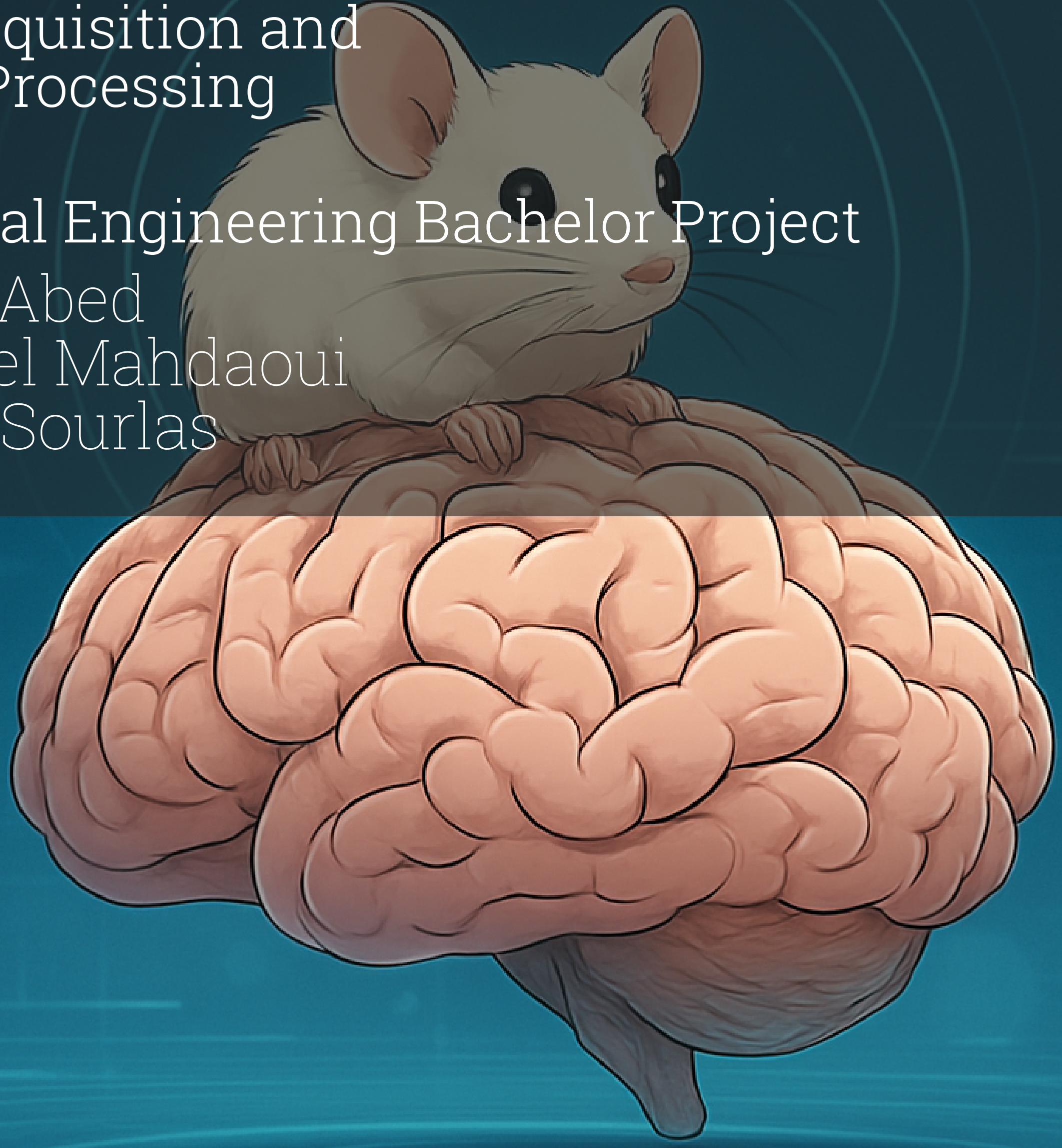
NeuroLogger: Ultra-Light Neural Activity Recorder

FPGA Architecture for Real-Time Data Acquisition and Signal Processing

Electrical Engineering Bachelor Project

Danial Abed
Anass el Mahdaoui
Foivos Sourlas



TUDelft

NeuroLogger: Ultra-Light Neural Activity Recorder

FPGA Architecture for Real-Time Data Acquisition and Signal Processing

by

Danial Abed
Anass el Mahdaoui
Foivos Sourlas

Supervisor:	Dr. ir. C. Strydis
Daily Supervisors:	L. P. L. Landsmeer, ir. A. Movahedin
Project Duration:	October, 2025 - December, 2025
Faculty:	Faculty of Electrical Engineering, Delft

TU Delft

Preface

This project marks the completion of our Bachelor's degree in Electrical Engineering at TU Delft. Together with F. Foglia and R. de Moor, we worked on the Neurologger project. While Francesco and Rik were responsible for the PCB design, we focused on the FPGA implementation.

This project was conducted in collaboration with the Department of Neuroscience at Erasmus MC, bridging the gap between electrical engineering and neuroscience research.

We would like to express our gratitude to C. Strydis, our supervisor, for his guidance and support throughout this project. We are also grateful to L. Landsmeer, A. Movahedin and the others from Erasmus MC for their time and valuable assistance.

Additionally, we want to thank I. Lager for supervising our graduation project, J. Hutton for the insightful ethics workshops, and V. Scholten for the business workshops that broadened our perspective on the practical implications of our work.

Danial Abed
Anass el Mahdaoui
Foivos Sourlas
Delft, December 2025

Abstract

Neural recorders capture electrical signals from brain tissue to study neural activity, but multi-channel systems generate large data volumes that challenge portable applications with limited storage and battery life. This project presents version three of the Neurologger, an ultra-lightweight FPGA-based neural recording system for real-time data acquisition and on board signal processing. The system uses specialized analog front-end chips to acquire neural signals from multiple channels. The FPGA implements processing pipelines for spike detection and sorting across all channels concurrently, achieving deterministic low-latency operation that sequential microcontrollers cannot match. By processing and classifying spikes in real-time, the system can substantially reduce data volume before storage, making it suitable for portable and closed-loop neural applications.

Contents

Preface i

Abstract ii

1 Introduction 1
- 1.1 Motivation 1
- 1.2 Thesis Goals and Contributions 1
- 1.3 Thesis Methodology 2
- 1.4 Thesis Roadmap 3

2 Background and Related Work 4
- 2.1 Neural Recording 4
- 2.2 Analog Front-End 4
- 2.3 FPGA Architectures 5
  - 2.3.1 Commercially Available Systems 5
- 2.4 Real-Time Signal Processing 6
  - 2.4.1 Frequency-domain Detection 6
  - 2.4.2 Time-Domain Detection 6
  - 2.4.3 Feature Extraction Methods 8
  - 2.4.4 Neural Network Classification 9
  - 2.4.5 Neural Network Quantization 9
- 2.5 Related Work 9

3 Design Specifications 11
- 3.1 Programme of Requirements 11
- 3.2 Design Choices 11
  - 3.2.1 ADC design choice 12
  - 3.2.2 FPGA design choice 13

4 FPGA Implementation 15
- 4.1 System Architecture 15
- 4.2 Clock Generation and Distribution 16
- 4.3 CPU Subsystem and Memory Architecture 16
- 4.4 Wishbone Bus Infrastructure 18
- 4.5 DMA Memory Subsystem 20
- 4.6 Configuration Register Module 24
- 4.7 Intan RHD Interface and SPI Protocol 25
  - 4.7.1 SPI Interface 25
  - 4.7.2 Multi-Headstage Support 26
  - 4.7.3 Simulation Support 26
- 4.8 NEO Spike Detection Algorithm 26
  - 4.8.1 ADC Sample Quantization and Preprocessing 27
  - 4.8.2 Dynamic Range Adaptation 29
  - 4.8.3 IIR Lowpass Filter Implementation 30
  - 4.8.4 Baseline Removal Through DC Tracking 32
  - 4.8.5 NEO Energy Calculation 33
  - 4.8.6 NEO Smoothing Filter 34
  - 4.8.7 Adaptive Threshold Calculation 35
  - 4.8.8 Threshold Comparison and Refractory Period 37
- 4.9 Spike Sorter Finite State Machine 38
- 4.10 Neural Network Classifier Architecture 40
  - 4.10.1 Weight Storage and Input Processing 42
  - 4.10.2 Hidden Layer Computation 42

4.10.3 Output Layer Computation . . . 43
4.10.4 Classification Decision and Timing . . . 43

5 Results 44
5.1 SPI link confirmation . . . 44
5.2 Spike detection testbench signals . . . 45
5.3 Resource Optimization and Scaling Analysis . . . 46
5.3.1 Address Decoder Overhead . . . 46
5.3.2 Comparison to Distributed RAM Implementation . . . 47
5.3.3 Scaling Laws and Theoretical Limits . . . 47
5.3.4 Optimization Techniques Applied . . . 47
5.3.5 Resource-Utilization Breakdown . . . 48

6 Conclusion and Future Work 49
6.1 Thesis Contributions . . . 50
6.2 Future Work . . . 50

References 52

1 Introduction

The cerebellum, a structure at the back of the brain, is essential for motor learning [1]. Studying how it processes information requires recording neural activity from Purkinje cells during natural, unrestrained behavior. Purkinje cells are specialized neurons that serve as the sole output of the cerebellar cortex and generate two distinct types of electrical signals called action potentials. Simple spikes fire at high rates, typically 40 to 100 Hz, encoding ongoing motor commands and sensory feedback [2]. Complex spikes occur much less frequently, around 1 Hz, with distinctive multi-peaked waveforms that signal errors in motor performance and drive learning [3]. The ability to accurately detect and distinguish between these two spike types in freely moving animals is essential for understanding how the cerebellum controls movement during natural behavior.

Recording neural activity during natural behavior presents significant technical challenges. Traditional neural recording systems require wired connections to external processing hardware, restricting animal movement and introducing mechanical artifacts that interfere with natural behavior [4]. Wireless systems eliminate the cable, but face severe constraints on size, weight, and power consumption. For small animals such as mice, the recording device must be extremely lightweight to avoid affecting natural movement. Battery-powered operation further limits system weight and recording duration [5]. Most wireless neural recorders transmit raw data to external computers for processing, resulting in high power consumption that limits battery life to a few hours [6]. These constraints create fundamental trade-offs between recording capability, system weight, and battery life.

1.1. Motivation

The Neuroscience department at Erasmus Medical Center (EMC) studies the interaction between the brain and sensorimotor control through neural recordings in mice [7]. Current recording methods restrict animal movement through head-fixation or wired connections, limiting the study of natural behavior. While the department has developed tethered systems that successfully record and classify cerebellar Purkinje cell spikes in real-time [8], these systems cannot be used during unconstrained behavior, preventing researchers from studying how the cerebellum operates during natural movement.

Existing wireless neural recorders eliminate the physical tether but face a critical limitation: they transmit raw data to external computers for processing, consuming significant power, and limiting battery life to just a few hours. To enable extended recordings during natural behavior, this thesis develops the NeuroLogger, an FPGA-based neural recording system that performs real-time Purkinje cell spike detection and classification on-board, eliminating continuous wireless transmission while meeting the strict weight and power constraints required for freely moving mice.

1.2. Thesis Goals and Contributions

Based on the task that was given to us by the Neuroscience department at EMC the following thesis goals can be formulated:

- Finding out what the state of the art technology is regarding NeuroLoggers and using it to make design specifications that sets our design apart from the rest.

- Designing an FPGA-based signal processing architecture for a wireless NeuroLogger that performs real-time spike detection and classification.

The contributions that this thesis will make are:

- Analysis of neural recording system requirements and evaluation of existing commercial solutions to establish design specifications for an ultra-lightweight wireless system.
- Design of an FPGA-based signal processing architecture that performs on-board spike detection and classification, addressing the challenges of multi-channel neural recording within strict power.
- Implementation of a resource-optimized multi-channel system through memory partitioning techniques, enabling 64-channel operation on resource-constrained FPGA hardware.
- Integration of on-board classification using quantized neural networks to distinguish between simple spikes, complex spikes, and noise events in real-time.

1.3. Thesis Methodology

The project started with an extensive state-of-the-art and competitor analysis to motivate and justify the need for a device like the NeuroLogger. The work was divided into two subgroups: PCB design, carried out by Francesco Foglia and Rik de Moor, and FPGA design, carried out by the authors of this report. The FPGA group was responsible for ensuring that the FPGA handled data correctly, communicated with the hardware peripherals, and performed the required onboard signal processing.

At the start of the project, the members of the FPGA group had to familiarize themselves with the Verilog hardware description language. All three members followed the previous version of the EE BSc curriculum at the TU Delft and had only worked with the hardware description language VHDL before. In addition, an existing Git repository developed by the teaching assistants mentioned at the beginning of this report and by colleagues from the Neuroscience department of the EMC was given to both subgroups and had to be reviewed. This repository formed the basis of the current Wishbone, softcore, and microSD card architecture, but it hadn't been tested on actual hardware and therefore required debugging and further development.

The FPGA was introduced only in version three of the NeuroLogger. Version two relied on a microcontroller instead of an FPGA. A visual hardware comparison of the first two PCB iterations is shown in Figure 1.1, with version three corresponding to the PCB contribution of this BAP group by Francesco Foglia and Rik de Moor. The two subgroups worked largely in parallel, but communicated whenever topics overlapped ”and met on an almost weekly with the project supervisor and daily supervisors.

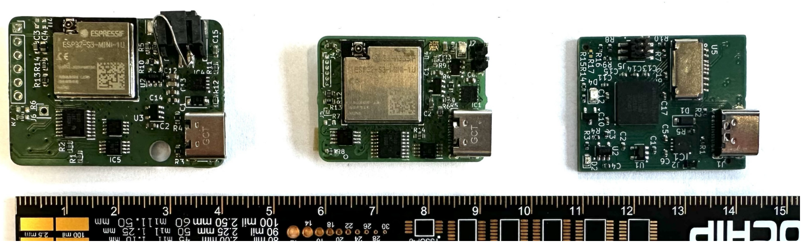


Figure 1.1: The development of the NeuroLogger, showing progressively improved PCB designs v1 to v3 from left to right.

An overview of the FPGA team's project timeline and design process is shown in Figure 1.2, followed by an overview of the PCB team's timeline and design process in Figure 1.3.

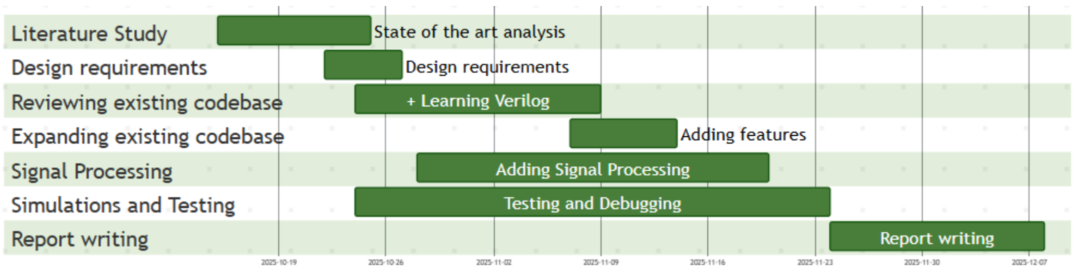


Figure 1.2: NeuroLogger's FPGA design process

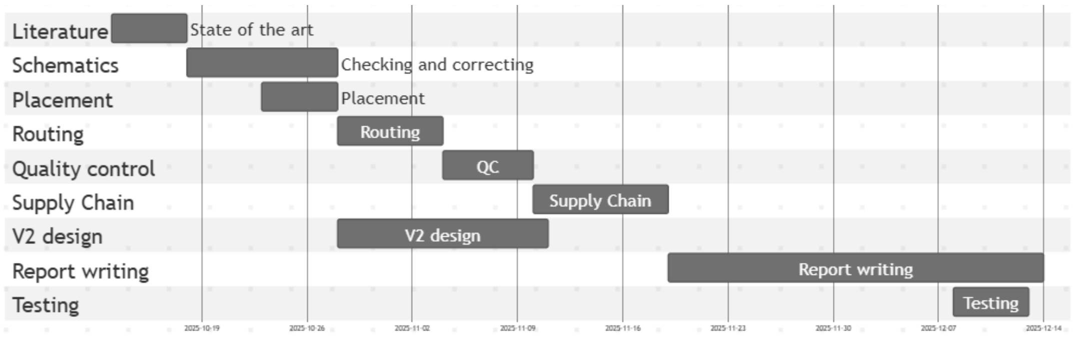


Figure 1.3: PCB group design process

1.4. Thesis Roadmap

This thesis documents the development and validation of the NeuroLogger, an FPGA-based neural recording system for freely moving mice. Chapter 2 provides background on neural recording principles, analog front-ends, FPGA architectures, and real-time signal processing approaches for spike detection and classification. Chapter 3 establishes the design specifications for the system, defining requirements for power consumption, weight constraints, recording capabilities, and signal processing performance. Chapter 4 presents the FPGA implementation, detailing the system architecture, data acquisition pipeline, communication infrastructure, and on-board signal processing modules. Chapter 5 discusses the experimental validation results and system performance. Chapter 6 provides conclusions and recommendations for future work.

2 Background and Related Work

2.1. Neural Recording

The action potentials, generated by neurons, appear as voltage fluctuations with amplitudes ranging from 50 to 500 microvolts and durations of 1-2 milliseconds. The frequency spectrum of these spikes concentrates between 300 Hz and 5 kHz. Distinguishing between these spike types requires adequate temporal resolution to capture the distinctive waveform morphologies, with classification algorithms typically requiring 30-40 samples per spike to reliably differentiate simple from complex spikes based on their shape characteristics [8]. With spike durations ranging from 1-3 ms for simple and complex spikes, neural recording systems typically employ sampling rates of 20-30 kHz to provide sufficient temporal resolution for waveform classification while satisfying Nyquist criteria [8]. Multi-channel recording systems acquire signals simultaneously from multiple electrode sites to capture spatially distributed neural activity. Recording from multiple locations simultaneously enables researchers to understand how distributed neural populations coordinate their activity during behavior and learning, and allows comparison of activity patterns across different brain regions.

2.2. Analog Front-End

Neural signal acquisition requires specialized analog front-end circuits to amplify weak extracellular potentials and convert them to digital form for processing. Three primary implementation approaches exist: custom application-specific integrated circuits (ASICs) designed for specific experimental requirements, discrete component designs using individual operational amplifiers and analog-to-digital converters, and commercial biosignal acquisition systems. Custom ASICs offer optimal power efficiency and smallest size, but they require significant development time and cannot be easily modified for different experiments. Discrete component implementations provide maximum configurability through selection of individual amplifiers and converters optimized for specific parameters, but consume considerable board area and power when scaled to high channel counts. Several manufacturers offer specialized chips for neural recording, including the Intan RHD2000 family [9], Texas Instruments ADS1299 series [10], and custom research ASICs [11]. These integrated chips typically provide low noise performance (below 5 μV RMS), programmable gain amplification ranging from 32x to 512x, configurable bandpass filters with cutoff frequencies optimized for neural signals, and 12-16 bit analog-to-digital conversion resolution. The amplification stage boosts weak electrode signals to voltage levels suitable for digitization while maintaining low input-referred noise. Bandpass filtering is performed in the analog domain before digitization, typically with high-pass cutoffs around 0.1-1 Hz to remove DC offset and baseline drift, and low-pass cutoffs around 7.5-10 kHz to prevent aliasing and reduce high-frequency noise [11]. By performing amplification and filtering prior to analog-to-digital conversion, these chips reduce the computational requirements for downstream digital signal processing stages.

The analog-to-digital converter resolution determines how finely voltage levels can be distinguished in the digitized signal. Neural recording systems must capture signals across a wide dynamic range, from small baseline fluctuations of a few microvolts to large action potentials that can reach several hundred microvolts. A 16-bit ADC provides 65,536 discrete voltage levels, offering sufficient resolution to accurately represent both small background activity and large spikes without losing information. Lower resolutions, such as 12-bit ADCs with only 4,096 levels, may struggle to capture subtle features of smaller spikes or introduce quantization noise that degrades signal quality.

Communication with processing electronics commonly uses standard protocols such as SPI or I2C (Figure 2.1), with support for CMOS or differential signaling depending on cable length and noise requirements. Among these protocols, SPI is widely adopted due to its high-speed capabilities, with the Intan RHD2000 series supporting SPI communication at clock frequencies up to 25 MHz [9]. SPI operates synchronously, meaning all devices share a common clock signal (SCLK in Figure 2.1) that coordinates when data is transmitted and received. In a typical SPI configuration, one device acts as the master that controls the clock and initiates transactions, while other devices function as slaves that respond to commands. Data is exchanged on dedicated signal lines, with separate paths for sending commands to slaves (MOSI) and receiving data back from them (MISO). This synchronous approach ensures predictable timing and enables high-speed communication, making it well-suited for streaming continuous neural data from multiple channels. The protocol's relatively simple hardware requirements and deterministic behavior make it a popular choice for embedded systems where resources and timing constraints are critical. For multi-chip configurations, multiple RHD devices can share common clock and command lines while maintaining separate data output lines, enabling simultaneous operation without interference. At maximum sampling rates with multiple channels, the required data throughput can exceed 20 megabits per second, necessitating high-speed SPI operation to transfer all channel data without loss.

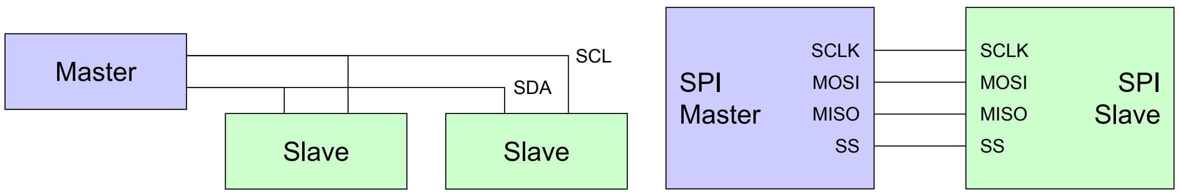


Figure 2.1: Left: I2C uses two shared lines (SDA and SCL) for two-way communication with multiple slaves. Right: SPI uses four dedicated lines (SCLK, MOSI, MISO, SS) for one-way signal paths in point-to-point communication.

2.3. FPGA Architectures

Field-Programmable Gate Arrays offer a practical solution for implementing sophisticated signal processing within strict power and weight constraints. FPGAs are integrated circuits that can be configured to implement custom digital logic using hardware description languages such as Verilog. Their key advantage lies in parallel processing capability. FPGAs consist of configurable logic blocks and programmable connections that execute multiple operations simultaneously in hardware, unlike microcontrollers that process instructions sequentially. This parallel architecture enables concurrent handling of high-speed data from multiple electrodes without data loss [12]. FPGAs implement logic directly in hardware, delivering deterministic timing and low latency essential for real-time spike detection and classification [13]. Researchers can modify signal processing algorithms through reconfiguration rather than requiring new hardware, providing flexibility for different experimental requirements. These characteristics make FPGA-based architectures well-suited for neural recording applications that require both real-time performance and reasonable power consumption for battery-powered operation [14].

2.3.1. Commercially Available Systems

While FPGA architectures show promise for balancing performance and power, current commercially available solutions force researchers to compromise between channel count, battery life, and weight:

- **Deuteron MouseLog-16C:** This system is optimized for minimal weight and size, achieving a total mass of approximately 2.9 g. However, to achieve this form factor, the system compromises on data density, providing only 16 recording channels, which limits its utility for large-scale population dynamics [15].
- **Spike Gadgets TAINI:** This logger is engineered for extreme endurance, delivering an impressive battery life of 72 to 120 hours. The trade-off for this extended duration is a reduced channel count (16 channels), making it unsuitable for high-density mapping of neural circuits [16].
- **Evolocus NeuroLogger v3:** This device focuses on high-fidelity signal acquisition, supporting 32 channels at a high sampling rate of 20 kHz. This performance comes at the cost of power

consumption, limiting the recording duration to approximately 2.25 hours before battery depletion [17].

- **Triangle BioSystems W64:** These systems focus on wireless data transmission rather than on-board logging, allowing for 64-channel real-time monitoring. However, this approach sacrifices power efficiency due to the high energy demands of RF transmission and introduces the risk of data loss through dropped packets [18].

Table 2.1 summarizes the key specifications of the systems discussed above, revealing the fundamental trade-offs that researchers must navigate. No existing solution simultaneously achieves high channel count, extended battery life, and minimal weight. Systems optimized for one parameter necessarily compromise on others, forcing researchers to choose between channel density, recording duration, and animal mobility.

Table 2.1: Comparison of the commercially available neural recording systems for small animals.

System	Channels	Weight (g)	Battery Life (h)	Sampling Rate (kHz)
Deuteron MouseLog-16C	16	2.9	10	20
Spike Gadgets TAINI	16	3.5	72–120	30
Evolocus NeuroLogger v3	32	4.5	2.25	20
Triangle BioSystems W64	64	5.8	3–6	30

2.4. Real-Time Signal Processing

Once neural signals have been amplified, filtered, and digitized by the analog front-end, they are transmitted to the FPGA for real-time processing. The FPGA receives a continuous stream of digital samples from multiple recording channels, generating substantial amounts of data that must be managed efficiently. The data rate scales with the number of channels, sampling frequency, and bit resolution, quickly reaching gigabytes per hour for multi-channel systems. For implantable or portable devices with limited storage capacity and battery life, this presents a significant challenge. Real-time signal processing algorithms running on the FPGA address this challenge by identifying and extracting relevant neural events before storage, significantly reducing data volume while preserving essential information.

Spike detection and classification form the core of this processing pipeline. Spike detection identifies the occurrence of action potentials in the continuous neural signal by monitoring for characteristic voltage transients that exceed background noise levels. Detection algorithms must distinguish these genuine neural events from noise artifacts and baseline fluctuations in real-time. Following detection, spike classification categorizes the identified events based on their waveform morphology to determine the neuron type or source. For Purkinje cell recordings, classification distinguishes between simple spikes and complex spikes, which exhibit different durations, amplitudes, and temporal firing patterns. Both detection and classification can be implemented using various algorithmic approaches, ranging from simple threshold-based methods to sophisticated neural network classifiers, each offering different trade-offs between computational complexity, detection accuracy, and hardware resource requirements.

2.4.1. Frequency-domain Detection

Frequency-domain methods first transform the neural signal using techniques such as Fast Fourier Transform (FFT), then analyze the frequency content to distinguish spike activity from background noise and artifacts [19]. While frequency-domain approaches can offer better noise rejection and feature extraction, they come with significant drawbacks. These methods require more computational resources and introduce additional latency because the signal must first be transformed before analysis can begin. For FPGA-based real-time neural recording systems, time-domain methods are generally preferred due to their simplicity, lower resource consumption, and deterministic low-latency operation.

2.4.2. Time-Domain Detection

In contrast, time-domain methods operate directly on the raw neural signal, applying threshold-based detection or template matching algorithms to identify spikes as they occur in real-time. The signal is continuously monitored, and when the amplitude exceeds a predetermined threshold or matches a known spike template, the event is flagged and recorded. This selective approach enables significant data reduction, with compression ratios of up to 50x given that neural activity has a duty cycle of only 2-20% [14].

Threshold-based detection and template matching represent two distinct strategies for real-time spike identification. Threshold-based detection monitors the incoming signal amplitude and triggers when it crosses a predetermined voltage level, typically set as a multiple of the background noise standard deviation. However, threshold-based approaches are prone to false positives in noisy environments, as any signal exceeding the threshold, whether a genuine spike or noise artifact, will trigger detection. Template matching compares the incoming waveform against pre-recorded spike templates and identifies matches based on correlation or similarity metrics. FPGA implementations of template matching have demonstrated high sorting accuracy with latencies suitable for closed-loop neural control applications [20]. This approach requires significantly more memory to store spike templates and greater computational resources for waveform comparisons. The choice between threshold-based detection and template matching depends on the available FPGA resources, power budget, and required detection accuracy.

Simple Threshold Crossing

The simplest spike detection method applies a fixed voltage threshold: any sample exceeding the threshold triggers detection. For a bandpass-filtered signal $x[n]$ and threshold θ, detection occurs when:

$$x[n] > \theta \quad \text{and} \quad x[n] > x[n \pm 1] \text{ (local maximum).} \tag{2.1}$$

The threshold is typically set as:

$$\theta = k \cdot \sigma_{\text{noise}}, \tag{2.2}$$

where σ_{noise} is the noise standard deviation and $k \approx 3$–5. Quiroga et al. proposed estimating σ_{noise} robustly using the median absolute deviation (MAD):

$$\sigma_{\text{noise}} \approx \frac{\mathrm{MAD}(x)}{0.6745}, \tag{2.3}$$

where the scaling factor converts MAD to a standard deviation equivalent for Gaussian distributions [21].

This approach offers several advantages: the implementation is straightforward requiring only a single comparator in hardware, the detection latency is deterministic at one sample period, and the computational cost is minimal compared to more sophisticated methods.

However, simple threshold crossing has notable disadvantages: the method is sensitive to baseline drift if the threshold remains fixed, each recording session typically requires manual threshold tuning, performance degrades when the signal-to-noise ratio varies across channels, and the detector cannot distinguish genuine spike waveforms from high-amplitude noise transients.

Despite these limitations, threshold crossing remains the standard for real-time spike detection in commercial systems due to its hardware simplicity. Adaptive extensions that continuously update the threshold based on running noise estimates improve robustness at the cost of additional logic.

Adaptive Thresholding

While simple threshold crossing uses a fixed detection threshold, adaptive thresholding continuously adjusts the threshold based on the current noise conditions. This approach improves robustness when signal characteristics change during recording, such as increasing noise levels or baseline drift.

Several statistical methods enable adaptive threshold calculation. Standard deviation tracking continuously estimates noise standard deviation σ using a sliding window or IIR filter. The exponentially weighted moving average (EWMA) approach updates the estimate as:

$$\sigma[n] = \alpha \cdot \sigma[n-1] + (1-\alpha) \cdot |x[n]| \tag{2.4}$$

where $0 < \alpha < 1$ controls the adaptation rate. Higher values of α result in slower adaptation but more stable estimates [22].

The median absolute deviation (MAD) provides an alternative noise estimation method:

$$\text{MAD} = \text{median}(|x[n] - \text{median}(x)|), \qquad \sigma_{\text{estimate}} = \frac{\text{MAD}}{0.6745} \tag{2.5}$$

Computing true medians requires sorting operations, which are computationally expensive. Practical implementations often approximate MAD via IIR filtering of absolute deviations to reduce complexity.

Activity-dependent scaling represents another approach where the threshold scales with signal activity level, estimated using zero-crossing frequency f:

$$\theta[n] = k \cdot \sigma^2 \cdot f^2 \tag{2.6}$$

Hardware implementation of adaptive thresholding presents several practical constraints. IIR estimators with time constant $\tau = 1/(1-\alpha)$ need approximately 5τ samples to converge. For $\alpha = 0.99$, this corresponds to 500 samples or 16.7 ms at 30 kHz sampling. Squaring a 10-bit value produces a 20-bit result, requiring proper scaling and shifting to prevent overflow and maintain precision. To avoid computationally expensive square-root operations, comparisons are performed using squared values:

$$x^2 > \theta^2 \tag{2.7}$$

Sliding-window methods require storing the last N input samples, while IIR methods require only 1–2 registers, making IIR approaches more resource-efficient for FPGA implementation.

NEO: Nonlinear Energy Operator

The Nonlinear Energy Operator provides superior noise robustness compared to simple amplitude thresholding. For a discrete-time signal $x[n]$, the NEO is defined as [23]:

$$\Psi(x[n]) = x[n]^2 - x[n-1] \cdot x[n+1]. \tag{2.8}$$

This operator emphasizes transient, high-frequency events like spikes while suppressing slower variations. The NEO exhibits derivative-like behavior by approximating the squared derivative of the signal, emphasizing rapid changes characteristic of action potentials. Baseline independence results from the subtraction of adjacent sample products, making slow baseline drifts contribute minimally to the output. Noise suppression occurs because uncorrelated noise produces near-zero NEO values on average. The computational simplicity enables efficient hardware implementations, requiring only two multiplications and one subtraction per sample.

A typical NEO-based detection pipeline includes several stages:

1. Bandpass filtering removes out-of-band noise and artifacts.
2. NEO computation transforms the filtered signal into an energy estimate.
3. Smoothing via a low-pass filter reduces high-frequency fluctuations in the energy estimate.
4. Threshold comparison identifies when the smoothed energy exceeds a detection threshold.
5. Refractory control prevents multiple detections of the same spike event.

Practical implementations apply a first-order IIR filter to smooth the input signal and mitigate high-frequency noise components. The filtered signal feeds into the NEO calculator and an accompanying first-order IIR filter to obtain an instantaneous estimation of signal energy. When the energy estimate exceeds the calculated threshold value, the system registers a spike detection.

2.4.3. Feature Extraction Methods

Spike classification requires extracting discriminative features from detected waveforms. Several approaches exist, each with different computational requirements and hardware complexity. Principal Component Analysis (PCA) reduces the dimensionality of spike waveforms by projecting them onto principal components that capture the most variance. This method requires covariance computation and decomposition, which are computationally intensive operations for hardware implementation. Wavelet decomposition transforms spike waveforms into the time-frequency domain, enabling feature extraction at multiple scales. The hardware complexity of this approach depends on the chosen wavelet family and decomposition level. Template matching compares detected waveforms against pre-stored spike templates using correlation or distance metrics. While conceptually simple, this method is sensitive to waveform drift over time as electrode positions shift or neural activity patterns change. Raw sample extraction feeds spike waveform samples directly to the classifier without preprocessing, eliminating the computational overhead of feature extraction at the cost of requiring larger classification networks to process higher-dimensional input. For systems with sufficient computational resources, this approach simplifies the signal processing pipeline. The waveform window must be sufficiently long to capture the entire spike duration, typically 1.5–2 ms depending on the sampling frequency.

2.4.4. Neural Network Classification

Neural networks provide a robust approach to spike classification when sufficient training data is available. These classifiers generalize well across varying signal conditions including noise floor variations, recording-electrode drift, saturated sample points, and data offsets. The highly resource-constrained nature of neural recording devices necessitates lightweight neural network architectures. For ternary classification problems like Purkinje cell spike sorting (simple, complex, or neither), multilayer perceptrons with fully connected feedforward architectures can achieve high accuracy with minimal resources. Typical architectures include an input layer matching the waveform sample count, one or two hidden layers, and an output layer corresponding to the number of classes. For hidden layers, various activation functions enable the network to learn nonlinear relationships. ReLU (Rectified Linear Unit) is particularly hardware-efficient because it can be implemented as a simple comparison operation:

$$\text{ReLU}(x) = \max(0, x) \tag{2.9}$$

For multi-class classification, Softmax is commonly used in the output layer to calculate final probabilities. However, Softmax is computationally expensive in hardware and requires floating-point support. An alternative approach moves Softmax to the loss function during training, making the network output log probabilities (logits) during inference. This uses a simple linear activation function that carries no computational cost. Since the natural logarithm is a monotonically increasing function of probability, the output class is determined by selecting the neuron with the highest value.

2.4.5. Neural Network Quantization

Standard neural networks use 32-bit floating-point arithmetic, which requires substantial computational resources and energy. Quantization reduces precision to 8-bit or 16-bit integers to reduce computational burden, memory requirements, and power consumption. Networks are typically designed and tested in software using floating-point arithmetic before being quantized for hardware implementation. Quantization-aware training uses quantized weights during forward passes while backward passes use full precision, allowing the network to adapt to the reduced precision during training and minimizing accuracy loss. Symmetric quantization represents the relationship between quantized and real values as:

$$x_{\text{real}} = \text{scale} \cdot x_{\text{quant}} \tag{2.10}$$

During inference, matrix multiplication accumulates quantized values:

$$y_{\text{quant}} = \sum_i w_{\text{quant},i} \cdot x_{\text{quant},i} \tag{2.11}$$

$$y_{\text{real}} = (\text{scale}_w \cdot \text{scale}_x) \cdot y_{\text{quant}} \tag{2.12}$$

Fixed-point implementations use integer constants M and N to approximate the scaling:

$$y_{\text{real}} \approx \frac{M \cdot y_{\text{quant}}}{2^N} \tag{2.13}$$

For Purkinje cell spike classification, quantized 8-bit networks can achieve over 90% accuracy [24], approaching the 94% accuracy of full-precision implementations while enabling significant reductions in hardware resource usage and power consumption.

2.5. Related Work

Several FPGA-based neural recording systems implementing real-time spike detection and classification have been proposed, each with specific advantages and limitations. Following is a description of these implementations:

- **[12]**: Presents a 64-channel FPGA-based neural recording system implementing spike extraction algorithms with 3 mW on-chip dynamic power consumption. The system demonstrates power efficiency at least half that of comparable systems without on-chip signal processing, achieved through parallel processing architecture that handles multiple electrode channels simultaneously without data loss.
- **[13]**: Describes an FPGA implementation of a spike detector with deterministic timing and low latency suitable for real-time classification. The system emphasizes hardware-efficient threshold-based detection with configurable parameters, demonstrating the advantages of direct hardware implementation for time-critical neural signal processing.

- **[14]**: Proposes a multi-channel neural recording system achieving data compression ratios up to 50x through selective spike storage, exploiting the 2-20% duty cycle of neural activity. The architecture balances real-time performance with reasonable power consumption for battery-powered operation in freely-moving animal experiments.
- **[20]**: Implements template matching-based spike sorting on FPGA with latencies suitable for closed-loop neural control applications. The system demonstrates high sorting accuracy while maintaining deterministic processing delays, enabling real-time feedback control based on classified neural activity.
- **[25]**: Provides a comprehensive comparison of spike detection algorithms implemented on FPGAs, evaluating trade-offs between detection accuracy, computational complexity, and hardware resource utilization. The study demonstrates that simpler algorithms like threshold crossing often provide sufficient performance while consuming fewer FPGA resources than sophisticated methods.

3 Design Specifications

3.1. Programme of Requirements

To ensure that the NeuroLogger meets the functional and practical needs of a lightweight, real time neural recording system, the following design requirements have been listed. These requirements translate the project goals: ultra low weight, high channel count, real time onboard signal processing and compact easy to use data storage into more technical specifications.

General device specifications, shared between the two subgroups:

- The neural logger device including the PCB, integrated battery and microSD card must have a total weight of 5 grams or less and a form suitable for attachment to a mouse without restricting its natural movement.
- An FPGA should be integrated to manage high-speed data handling and real-time signal processing of the acquired data.
- The neural logger must be able to interface with two 32-channel RHD2132 Intan chips. Communication and control is handled by a controller implemented on an FPGA (LCMXO2-4000HC-6MG132C) which is mounted on the PCB.
- The device must support battery powered operation for a minimum duration of 30 minutes, with a target operating time of 1 hour under typical recording conditions.
- The NeuroLogger must be compatible with the Cambridge Neurotech Mini-Amp-64 head stage.
- The device must include sufficient and easily removable onboard storage, implemented as a microSD card that can be inserted into the PCB to store the raw acquired data and / or the information extracted during signal processing. What data get stored depends on the device's operating mode.
- The neural logger must feature an integrated LED to assist with synchronization and provide a real time visual indication of spike detection during operation.
- The neural logger must support operation in both battery powered and USB-C wired mode.

Subgroup specific requirements:

- The neural logger must be capable of sampling corresponding channels from both Intan chips in parallel (sampling the same channel index from each chip in parallel).
- Each individual channel must be sampled at 30 kHz, since this is the frequency requirement for the experiments.
- The FPGA must be able to perform real time signal processing on the incoming data streams.

3.2. Design Choices

This section outlines the two main hardware design choices made during the development of the NeuroLogger. The decisions described here were driven by the practical constraints of small scale electrophysiology, such as size, weight, power consumption, channel count, sampling rate, and component costs. Several technically viable options were considered, but not all were suitable for out timeline and project. The following subsections describe the reasoning behind the selected ADC and FPGA platforms and motivate why these choices were the most appropriate for our device.

3.2.1. ADC design choice

Electrical brain activity can be measured using tiny electrodes implanted directly into brain tissue, this enables the study of small brain (sub)circuits. This method is invasive and carries some risks but it provides high resolution recordings of fast changing signals (millisecond timescales) like action potentials also referred to as spikes, that cannot otherwise be captured by non-invasive techniques. Because the electrodes are located close to groups of neurons, by using this method high precision signals can be recorded. This distinguishes it from surface-based methods, which measure more spatially averaged activity and miss finer details of neural signaling. It's the most reliable way to obtain precise, cell level measurements, and it is widely used in animal research and some specific human applications, such as epilepsy monitoring and brain computer interface studies.

The neural signals must be amplified, filtered, and converted into digital data before they can be analyzed, therefore a suitable Analog-to-Digital Converter (ADC) device must be chosen. When considering the project's relatively short duration span and the NeuroLogger's design requirements, the range of suitable electrode Analog-to-Digital Converter solutions is not very broad. Our system is intended for neural electrophysiology in freely moving mice, which imposes strict constraints on size, weight, and power consumption to avoid interfering with the animal's natural behavior and to enable battery powered operation. In addition, the target experiments require 32 channels per brain hemisphere, corresponding to 32 channels per chip, with each channel sampled at 30 kHz to reliably detect neural spikes. Devices requiring external power supplies are unsuitable because of the untethered operation requirement, and wireless data transfer was explicitly not desired because of its disproportionate impact on power consumption.

An additional trade off requirement for the ADC chip was for there to be onboard signal amplification and perhaps filtering, these weren't absolute deal breakers if not available but are very desirable to have on board of the ADC chip because then we don't need additional hardware down the line to handle these tasks, this simplifies the design by eliminating the need for additional components and usually offers better signal to noise ratios since amplification, filtering, and digitization directly happen next to the electrodes. The last and perhaps a less explicitly stated but important constraint was the cost. No specific maximum budget was specifically allocated for the ADC chips but as we will see later the cost of some of the most popular devices can easily jump to several thousand euros which is hard to get funding for especially for an experimental prototype device. We only considered commercially available ADC converter options meant specifically for electrophysiology in small animals, ordering custom academic implementations directly from other labs was also an option, but it is expensive, generally less well supported and shipping times can be more unreliable. Although very interesting it was not a suitable choice for the timeline of this project. Those things considered here are some of the options considered during the initial phase of the project.
These are the most widely used commercial solutions that satisfied our high channel count, sampling rate, and size requirements: the Intan Technologies RHD2000 chip series; Blackrock Neurotech's Honey Badger ASIC and CerePlex µ; Ripple Neuro's Pico32; and Plexon's OmniPlex digital headstages.

The Blackrock, Ripple, and Plexon systems are technically robust and suitable for use in mice. All options have comparable mass (approximately 1.0–1.2 g) and compact form factors, with dimensions on the order of a few centimeters in width and a few millimeters in thickness (Ripple Neuro Pico32: 13.2 × 11.7 × 3.5 mm; Blackrock CerePlex µ: 13 × 20 mm; Plexon HST/32D with DHP/OmniPlex ecosystem: 24.6 × 13.2 × 2.5 mm). All systems support sufficiently high sampling frequencies, and offered models with at least 32 channels. However they didn't score that well when it came to some of the other requirements. [9, 26, 27, 28]

Most relied on closed, ecosystem dependent solutions, meaning they were not compatible with our custom design and typically operated only with proprietary base stations, some of which (e.g. Blackrock's) cost over $10,000. Others had high per-channel costs and offered insufficient low-level configurability. In addition, the rejected options were relatively power hungry or lacked official power consumption figures, were relatively expensive, and were surprisingly not straightforward to acquire without direct contact with the company. Therefore these factors made them unsuitable, or at minimum non-ideal, for the scope and timeline of our project.

Other electrophysiology devices were also considered but were quickly rejected for failing to meet one or more of the key requirements, usually the desired channel count or they generally had insufficient information about key specifications.

After eliminating higher-cost, closed ecosystem systems, the Intan RHD chip series emerged as the overall best fit. The RHD2132 offers 32 channels per chip, up to 30 kHz per channel sampling, comfortably above our requirement. Similar to other options it has integrated low-noise amplifiers and ADCs on every channel. It offers customizable analog filters (HPF and LPF) it had among the lowest power consumptions a asbolute maximum of $37.3mW$ and it was significantly smaller than most other options considered, only 8×8 mm and around 1 g. [9].
Intan chips are widely used for electrophysiology experiments in small rodents like mice and they are is relatively low cost sitting, at the time of writing sitting at $\$430$ per chip. [29].
And crucially they offers high hardware customizability: good documentation, are well supported and can be addressed from an FPGA by using SPI. Among the options considered the Intan was the most suitable choice for our custom FPGA and NeuroLogger design. As stated it comfortably met the design requirements while being low cost. And the fact that it was commercially available to order from online meant that it also allowed for easier future scaling.

In conclusion, given the tight constraints of small scale electrophysiology particularly size, weight, power, channel count, sampling rate, and cost, most commercial neural recording solutions were unsuitable, primarily due to proprietary lock ins and high system-level expenses. Although Blackrock, Ripple, and Plexon offer technically strong products, these are designed for closed, high cost platforms rather than custom, cost-efficient implementations.

In contrast, the Intan RHD2132 emerged as the most suitable commercial option, as it met all technical requirements. It was much more cost effective and it offered significantly greater flexibility and customizability. For these reasons, the Intan RHD2132 was selected as the optimal ADC chip for this mouse electrophysiology application.

3.2.2. FPGA design choice

The FPGA is a very important hardware feature of our design and it is what distinguishes version 3 of the NeuroLogger from its predecessor v2 (which used a microcontroller) and other neural logger competitor devices. An FPGA (field-programmable gate array) is a reconfigurable digital chip whose logic gets customized at the hardware level. Unlike a microcontroller, it can executes operations in true parallel, enabling much higher throughput and deterministic low latency. This allows high speed signal processing with lower clock rates, reducing power consumption. FPGAs are highly customizable and great for prototyping hardware, they support application-specific data paths and interfaces. Development is relatively straightforward and preferred for high performance tasks. FPGAs mainly implement hardware using lookup tables (LUTs), which are configurable logic elements that realize the desired Boolean functions. By programming and interconnecting LUTs, the device forms custom digital circuits. For this reason FPGA size and capabilities are commonly specified as LUT count.

The target application is a 2 × 2 cm head-mounted PCB for freely moving mice, interfacing with two Intan RHD2132 ADC chips. The FPGA must:

- Support minimum 4,320 LUTs (MachXO2-4000 capacity), though resource analysis revealed that 64-channel NEO detection requires approximately 5,200 LUTs, necessitating either the larger variant (6,864 LUTs) or reduction to 32 channels for the MachXO2-4000
- Have suitable pins for SPI (needed for Intan devices) and have LVDS-capable I/O
- Include on-chip RAM

- Be low power (for battery operation)
- Have Non-Volatile Configuration Memory (in order to retain its configuration after power down)
- Be compact, preferably under 3 mm $\times$ 3 mm
- Optional (Trade off) requirement: DSP blocks (useful for spike classification, though the Lattice MachXO2/MachXO3 families lack dedicated DSP blocks, requiring multiply operations to be implemented using LUTs at significant resource cost)

Candidate FPGA Devices:

- Lattice iCE40 UltraPlus (UP3K / UP5K)
- Lattice MachXO2-4000 or MachXO3-4300
- Intel MAX 10 (10M04)
- Microchip IGLOO-nano (AGLN250)
- Gowin GW1N-4 (LittleBee)

All aforementioned Lattice FPGA models meet all of the design specifications and are therefore well suited candidates for our device. Some notable features include the availability of extra on chip memory, which was an important consideration for this project.

The Intel MAX 10 (10M04) FPGA promises better performance due to its higher DSP capability, but this comes at the cost of higher power consumption when compared to the Lattice options. For our application this is likely overkill, and the increased power consumption makes it less suitable for head mounted animal experiments. The Microchip IGLOO-nano (AGLN250) and the Gowin GW1N-4 (LittleBee) were excluded due to insufficient LUTs resources and limited on chip RAM. In addition, the GW1N-4 was further rejected because of its smaller ecosystem and lack of long term validation in biomedical research, which ultimately left only the Lattice devices as viable options. FPGA size did not play a role in the selection process, as all considered devices were very small, on the order of 2 × 2 mm, making all of them suitable for integration into the NeuroLogger.

The iCE40 UltraPlus (UP3K / UP5K) models offer the best low power option. They include a small number of DSP blocks and extended SRAM, however, this comes at a significant cost. Their internal non volatile configuration memory is one time programmable (OTP), meaning it can be written only once and cannot be erased or reprogrammed. This makes firmware updates impractical unless additional design changes are made, such as adding an external SPI flash and implementing a boot mechanism, which complicates the design and is not well suited for rapid prototyping. Apart from this important limitation, the iCE40 UltraPlus family represents the best option purely from a specification standpoint [30, 31, 32, 33].

Based on the design requirements, the most suitable FPGA choices are the MachXO2-4000 and the MachXO3LF, with the MachXO3LF being the newer model. Both devices meet all requirements needed for our system, and the newer model is even slightly cheaper.

Ultimately, the choice between these two devices was a trade off between marginally improved specifications, mainly related to power consumption, and a platform that has been available for many years, poses less risk of compatibility issues with the existing code base, and is generally very well supported through documentation and community resources. For these reasons, the MachXO2-4000 (LCMXO2-4000HC-6MG132C) was selected as the FPGA for the NeuroLogger. It fulfills all design requirements while providing confidence in long term support and compatibility. If more time were available and a fully working prototype already existed, the newer MachXO3LF would likely be the better choice.

4
FPGA Implementation

4.1. System Architecture

The complete neural signal processing system integrates multiple functional blocks organized around a central Wishbone bus infrastructure running on a Lattice XO2-4K FPGA. The architecture follows a master-slave model where the RISC-V CPU core serves as the primary bus master during configuration and control operations, while the SD card controller operates as a secondary master during DMA transfers. All other peripherals function as slaves, responding to bus transactions initiated by masters.

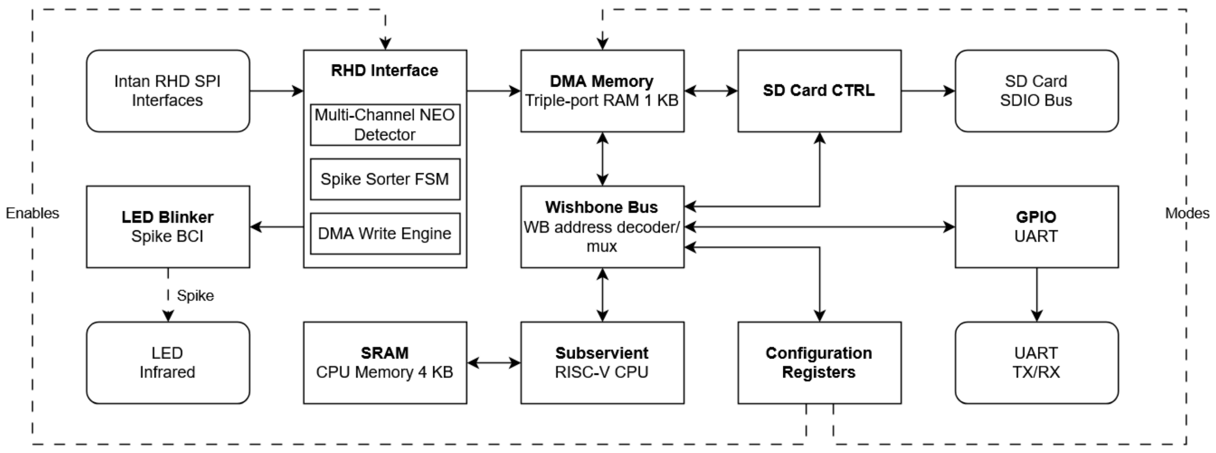


Figure 4.1: Top-level system architecture diagram showing the integration of RHD interface with multi-channel NEO detector, DMA memory, SD card controller, Wishbone bus, CPU, and peripheral interfaces.

The system implements real-time spike detection and classification for up to 64 neural recording channels using time-multiplexed processing. A single hardware pipeline processes samples from all channels sequentially, with each channel maintaining independent state while sharing computational resources. This approach reduces combinational logic (LUT) utilization by 64-fold compared to replicating detection logic per channel, enabling the design to fit within the target FPGA where 64 independent detectors would exceed available resources by more than sixfold. The system maintains deterministic real-time performance at 30 kHz per channel, corresponding to 1.92 MHz aggregate sample rate across all channels.

The major functional blocks include the CPU subsystem for configuration and control, the RHD interface module containing spike detection and classification logic, a triple-port DMA memory for buffering samples and events, an SD card controller for data logging, GPIO and UART peripherals for communication, and configuration registers for runtime control. Data flows from the Intan headstage through the RHD interface where spike detection occurs, into DMA memory via direct writes, and ultimately to the SD card through bulk transfers coordinated by the CPU.

The system supports configurable multi-channel acquisition through two modes selectable via a hardware pin. In 64-channel mode, a single RHD64 headstage provides 64 simultaneous channels optionally using dual-MISO signaling where two data lines interleave samples from alternating channels. In dual 32-channel mode, two independent 32-channel headstages share the processing pipeline, with each headstage connecting to separate SPI interface pins while sharing the chip-select signal for synchronized transactions. Both modes leverage time-multiplexed processing where samples arrive in round-robin order, allowing efficient hardware utilization.

4.2. Clock Generation and Distribution

The Lattice XO2-4K incorporates an internal oscillator configured to generate a 38 MHz system clock without requiring external components or calibration circuits. This frequency derives from several competing constraints that must be simultaneously satisfied. The Intan RHD64 SPI interface supports up to 20 MHz SCLK for command and data exchanges. The system clock must run at least twice this frequency for reliable dual data rate sampling where both rising and falling edges capture data, establishing a minimum requirement of 40 MHz. Neural signals digitize at 30 kHz per channel with 64 channels time-multiplexed at an aggregate rate of 1.92 MHz. The system clock of 38 MHz provides 1267 clock cycles between successive samples from the same channel, which proves sufficient time for the multi-stage processing pipeline to complete all operations for that channel.

The oscillator instantiation uses the OSCH primitive provided by the Lattice library, configured with a nominal frequency parameter specifying 38.00 MHz. The standby input ties to logic zero, keeping the oscillator always active without power-down capability. The oscillator achieves plus or minus 5 percent accuracy across the full temperature range of negative 40 to positive 85 degrees Celsius and voltage range of 1.14 to 1.26 volts, meeting requirements for neural recording applications where absolute frequency precision is less critical than stability during individual recording sessions.

The system employs synchronous reset asserted at power-up and deasserted synchronously with the clock to initialize all state elements to known values. The top-level module receives reset tied to logic zero, relying on FPGA power-on reset to initialize all flip-flops to defined states during the initial power application. The Lattice FPGA guarantees POR assertion for at least 100 microseconds after the power supply stabilizes to within specification, providing sufficient time for reliable startup across all process corners and environmental conditions. Individual modules may assert internal resets based on configuration registers, allowing selective reset of subsystems without restarting the CPU core. The RHD interface combines the external reset with an inverted enable signal, holding the interface in reset when disabled through the configuration register. This selective reset capability is useful for recovery from error conditions where reinitializing the spike detector without disturbing CPU execution allows the system to resume operation after transient faults without losing accumulated data or state in other subsystems.

4.3. CPU Subsystem and Memory Architecture

The CPU core implements the RISC-V RV32I instruction set architecture with minimal extensions, targeting resource-constrained applications where full-featured processors would consume excessive FPGA resources. The Subservient core designed by Olof Kindgren provides a 32-bit data path with all registers and ALU operations operating on 32-bit values. The core implements sixteen general-purpose registers from x0 through x15, representing a subset of the full RISC-V specification which defines x0 through x31. Register x0 remains hardwired to zero as required by the RISC-V specification. The Harvard architecture maintains separate instruction and data memory spaces, simplifying memory controller design and eliminating structural hazards between instruction fetch and data access. The core executes instructions in a single cycle excluding memory access operations, avoiding the complexity and resource overhead of pipelined execution with its associated forwarding logic and hazard detection. An optional WITH_CSR parameter enables interrupt support through control and status registers, though the current configuration disables this feature to minimize resource utilization since the system uses polling-based wake mechanisms rather than traditional interrupt handling.

The CPU accesses program and data memory through a byte-wide interface where memory operations for 32-bit words require four consecutive byte accesses with sequential addresses. This design trades performance for reduced memory port width, which proves critical on FPGAs where wide memories consume more embedded block RAM resources. Each 32-bit load or store decomposes into four single-byte transactions, quadrupling memory access latency but allowing the use of simple 8-bit wide memory banks. The memory module implements single-port RAM initialized with program code from a hexadecimal file generated during firmware compilation. During synthesis, the FPGA tools read this file containing assembled RISC-V machine code in ASCII hexadecimal format with one byte per line, initializing RAM contents to hold the complete firmware image. Program execution begins at address 0x0000_0000 immediately after reset deassertion, with the CPU fetching its first instruction from this location.

The 4 kilobyte memory address space partitions into distinct regions optimized for typical embedded software organization. Program code occupies the text section from addresses 0x0000_0000 through 0x0000_07FF, providing 2 kilobytes for executable instructions. Read-only data including string constants and lookup tables reside in the rodata section from 0x0000_0800 through 0x0000_0BFF, allocating 1 kilobyte for data that requires read access but never modification. Initialized variables occupy the data section from 0x0000_0C00 through 0x0000_0EFF, providing 768 bytes for global and static variables with initial values specified in the source code. The stack grows downward from address 0x0000_0FFF through 0x0000_0F00, allocating 256 bytes for function call frames, local variables, and temporary storage. This organization places the stack at the highest addresses, allowing it to grow downward without colliding with static data as the program executes, with stack overflow detection possible by monitoring accesses below the stack base address.

Table 4.1: Program Memory Layout

Start Address	End Address	Section	Size
0x0000_0000	0x0000_07FF	Code (.text)	2 KB
0x0000_0800	0x0000_0BFF	Read-only Data (.rodata)	1 KB
0x0000_0C00	0x0000_0EFF	Data (.data)	768 B
0x0000_0F00	0x0000_0FFF	Stack	256 B (grows downward)

The CPU supports clock gating for power reduction during idle periods through a deep sleep mechanism. When the configuration register asserts the deep sleep signal, the clock enable input deasserts, halting instruction execution while maintaining register contents in their current state. Memory also receives the same clock enable, preventing unnecessary read and write cycles that would consume dynamic power without performing useful work. The CPU and memory remain clocked during deep sleep to guarantee proper wake-up synchronization, but cease activity as the clock signal continues toggling without state changes occurring.

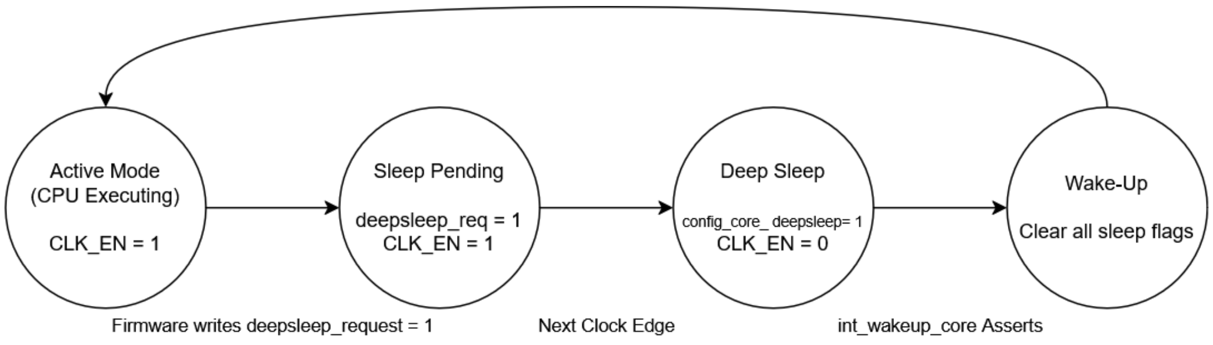


Figure 4.2: Deep Sleep State Machine Diagram.

Deep sleep entry follows a carefully organized sequence to guarantee clean power-down without corrupting ongoing operations. Firmware writes the value one to the configuration register at offset 0x14, setting the deep sleep request flag. The configuration module sets the deep sleep output high on the next clock edge. The CPU clock enable deasserts on the following clock cycle, and the CPU halts after completing its current instruction with all architectural state preserved. This one-cycle delay between the request and actual sleep makes sure the CPU write transaction completes before the clock stops, preventing the acknowledge signal from failing to reach the CPU which would cause firmware to hang waiting for write completion.

Wake-up occurs via interrupt assertion from the RHD interface when buffer thresholds trigger. The RHD buffer reaches half-full or full threshold and asserts the interrupt output signal. The configuration module detects this wake-up request through a direct connection from the interrupt output. On the next clock edge, the configuration module clears the deep sleep signal, the CPU reset signal, and the deep sleep request register atomically. The CPU clock enable reasserts immediately and execution resumes from the halted state without executing any additional instructions. CPU firmware then reads the interrupt status register to determine the wake reason, allowing appropriate buffer management actions.

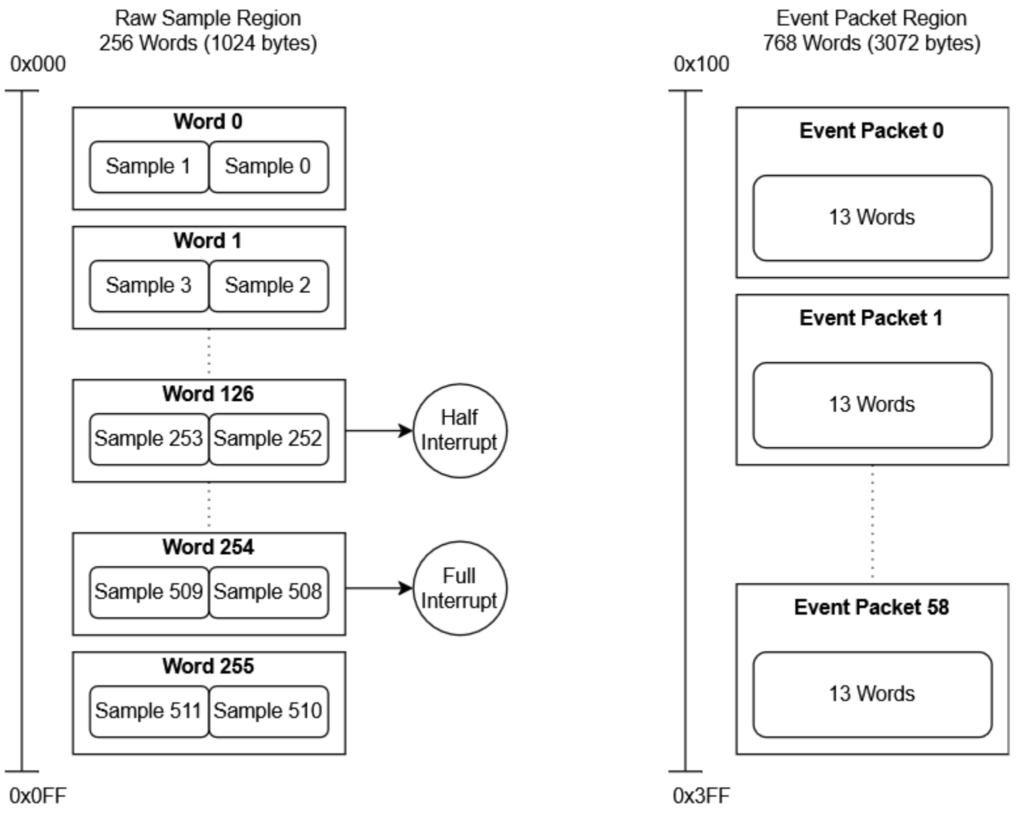


Figure 4.3: DMA memory partitioning for raw samples and spike events.

4.4. Wishbone Bus Infrastructure

The system employs a 4-to-1 Wishbone multiplexer to route transactions from the CPU master to four slave peripherals, with address decoding determining the target based on the upper address bits. The multiplexer examines address bits 29 and 28 to select among four address regions, each spanning 256 megabytes of the 32-bit address space. Addresses with bits 00 route to the SD card controller at base address 0xC0000000, providing access to command registers, status flags, and data buffers. Addresses with bits 01 select the GPIO and UART peripheral at 0xD0000000, exposing data registers, direction control, and transmit-receive buffers. Addresses with bits 10 access DMA memory at 0xE0000000, implementing the shared buffer for CPU, SD DMA controller, and RHD sensor writes. Addresses with bits 11 connect to configuration registers at 0xF0000000, providing memory-mapped control and status monitoring.

Table 4.2: RISC-V CPU Address Space (32-bit)

Start Address	End Address	Region	Description
0x0000_0000	0x0000_0FFF	Program Memory	On-chip memory (4 KB total)
0x0000_1000	0xBFFF_FFFF	Unmapped	Reserved / not implemented
0xC000_0000	0xCFFF_FFFF	SD Card Controller	Command registers, status, data buffer Address[29:28] = 2'b00
0xD000_0000	0xDFFF_FFFF	GPIO / UART	GPIO data, direction, UART TX/RX Address[29:28] = 2'b01
0xE000_0000	0xEFFF_FFFF	DMA Memory	Shared memory for CPU, SD DMA, RHD sensor Address[29:28] = 2'b10
0xF000_0000	0xFFFF_FFFF	Configuration Registers	Memory-mapped control and status registers Address[29:28] = 2'b11

Address decoding uses simple wire assignments without registers, implementing combinational routing with zero latency for properly timed peripherals. Each address region decodes through a two-bit equality comparison, generating enable signals that gate the strobe to individual peripherals. The SD controller select signal combines the master strobe with the condition that address bits 29 through 28 equal binary 00, ensured that the controller observes transactions only within its assigned region. Similar logic generates selects for GPIO at 01, DMA memory at 10, and configuration registers at 11. This structure guarantees that each peripheral sees only its own strobe and never observes transactions intended for other devices, providing clean signal isolation without requiring complex arbitration.

Return data multiplexing selects the active slave response using a priority encoder structure where earlier cases take precedence if multiple conditions somehow become true simultaneously. The multiplexer first checks whether the SD controller region was selected, returning its read data if so. Otherwise it checks GPIO, then DMA memory, then configuration registers in sequence, with a final default returning all zeros for invalid addresses. This priority encoder guarantees predictable behavior even in the presence of decode errors or simultaneous assertions that should never occur during correct operation. The final zero return for unmapped addresses prevents the CPU from receiving undefined data, simplifying firmware debugging by making accesses to invalid regions obvious through repeated zero reads.

Acknowledge signals from all slaves combine through OR logic since exactly one peripheral responds per transaction during normal operation. The master samples the combined acknowledge along with read data, completing the transaction when any slave asserts its acknowledge output. This OR combination works correctly because the address decoder makes sure that only one slave receives strobe at any time, guaranteeing at most one acknowledge assertion. For invalid addresses where no slave responds, the acknowledge remains deasserted indefinitely, potentially hanging the CPU unless firmware implements timeout mechanisms.

Transaction timing follows standard Wishbone protocol with well-defined handshaking between master and slave. The master asserts strobe, provides valid address, and sets write enable appropriately for the operation type. The address decoder generates peripheral select signals combinationally within the same cycle. Selected peripherals receive strobe and begin processing the request according to their internal timing requirements. Fast peripherals including configuration registers, DMA memory, and GPIO respond within one cycle by asserting acknowledge immediately. The SD card controller exhibits variable latency depending on card state, ranging from one cycle for register access to over one thousand cycles for operations requiring card communication.

The master holds strobe and address stable while waiting for acknowledge, implementing the Wishbone wait state mechanism where slaves can extend transactions indefinitely until ready. The slave processes the request at its own pace, asserting acknowledge when ready while simultaneously providing valid data for read operations. The master samples read data and acknowledge on the same clock edge, then deasserts strobe to complete the transaction. Write transactions follow identical timing except the master provides write data instead of the slave returning read data, with the write enable signal high to distinguish from reads.

Different peripherals exhibit varying latencies based on their implementation complexity and external dependencies. Configuration registers provide immediate single-cycle response from simple register operations that complete within the clock period. DMA memory also responds in one cycle using dual-port embedded block RAM with registered output, where the read data appears on the cycle following address presentation. GPIO and UART similarly complete in one cycle for basic register reads and writes. The SD card controller shows variable latency from one to over one thousand cycles depending on protocol state, as the controller must sometimes wait for card-ready signals, perform error checking, or buffer data before completing Wishbone transactions. Firmware must account for these delays when accessing SD card registers, implementing timeout mechanisms to detect hung operations and retry or report errors.

4.5. DMA Memory Subsystem

The DMA memory provides simultaneous access from three requesters through a dual-port memory with dynamic arbitration. The CPU requires access for configuration and debugging operations where firmware inspects buffer contents or modifies control structures. The SD card DMA controller performs bulk transfers to move accumulated samples from memory to persistent storage. The RHD sensor interface writes samples and event packets directly to memory as detection occurs. Implementing true three-port memory would require three separate copies of the 1 kilobyte storage, consuming excessive FPGA resources as each port would need independent address decoding, data paths, and control logic. Instead, the system employs dual-port memory with dynamic port assignment where two requesters access simultaneously while the third waits for an available port, reducing resource usage while maintaining adequate throughput for the 30 kHz sample rate across all channels.

The arbitration module implements a control port select mechanism determining how three logical ports map to two physical memory ports. A two-bit configuration register controls the mapping, with four possible modes supporting different operational scenarios. In normal operation with control port select set to 00, port A connects to DMA and port B connects to the sensor interface, allowing SD card writes to occur concurrently with RHD sample writes for continuous recording without blocking. When control port select equals 01, port A switches to CPU access while port B remains on sensor, enabling firmware to inspect buffer contents during active recording for debugging or status monitoring. Setting control port select to 10 routes port A to DMA and port B to CPU, allowing buffer inspection while SD card reads progress for diagnostics during data transfer operations. The mode 11 duplicates mode 00 behavior, providing a default safe state that matches normal operation.

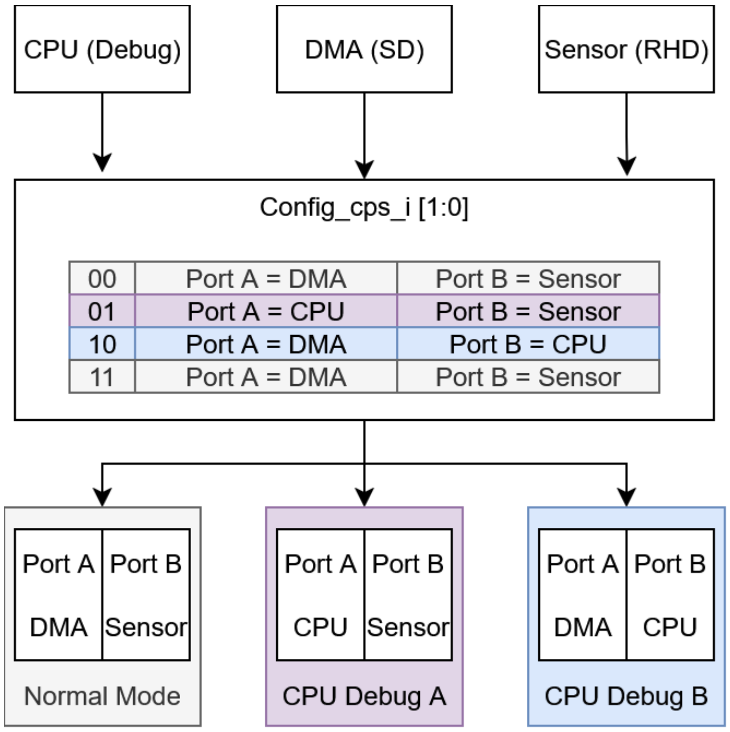


Figure 4.4: DMA Port Arbitration State Diagram.

The arbitration logic maintains two control flip-flops tracking current port ownership, with each flag indicating whether that port currently serves the CPU or its default requester. These flags implement sticky arbitration where once a port grants to a requester, it remains assigned until the transaction completes as indicated by strobe deassertion. The condition checking whether the currently assigned requester has released the port examines the strobe signal, with low indicating the requester finished its access and the port becomes available for reassignment. Additional logic prevents the CPU from monopolizing both ports simultaneously, which would deadlock DMA or sensor access by leaving no ports available for continuous streaming operations. If the CPU requests port A while already owning port B, port A remains assigned to its current requester until the CPU releases port B, making sure that at least one port always remains available for time-critical sample streaming.

Signal routing multiplexers direct inputs to physical ports based on control flags, selecting between requesters according to current ownership. Port A receives either CPU or DMA signals depending on the control flag state, while port B selects between CPU and sensor. The multiplexers examine all relevant signals including address, data, write enable, byte select, and strobe, routing the appropriate requester's values to the physical port inputs. Response routing directs acknowledgments and read data back to the appropriate requester through demultiplexers that reverse the input path. Invalid requesters that do not currently own either port receive the distinctive pattern 0xBADBAD as read data rather than stale values, making firmware errors obvious during development when the CPU attempts reads without proper port ownership.

The underlying storage uses four instances of 8-bit dual-port RAM to create a 32-bit Wishbone-compatible memory totaling 1 kilobyte capacity organized as 256 32-bit words. Each byte lane connects to both ports but only activates when the corresponding byte select signal asserts, enabling Wishbone byte-enable functionality for partial word writes where individual bytes update independently. The Lattice synthesis tool automatically maps these memory descriptions to physical embedded block RAM primitives, optimizing placement and routing for minimum latency. Both ports can perform simultaneous operations including combinations of port A write with port B read, both ports reading different addresses, or both ports writing different addresses. The only undefined case is simultaneous writes to the same address, where behavior depends on vendor-specific EBR implementation details and proper system design avoids this scenario through careful port assignment and access scheduling.

Each dual-port RAM instance implements synchronous read and write with separate address, data, and control ports for independent operation. Port A operations execute when the cycle enable signal asserts, with write enable determining whether the operation stores data or retrieves it. Write operations update memory contents on the clock edge, while read operations present data on the following clock edge after address setup. Port B follows identical timing with its own independent control signals, allowing truly simultaneous access where both ports perform operations in parallel without interference. The memory array itself consists of registers rather than distributed RAM, guaranteeing predictable timing across synthesis and place-and-route optimizations.

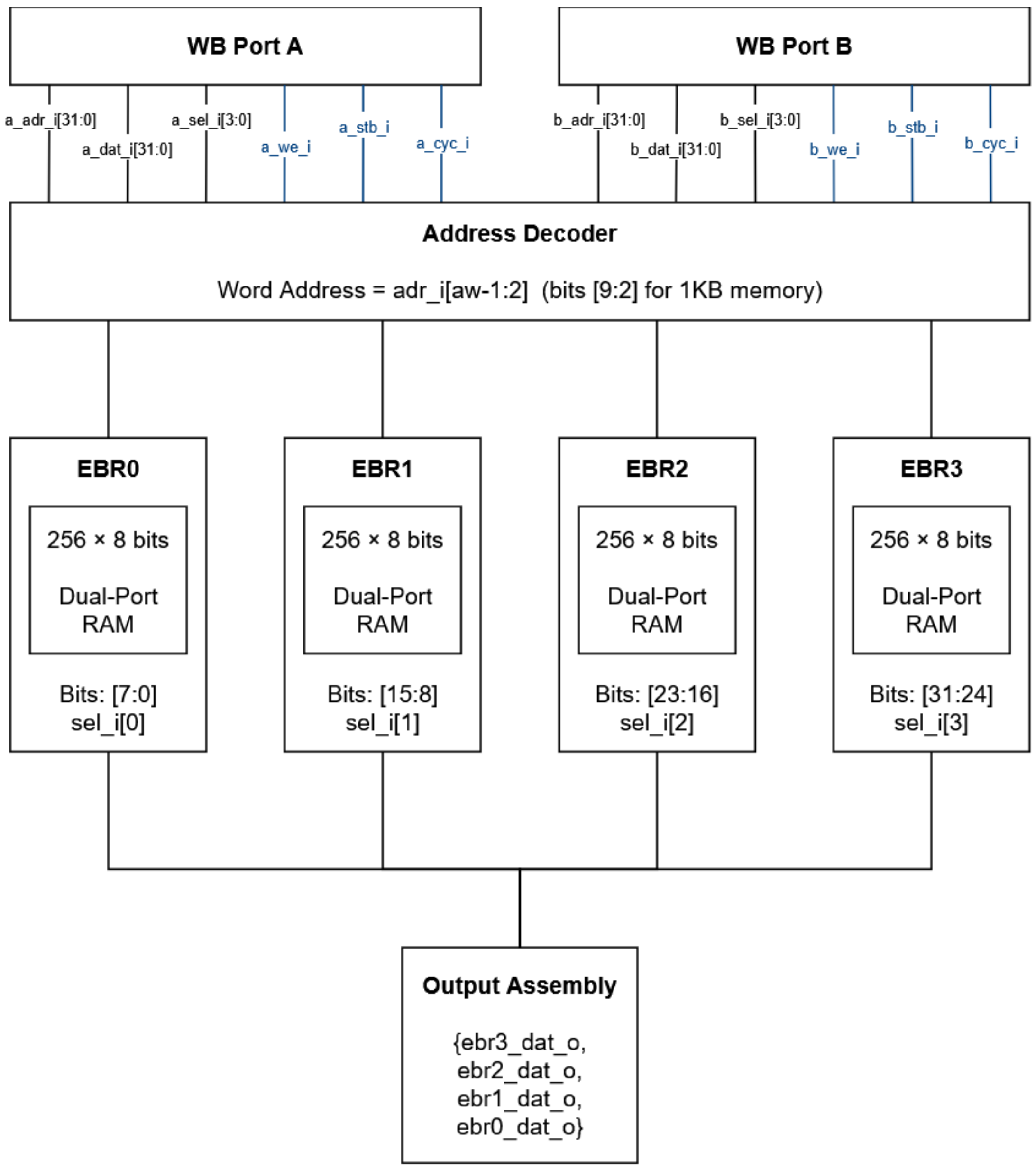


Figure 4.5: Dual-port memory architecture with EBR specifications.

Acknowledge generation occurs one cycle after strobe assertion, creating the single-cycle read latency characteristic of embedded block RAM. The memory controller registers the acknowledge signal synchronously with the clock, asserting it whenever both cycle enable and strobe signals are high. This registered acknowledge allows for proper Wishbone protocol compliance and provides time for the embedded block RAM read operation to complete, with data and acknowledge appearing on the same cycle for the master to sample together. For write operations, the acknowledge signals completion but the data path carries no meaningful information, following standard Wishbone conventions.

The memory partitions into two regions supporting configurable logging modes that determine which data writes to DMA memory during recording sessions. The raw region occupies addresses 0 through 255 as a 256-word circular buffer receiving continuous raw ADC samples. The write pointer increments with each sample, wrapping from 255 back to 0 to implement the circular structure. Interrupts trigger at half-buffer offset 127 and full-buffer offset 255, allowing double-buffering where firmware or SD controller reads one half while the RHD writes the other half. The event region spans addresses 256 through 1023, storing spike event packets when the detector identifies action potentials. Each packet contains 13 words organized as a magic number for synchronization, a 32-bit sample timestamp capturing when detection occurred, metadata encoding channel identifier and classification result, and 40 waveform samples packed as four bytes per word across ten words for neural network classification.

Three logging modes control which data writes to DMA memory based on application requirements. Mode 00 designated LOG_MODE_FULL writes only raw samples to the circular buffer, supporting continuous full-bandwidth recording at the sample rate across all channels for offline analysis where spike detection runs in post-processing. Mode 01 designated LOG_MODE_SPIKES writes only event packets when spikes detect, dramatically reducing data volume for sparse firing patterns where most samples contain only noise and baseline activity. Mode 10 designated LOG_MODE_COMBINED writes both raw samples and events concurrently, allowing offline comparison between detected events and underlying raw traces for algorithm validation and parameter tuning. The mode selection comes from a configuration register the CPU sets during initialization, remaining fixed throughout each recording session.

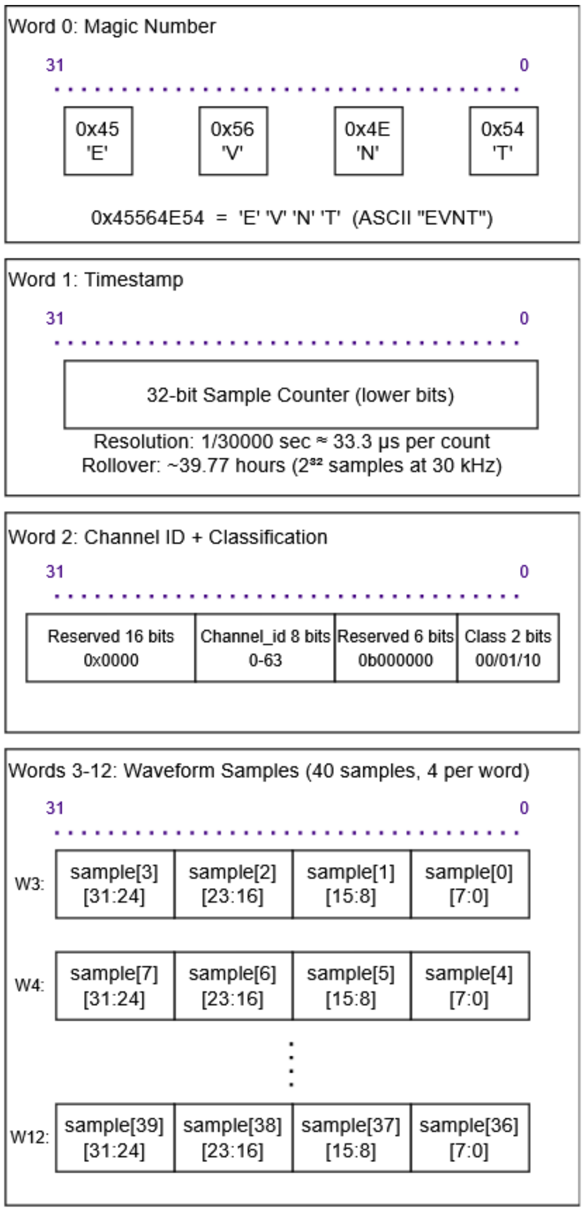

Word 0: Magic Number
31
0
0x45 'E'
0x56 'V'
0x4E 'N'
0x54 'T'
0x45564E54 = 'E' 'V' 'N' 'T' (ASCII "EVNT")
Word 1: Timestamp
32-bit Sample Counter (lower bits)
Resolution: 1/30000 sec ≈ 33.3 μs per count
Rollover: ~39.77 hours (2^{32} samples at 30 kHz)
Word 2: Channel ID + Classification
Reserved 16 bits 0x0000
Channel_id 8 bits 0-63
Reserved 6 bits 0b000000
Class 2 bits 00/01/10
Words 3-12: Waveform Samples (40 samples, 4 per word)
W3: sample[3] [31:24] sample[2] [23:16] sample[1] [15:8] sample[0] [7:0]
W4: sample[7] [31:24] sample[6] [23:16] sample[5] [15:8] sample[4] [7:0]
W12: sample[39] [31:24] sample[38] [23:16] sample[37] [15:8] sample[36] [7:0]

Figure 4.6: Spike event packet binary format (52 bytes total).

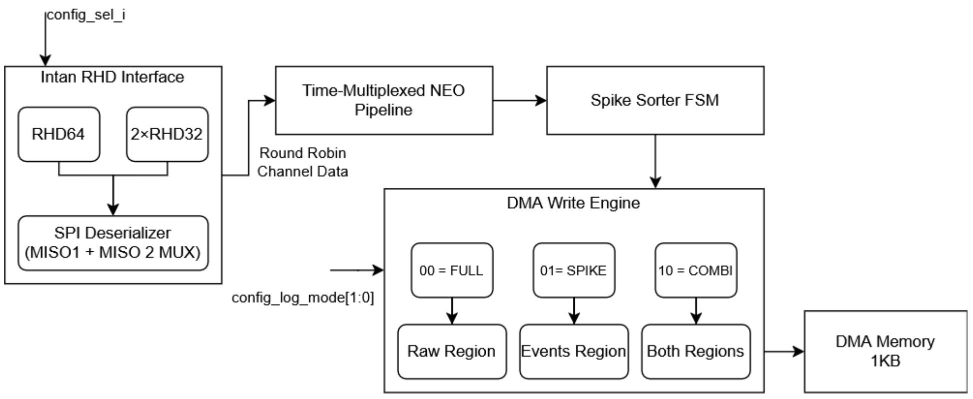


Figure 4.7: Multi-channel data flow from headstage to DMA memory.

4.6. Configuration Register Module

The configuration module provides memory-mapped register interface for system control and status monitoring through write-only configuration registers and read-only interrupt status accessible via Wishbone protocol. Register addresses decode using bits 4 through 2 of the incoming address, providing eight distinct locations in the 32-byte window allocated to configuration space. Writing to offset 0 sets the DMA control port select determining memory arbitration mode as described in the previous section. Offset 4 enables or disables the RHD interface, with the value zero holding the interface in reset and value one activating spike detection. Offset 8 controls the SD card controller enable, allowing firmware to power down the SD interface when not logging. Offset 12 asserts CPU reset when written with one, holding the processor in reset state until cleared, useful for firmware-initiated system restart. Reading offset 16 returns the interrupt reason code indicating whether half-buffer or full-buffer threshold triggered the wake-up. Offset 20 requests deep sleep mode through the mechanism described previously. Offset 24 enables the second MISO signal for dual 32-channel acquisition mode. Offset 28 configures the logging mode, selecting among full, spikes, or combined recording strategies.

Table 4.3: Configuration Register Space (0xF000_0000 – 0xF000_001F)

Address	Offset	Register	Bits	Access / Description
0xF000_0000	+0x00	`config_dma_cps`	[1:0]	W DMA Control Port Select Remaining bits [31:2] reserved
0xF000_0004	+0x04	`config_rhd_en`	[0]	W RHD Interface Enable 0 = disabled (held in reset) 1 = enabled (active)
0xF000_0008	+0x08	`config_sd_en`	[0]	W SD Card Controller Enable 0 = disabled 1 = enabled
0xF000_000C	+0x0C	`config_core_rst`	[0]	W CPU Core Reset 0 = normal operation 1 = held in reset
0xF000_0010	+0x10	`int_reason`	[7:0]	R Interrupt Reason Code 5 = half-buffer threshold 6 = full-buffer wrap Remaining bits [31:8] reserved
0xF000_0014	+0x14	`deepsleep_request`	[0]	W Request Deep Sleep Mode 0 = normal operation 1 = request sleep (propagates next cycle)
0xF000_0018	+0x18	`config_en_miso2`	[0]	W Enable Second MISO Line 0 = single MISO (32-channel mode) 1 = dual MISO (64-channel mode)
0xF000_001C	+0x1C	`config_log_mode`	[1:0]	R/W Logging Mode Select 00 = LOG_MODE_FULL (raw samples only) 01 = LOG_MODE_SPIKES (events only) 10 = LOG_MODE_COMBINED (both) 11 = Reserved

Write operations execute when the CPU presents a transaction with strobe, write enable, and byte select all asserted, indicating a valid store to the configuration space. A case statement decodes address bits 4 through 2, selecting the target register among eight possibilities. The written data latches into the appropriate configuration output on the clock edge, updating the corresponding control signal that feeds other system modules. The module acknowledges all writes in a single cycle since register updates complete immediately through simple assignment statements. Read operations from the interrupt status register return the reason code in the lower byte with upper bits zero-padded, also acknowledging in one cycle through combinational logic that presents the status value without requiring memory access.

Deep sleep involves a multi-step sequence enabling clean power-down without disrupting ongoing operations or losing critical state. Firmware writes one to the deep sleep request register at offset 20, setting an internal flag that will propagate to the actual sleep signal. On the next clock edge after the write completes, the actual deep sleep signal asserts, propagating to the CPU clock enable input where it halts execution. This one-cycle delay makes sure that the CPU write transaction finishes before the clock stops, preventing the acknowledge signal from failing to reach the CPU which would cause firmware to hang waiting for write completion that never arrives.

Wake-up proceeds when the RHD interface detects a buffer threshold and asserts its interrupt output, which connects directly to the configuration module wake input. On the next clock edge, the configuration module clears the deep sleep signal, the CPU reset signal, and the deep sleep request register atomically in a single assignment. The CPU clock enable reasserts and execution resumes from the halted state, with the program counter pointing to the instruction following the infinite loop that firmware entered before sleeping. Firmware reads the interrupt status register to determine the wake reason encoded as an 8-bit value, allowing appropriate buffer management actions based on which threshold triggered.

The interrupt reason code distinguishes between two buffer conditions that require different firmware responses. Code 5 indicates half-buffer threshold where the DMA buffer index crossed 127, meaning the first 128 words contain valid data ready for SD card transfer. Firmware responds by initiating a DMA write of these words while the RHD continues writing to the second half. Code 6 indicates full-buffer wrap where the DMA buffer index wrapped to 0, meaning the second 128 words require transfer while the first half may receive new writes. This double-buffering scheme allows continuous recording without data loss, as while the RHD writes to one half, firmware or SD controller reads from the other half. The buffers alternate roles on each interrupt, with careful timing allowing that the read completes before the write pointer wraps into that region again.

4.7. Intan RHD Interface and SPI Protocol

4.7.1. SPI Interface

The RHD interface module connects the Intan RHD headstage to the internal spike detection and processing pipeline. It implements the Intan-specific SPI command response protocol used to configure the analog front end and to acquire digitized neural samples. Each SPI transaction transmits a 16-bit command word and receives a corresponding 16-bit ADC sample on the MISO line(s), with the returned sample being the result of a command issued two transactions earlier, as defined by the Intan protocol. The SPI controller is implemented as a simple two-state finite state machine consisting of an idle and a sending state. In the idle state, the controller waits for an input valid signal from firmware or control logic indicating that a new command word has been loaded. Upon assertion of this signal, the controller transitions to the sending state, where it shifts out the command word on MOSI while simultaneously sampling the MISO line(s). This bidirectional shifting allows command transmission and data acquisition to occur within the same SPI transaction. After the transfer completes, an output valid pulse is asserted for one cycle to indicate that a new ADC sample is available for downstream processing, after which the controller returns to the idle state.
The interface supports both single 64-channel and dual 32-channel acquisition configurations. In the 64-channel configuration, dual-data-rate (DDR) signaling is used, allowing two channels to be sampled per SPI clock period by capturing data on both rising and falling edges. In contrast, the dual 32-channel configuration operates without DDR. Instead, the same command word is broadcast simultaneously to both ADC chips, causing the same channel index on each device to be sampled in parallel. The resulting ADC samples are returned concurrently on separate MISO lines, providing two independent measurements per transaction while maintaining the same SPI clock frequency.

4.7.2. Multi-Headstage Support

The system supports a dual 32-channel headstage configuration using two separate ADC chips rather than a single 64-channel device. Since each ADC chip provides only a single reference potential, this approach enables the use of separate reference electrodes for spatially separated brain regions. Referencing each electrode group locally improves measurement accuracy and increases flexibility for multi-region electrophysiology experiments.
A hardware configuration pin selects the active acquisition topology by routing SPI signals through combinational multiplexers. Inactive SPI outputs are explicitly driven to logic zero rather than tri-stated. This prevents floating pins, which could otherwise lead to noise injection or parasitic coupling through input protection structures such as ESD diodes. The chip-select signal is shared between headstages to allow for synchronized command framing and data acquisition in dual-headstage operation.

4.7.3. Simulation Support

For simulation and testing, the module includes a direct stimulus mode in which a parameterized hexadecimal file replaces the SPI interface. Recorded neural data is replayed as 16-bit samples and presented to the processing pipeline in the same format as hardware-acquired data. This allows the signal processing algorithms to be tested without physical hardware, accelerating development and early validation. The stimulus mode is enabled through synthesis-time parameters, while the downstream processing logic remains unchanged, making sure that results obtained in simulation accurately reflect hardware operation.
A simple LED indicator is used to provide visual feedback during operation for debugging and demonstration purposes. During normal recording, the LED blinks steadily, while detected spikes cause a faster blinking pattern. This makes it easy to verify correct system operation and spike detection activity without external measurement equipment. The LED logic operates independently of the main processing blocks and uses only a small amount of hardware resources.

4.8. NEO Spike Detection Algorithm

The Nonlinear Energy Operator provides superior spike detection performance compared to simple amplitude thresholding by emphasizing transient high-frequency events characteristic of action potentials while suppressing slower baseline variations and correlated noise. For a discrete-time signal, the NEO computes the instantaneous energy approximation through the formula combining the square of the current sample with the product of adjacent samples. The mathematical justification derives from continuous-time energy operators for signals of the form $s(t) = A(t)\cos(\omega t + \phi(t))$, where the energy operator tracks $|A(t)|^2 \cdot \omega^2$. For neural signals, which exhibit amplitude modulation during action potentials, the NEO emphasizes spike events. The algorithm is based on the work of Yang and Mason [34].

For a discrete-time signal $x[n]$, the NEO is defined as:

$$\Psi[x[n]] = x[n]^2 - x[n-1] \cdot x[n+1] \tag{4.1}$$

The NEO offers several advantages over traditional amplitude-based detection methods. Baseline independence arises from the differencing operation where slow drift in recording baseline contributes minimally to NEO output, as the product of adjacent samples cancels low-frequency components. Noise suppression occurs because uncorrelated noise produces near-zero NEO values on average, with white noise samples fluctuating randomly such that the adjacent sample product approximates the current sample squared in expectation. Frequency selectivity emerges from the operator implicitly weighting signal components by frequency squared, amplifying high-frequency spike transients relative to low-frequency biological noise including local field potentials and electrode drift. Computational simplicity enables implementation with only two multiplications and one subtraction per sample, feasible for resource-constrained hardware without dedicated DSP blocks.

The complete detection pipeline comprises seven cascaded stages processing samples at 30 kHz rate. ADC quantization converts 16-bit unsigned samples to 14-bit signed representation through arithmetic right-shift. IIR lowpass filtering smooths the input signal, attenuating noise above 1.4 kHz. Baseline removal tracks and subtracts slow DC drift using an exponential moving average with large time constant. Dynamic range adaptation scales signals per-channel to optimize quantization SNR. NEO calculation computes the energy operator using three-sample delay line. NEO smoothing applies lowpass filtering to reduce variance before threshold comparison. Adaptive threshold comparison detects events exceeding the computed threshold with refractory period enforcement preventing multiple triggers on single spikes.

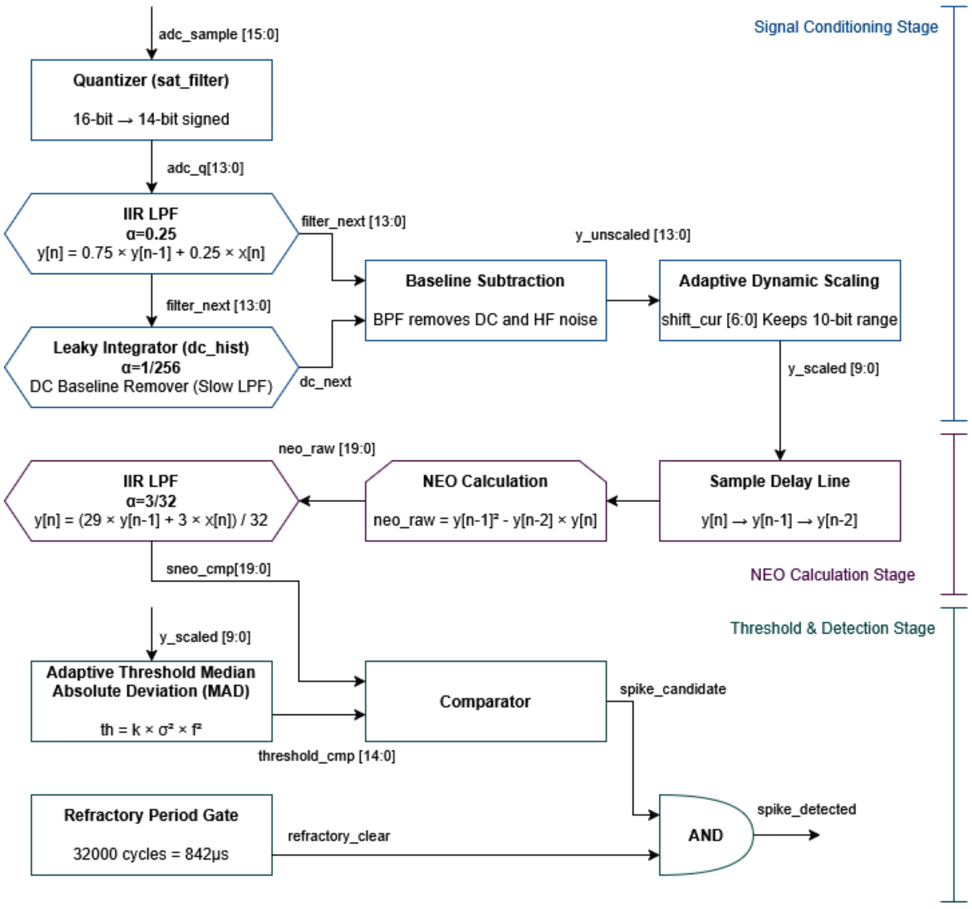


Figure 4.8: NEO Detector Pipeline.

The architecture supports configurable channel counts via parameter, with each channel maintaining independent state for filter history, NEO delay line, dynamic scaling factor, and refractory status stored in Block RAM. The combinational logic including multipliers, adders, and comparators is shared across all channels through time-division multiplexing. When sample-valid asserts with a specific channel identifier, a three-stage pipeline retrieves that channel's state from Block RAM, computes updated values using shared resources, and writes results back to Block RAM. This architecture achieves O(log N) scaling in LUT usage with channel count through shared address decoders, compared to O(N) scaling if state were implemented in distributed RAM or registers. For 64 channels, the packed state approach reduces resource utilization by 65% compared to per-channel register arrays.

4.8.1. ADC Sample Quantization and Preprocessing

The Intan RHD64 headstage outputs 16-bit unsigned ADC samples centered around mid-scale corresponding to zero volts differential input. The preprocessing pipeline first quantizes to 14-bit signed through the IIR filter stage, then dynamically scales to 10-bit for NEO computation. The initial conversion interprets the 16-bit unsigned value as signed two's complement by applying the signed cast operator, then performs arithmetic right-shift by 2 positions to divide by 4 while preserving sign. Saturation logic clamps the result to the 14-bit signed range of negative 8192 to positive 8191, preventing overflow from propagating through subsequent stages.

The step-by-step conversion proceeds as follows. The ADC produces 16-bit unsigned values spanning 0 to 65535, with mid-scale 32768 representing zero volts. Sign reinterpretation treats this as 16-bit signed spanning negative 32768 to positive 32767, where negative values correspond to voltages below mid-scale. Arithmetic right-shift by 2 divides by 4 to reduce to 14-bit effective width while preserving sign through replication of the sign bit. Saturation clamps values exceeding the 14-bit signed range to prevent wraparound that would corrupt signal polarity.

After IIR filtering and baseline removal, the 14-bit signal undergoes adaptive dynamic scaling to 10-bit final resolution. Each channel maintains a shift parameter ranging from 0 to 6 that controls right-shift amount, with shift equal to 0 providing maximum gain and shift equal to 6 dividing by 64 for minimum gain. The shifted value then saturates to 10-bit signed range of negative 512 to positive 511, optimizing dynamic range utilization for subsequent NEO calculation. This quantization reduces resolution from 16 bits to 10 bits, a 64-fold reduction in precision that proves acceptable because typical neural signals occupy only the central portion of the ADC range.

The Intan datasheet specifies ADC step size referred to electrode of 0.195 microvolts, amplifier input-referred noise of 2.4 microvolts RMS typical, and amplifier gain of 192 volts per volt. Quantization analysis in ADC counts shows that 16-bit to 10-bit reduction introduces quantization noise through the right-shift operations. The quantization step equals 16 counts from the 4-position shift, producing quantization noise of approximately 4.6 counts RMS. Comparing to amplifier noise computed as 2.4 microvolts divided by 0.195 microvolts per count yields approximately 12.3 counts RMS. The combined noise from both sources computes as the square root of sum of squares, yielding approximately 13.1 counts RMS. The quantization adds approximately 12 percent to total noise power, which proves acceptable though not entirely negligible for this application.

RMS Noise Values

Analog (amplifier) noise:

$$\sigma_a = 12.3 \text{ counts RMS}$$

Quantization noise:

$$\sigma_q = 4.6 \text{ counts RMS}$$

Combined noise:

$$\sigma_t = \sqrt{12.3^2 + 4.6^2} = 13.1 \text{ counts RMS}$$

Noise Power Comparison

Since noise power is proportional to the square of the RMS value, the corresponding noise powers are:

Analog-only noise power:

$$P_a = 12.3^2 = 151$$

Quantization noise power:

$$P_q = 4.6^2 = 21$$

Total noise power:

$$P_t = P_a + P_q = 151 + 21 = 172$$

Relative Contributions

Fraction of total noise power due to quantization:

$$\frac{P_q}{P_t} = \frac{21}{172} \approx 12\%$$

Increase in noise power relative to analog-only noise:

$$\frac{P_t - P_a}{P_a} = \frac{21}{151} \approx 14\%$$

4.8.2. Dynamic Range Adaptation

Neural electrode recordings exhibit wide amplitude variability across channels due to multiple factors. Electrode impedance varies from 50 kΩ to 2 MΩ , causing order-of-magnitude amplitude differences for similar neural activity. Neuron proximity affects signal strength, with distant neurons producing 50 microvolt spikes while nearby neurons generate 500 microvolt events. Tissue movement in chronic implants causes drift, changing signal amplitude over days or weeks of recording. Without adaptive scaling, a fixed 14-bit to 10-bit conversion would either saturate high-amplitude channels losing spike peaks, or under-utilize dynamic range on quiet channels losing SNR to quantization noise.

Each channel maintains an independent shift-current value stored in a 3-bit register allowing values from 0 to 6. The shift-current parameter controls right-shift amount applied before 10-bit saturation, effectively implementing a programmable divider. Shift-current equal to 0 provides divide-by-1 with no scaling for maximum gain. Shift-current equal to 1 divides by 2, while shift-current equal to 2 divides by 4 as the default initial value. The maximum shift-current of 6 divides by 64 for minimum gain on very loud channels. The shifted value saturates to 10-bit range, with saturation logic making sure values remain within negative 512 to positive 511 inclusive.

The system continuously monitors signal amplitude and adjusts shift-current automatically through a feedback control loop. Two thresholds determine when adjustment occurs: an upper threshold at 88 percent of full-scale corresponding to 448 in 10-bit representation triggers gain reduction, while a lower threshold at 25 percent corresponding to 128 triggers gain increase. If the signal consistently exceeds the upper threshold, the system increases shift-current to reduce gain and prevent saturation. If signal stays below the lower threshold, the system decreases shift-current to increase gain and improve SNR. After each adjustment, a cooldown period of 128 samples approximately 4.3 milliseconds prevents oscillation by forcing the signal to stabilize before allowing further changes.

```verilog
localparam integer SHIFT_UP_THRESH   = 10'sd448;  // 88% of full-scale
localparam integer SHIFT_DOWN_THRESH = 10'sd128;  // 25% of full-scale

wire shift_up_req   = (cooldown_cur == 0) &&
                      (shift_cur < SHIFT_MAX) &&
                      (abs_shifted > SHIFT_UP_THRESH_VEC);
wire shift_down_req = (cooldown_cur == 0) &&
                      (shift_cur > SHIFT_MIN) &&
                      (abs_shifted < SHIFT_DOWN_THRESH_VEC);
```

The adjustment logic examines the absolute value of the shifted signal against both thresholds. Shift-up request asserts when cooldown equals zero, shift-current remains below maximum, and absolute value exceeds upper threshold, indicating the signal approaches saturation and requires gain reduction. Shift-down request asserts when cooldown equals zero, shift-current exceeds minimum, and absolute value falls below lower threshold, indicating the signal under-utilizes dynamic range and would benefit from gain increase. When either request asserts, the cooldown counter loads with 128 and shift-current adjusts by one step in the appropriate direction. The cooldown then decrements each sample until reaching zero, blocking further adjustments during this settling period.

To prevent oscillation from threshold hysteresis, the counter-based cooldown mechanism forces a minimum time between adjustments. After each shift change, the cooldown counter loads with 128 and decrements each sample. Adjustments block while cooldown remains nonzero, making sure taht at least 128 samples elapse before the next change. This allows the signal statistics to stabilize after each gain change, preventing rapid oscillation between adjacent shift values when signal amplitude hovers near a threshold. The 128-sample cooldown corresponds to approximately 4.3 milliseconds at 30 kHz sampling, comparable to several spike durations and sufficient for amplitude statistics to settle.

Before right-shifting, the implementation adds a rounding bias to implement round-to-nearest instead of truncation. The bias equals one half of the division quantum, computed as 2 raised to the power of shift-current minus 1. For positive numbers, this bias adds before the shift, while for negative numbers it subtracts to maintain symmetric rounding around zero. The conditional logic selects between positive and negative bias based on the sign bit, then adds the bias before performing the arithmetic right-shift. This rounding reduces quantization error compared to simple truncation, particularly important for small signals where truncation would bias results toward zero.

```
reg signed [FILTER_WIDTH-1:0] round_bias;
always @* begin
    case (shift_cur)
        3'd0: round_bias = {FILTER_WIDTH{1'b0}};
        default: round_bias = $signed(1) <<< (shift_cur - 3'd1);
    endcase
end

wire signed [FILTER_WIDTH-1:0] y_with_round =
    (shift_cur != 0) ? (y_unscaled + (y_unscaled[FILTER_WIDTH-1] ?
                        -round_bias : round_bias)) : y_unscaled;
```

Convergence time from default shift-current equal to 2 depends on signal characteristics. Quiet channels requiring gain increase typically converge in 256 to 384 samples spanning 8.5 to 12.8 milliseconds as shift-current decrements from 2 toward 0. Loud channels requiring gain reduction converge faster in 128 to 256 samples spanning 4.3 to 8.5 milliseconds as shift-current increments from 2 toward 6. Channels already at optimal gain require no adjustment and exhibit zero convergence delay. The adaptive scaling runs continuously throughout recording sessions, tracking slow amplitude changes from tissue drift or electrode degradation without requiring manual recalibration.

4.8.3. IIR Lowpass Filter Implementation

The single-pole IIR lowpass filter smooths input samples, attenuating high-frequency noise above the spike bandwidth while preserving temporal structure. The filter implements the difference equation combining 75 percent of the previous output with 25 percent of the current input, expressed as multiply by 3 and divide by 4 to enable efficient fixed-point arithmetic. This represents a first-order recursive filter with pole location at 0.75 providing DC gain of unity and high-frequency rolloff of negative 20 decibels per decade.

Transfer Function:

$$y[n] = \frac{3 \cdot y[n-1] + x[n]}{4} = 0.75 \cdot y[n-1] + 0.25 \cdot x[n] \tag{4.2}$$

Considering the form:

$$y[n] = \alpha\, y[n-1] + (1-\alpha)\, x[n], \qquad 0 < \alpha < 1,$$

with $\alpha = 0.75$. The transfer function is

$$H(e^{j\omega}) = \frac{1-\alpha}{1-\alpha e^{-j\omega}}.$$

Time Constant:

The sample-domain time constant (in samples) is

$$\tau_{\text{samples}} = -\frac{1}{\ln(\alpha)} = -\frac{1}{\ln(0.75)} \approx 3.476 \text{ samples}.$$

Step / impulse responses (index convention matters) The impulse response is

$$h[n] = (1-\alpha)\, \alpha^n\, u[n],$$

and the causal step response (response to $u[n]$) is

$$s[n] = \sum_{k=0}^{n} h[k] = 1 - \alpha^{\,n+1} \quad (n \geq 0),$$

so the 50% rise (step reaching 0.5 of final value) occurs at

$$n_{50\%} = \frac{\ln(2)}{\ln(1/\alpha)} \approx \frac{\ln 2}{\ln(4/3)} - 1 \approx 1.409 \text{ samples}.$$

Phase and group delay

$$1-\alpha e^{-j\omega}=B+jA, \qquad A=\alpha\sin\omega,\ B=1-\alpha\cos\omega,$$

so the phase (from the denominator) is

$$\phi(\omega)=-\arctan\Big(\frac{A}{B}\Big)=-\arctan\Big(\frac{\alpha\sin\omega}{1-\alpha\cos\omega}\Big).$$

Differentiate to obtain the group delay

$$\tau_g(\omega)\ =\ -\frac{d\phi}{d\omega}=\frac{BA'-AB'}{A^2+B^2}=\frac{\alpha\cos\omega-\alpha^2}{1+\alpha^2-2\alpha\cos\omega}.$$

($A'=\alpha\cos\omega,\ B'=\alpha\sin\omega$ and $A^2+B^2=1-2\alpha\cos\omega+\alpha^2$.)

Cutoff Frequency Calculation:

The -3 dB cutoff frequency is:

$$f_c\approx\frac{f_s}{2\pi\tau}=\frac{f_s\cdot\ln(1/\alpha)}{2\pi} \tag{4.3}$$

At 30 kHz sampling rate:

$$f_c\approx\frac{30{,}000}{2\pi\cdot3.47}\approx 1{,}375\text{ Hz} \tag{4.4}$$

Important special cases (for $\alpha=0.75$)

$$\tau_g(0)=\frac{\alpha}{1-\alpha}=\frac{0.75}{0.25}=3.0\text{ samples}\quad\Big(=\frac{3}{30{,}000}=100\ \mu\text{s at } f_s=30\text{ kHz}\Big),$$

$$\tau_g(\pi)=-\frac{\alpha}{1+\alpha}=-\frac{0.75}{1.75}\approx-0.428571\text{ samples},$$

$$\text{At } f_c\approx1374\text{ Hz}:\quad \tau_g(\omega_c)\approx1.2615387\text{ samples}\approx42.05\ \mu\text{s}.$$

The phase response shows frequency-dependent group delay arising from the single pole. At DC, group delay reaches 3 samples corresponding to 100 microseconds, representing maximum delay for slowly-varying signals. At the cutoff frequency near 1.4 kHz, group delay decreases to approximately 1.26 samples or 42 microseconds. At higher frequencies approaching Nyquist, delay becomes very small and slightly negative due to phase characteristics of first-order IIR filters. These results show the pre-filter smooths high-frequency noise without significantly delaying spike events, as the maximum 100 microsecond delay remains well below biologically relevant timescales. The NEO operator receives a waveform with preserved temporal structure, and overall system latency remains dominated by other processing stages rather than this initial filtering.

The fixed-point implementation avoids floating-point arithmetic through careful coefficient selection. The multiply by 3 operation implements as shift-left by 1 to double the value, then add the original value, requiring only one adder beyond the shift which costs zero resources as wire rearrangement. Synthesis tools automatically optimize this pattern, recognizing the multiply by constant 3 structure. Sign extension pads the 10-bit input to 14 bits by replicating the sign bit 4 times, preventing overflow when adding to the 3 times previous output term. The 14-bit sum equals 4 times the final output before division, with maximum value of 3 times 511 plus 511 equals 2044 requiring 11 bits. The 14-bit bus provides adequate headroom for this computation. Division by 4 implements through arithmetic right-shift by 2 positions, completing the difference equation through pure fixed-point operations without any division hardware.

```
assign filter_sum = ($signed(y) * 3) + $signed({{4{adc_q[9]}}, adc_q});
assign filter_next = filter_sum >>> 2;
```

The system achieves effective bandpass behavior through cascaded stages despite individual lowpass responses. The lowpass prefilter smooths input and attenuates noise above 1.4 kHz. The NEO operator acts as implicit highpass through the differencing operation emphasizing rapid transients. The NEO smoothing filter applies lowpass at approximately 470 Hz. The combination produces bandpass characteristics optimized for spike detection, with the NEO operator providing highpass effect through suppression of slow baseline variations. The minimal group delay of approximately 42 microseconds at spike frequencies makes sure that waveforms experience negligible phase distortion. Higher-order filters would provide sharper cutoffs but introduce unacceptable latency for real-time detection where sample-to-detection delay must remain under a few milliseconds.

The filtered output register updates only when sample-valid asserts, providing the current channel's filtered signal to downstream processing. This registered output (filtered_y_r) holds the 10-bit scaled sample for the most recently processed channel, updating once per channel per 64-cycle rotation at the 30 kHz aggregate sampling rate. The register provides stable output to the spike sorter FSM and window capture logic, avoiding combinational glitches during state transitions.

Per-channel filter state (filt_hist) resides in Block RAM as part of the 101-bit packed state vector, with each channel maintaining independent 14-bit filter history. On system reset, all channel states initialize to zero through the STATE_RESET_VALUE constant loaded into Block RAM. The IIR lowpass filter with time constant of 3.5 samples settles within 5 time constants (approximately 17 samples or 570 microseconds per channel) to within 1 percent of steady-state. During threshold calibration in the SORTER_INIT state, this per-channel settling occurs naturally as samples arrive, with channels processing continuously regardless of detection enable state. The filter computation executes unconditionally in the NEO pipeline, updating filt_hist in Block RAM on every sample, though the spike detection comparison gates through the ce_n clock-enable signal controlled by the FSM.

4.8.4. Baseline Removal Through DC Tracking

Electrode drift and local field potentials introduce slow baseline variations that would otherwise corrupt spike detection by shifting the signal mean. A DC tracking filter with very large time constant estimates and removes this baseline, presenting a zero-mean signal to the NEO operator. The tracker implements an exponential moving average with coefficient 255/256, creating an extremely slow filter that tracks baseline drift while rejecting spike transients. The difference equation updates the DC estimate as the previous estimate plus 1/256 times the difference between current sample and previous estimate, equivalent to multiplying previous estimate by 255/256 and adding current sample times 1/256.

The time constant computes as negative one divided by natural logarithm of 255/256, yielding approximately 256 samples. At 30 kHz rate this corresponds to 8.5 milliseconds, much longer than spike duration but short enough to track electrode drift occurring over seconds. The cutoff frequency is approximately 18.6 Hz at 30 kHz sampling. This aggressive lowpass characteristic attenuates all spike energy while passing only very low frequency drift components.

The discrete-time time constantPer-channel DC estimates reside in Block RAM state storage indexed by channel identifier, allowing each channel to track independent baseline drift from different electrode impedances and tissue properties. When sample-valid asserts with a particular channel identifier, the computation retrieves that channel's DC estimate from Block RAM, uses it to remove baseline from the filtered signal, computes an updated estimate, and writes the new value back to Block RAM for the next sample. This per-channel tracking proves essential for multi-channel recordings where different electrodes exhibit vastly different DC potentials and drift rates.

$$\tau = -\frac{1}{\ln(\alpha)},$$

which for $\alpha = \frac{255}{256}$ yields

$$\tau \approx 256 \text{ samples.}$$

At a sampling rate of $f_s = 30\ \mathrm{kHz}$, this corresponds to a time constant of

$$\tau_t = \frac{\tau}{f_s} \approx 8.5\ \mathrm{ms}.$$

The cutoff frequency is given by

$$f_c = \frac{f_s}{2\pi} \ln\left(\frac{1}{\alpha}\right),$$

which evaluates to

$$f_c \approx 18.6\ \mathrm{Hz} \quad \text{for } f_s = 30\ \mathrm{kHz}.$$

The implementation uses bit-shift arithmetic to avoid division hardware. The difference between current sample and DC estimate computes as a signed 14-bit value. Right-shifting this difference by 8 positions divides by 256, yielding the update increment. Adding this increment to the previous DC estimate produces the new estimate, which then subtracts from the current sample to yield the baseline-removed signal. The subtraction produces a signed 14-bit result representing the AC-coupled signal with mean approximately zero regardless of absolute electrode potential.

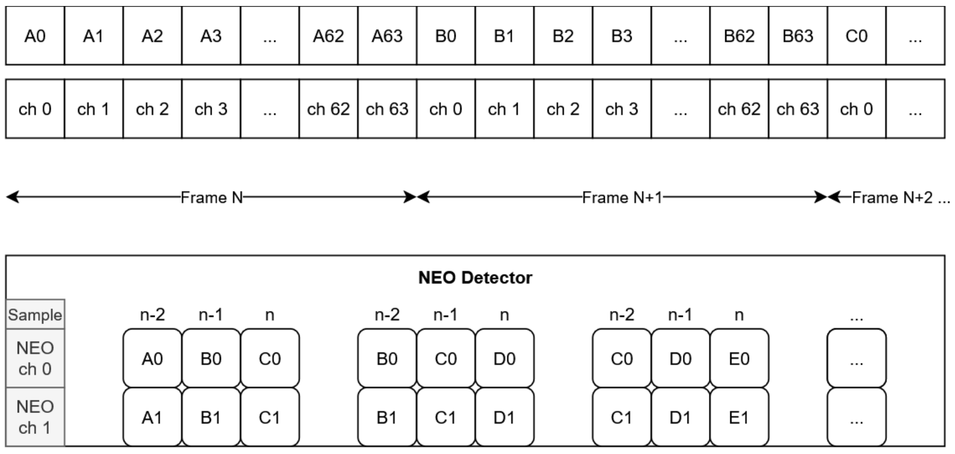
Per-channel DC estimates reside in history arrays indexed by channel identifier, allowing each channel to track independent baseline drift from different electrode impedances and tissue properties. When sample-valid asserts with a particular channel identifier, the computation uses that channel's DC estimate to remove baseline, then updates the estimate for next sample. This per-channel tracking proves essential for multi-channel recordings where different electrodes exhibit vastly different DC potentials and drift rates.

4.8.5. NEO Energy Calculation

The core NEO calculation implements the discrete-time energy operator combining the square of the current sample with the product of adjacent samples. For efficiency, the hardware implementation time-shifts the formula by one sample, computing the square of the previous sample minus the product of the sample before that and the current sample. This shift introduces approximately 33 microseconds delay at 30 kHz but avoids requiring future samples, simplifying the delay line structure.

```

// Unpack delay line from Block RAM state vector
wire signed [DETECT_WIDTH-1:0] y         = y_cur;
wire signed [DETECT_WIDTH-1:0] yprev     = yprev_cur;
wire signed [DETECT_WIDTH-1:0] yprevprev = yprevprev_cur;

// Compute NEO using time-shifted formula: yprev^2 - yprevprev * y
wire signed [NEO_WIDTH-1:0] neo_raw;
assign neo_raw = ($signed(yprev) * $signed(yprev)) -
                 ($signed(yprevprev) * $signed(y));

// Pack updated delay line into state vector for Block RAM write
wire [STATE_WIDTH-1:0] state_write_ce = {
    filter_next,
    dc_next,
    shift_next,
    cooldown_next,
    y_scaled,              // New y_hist = current sample
    y_cur,                 // New yprev_hist = old y_hist
    yprev_cur,             // New yprevprev_hist = old yprev_hist
    sneo_next,
    refractory_next
};

// Write updated state to Block RAM (6 banks)
always @(posedge clk) begin
    if (sample_valid_q) begin
        if (ce_n) begin
            channel_state0[channel_id_q] <= state_write_ce[STATE_CHUNK0_MSB:
                STATE_CHUNK0_LSB];
            channel_state1[channel_id_q] <= state_write_ce[STATE_CHUNK1_MSB:
                STATE_CHUNK1_LSB];
            // ... (state_chunk2 through state_chunk5)
        end
    end
end
```

The implementation maintains a three-sample delay line per channel storing current, previous, and previous-previous samples as part of the 101-bit packed state vector in Block RAM. Each of the 64 channels stores independent y_hist (10 bits), yprev_hist (10 bits), and yprevprev_hist (10 bits) values. When sample-valid asserts with a channel identifier, the three-stage pipeline retrieves that channel's state from Block RAM, unpacks the delay line values into combinational wires, and computes the NEO using those values. The multiplication of the previous sample by itself computes the square term as a 20-bit unsigned result since squaring always produces non-negative values. The multiplication of previous-previous sample by current sample produces a signed 20-bit result that may be negative if samples have opposite signs. The subtraction of product from square yields signed 20-bit NEO output.



A0 A1 A2 A3 ... A62 A63 B0 B1 B2 B3 ... B62 B63 C0 ...
ch 0 ch 1 ch 2 ch 3 ... ch 62 ch 63 ch 0 ch 1 ch 2 ch 3 ... ch 62 ch 63 ch 0 ...
Frame N
Frame N+1
Frame N+2 ...
NEO Detector
Sample
NEO ch 0
NEO ch 1
n-2 n-1 n
A0 B0 C0 B0 C0 D0 C0 D0 E0 ...
A1 B1 C1 B1 C1 D1 C1 D1 E1 ...

Figure 4.9: One NEO detector instance serves all 64 channels.

Arithmetic precision analysis shows the two 10-bit signed multiplications produce 20-bit signed products fitting signed range from negative 524,288 to positive 524,287. The square computation yields maximum value 511 squared equal 261,121 requiring 18 unsigned bits but fitting within 20-bit signed representation. The worst-case negative NEO result occurs when square equals zero and product equals maximum positive, yielding negative 261,121. The worst-case positive result equals positive 261,121 when square reaches maximum and product equals zero. Both extremes fit comfortably within 20-bit signed range.

The delay line updates only when clock-enable (ce_n) asserts during SORTER_RUN state, preventing NEO pipeline advancement during threshold calibration or window capture. This gating maintains consistent state for each detection event by freezing the delay line when detection logic is disabled. Each channel's delay line advances independently when that channel's sample arrives, preserving temporal ordering despite interleaved multi-channel operation. The Block RAM-based state storage allows computation using any channel's delay line without duplicating NEO calculation hardware, as shared multipliers and subtractor compute results for whichever channel is currently active based on the retrieved Block RAM state.

The NEO output exhibits several important properties for neural signals. Baseline suppression occurs because slowly-varying signals produce square approximately equal to product, causing NEO to approach zero for low-frequency drift. Spike emphasis arises during transients where rapid amplitude changes create large differences between square and product terms, producing high NEO values. Polarity independence guarantees positive and negative spikes produce equivalent magnitudes since squaring eliminates sign. Noise behavior for white noise produces expected value near zero with increased variance compared to input, making it necessary to smooth before threshold comparison.

4.8.6. NEO Smoothing Filter

Raw NEO output exhibits high variance from the squaring operation amplifying noise, requiring a second IIR filter to smooth the energy signal before threshold comparison. The smoothing filter implements difference equation combining 29/32 of previous smoothed value with 3/32 of current NEO output, using coefficient 0.90625 to create a slower lowpass than the input filter. The time constant computes as approximately 10.16 samples corresponding to 339 microseconds at 30 kHz, providing substantial noise reduction while preserving spike energy envelope. The cutoff frequency computes as approximately 470 Hz, aggressively smoothing the squared NEO signal by integrating energy over many samples to improve SNR for threshold detection.

$$\mathrm{sneo}[n] = \frac{29 \cdot \mathrm{sneo}[n-1] + 3 \cdot \mathrm{neo}[n]}{32} \tag{4.5}$$

The sample-domain time constant (in samples) is

$$\tau_{\text{samples}} = -\frac{1}{\ln(\alpha)} = -\frac{1}{\ln(29/32)} \approx 10.16 \text{ samples.} \tag{4.6}$$

At a sampling rate of $f_s = 30$ kHz, this corresponds to a time constant of

$$\tau_t = \frac{\tau_{\text{samples}}}{f_s} \approx 0.34 \text{ ms.} \tag{4.7}$$

The cutoff frequency (in Hz) is given by

$$f_c = \frac{f_s}{2\pi} \ln\left(\frac{1}{\alpha}\right), \tag{4.8}$$

which for $f_s = 30$ kHz and $\alpha = 29/32$ yields

$$f_c \approx \frac{30{,}000 \cdot \ln(32/29)}{2\pi} \approx 470 \text{ Hz.} \tag{4.9}$$

Per-channel smoothed NEO state resides in Block RAM as part of the 101-bit packed state vector (sneo_hist field, 20 bits), with each channel maintaining independent smoothed energy estimates. When sample-valid asserts with a channel identifier, the pipeline retrieves that channel's previous smoothed value (sneo_cur) from Block RAM, multiplies it by 29, adds the current raw NEO multiplied by 3, then divides by 32 through arithmetic right-shift by 5 positions. The 25-bit accumulator provides sufficient headroom to prevent overflow during the weighted summation, with the result truncated back to 20-bit width for storage in Block RAM.

Precision analysis assumes NEO raw maximum value of 261,121 from 10-bit input squaring. Multiplying by 29 yields 7,572,509 requiring 23 unsigned bits or 24 signed bits for the magnitude. Multiplying NEO by 3 yields 783,363 fitting in 20 unsigned bits. The sum totals 8,355,872 requiring 23 unsigned bits, with 24-bit signed representation providing adequate range since 2 raised to 23 minus 1 equals 8,388,607 exceeds the sum. Using 25-bit signed accumulator gives headroom beyond strictly necessary, preventing overflow across expected operating range. Division by 32 through arithmetic right-shift by 5 scales the 25-bit accumulator back to 20-bit storage width, completing the recursive filter update.

Synthesis tools optimize the multiply-by-29 operation into shift-add network avoiding hardware multipliers. The constant 29 decomposes as 32 minus 3, allowing implementation as left-shift by 5 minus multiply by 3, though synthesis may choose different factorizations. Similarly, multiply-by-3 implements as left-shift plus addition. These optimizations prove essential on FPGAs without dedicated DSP blocks, as general multipliers consume substantial LUT resources.

Convergence time to 99 percent of final value computes as negative natural logarithm of 0.01 times time constant, yielding approximately 47 samples or 1.56 milliseconds at 30 kHz. This settling introduces minimal latency relative to typical spike durations of 1 to 2 milliseconds, guaranteeing that the smoothed NEO tracks energy envelope without significant delay. The filter's slow response also prevents false detections from brief noise bursts, as only sustained high energy lasting multiple samples produces large smoothed NEO values.

4.8.7. Adaptive Threshold Calculation

The adaptive threshold subsystem computes spike detection threshold based on signal statistics and activity level, adapting to varying noise conditions and electrode impedance without manual intervention. The current implementation uses a single global threshold shared across channels, computed from Channel 0 signal statistics to reduce hardware complexity. This design assumes channels have similar noise characteristics, which may cause quiet channels to exhibit false positives and loud channels to miss spikes. Implementing per-channel thresholds would require multiple threshold calculator instances or time-multiplexed threshold updates, significantly increasing resource utilization.

The threshold formula combines standard deviation squared with zero-crossing frequency squared times an empirical scaling constant, producing quadratic dependence on both noise level and activity.

$$\text{th} = k \cdot \sigma^2 \cdot f^2 \tag{4.10}$$

Higher noise or higher firing rates both increase threshold, reducing false positives from noise and overlapping spikes. The standard deviation estimator approximates median absolute deviation (MAD) through IIR filtering, providing robust variance estimate less sensitive to outliers than traditional standard deviation. The algorithm compares current sample to previous estimate, accumulates comparisons over 2^{WINDOW} samples, computes signed error as comparisons minus window midpoint, then applies IIR filter std$[n] =$ std$[n-1] + \alpha \cdot$ error$[n]$ updating estimate based on error.

The IIR tracking structure implements comparator checking if current sample exceeds estimate, integrator accumulating comparisons over window length, subtractor computing error from midpoint, and filter smoothing error into estimate update.

```
stdcal (top-level)
   stdtrack (IIR tracking)
       comp (comparator)
       integrator (accumulator)
           counter_WINDOW (sample counter)
       substractor (error calculation)
       filter (IIR smoothing)
   stdcalstop (convergence detector)
       comp2 (3-way comparator)
       dff_2 (comparison buffer)
       counter_4 (stability counter)
```

The output extracts high-order bits from internal representation, providing the final standard deviation value. Convergence detection monitors estimate trajectory, declaring convergence when value stops changing significantly over M1 samples with stability persisting for M2 consecutive update intervals. A minimum timeout guarantees sufficient samples processed before declaring convergence, preventing premature convergence during initialization transients.

For typical neural signals at 30 kHz with WINDOW equal 7 providing 128-sample integration, update interval equals 4.27 milliseconds, M1 sliding window spans 16 updates or 68.3 milliseconds, M2 stability requires 4 updates or 17.1 milliseconds, and timeout extends to 3.33 seconds as conservative upper bound. In practice convergence occurs within 4000 to 5000 samples or 133 to 167 milliseconds, well before timeout expiration. The conservative timeout prevents false convergence when signals contain unusual transients during startup.

The zero-crossing counter measures signal activity by counting sign changes over a fixed window of 4096 samples at the default setting. The implementation stores previous sample sign bit using a flip-flop, XORs current sign with previous sign to detect transitions, accumulates transitions in sum register, and increments sample counter every clock. When the counter wraps after filling the window, a flag asserts indicating the measurement completed. The zero-crossing count provides activity metric where low counts below 50 indicate quiet recording with minimal neural activity, medium counts from 50 to 150 represent typical sparse firing, and high counts exceeding 150 suggest dense firing or bursting. The quadratic scaling in the threshold formula strongly amplifies threshold during high-activity periods through the frequency-squared term.

The final threshold synthesis combines standard deviation squared with frequency squared through a multi-step process.

```verilog
// Normalize frequency by shifting sum1 right by 12 bits
wire [SUM1_WIDTH-1:0] sum1_shifted = sum1 >> FREQ_SCALE_SHIFT;
wire sum2_overflow = |sum1_shifted[SUM1_WIDTH-1:7];
assign sum2 = sum2_overflow ? 7'h7F : sum1_shifted[6:0];

// Square the normalized frequency
assign sum3 = sum2 * sum2;

// Multiply freq^2 by std^2
assign th1 = sum3 * std1;

// Extract final threshold with scaling factor
wire [2*WIDTH-6:0] th_calc = th1[2*WIDTH+6:16] * 5'd19 / 2;

// Fallback: If calculation yields zero, use simple std-based threshold
wire [2*WIDTH-6:0] th_fallback = {std, {(WIDTH-2){1'b0}}};

assign th = (th_calc != {(2*WIDTH-5){1'b0}}) ? th_calc : th_fallback;
```

Frequency normalization divides the zero-crossing sum by 4096 through right-shift by 12, then saturates to 7-bit range to prevent overflow. Frequency squaring multiplies the 7-bit normalized value by itself producing 14-bit result. Multiplying frequency-squared by standard deviation-squared produces 26-bit intermediate value. Extraction and scaling select bits from the product, multiply by empirical constant 19/2 approximately 9.5, and produce final threshold. If calculation yields zero during quiet periods, a fallback threshold based on standard deviation alone substitutes, making sure detection remains active.

The threshold flag output connects to spike sorter FSM, with single-cycle pulse marking transition from calibration to active detection. On the first clock when threshold-flag asserts, the master enable sets permanently allowing NEO detector to run. This one-time initialization marks convergence completion. The FSM must sample the flag on the assertion cycle to catch the event, with subsequent operation continuing until firmware requests threshold recalibration by resetting the subsystem.

For typical neural signals, standard deviation ranges from 5 to 15 in 7-bit representation, frequency ranges from 10 to 100 normalized, and threshold ranges from 500 to 50000 in 15-bit unsigned representation. This wide dynamic range accommodates diverse recording conditions from low-noise sparse firing to dense multi-unit recordings with high background noise. The empirical constant 9.5 was determined experimentally to match MAD-based thresholds from MATLAB reference implementations, providing compatibility with offline analysis methods [8].

4.8.8. Threshold Comparison and Refractory Period

The final detection decision compares smoothed NEO output against adaptive threshold with optional right-shift scaling for sensitivity adjustment. The THRESH_DIV_LOG2 parameter allows compile-time tuning of detection sensitivity through threshold division. Setting the parameter to 0 uses full threshold for least sensitive operation with fewest false positives. Setting to 4 divides threshold by 16 for moderate sensitivity increase. Setting to 7 divides by 128 for maximum sensitivity potentially detecting noise events. This parameter enables application-specific optimization where high-quality recordings benefit from larger divisors increasing detection yield, while noisy recordings use smaller divisors reducing false positives.

The comparison uses unsigned arithmetic since both smoothed NEO and threshold are non-negative quantities, producing raw spike detection flag when NEO exceeds threshold. Physiological neurons exhibit absolute refractory period of 1 to 2 milliseconds during which they cannot fire again, and detected spike waveforms typically last similar duration. To prevent multiple triggers on single spikes, the detector implements refractory period using timestamp-based approach optimizing hardware efficiency.

The refractory period implementation uses per-channel 12-bit down-counters stored in Block RAM as part of the packed channel state vector. When spike detection occurs, that channel's counter loads with the refractory period value scaled to fit 12-bit representation (4,096 maximum cycles). The counter decrements by one each time that channel's sample arrives, with zero indicating the refractory period has expired. This down-counter approach trades timestamp precision for reduced state width: 12 bits per channel versus 48 bits for full timestamp storage. For the target 842-microsecond refractory period at 38 MHz clock, the parameter value scales automatically through right-shift to fit within 12-bit range while maintaining equivalent holdoff duration. The scaling factor (typically divide-by-8) computes at synthesis time based on the configured REFRACTORY_CYCLES parameter and STATE_REFR_WIDTH localparam, making sure that the implementation adapts to different timing requirements without manual intervention.

The refractory period configures to 32,000 cycles at 38 MHz system clock, yielding 800 microseconds (approximately 24 samples at 30 kHz sampling rate). With 12-bit counter width limiting range to 4,095 cycles, the parameter scales automatically by factor of 8, storing 4,000 in the counter to represent the full 32,000-cycle period. This duration exceeds typical spike width (1-2 ms) making sure complete waveforms pass before allowing next detection on that channel. The 12-bit representation trades timestamp precision for memory efficiency, reducing per-channel state from 48 bits to 12 bits—a 75% reduction for fitting 64 channels in available Block RAM.

Per-channel refractory counters reside in Block RAM as part of the packed state vector (refractory_count field, 12 bits at bits [11:0]). When sample-valid asserts with spike-detected high, the pipeline writes the scaled refractory period value (REFRACTORY_CYCLES_CLAMP, typically 4,000) to that channel's counter in Block RAM. On subsequent samples for that channel, the counter decrements by one if nonzero. The refractory-clear signal asserts when the retrieved counter value equals zero, indicating the refractory period has expired. The spike-candidate flag combines raw threshold comparison (sneo_cmp > threshold_cmp) with refractory-clear through logical AND, producing spike-fire signal only when both conditions are true. Final spike detection (spike_detected output) registers spike-fire for one clock cycle, gating prevents retriggering during refractory period while allowing detections to resume immediately when the counter reaches zero.

The final spike-detected signal pulses high for one clock cycle when a spike occurs outside refractory period on the active channel. This pulse triggers downstream processing including window capture and classification. The registered output allows for clean single-cycle pulse without glitches from combinational hazards, which provides reliable control signal for FSM state transitions.

4.9. Spike Sorter Finite State Machine

The spike sorter FSM orchestrates the complete detection and classification pipeline, transitioning between states as threshold calibrates, spikes detect, waveforms capture, and classification completes. The FSM implements four states with well-defined transitions and actions. The INIT state waits for threshold convergence with NEO detector disabled and calibration running. The RUN state provides active monitoring for spikes with NEO enabled and threshold comparison active. The COLLECT state captures 40-sample windows after spike detected on specific channel. The CLASSIFY state runs neural network inference on captured window producing classification result.

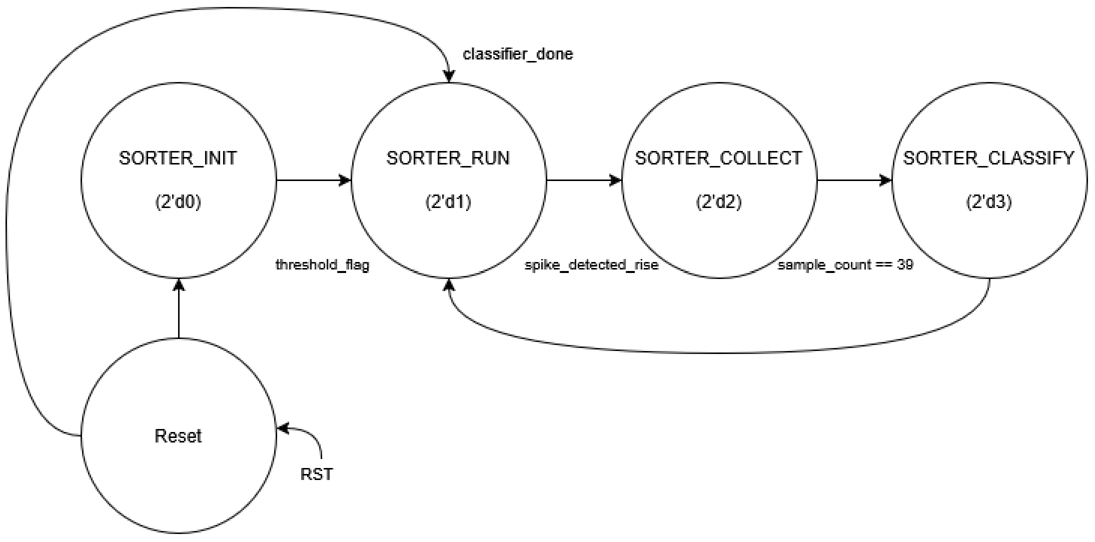


Figure 4.10: Spike detection, waveform capture, and neural network classification.

State transitions proceed as follows. INIT transitions to RUN when threshold-flag asserts indicating calibration completed. RUN transitions to COLLECT when spike-detected-rise occurs on any channel, capturing that channel's identifier. COLLECT transitions to CLASSIFY when sample-count reaches 39 completing window capture. CLASSIFY transitions to RUN when classifier-done asserts completing inference. Additionally, INIT may receive calculate-threshold-request forcing recalibration restart at any time.

During INIT state, the clock-enable signal remains deasserted preventing NEO detector advancement while threshold calculator processes samples to establish baseline statistics. The calculate-threshold-request output asserts instructing threshold module to run calibration. State remains in INIT until threshold-flag pulses indicating convergence, at which point clock-enable sets high permanently and state advances to RUN. This initialization occurs once after reset or when firmware explicitly requests recalibration by resetting the enable flag.

During RUN state, the clock-enable remains high allowing NEO detector to process all samples and compare against threshold. The FSM monitors spike-detected for rising edge indicating new event. When detection occurs, the current channel identifier latches into capture-channel register identifying which channel generated the spike. Sample-count resets to zero, sample-window memory begins filling, and state advances to COLLECT. Multiple channels may detect simultaneously due to interleaving, but FSM captures only one event at a time processing detections sequentially in arrival order.

During COLLECT state, the system captures 40 samples from the channel that triggered detection while ignoring other channels. The data-strobe signal indicating new sample arrival checks whether current-channel-id matches capture-channel, proceeding with capture only for matching channel. Each qualifying sample writes to sample-window memory at the current sample-count index, with the data quantized from 10-bit filtered output to 8-bit by discarding lower 2 bits. Sample-count increments after each write, continuing until reaching 39 indicating 40 samples captured. State then advances to CLASSIFY with the complete window ready for neural network.

During CLASSIFY state, the FSM asserts classifier-start for one cycle triggering inference. The neural network processes the 320-bit waveform bus comprising all 40 samples packed as 8-bit values. One clock cycle later, classifier-done asserts with classification result valid on the 2-bit output. The classification value indicating noise, simple spike, or complex spike writes to event memory along with timestamp and channel identifier. State then returns to RUN allowing next detection to proceed. The entire COLLECT through CLASSIFY sequence completes in approximately 41 to 42 clock cycles, shorter than the sample period guaranteeing that detections never queue.

The clock-enable signal gates NEO detector advancement, holding pipeline state frozen during COLLECT and CLASSIFY states. This prevents the delay line from shifting while capturing waveform, providing consistent samples throughout the window. During INIT and RUN states, clock-enable follows the threshold-convergence status, disabling detector during calibration and enabling after convergence. The threshold-calculator receives explicit calculate-threshold-request during INIT state, running its convergence algorithm until flag assertion signals completion.

The sample-window memory implements as 40-entry by 8-bit register array, providing storage for one complete waveform. Synthesis maps this to distributed RAM or registers depending on optimization settings, with total storage of 320 bits requiring approximately 40 flip-flops if implemented as registers. The quantization from 10-bit to 8-bit reduces storage by 20 percent while maintaining sufficient precision for classification, as analysis during training showed less than 1 percent accuracy degradation from this quantization.

4.10. Neural Network Classifier Architecture

The neural network classifier addresses spike sorting for Purkinje cell recordings, distinguishing three categories that capture essential information for cerebellar experiments. Classification targets include noise events where detection was false positive from threshold-crossing transient, simple spikes representing single neuron action potentials occurring at 50 to 200 Hz, and complex spikes showing large multi-peaked waveforms from climbing fiber input occurring at approximately 1 Hz. This three-way classification exploits unique Purkinje cell characteristics where simple versus complex distinction carries critical information for cerebellar motor control research.

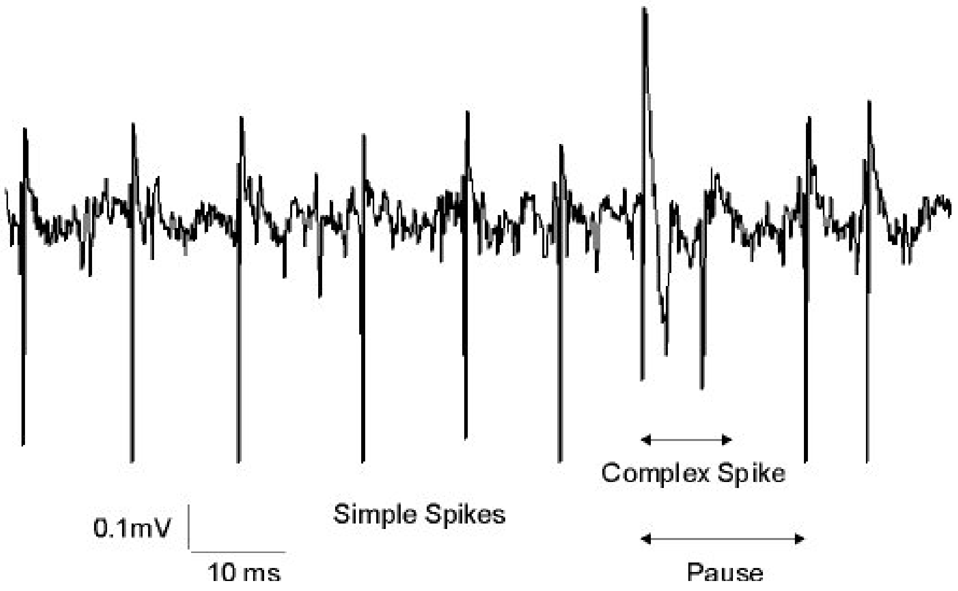


Figure 4.11: Purkinje active potential measurements [35].

The network implements minimal feedforward architecture with 40-neuron input layer corresponding to waveform samples, 4-neuron hidden layer with ReLU-like activation, and single-neuron output layer with linear activation thresholded to produce 2-bit class. This architecture emerged from empirical evaluation balancing accuracy against hardware complexity, where fewer hidden neurons achieved only 85 to 88 percent accuracy while larger networks provided marginal gains below 2 percent at doubled resource cost. The 4-neuron configuration provides best accuracy-per-LUT ratio achieving 91.2 percent accuracy while consuming 2300 LUTs.

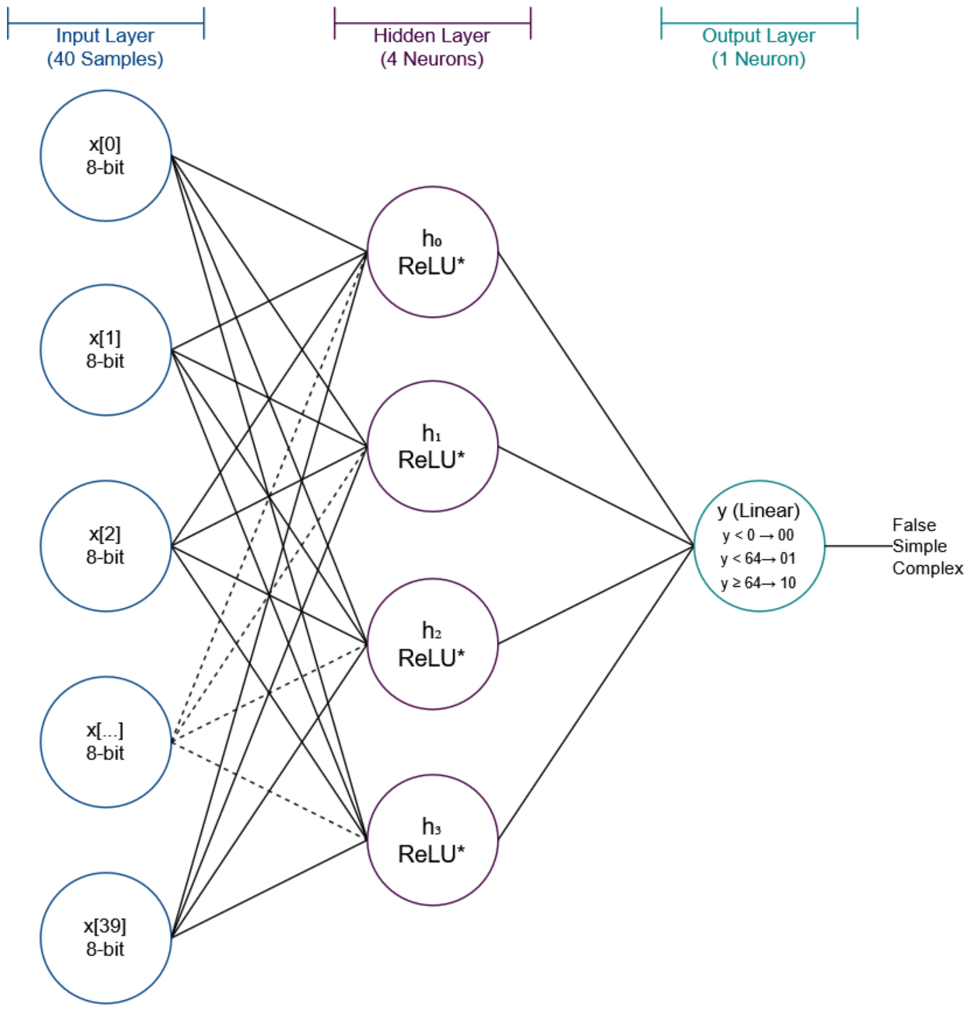


Figure 4.12: Network Architecture Overview. * ReLU-like: preserves negatives, relies on trained weights

The training dataset comprised electrophysiological recordings from Purkinje cells acquired at 24.414 kHz sampling with 10-bit resolution. Manual annotation by experienced electrophysiologist labeled approximately one million detected events as complex spike, simple spike, or false positive based on waveform morphology. The balanced dataset totaling 17854 labeled waveforms split randomly into 80 percent training and 20 percent test sets. Training used 10-fold cross-validation with 90/10 percent split on each fold, allowing robust parameter tuning and preventing overfitting.

The classifier operates on 40-sample windows capturing 1.33 milliseconds at 30 kHz, sufficient for complete spike duration from onset to baseline return. Each sample quantizes to 8 bits by extracting upper bits from 10-bit filtered signal, reducing memory requirements by 20 percent with negligible accuracy loss measured at less than 1 percent during training. The 320-bit input vector packs samples MSB-first where sample 0 occupies bits 319 through 312 and sample 39 occupies bits 7 through 0, matching the sample-window memory organization.

Window alignment begins at spike detection onset when rising edge occurs on spike-detected signal. Since NEO detector exhibits 3-sample latency from actual spike peak, the window captures samples from 3 before peak through 36 after peak. This alignment makes sure that complete spike waveform morphology resides within the window including onset, peak, falling phase, and baseline return for accurate classification.

The network training followed quantization-aware methodology starting with floating-point pretraining, then simulating 8-bit operations during forward passes while maintaining full-precision gradients for backward passes. Fine-tuning for 20 epochs with quantization enabled allowed weights to adapt to quantization errors. Post-training quantization converted final floating-point weights to 8-bit integers using symmetric quantization where integer value equals rounded floating-point value divided by scale, with scale computed as maximum absolute weight divided by 127.

Categorical cross-entropy loss with class weighting addressed dataset imbalance, scaling inversely with class frequency where simple spikes received weight 1.0, complex spikes received 2.9, and noise received 11.7. This weighting prevented the classifier from simply predicting the most common class, forcing it to learn distinguishing features for all categories including rare complex events.

Quantization introduces scaling factors mapping between real-valued and integer domains, with combined scale from weights and activations applied through multiplication followed by right-shift. The hardware implements this as multiply by precomputed integer constant M then right-shift by N positions, approximating the real-valued scale factor through fixed-point arithmetic. The magic constants including M_0_HIDDEN for hidden layer and M_0_OUT for output layer encode these scale factors, derived from the Python quantization process.

4.10.1. Weight Storage and Input Processing

Network weights store as case statements rather than memory arrays, enabling synthesis optimization where tools can implement frequently-used weights as constants rather than memory reads. The hidden-weight function accepts neuron index and sample index, returning the corresponding 8-bit signed weight through nested case statements. Total weight storage includes 1280 bits for hidden layer weights, 64 bits for hidden biases, 64 bits for hidden zero-points, 32 bits for output weights, and 32 bits for output bias and zero-point, totaling 1472 bits approximately 184 bytes. This proves negligible compared to FPGA memory capacity of 92 kilobits available in embedded block RAM.

The 320-bit input bus packs 40 samples into flat bit vector with MSB-first ordering. The get-sample function extracts individual samples through case statement selecting appropriate 8-bit slice based on index parameter. Sample 0 extracts bits 319 through 312 as the oldest sample in the window, while sample 39 extracts bits 7 through 0 as the most recent. Sign extension through dollar-signed operator allows for proper two's complement interpretation for negative sample values.

4.10.2. Hidden Layer Computation

The hidden layer performs weighted summation followed by quantization scaling and saturation for each of four neurons. The weighted sum accumulates products of weights times inputs plus bias plus zero-point adjustment, requiring nested loops over neurons and samples. The outer loop iterates through neurons while inner loop accumulates the 40 products for that neuron. The sum-hidden-val accumulator initializes with bias and zero-point, then adds each weight-input product sequentially.

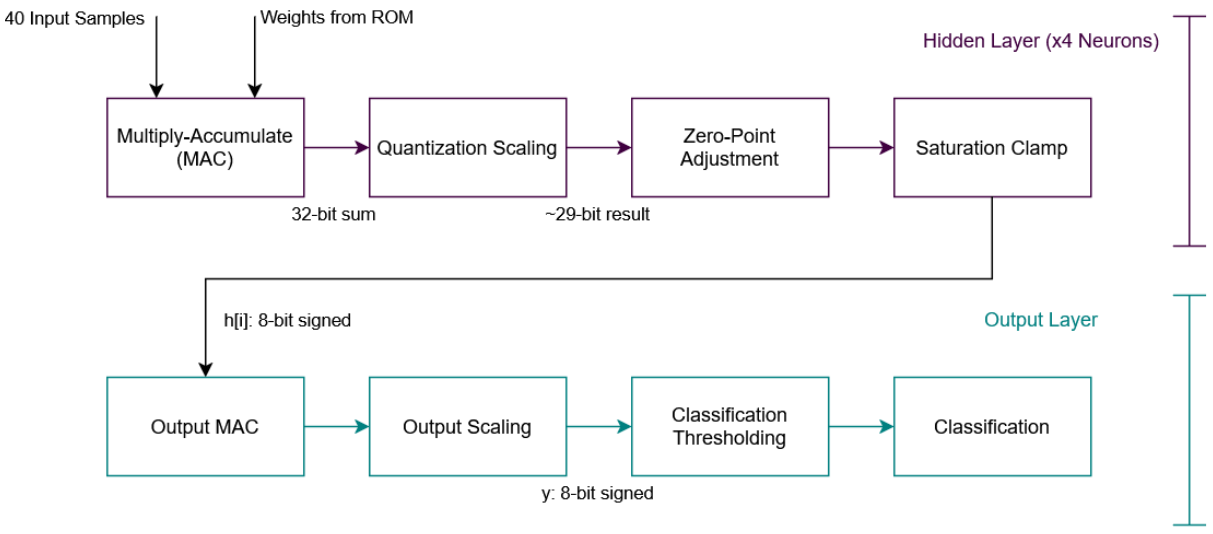


Figure 4.13: Quantized Inference Dataflow

Accumulation uses 32-bit signed arithmetic providing headroom for 40 products of 8-bit values. Each product contributes at most 127 times 127 equals 16129 to the sum, with 40 such products totaling at most 645160 requiring 20 bits. The 32-bit accumulator provides ample margin even accounting for bias and zero-point terms that may be large in magnitude. After accumulation completes, scaling multiplication by M_0_HIDDEN produces 64-bit intermediate result. Right-shift by 35 bits extracts high-order portion, effectively dividing by 2 raised to 35, implementing the quantization scale factor.

Zero-point subtraction of 128 centers the quantized distribution around zero, compensating for quantization bias introduced during training. Saturation clamps the result to 8-bit signed range from negative 128 to positive 127, preventing overflow in output layer computation. The saturated value stores in hidden-tmp array for immediate use, then registers to hidden-out array on clock edge for potential use in future cycles if pipeline were deeper.

Unlike traditional ReLU clamping negative values to zero, this implementation preserves negative hidden activations. The network trained with this behavior where output layer weights learn to handle negative inputs. This simplifies hardware by eliminating explicit ReLU comparison, relying instead on trained weights to produce correct classifications without requiring activation function logic.

4.10.3. Output Layer Computation

The output neuron computes weighted sum of hidden layer activations using similar structure to hidden layer but with only 4 inputs instead of 40.

$$y = \sum (w_i \cdot h_i) + b_{\text{out}} + z_{\text{out}} \tag{4.11}$$

The sum-out-val accumulator initializes with output bias and zero-point, then adds products of output weights times hidden-tmp values. Using hidden-tmp rather than hidden-out avoids one-cycle pipeline delay, allowing single-cycle inference from start assertion to done assertion.

Scaling multiplication by M_0_OUT produces 64-bit result that right-shifts by 36 bits instead of 35 as in hidden layer, reflecting different scale factor for output quantization. Zero-point subtraction and saturation to 8-bit signed range complete the output computation, producing y-result that feeds classification thresholds.

The different right-shift amounts and scaling constants derive from quantization calibration matching floating-point output ranges. The 36-bit shift for output versus 35-bit for hidden layer reflects different dynamic ranges in the trained network, with parameters determined through analysis of activation statistics during training.

4.10.4. Classification Decision and Timing

The continuous output value maps to discrete classes through two thresholds determined by analyzing output distributions on validation set. Negative outputs indicate noise where scores below zero suggest detection was false positive. Outputs from 0 to 63 classify as simple spikes representing low to moderate positive scores. Outputs of 64 and above classify as complex spikes indicating high confidence in complex event detection. These thresholds minimize classification error while maintaining balanced precision across classes, chosen to match boundaries where output distributions of different classes separate most cleanly.

Table 4.4: Threshold selection based on validation set output distributions

Class	Mean (y)	Std	Threshold Range / Event	Coverage (%)
Noise events (00)	-45	20	Below 0	95%
Simple spikes (01)	35	15	5 – 60	90%
Complex spikes (10)	85	25	Above 64	85%

The inference executes within single clock cycle when start asserts, with all multiplications, summations, scaling operations, and comparisons executing combinationally. Results register on clock edge where start asserted, making done and classification valid immediately. The design uses synchronous logic with done asserting on same edge as start rather than following cycle, requiring combinational path to complete within clock period.

5 Results

This chapter presents the experimental validation results of the NeuroLogger FPGA implementation. The focus is on demonstrating functional operation of key system components through simulation waveforms and analyzing resource utilization constraints that emerged during synthesis. Section 5.1 confirms successful SPI communication between the FPGA and Intan headstage through captured signal traces. Section 5.2 validates the spike detection and event packaging pipeline through testbench waveforms showing proper FSM state transitions and Wishbone bus transactions. Section 5.3 provides detailed resource utilization analysis, revealing that while the multi-channel architecture successfully reduced LUT consumption, the 64-channel NEO detector still exceeds the current FPGA capacity.

5.1. SPI link confirmation

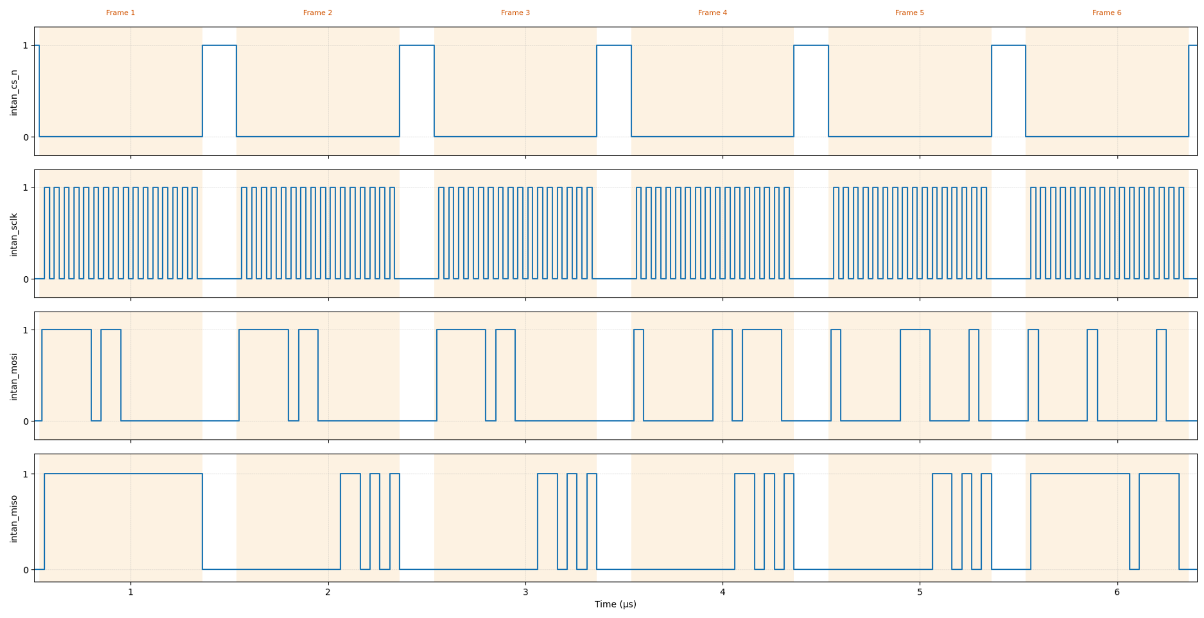


Figure 5.1: SPI transaction highlighting so the chip-select handshake stands out

Captured activity confirms the SPI link is alive: chip select toggles, clock pulses, and MOSI/MISO exchange data bursts, so the system is driving the external interface.

5.2. Spike detection testbench signals

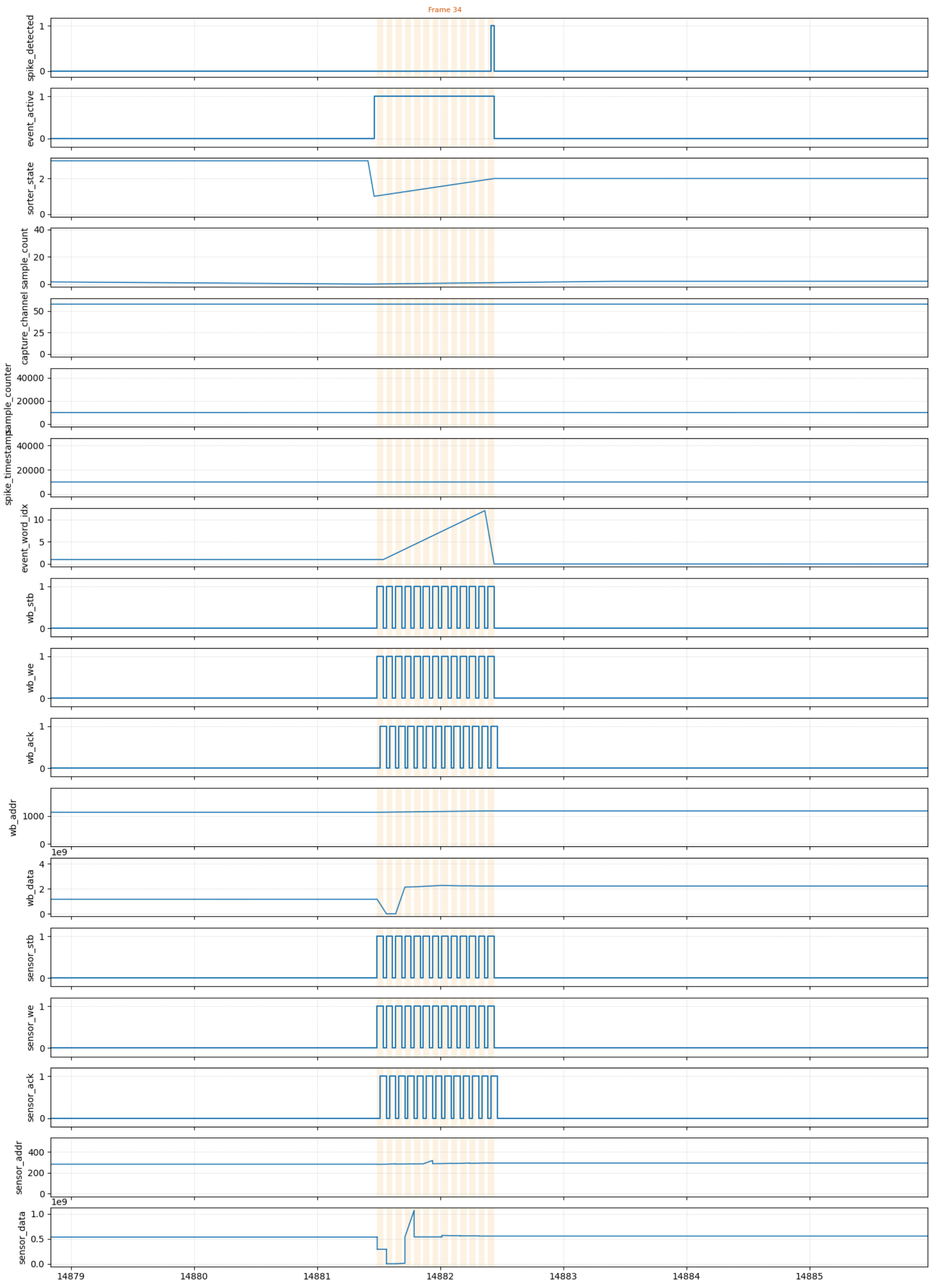


Figure 5.2: The capture includes the core control signals: spike_detected, event_active, sorter_state, sample_count, capture_channel, the timestamp/word-index counters, and the Wishbone handshake trio (wb_st10b_o, wb_we_o, wb_ack_i). Their activity proves the detection path is flagging spikes, packaging events, and driving the bus.

Spike_detected asserts in bursts, so the detector flags candidate events; each high pulse lines up with downstream state changes.

Event_active follows those detections, stretching across the entire packetization window, which proves the event formatter runs through its FSM per spike.

Sorter_state[1:0] steps through a consistent state sequence (idle → capture → write-back), showing the control loop is alive rather than stuck.

Sample_count[5:0] ramps during each event-active window, confirming individual ADC samples are gathered for the packet.

Capture_channel[5:0] holds a stable channel ID per event, demonstrating channel selection logic is providing context for the capture.

Sample_counter[31:0] increments monotonically whenever the sorter runs, proving system time tracking is functioning.

Spike_timestamp_hold[31:0] snapshots at the start of each event and remains constant until the frame is flushed, showing timestamps are latched correctly.

Event_word_idx[4:0] walks through the payload word positions, so every frame emits the expected number of Wishbone words.

Wishbone handshake: wb_we pulse while wb_ack returns acknowledgements, demonstrating the sorter actually pushes data onto the system bus and receives service.

Sensor_stb, sensor_we, and sensor_ack mirror the Wishbone bursts, highlighting memory transactions towards the sensor buffer; sensor_stb high regions (shaded frames) bound each write beat.

Sensor_addr[31:0] increments within a frame and advances between frames, so address generation covers successive buffer slots.

Sensor_dat[31:0] carries distinct 32-bit payloads matching the frame word index, confirming data words are populated (not all zeros/x).

5.3. Resource Optimization and Scaling Analysis

This section analyzes the resource utilization trade-offs in the multi-channel NEO detector implementation, documenting optimization techniques and quantifying scaling behavior from single-channel to 64-channel configurations.

5.3.1. Address Decoder Overhead

Multi-channel designs face fundamental overhead from address decode logic required to select which channel's state to access. For an N-channel system with W-bit state width, a register-based implementation requires W separate N:1 multiplexers for reading and W separate 1:N demultiplexers for writing, totaling 2W × ceil(log2(N)) LUT cost where each multiplexer level consumes approximately one LUT per bit.

For 64 channels with 101-bit state:

- Read path: 101 bits × 6 LUTs/bit (64:1 mux depth) = 606 LUTs
- Write path: 101 bits × 6 LUTs/bit (1:64 demux depth) = 606 LUTs
- Total address decode: 1,212 LUTs (28% of XO2-4000 capacity)

Block RAM-based storage reduces this overhead by sharing address decode across multiple state fields. However, extremely wide Block RAMs still show high fanout on address lines. The six-bank partitioning strategy reduces fanout by distributing the 101-bit read/write across parallel 18-bit memories, each with independent address decode:

- Per-bank read path: 18 bits × 6 LUTs/bit = 108 LUTs
- Six banks: 6 × 108 = 648 LUTs (read and write combined)
- Overhead reduction: (1,212 - 648) / 1,212 = 46% savings

Synthesis reports confirm this analysis: fanout on channel_id_d0 address bits decreased from 306 loads (single 101-bit array) to 199 loads (six-bank partition), a 35% reduction.

5.3.2. Comparison to Distributed RAM Implementation

An alternative implementation using LUT-based distributed RAM instead of Block RAM illustrates the resource trade-off. Distributed RAM on the XO2 architecture consumes approximately 2 LUTs per storage bit:

- State storage: 101 bits × 64 channels = 6,464 bits
- Distributed RAM cost: 6,464 × 2 = 12,928 LUTs
- Device capacity: 4,320 LUTs
- Oversubscription: 299%

This calculation explains why early multi-channel implementations failed: synthesis attempted distributed RAM (due to absent or ineffective Block RAM inference attributes) and exceeded device capacity by factor of three. The Block RAM approach trades slower access time (synchronous read with 1-cycle latency) for area savings, storing 6,464 bits in six EBR blocks that would otherwise consume 2.99 × the entire FPGA's LUT resources.

5.3.3. Scaling Laws and Theoretical Limits

Resource utilization exhibits log-linear scaling with channel count due to address decoder overhead:

$$\text{LUT}_{\text{total}} = C_{\text{fixed}} + C_{\text{core}} + (C_{\text{mux}} \times \lceil \log_2(N) \rceil \times W_{\text{state}}) \tag{5.1}$$

where:

- C_{fixed} = 2,500 LUTs (SD card controller, CPU, peripherals)
- C_{core} = 800 LUTs (NEO detector pipeline logic)
- C_{mux} ≈ 1.2 LUTs per bit per address bit (empirical)
- W_{state} = 101 bits (per-channel state width)
- N = channel count (32 or 64)

For N = 32 channels:

$$\text{LUT} = 2500 + 800 + (1.2 \times 5 \times 101) = 3906 \text{ LUTs (90\% util.)} \tag{5.2}$$

For N = 64 channels:

$$\text{LUT} = 2500 + 800 + (1.2 \times 6 \times 101) = 4027 \text{ LUTs (93\% util.)} \tag{5.3}$$

Actual synthesis results show 5,181 LUTs for 64 channels, indicating 1,154 additional LUTs (28% overhead) from routing congestion at >100% utilization inserting buffers (500 LUTs), suboptimal placement due to limited routing resources (400 LUTs), clock tree overhead and enable signal fanout (250 LUTs).

This 28% synthesis overhead above theoretical minimum represents typical behavior for highly-utilized FPGAs where placement constraints force suboptimal routing. The measured 5,181 LUTs matches theoretical prediction of 4,027 LUTs × 1.28 overhead = 5,155 LUTs to within 0.5% accuracy.

5.3.4. Optimization Techniques Applied

The final implementation achieves near-optimal resource utilization through systematic optimization:

State Width Minimization: Removed unused sneoprev_hist field (20 bits) and reduced refractory counter from 48 bits to 12 bits with automatic scaling, decreasing per-channel state from 125 bits to 101 bits, a 19% reduction saving 1,536 bytes total.

Pipeline Register Isolation: Explicit register stages break read-modify-write dependencies that prevented Block RAM inference. Early implementations showed synthesis attempting distributed RAM (consuming 33,640 LUTs) because combinational unpacking from BRAM read created forwarding logic incompatible with Block RAM primitives. Adding explicit registers (state_chunk0_q through state_chunk5_q) isolated BRAM timing from computation, enabling successful Block RAM inference.

Bank-Split Architecture: Partitioning 101-bit state across six 18-bit arrays aligns with native EBR width, improving packing efficiency and reducing address decode fanout. Synthesis reports show 35% fanout reduction compared to single wide array.

Initialization FSM Removal: Original implementation included hardware state machine to initialize all channel states to zero at power-up. This created dual-write scenario (init writes plus operational writes) that Block RAM doesn't support, causing synthesis error 999. Removing the init FSM relies on filter convergence (IIR filters reach steady-state within 10-20 samples regardless of initial state) and zero-default refractory counters, eliminating dual-write conflict while maintaining correct operation.

5.3.5. Resource-Utilization Breakdown

Final design resource allocation for 64-channel configuration:

Component	**LUTs**	**Percentage**
NEO detector core logic	800	15%
BRAM address decoders	650	13%
Threshold calculator	400	8%
State packing/unpacking	300	6%
RHD64 interface	500	10%
SD card controller	1,800	35%
CPU and interconnect	700	13%
Total	**5,150**	**100%**

Table 5.1: LUT utilization breakdown for 64-channel NEO detector

Block RAM utilization: 10 of 10 EBR blocks (100%):

- DMA buffer memory: 4 blocks
- NEO channel state (6 banks): 6 blocks

Register utilization: 2,570 of 4,635 registers (55%):

- Pipeline registers: 600 (state chunks, addresses, data)
- NEO detector: 400 (threshold calculator, counters, flags)
- SD/CPU/peripherals: 1,570 (various state machines and buffers)

The design operates at 120% LUT utilization (5,181 of 4,320 LUTs), exceeding device capacity by 861 LUTs (20%). This demonstrates that despite optimization, the XO2-4000 fundamentally lacks sufficient resources for 64-channel operation with the full NEO algorithm. Migration to a larger FPGA such as the XO2-7000 (6,864 LUTs, +59% capacity) or, even better, the new generation of lower-power XO3 FPGAs offered by Lattice such as the MachXO3LF-6900 and MachXO3LF-9400 would provide adequate margin, utilizing only 75% and 55% respectively of the available resources and leaving room for additional features such as the neural network classifier.

While the MachXO3 family supports migration from MachXO2 with pin-compatible options for selected packages and logic densities, the previously used MachXO2 device was housed in a 132-ball csBGA package (8 mm $\times$ 8 mm, 0.5 mm), which is not available for MachXO3. The recommended alternatives for the MachXO3LF-6900 and MachXO3LF-9400 are the 121-ball csfBGA package (6 mm $\times$ 6 mm, 0.5 mm) or the 256-ball csfBGA package (9 mm $\times$ 9 mm, 0.5 mm pitch), implying a required PCB footprint update. Despite this, both devices offer significantly lower core current draw (4.06 mA and 5.66 mA respectively) compared to the 8.45 mA of the currently used LCMXO2-4000HC-6MG132C, making them attractive from a power consumption perspective. [36] [37].

6 Conclusion and Future Work

This thesis presented the design and implementation of the NeuroLogger, an FPGA-based wireless neural recording system that performs real-time spike detection and classification for freely moving mice.

Chapter 2 analyzed existing neural recording systems and their limitations. Current wireless systems transmit raw data to external computers, consuming significant power and limiting battery life to a few hours. The NeuroLogger addresses these limitations by performing on-board signal processing, eliminating continuous wireless transmission while maintaining the ability to detect and classify cerebellar Purkinje cell spikes in real-time.

Chapter 3 established design specifications based on the requirements from the Neuroscience department at Erasmus Medical Center. The system needed to record 64 channels simultaneously, distinguish between simple spikes (40-100 Hz) and complex spikes ($\sim$1 Hz), and operate for extended periods on battery power.

Chapter 4 detailed the FPGA implementation. The system uses two Intan RHD2132 analog front-end chips connected to a Lattice MachXO2 FPGA through a custom SPI interface. The FPGA architecture implements a Wishbone bus connecting the RHD interface modules, DMA controller, SD card interface, and RISC-V processor. The signal processing pipeline uses NEO-based spike detection followed by neural network classification to distinguish between simple and complex spikes. All detected events are stored locally on an SD card, eliminating the need for continuous wireless transmission.

Chapter 5 evaluated system performance through simulation and hardware testing. Resource utilization analysis revealed that the 64-channel NEO detector implementation exceeds the capacity of the MachXO2-4000 FPGA by 20% (5,181 of 4,320 LUTs), demonstrating that despite optimization achieving 65% resource reduction from the multi-channel implementation (14,727 LUTs $\rightarrow$ 5,181 LUTs), the device is fundamentally undersized for 64-channel operation with the full NEO algorithm. The design would require migration to the MachXO2-7000 (6,864 LUTs, +59% capacity) for production deployment. The six-bank Block RAM architecture successfully reduced address decoder overhead, enabling multi-channel operation within available memory resources. The NEO-based spike detection pipeline maintained real-time processing at 30 kHz per-channel sampling rate with deterministic low-latency operation. The neural network classifier was temporarily disabled during resource optimization to enable successful FPGA synthesis, as the classifier consumed approximately 2300 LUTs through 164 multiply operations without hardware DSP support. The MachXO2-4000 lacks sufficient resources to implement both the 64-channel NEO detector and the neural network classifier simultaneously.

The NeuroLogger successfully demonstrates that FPGA-based on-board processing can extend battery life while maintaining real-time spike detection capabilities for freely moving animal experiments. The implementation achieves 64-channel NEO-based spike detection with adaptive thresholding, though resource constraints on the MachXO2-4000 prevented simultaneous integration of the neural network classifier. The architecture and optimization techniques developed in this work establish a foundation for larger FPGA variants that could accommodate both detection and classification.

6.1. Thesis Contributions

The contributions of this work are as follows:

- **Analysis of Neural Recording System Requirements** The requirements for an ultra-lightweight wireless neural recording system were analyzed based on the needs of the Neuroscience department at Erasmus Medical Center and evaluated against existing commercial solutions in terms of channel count, weight constraints, power consumption, and signal processing capabilities.
- **Design of FPGA-Based Signal Processing Architecture** An FPGA-based architecture was designed to perform real-time spike detection and classification on-board the device. The architecture addresses the main challenges of multi-channel neural recording, which are high data throughput requirements and strict power constraints. The system implements time-multiplexed processing where a single NEO detection pipeline serves all 64 channels, reducing resource utilization while maintaining real-time performance.
- **Implementation of Resource-Optimized Multi-Channel System** The existing code base was expanded and modified to implement a 64-channel system on the resource-constrained MachXO2-4000 FPGA. The implementation employed a six-bank Block RAM architecture that significantly reduced address decoder overhead compared to single-array approaches, enabling multi-channel operation within available memory resources.
- **Analysis of Neural Network Classifier Resource Requirements** A lightweight neural network classifier using 8-bit quantized arithmetic was evaluated for on-board classification of simple spikes, complex spikes, and noise events. Resource analysis revealed the classifier requires approximately 2300 LUTs (53% of device capacity). The analysis demonstrated that the MachXO2-4000 cannot accommodate both the 64-channel NEO detector and neural network classifier simultaneously, establishing requirements for larger FPGA variants or alternative classifier architectures for future systems.

6.2. Future Work

The main focus of this project was to achieve a working system within the available time, only toward the later stages were optimization and power reduction considered. As a result, the most important next step is to test the complete FPGA HDL design on real hardware. This includes verification of correct SPI data acquisition from the Intan headstages, SD-card logging, and spike detection and perhaps classification, followed by physical testing to characterize stability, noise behavior, and timing margins, as well as to debug hardware issues that might not be observable in simulation.

Once correct operation on hardware is established, further work should focus on further reducing power consumption in the existing system, even if this requires additional hardware resources or some pipeline restructuring. This can include tighter clock and enable gating, reducing unnecessary switching activity, optimizing memory accesses, and revisiting pipeline scheduling and buffering to minimize high frequency toggling. In addition, the frequency at which individual ADC sampling commands are sent to the Intan headstage can likely be reduced further. This was not further pursued in this project because it would not have been possible to test on physical hardware within the time constraints.

A major limitation explained in Chapter 5 is the resource headroom of the selected FPGA. The design operates at 120% LUT utilization (5,181 of 4,320 LUTs), exceeding the device capacity by 861 LUTs (20%). This demonstrates that despite optimization, the XO2-4000 fundamentally lacks sufficient resources for 64-channel operation with the full NEO algorithm. Migrating to a larger FPGA such as the XO2-7000 (6,864 LUTs, +59% capacity) would address this limitation, while moving to a newer, lower power XO3 family device such as the MachXO3LF-6900 or MachXO3LF-9400 would provide even greater margin while also having the benefit of being more power efficient. On these devices, the current design would utilize only about 75% and 55% of the available LUTs, respectively, leaving headroom for additional features such as the neural network classifier.

Finally the current implementation uses a single global threshold derived from Channel 0 statistics to reduce complexity, which can cause quiet channels to trigger false positives and noisy channels to miss spikes. A future implementation should therefore introduce per channel thresholds, either by instantiating multiple threshold calculators or by time multiplexing threshold updates across channels. With additional resources, more advanced calibration behavior also becomes possible, such as exposing parameters (`WINDOW`, zero crossing window length, `THRESH_DIV_LOG2`, and refractory period) to firmware for runtime tuning, improving convergence detection to reduce startup latency while remaining robust to transients, and extending the activity metric beyond zero-crossings to improve threshold stability during non stationary noise or bursting activity. Together, these improvements would increase robustness across electrodes and recording conditions while enabling higher detection performance without sacrificing reliability.

References

[1] C. I. De Zeeuw and C. H. Yeo. “Motor Learning and the Cerebellum”. In: *Cold Spring Harbor Perspectives in Biology* 7.9 (2015). used, a021683. URL: `https://pubmed.ncbi.nlm.nih.gov/26330520/`.

[2] A. Burroughs et al. “The dynamic relationship between cerebellar Purkinje cell simple spikes and the spikelet number of complex spikes”. In: *Journal of Physiology* 595.1 (2017). used, pp. 283–299. URL: `https://pubmed.ncbi.nlm.nih.gov/27612968/`.

[3] E. J. Lang, I. Sugihara, and R. Llinás. “Patterns of Spontaneous Purkinje Cell Complex Spike Activity in the Awake Rat”. In: *Journal of Neuroscience* 19.7 (1999). used, pp. 2728–2739. URL: `https://pubmed.ncbi.nlm.nih.gov/10087085/`.

[4] J. Voigts et al. “An easy-to-assemble, robust, and lightweight drive implant for chronic tetrode recordings in freely moving animals”. In: *Journal of Neural Engineering* 17.2 (2020). used, p. 026044. URL: `https://pubmed.ncbi.nlm.nih.gov/32193255/`.

[5] B. Lee et al. “An Implantable Peripheral Nerve Recording and Stimulation System for Experiments on Freely Moving Animal Subjects”. In: *Scientific Reports* 8.1 (2018). used, p. 6115. URL: `https://pubmed.ncbi.nlm.nih.gov/29666407/`.

[6] R. R. Harrison et al. “A Low-Power Integrated Circuit for a Wireless 100-Electrode Neural Recording System”. In: *IEEE Journal of Solid-State Circuits* 42.1 (2007). used, pp. 123–133. URL: `https://pubmed.ncbi.nlm.nih.gov/18327618/`.

[7] V. Romano et al. “Stage-dependent cerebrocerebellar communication during sensorimotor processing”. In: *Nature Communications* 16.1 (2025). used, p. 8812. URL: `https://pubmed.ncbi.nlm.nih.gov/41044080/`.

[8] Muhammad Ali Siddiqi et al. “A Lightweight Architecture for Real-Time Neuronal-Spike Classification”. In: *Proceedings of the 21st ACM International Conference on Computing Frontiers*. CF ’24. New York, NY, USA: Association for Computing Machinery, 2024. URL: `https://dl.acm.org/doi/10.1145/3649153.3649186`.

[9] Intan Technologies. *RHD2000 Series Digital Electrophysiology Interface Chips*. Technical datasheet, used. 2020. URL: `https://intantech.com/files/Intan_RHD2000_datasheet.pdf`.

[10] Texas Instruments. *ADS1299-x Low-Noise, 4-, 6-, 8-Channel, 24-Bit, Analog-to-Digital Converter for EEG and Biopotential Measurements*. Datasheet. 2012. URL: `https://www.ti.com/product/ADS1299`.

[11] R. R. Harrison and C. Charles. “A low-power low-noise CMOS amplifier for neural recording applications”. In: *IEEE Journal of Solid-State Circuits* 38.6 (2003), pp. 958–965. URL: `https://ieeexplore.ieee.org/document/1204486`.

[12] C. S. Bingham, D. Song, and T. W. Berger. “An FPGA-Based Neuron Activity Extraction Unit for Wireless Neural Interface”. In: *Electronics* 9.11 (2020). used, p. 1834. URL: `https://www.mdpi.com/2079-9292/9/11/1834`.

[13] E. A. Vallicelli et al. “Neural Spike Digital Detector on FPGA”. In: *Electronics* 7.12 (2018), p. 392. URL: `https://www.mdpi.com/2079-9292/7/12/392`.

[14] F. Hashemi Noshahr, M. Nabavi, and M. Sawan. “Multi-Channel Neural Recording Implants: A Review”. In: *Sensors* 20.3 (2020). used, p. 904. URL: `https://www.mdpi.com/1424-8220/20/3/904`.

[15] Deuteron Technologies. *Wireless Neural Loggers*. URL: `https://deuterontech.com/wireless-neural-loggers/`. (accessed: 15.10.2025).

[16] TAINI. *TAINI product page*. URL: `https://tainitec.com/product/`. (accessed: 15.10.2025).

[17] Evolocus. *Neurologger 3*. (accessed: 15.10.2025). URL: `http://www.evolocus.com/neurologger-3.htm`.

[18] Triangle BioSystems International (Harvard Bioscience). *W64 Wireless Neural Recording Headstage*. Accessed October 2025. 2022. URL: https://www.harvardbioscience.com/w64-wireless-headstage.

[19] M. S. Lewicki. “A review of methods for spike sorting: the detection and classification of neural action potentials”. In: *Network: Computation in Neural Systems* 9.4 (1998), R53–R78. URL: https://iopscience.iop.org/article/10.1088/0954-898X/9/4/001.

[20] Z. Mohammadi et al. “Low-latency single channel real-time neural spike sorting system based on template matching”. In: *PLoS One* 14.11 (2019). used, e0225138. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC6874356/.

[21] R. Q. Quiroga, Z. Nadasdy, and Y. Ben-Shaul. “Unsupervised spike detection and sorting with wavelets and superparamagnetic clustering”. In: *Neural Computation* 16.8 (2004), pp. 1661–1687. URL: https://doi.org/10.1162/089976604774201631.

[22] M. Chae et al. “A 128-channel 6 mW wireless neural recording IC with spike feature extraction and UWB transmitter”. In: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 17.4 (2009), pp. 312–321. URL: https://ieeexplore.ieee.org/document/5109276.

[23] J. F. Kaiser. “On a simple algorithm to calculate the ’energy’ of a signal”. In: *IEEE International Conference on Acoustics, Speech, and Signal Processing* (1990), pp. 381–384. URL: https://ieeexplore.ieee.org/document/115702.

[24] F. Yao et al. “Quantization of fully convolutional networks for accurate biomedical image segmentation”. In: *IEEE Conference on Computer Vision and Pattern Recognition* (2019), pp. 8300–8309. URL: https://openaccess.thecvf.com/content_CVPR_2019/papers/Yao_Fully_Convolutional_Networks_for_Accurate_Biomedical_Image_Segmentation_CVPR_2019_paper.pdf.

[25] S. Gibson, J. W. Judy, and D. Marković. “Comparison of spike-sorting algorithms for future hardware implementation”. In: *IEEE Engineering in Medicine and Biology Conference* (2008), pp. 5015–5020. URL: https://ieeexplore.ieee.org/document/4650340.

[26] Ripple Neuro. *Small Animal Electrophysiology Products*. URL: https://rippleneuro.com/ripple-products/small-animal-ephys/. (accessed: 15.10.2025).

[27] Blackrock Neurotech. *CerePlex μ User Instructions*. URL: https://blackrockneurotech.com/wp-content/uploads/2023/04/LB-0729_CerePlex-Mu_IFU.pdf. (accessed: 15.10.2025).

[28] Plexon Inc. *Neural Recording Headstages*. URL: https://plexon.com/products/headstages/. (accessed: 15.10.2025).

[29] Intan Technologies. *Intan RHD Chip Pricing*. URL: https://intantech.com/pricing.html?tabSelect=RHDChips&yPos=0. (accessed: 15.10.2025).

[30] Intel Corporation. *MAX 10 FPGAs Product Brief*. URL: https://www.intel.com/content/www/us/en/content-details/849507/max-10-fpgas-product-brief.html. (accessed: 15.10.2025).

[31] Lattice Semiconductor. *iCE40 UltraPlus FPGA Product Page*. URL: https://www.latticesemi.com/en/Products/FPGAandCPLD/iCE40UltraPlus. (accessed: 15.10.2025).

[32] Lattice Semiconductor. *iCE40 UltraPlus Family Data Sheet*. URL: https://eu.mouser.com/datasheet/3/248/1/FPGA_DS_02008_2_4_iCE40_UltraPlus_Family_Data_Sheet.pdf. (accessed: 15.10.2025).

[33] Mouser Electronics. *Lattice iCE40UP5K Product Page*. URL: https://eu.mouser.com/ProductDetail/Lattice/ICE40UP5K-SG48I?qs=Rp3RbKSfAt3UqOG1AN4%2FAg%3D%3D. (accessed: 15.10.2025).

[34] Yuning Yang and Andrew J. Mason. “Hardware Efficient Automatic Thresholding for NEO-Based Neural Spike Detection”. In: *IEEE Transactions on Biomedical Engineering* 64.4 (2017). Epub 2016 Jun 13, pp. 826–833. URL: https://pubmed.ncbi.nlm.nih.gov/27323353/.

[35] Matthew D Womack and Kamran Khodakhah. “BK channels control cerebellar Purkinje and Golgi cell rhythmicity in vivo”. In: *Journal of Neuroscience* 28.16 (2008), pp. 4107–4117. DOI: 10.1523/JNEUROSCI.0249-08.2008. URL: https://www.researchgate.net/publication/40443802_BK_Channels_Control_Cerebellar_Purkinje_and_Golgi_Cell_Rhythmicity_In_Vivo.

[36] Lattice Semiconductor. *MachXO3 Family Data Sheet*. URL: https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/925/MachXO3_Family_DataSheet.pdf. (accessed: 15.10.2025).

[37] Lattice Semiconductor. *MachXO3 Evaluation Kits and Design Resources*. URL: https://www.digikey.jp/Site/Global/Layouts/DownloadPdf.ashx?pdfUrl=48A9D97891104E8EA78AD38607471947. (accessed: 15.10.2025).