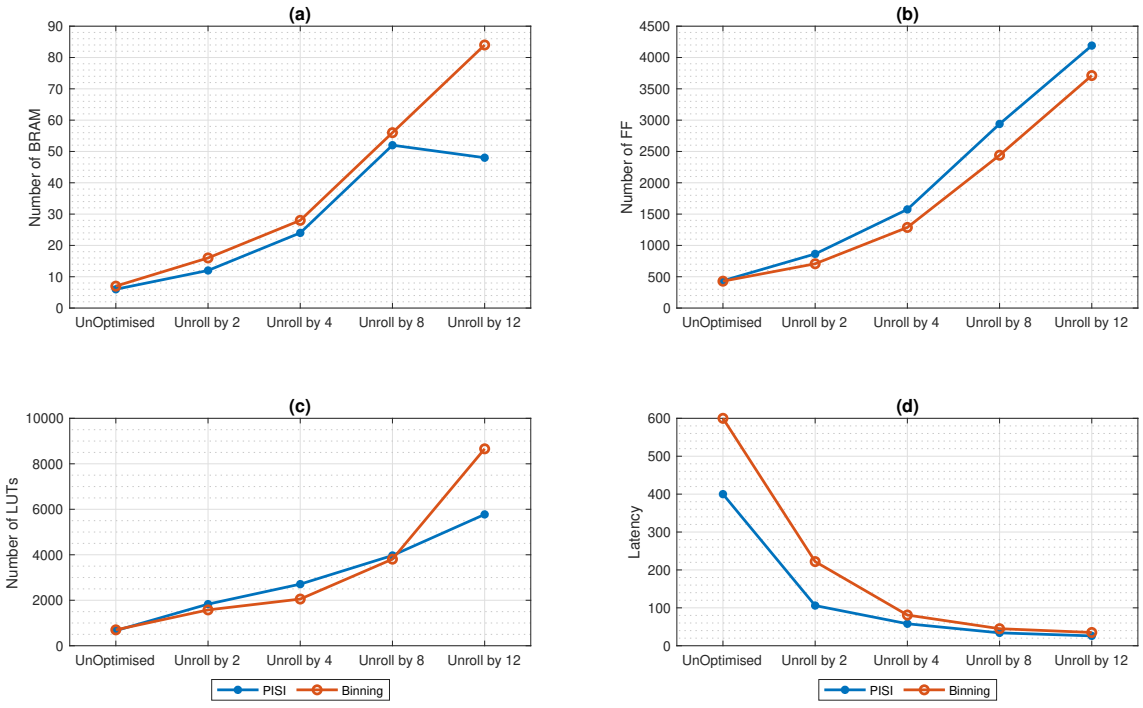

Figure 4.9: The relationship between resource utilisation and throughput performance of the parallel operation (Unroll+Pipeline+Partition). The partition factors are same as unroll factors and all loops are implemented in pipeline. (a) Number of BRAM_18K required (b) Number of FF (c) Number of LUTs (d) Latency improvement

Different parallel architectures are investigated to reduce the overall latency of IP cores. The pipeline is applied in all the loops, and the loop for the bit-packing process is fully unrolled. Since the primary function loop consumes most of the latency and consists of many operations, fully unrolling leads to many resources, and high unroll factors will reach the bottleneck of throughput improvement. Therefore, the range of factors ({2,4,6,8,12}) is selected for unrolling and partitioning. The partition factors are

the same as the unroll factors for effective resource usage. The detailed trade-off between resource utilisation and throughput improvement can be seen in Figure 4.9.

An approximately linear relationship between different types of hardware components and unrolling factors has been observed. It suggests that unrolling by factor f will multiply the resources by f. When increasing the factor from 8 to 12, it is a surprise that the BRAM usage of PISI reduces from 52 to 48. The possible reason is that the balanced factors between the main loop and bit-packing loop (fully unrolled by 12) lead to the efficient storage of partitioned arrays. Compared with binning, PISI utilises slightly more FF and LUTs but fewer BRAMs.

As demonstrated in Figure 4.9 (d), the latency decreases exponentially with the factors. The bottleneck of improvement is reached if further increasing the factor. Also, partitioning the arrays by factor f requires the function to call by f times for each array in the AXI-Lite, resulting in a longer processing time. Therefore, the overall throughput performance is limited by the latency of custom IP and the data communication speed. To conclude, it is highly suggestive to only unroll by factor 2 to improve IP latency without utilising too many hardware resources and degrading the communication performance.

4.2.3 Overall Hardware Performance

Static Timing Violation

The timing constraint should be met to ensure no loss of data accuracy. In the timing report, the critical paths from both PISI and binning IP core do not violate the CPU default clock period (10 ns), which results in zero failing endpoints. Also, it is essential to note that the hardware design from throughput optimisation only needs slightly higher timing to process data, but they are still within the minimum timing period. The approximate clocking periods for PISI and binning are 9.5 and 7 ns, respectively.

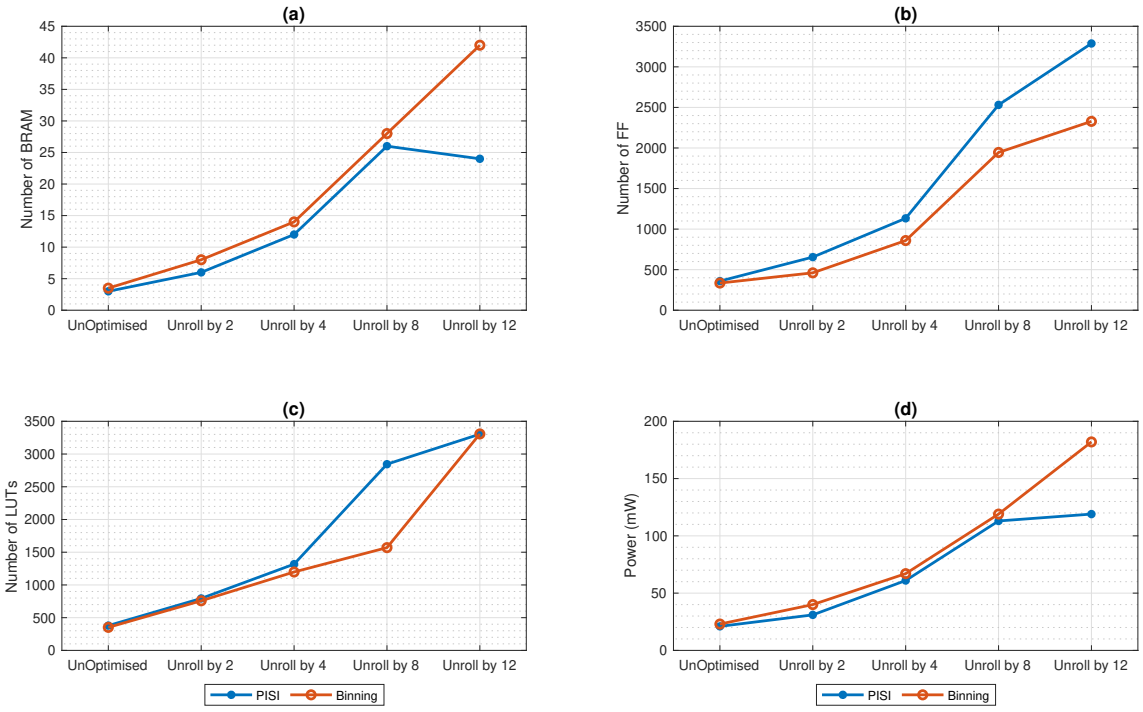

Figure 4.10: The resource utilisation and dynamic power consumption of custom IP core. (a) Number of BRAM_36K (b) Number of FF (c) Number of LUTs (d) On-chip Power Consumption (mW) of IP cores

Power Consumption and Resource Utilisation

When synthesising in Vivado, the actual hardware utilisation after place and route are less than the estimated values from Vivado HLS. It is partly due to the more rigour optimisations implemented by Vivado and very rough estimation done by Vivado HLS. In Vivado, the BRAM_36K is utilised rather than BRAM_18K (half size) in Vivado HLS. No DSPs are required in both algorithms.

Figure 4.10 demonstrates the actual resource utilisation and dynamic power consumption of IP cores. A similar trend of resource (FF, LUT, BRAM) usage with unroll factors are observed in both HLS estimation and actual implementation. Regarding LUT and FF, the gap between the two algorithms slightly grows with the factor number, in which binning employs less. While in BRAM memory usage, binning always consume two more than PISI for the counters variable discussed before. Especially in factor 12, PISI utilises BRAMs effectively, but this is not observed in binning.

The synthesis report provides static and dynamic on-chip power consumption in a hierarchical broad design, including processing system, IP core, AXI interface and reset system. The power consumption is affected by various factors, like environmental setting. Depending on the unroll factor, the total on-chip power consumption with default configuration varies between 1.6 and 1.8 W, around 92% for dynamic power and 8% for device static power. The Zynq processing system generally consumes almost 90% of total power to operate phase-locked loop, memory and I/O interface. In contrast, both IP core and AXI peripheral interface consume less than 1% individually. As seen in Figure 4.10, with the rise of unrolling factors, the dynamic power consumption of the IP block grows linearly. Most of the contribution comes from using logic (FF, LUT) and BRAM in the report. Based on the empirical testing, BRAM consumes more power than logic units. Therefore, even though binning uses less logic, the unoptimised version of PISI (21 mW) still slightly outperforms binning (23 mW).

Processing Speed

By considering the power consumption and bottleneck of latency improvement, only parallel architecture with factor 2 and an unoptimised version of both algorithms are implemented in FPGA. As shown in Equation 4.7, the overall performance of speed (t_{total}) is affected by multiple factors, including execution time in IP core (t_{IP}), transfer overhead (t_{com}) and time for function calls (t_{func}). The timer function provided by SDK only produce t_{total} without the segmentation of each factor. However, given the (f_{clk}) and estimated latency for IP cores, the t_{IP} can be roughly measured by Equation 4.6.

$$t_{IP} = \frac{1}{f_{clk}} * latency \tag{4.6}$$

$$t_{total} = t_{IP} + t_{com} + t_{func} \tag{4.7}$$

Table 4.5 shows the overall performance of processing speed per sample. Surprisingly, although parallel architecture lowers the latency for t_{IP}, the total time increases slightly along with less IP contribution. The possible reason is that both algorithms are

low complexity, contributing a minimal amount of time (less than 10%). On the contrary, the parallel architecture with factor 2 needs to call the function twice to send and receive each array, leading to double t_{func} approximately. Furthermore, since both contributions of t_{func} and t_{com} are unknown, it is hard to improve the processing speed in the AXI-Lite, where an alternate interface approach, like direct memory access, may be needed to enhance t_{com} further. Overall, due to the less latency in PISI, its speed is slightly better than binning.

Table 4.5: Comparison of processing speed per sample. IP Contribution: percentage of t_{IP} in t_{total}

Type	**Estimated t_{IP}(us)**	**t_{total} (us)**	**IP Contribution (%)**
Unoptimised PISI	4	62.88	6.36
PISI (Unrolled by 2)	1.06	63.47	1.67
Unoptimised Binning	6	64	9.23
Binning (Unrolled by 2)	2.22	64.23	3.4

Chapter 5

Conclusion and Future Work

We have presented a hardware-friendly method, referred to PISI, for characterising a single-trial MUA spike train in a real-time. Since the spike interval implicitly indicates the frequency of spikes, the critical idea of PISI is to adjust the spike interval by checking the difference with previous values. Compared with binning, this method facilitates instantaneous frequency between spikes, which may contain helpful information.

Both asynchronous and synchronous organisations are proposed to send out the multi-channel data effectively. The results show that the full implementation can reduce the data rates from originally 36000 kbps to only around 1 kbps. In Huffman encoding, the estimated ISIs are normalised to maximise the entropy and average codeword length by four times. On the synthetic datasets, PISI is suitable to estimate the spike dynamic with low intensity and frequency. On the real-world test data, it has been shown that PISI achieves high decoding performance on the position (CC $\approx$ 0.72, RMSE $\approx$ 25) and velocity (CC $\approx$ 0.62, RMSE $\approx$ 45), even when using simple decoders.

In the hardware synthesis, all floating-point operations are correctly converted to the fixed-point equivalent. Both PISI and binning utilise similar resources, such as LUTs, BRAMs and FF, but PISI facilitates slightly lower power consumption (21 mW) and higher throughput performance (around 400 clock cycles). In FPGA testing, both algorithms are required around 60 us to process each spike sample. Interestingly, the throughput optimisation on IP functions negatively impacts the processing speed since the execution

time of IP cores only contributes a small amount of total time. It suggests that it is costly and ineffectively to implement a parallel structure on the simple algorithms.

To conclude, PISI can estimate accurately in the rare spike activities with low complexity. In terms of future work, the hardware implementation of PISI estimates in an asynchronous way can be investigated, potentially further reducing the power consumption and data rates. Besides, some deep learning models, like LSTM and GRU, can be employed on PISI estimates to improve the decoding accuracy.

Bibliography

[1] E. López-Larraz, A. Sarasola-Sanz, N. Irastorza-Landa, N. Birbaumer, and A. Ramos-Murguialday, "Brain-machine interfaces for rehabilitation in stroke: A review," vol. 43, no. 1, pp. 77–97, publisher: IOS Press. [Online]. Available: https://content.iospress.com/articles/neurorehabilitation/nre172394

[2] N. Ahmadi, T. Constandinou, and C.-S. Bouganis, "Robust and accurate decoding of hand kinematics from entire spiking activity using deep learning."

[3] S. D. Stavisky, J. C. Kao, P. Nuyujukian, C. Pandarinath, C. Blabe, S. I. Ryu, L. R. Hochberg, J. M. Henderson, and K. V. Shenoy, "Brain-machine interface cursor position only weakly affects monkey and human motor cortical activity in the absence of arm movements," vol. 8, no. 1, p. 16357, number: 1 Publisher: Nature Publishing Group. [Online]. Available: https://www.nature.com/articles/s41598-018-34711-1

[4] J. E. Niemeyer, "Brain-machine interfaces: assistive, thought-controlled devices," vol. 45, no. 10, pp. 359–361, number: 10 Publisher: Nature Publishing Group. [Online]. Available: https://www.nature.com/articles/laban.1115

[5] L. R. Hochberg, D. Bacher, B. Jarosiewicz, N. Y. Masse, J. D. Simeral, J. Vogel, S. Haddadin, J. Liu, S. S. Cash, P. van der Smagt, and J. P. Donoghue, "Reach and grasp by people with tetraplegia using a neurally controlled robotic arm," vol. 485, no. 7398, pp. 372–375, number: 7398 Publisher: Nature Publishing Group. [Online]. Available: https://www.nature.com/articles/nature11076

[6] N. Ahmadi, T. G. Constandinou, and C.-S. Bouganis, "Estimation of neuronal firing rate using bayesian adaptive kernel smoother (BAKS)," vol. 13, no. 11,

p. e0206794, publisher: Public Library of Science. [Online]. Available: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0206794

[7] Z. Zhang and T. G. Constandinou, "Adaptive spike detection and hardware optimization towards autonomous, high-channel-count BMIs," vol. 354, p. 109103. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0165027021000388

[8] C. J. Keller, C. Chen, F. A. Lado, and K. Khodakhah, "The limited utility of multiunit data in differentiating neuronal population activity," vol. 11, no. 4, p. e0153154.

[9] S. Han, J.-U. Chu, H. Kim, J. W. Park, and I. Youn, "Multiunit activity-based real-time limb-state estimation from dorsal root ganglion recordings," vol. 7, no. 1, p. 44197, number: 1 Publisher: Nature Publishing Group. [Online]. Available: https://www.nature.com/articles/srep44197

[10] N. Even-Chen, D. G. Muratore, S. D. Stavisky, L. R. Hochberg, J. M. Henderson, B. Murmann, and K. V. Shenoy, "Power-saving design opportunities for wireless intracortical brain–computer interfaces," vol. 4, no. 10, pp. 984–996, number: 10 Publisher: Nature Publishing Group. [Online]. Available: https://www.nature.com/articles/s41551-020-0595-9

[11] J. C. Kao, S. D. Stavisky, D. Sussillo, P. Nuyujukian, and K. V. Shenoy, "Information systems opportunities in brain–machine interface decoders," vol. 102, no. 5, pp. 666–682. [Online]. Available: http://ieeexplore.ieee.org/document/6786380/

[12] E. M. Trautmann, S. D. Stavisky, S. Lahiri, K. C. Ames, M. T. Kaufman, D. J. O'Shea, S. Vyas, X. Sun, S. I. Ryu, S. Ganguli, and K. V. Shenoy, "Accurate estimation of neural population dynamics without spike sorting," vol. 103, no. 2, pp. 292–308.e4. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0896627319304283

[13] R. Tomar, "Review: Methods of firing rate estimation," vol. 183.

[14] S. Shinomoto, "Estimating the firing rate," in *Analysis of Parallel Spike Trains*, pp. 21–35, journal Abbreviation: Analysis of Parallel Spike Trains.

[15] J. P. Cunningham, V. Gilja, S. I. Ryu, and K. V. Shenoy, "Methods for estimating neural firing rates, and their application to brain–machine interfaces," vol. 22, no. 9, pp. 1235–1246. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S089360800900032X

[16] G. L. Gerstein and N. Y.-S. Kiang, "An approach to the quantitative analysis of electrophysiological data from single neurons," vol. 1, no. 1, pp. 15–28. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1366309/

[17] H. Shimazaki and S. Shinomoto, "A method for selecting the bin size of a time histogram," vol. 19, pp. 1503–27.

[18] D. Endres and P. Földiák, "Estimating mutual information by bayesian binning."

[19] M. Rosenblatt, "Remarks on some nonparametric estimates of a density function," vol. 27, no. 3, pp. 832–837, publisher: Institute of Mathematical Statistics. [Online]. Available: https://projecteuclid.org/journals/annals-of-mathematical-statistics/volume-27/issue-3/Remarks-on-Some-Nonparametric-Estimates-of-a-Density-Function/10.1214/aoms/1177728190.full

[20] P. Bessou, Y. Laporte, and B. Pagès, "Frequencygrams of spindle primary endings elicited by stimulation of static and dynamic fusimotor fibres," vol. 196, no. 1, pp. 47–63, _eprint: https://physoc.onlinelibrary.wiley.com/doi/pdf/10.1113/jphysiol.1968.sp008493. [Online]. Available: https://physoc.onlinelibrary.wiley.com/doi/abs/10.1113/jphysiol.1968.sp008493

[21] D. A. Huffman, "A method for the construction of minimum-redundancy codes," vol. 40, no. 9, pp. 1098–1101, conference Name: Proceedings of the IRE.

[22] C. E. Shannon, "A mathematical theory of communication," p. 55.

[23] O. W. Savolainen and T. G. Constandinou, "Lossless compression of intracortical extracellular neural recordings using non-adaptive huffman encoding," in *2020 42nd Annual International Conference of the IEEE Engineering in Medicine Biology Society (EMBC)*, pp. 4318–4321, ISSN: 2694-0604.

[24] "Delta encoding," page Version ID: 1038092869. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Delta_encoding&oldid=1038092869

[25] M. Arif and R. Anand, "Effect on speech compression by combined delta encoding and huffman coding scheme," vol. 79, pp. 2371–2381.

[26] J. I. Glaser, A. S. Benjamin, R. H. Chowdhury, M. G. Perich, L. E. Miller, and K. P. Kording, "Machine learning for neural decoding," vol. 7, no. 4, publisher: Society for Neuroscience Section: Research Article: Methods/New Tools. [Online]. Available: https://www.eneuro.org/content/7/4/ENEURO.0506-19.2020

[27] Z. Xu, W. Wu, S. S. Winter, M. L. Mehlman, W. N. Butler, C. M. Simmons, R. E. Harvey, L. E. Berkowitz, Y. Chen, J. S. Taube, A. A. Wilber, and B. J. Clark, "A comparison of neural decoding methods and population coding across thalamo-cortical head direction cells," vol. 13, publisher: Frontiers. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fncir.2019.00075/full

[28] W. Wu, M. J. Black, Y. Gao, E. Bienenstock, M. Serruya, A. Shaikhouni, and J. P. Donoghue, "Neural decoding of cursor motion using a kalman filter," p. 8.

[29] J. M. Carmena, M. A. Lebedev, R. E. Crist, J. E. O'Doherty, D. M. Santucci, D. F. Dimitrov, P. G. Patil, C. S. Henriquez, and M. A. L. Nicolelis, "Learning to control a brain-machine interface for reaching and grasping by primates," vol. 1, no. 2, p. E42.

[30] Z. Li, J. E. O'Doherty, T. L. Hanson, M. A. Lebedev, C. S. Henriquez, and M. A. L. Nicolelis, "Unscented kalman filter for brain-machine interfaces," vol. 4, no. 7, p. e6243, publisher: Public Library of Science. [Online]. Available: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0006243

[31] D. Sussillo, P. Nuyujukian, J. M. Fan, J. C. Kao, S. D. Stavisky, S. Ryu, and K. Shenoy, "A recurrent neural network for closed-loop intracortical brain-machine interface decoders," vol. 9, no. 2, p. 026027.

[32] J. E. O'Doherty, M. M. B. Cardoso, J. G. Makin, and P. N. Sabes, "Nonhuman primate reaching with multichannel sensorimotor cortex electrophysiology," May 2017. [Online]. Available: https://doi.org/10.5281/zenodo.788569

[33] AXI4-lite IP interface (IPIF). [Online]. Available: https://www.xilinx.com/products/intellectual-property/axi_lite_ipif.html

Appendix A

Github

Please access the code by https://github.com/Alanzjc-pro/MSc_project, including Matlab (Software Development), Vivado_HLS (IP block Creation), Vivado (FPGA testing) and Datasets (Synthetic, Real-world)

Appendix B

Binning

The same hardware implementation flow is applied to the binning method, from the floating-point operation to fixed-point representation.

B.1 Floating-point Implementation

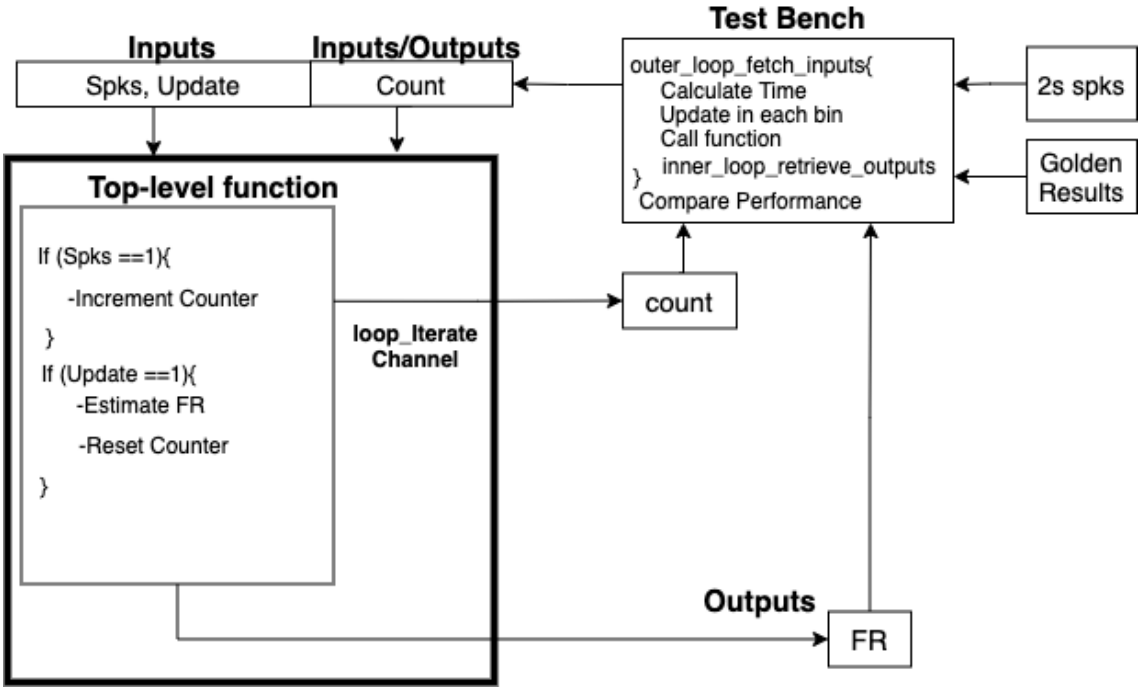

Figure B.1: The high-level interface between test bench and binning function in Vivado HLS

Similarly, a test bench is written in floating-point operation to fetch inputs, receive

outputs, and compare the estimated outputs with golden results.

As discussed, time is measured from current samples in the loop with the given sampling frequency. The spikes are passed to the top-level function to increment the value of the count in each channel. Then these count values are stored by transferring back to the test bench. The update signal is set to 1 only when the edge of a bin is reached. In this case, the multi-channel firing rates are measured, and the counters are then reset. The firing rates are finally retrieved and compared with the golden results in terms of accuracy.

B.2 Top-level Data Interface

Due to the requirement in the AXI-Lite interface, bit-packing is utilised to reduce the top-level arguments. The update and spike are packed to *Inputs*, the counts are packed to *Inputs/Outputs* and firing rate to *Outputs*. The original four arrays are reduced to three, saving the communication time.

Depending on the bit width, the multiple-channel counts and firing rates can be packed in corresponding variables, which reduces the array sizes.

B.3 Fixed-point Optimisation

The bit configuration of each variables can be seen in the Table B.1, which ensures no loss of precision. The bit usage of count and firing rate depends on the maximum values calculated from the test datasets. Both spike and update signal are represented by one bit to indicate true and false.

Table B.1: Bit configuration for fixed-point operation in binning.$< a, b >$: a represents word length and b represents integer bits

Variable	Configuration
Spks	$< 1, 1 >$
Update	$< 1, 1 >$
Count	$< 6, 6 >$
Firing Rate	$< 8, 8 >$

Therefore, four-channel count and firing rate are packed into *Inputs/Outputs* and *Outputs* with size $\frac{N}{4}$, respectively. In this study, the bin size is 200 ms. The division for the measurement of firing rate can be replaced by bit-shifting shown below:

$$\lambda(t) = \frac{count}{0.2} \tag{B.1}$$

$$\lambda(t) = count >> 2 + count \tag{B.2}$$