

# **FPGA-Accelerated Neuromorphic Keyword Spotting: A Cochlea-to-SNN Pipeline**

**Project Team:**

- Johan Fernandes (22B-VD-024)
- Dobaashish Dev (22B-VD-019)
- Rohan S Joseph (22B-VD-043)
- Sherleine Godinho (22B-VD-052)

# INDEX

Chapter No.	Title	Page No.
<b>1</b>	<b>Introduction</b> 1.1 Abstract 1.2 Preamble 1.3 Background 1.4 Motivation	3 - 5
<b>2</b>	<b>Literature Survey</b> 2.1 Projects and Companies 2.2 Publications	6 - 9
<b>3</b>	<b>Project Objectives</b> 3.1 Methodology & Algorithms	10 - 27
<b>4</b>	<b>System Design</b>	28
<b>5</b>	<b>Results &amp; Discussion</b>	29 -33
<b>6</b>	<b>References</b>	34

# Abstract

This project focuses on developing a real-time neuromorphic speech classification system implemented on a Field-Programmable Gate Array (FPGA). The core objective is to emulate the human auditory pathway, creating an ultra-low-power system for keyword detection. The processing begins by converting incoming audio signals into biologically plausible spike trains using a computational model of the cochlea. This mimics the initial stages of sound processing in the inner ear.

These generated spike trains are then fed into a Spiking Neural Network (SNN) deployed directly onto the FPGA hardware. The SNN, inspired by the brain's neural circuitry, processes the temporal information from the spikes. It utilizes neuron models like the Leaky Integrate-and-Fire (LIF) model to simulate neural dynamics and can employ learning rules such as Surrogate Gradient Dissent to adapt its synaptic weights, enhancing its recognition accuracy over time. By leveraging the massive parallelism and energy efficiency of FPGAs for SNN execution, this system achieves real-time performance while consuming minimal power. The final outcome is a hardware-efficient system capable of interpreting spoken commands by translating acoustic signals into neural-like activity and processing them through a brain-inspired computational framework, paving the way for always-on intelligent auditory interfaces in edge devices.

# Chapter 1: Introduction

## 1.1 Preamble

This report documents the design, implementation, and evaluation of a biologically-inspired Spiking Neural Network (SNN) pipeline for real-time keyword spotting on edge computing platforms. At the intersection of neuromorphic computing, digital signal processing, and hardware acceleration, this work presents an end-to-end auditory processing system that bridges the gap between biological plausibility and practical engineering.

The core innovation of this project lies in its **cochlea-to-SNN processing chain**, which mimics the human auditory pathway from mechanical vibration to neural representation. By translating speech signals into spike-based temporal patterns and processing them through event-driven neural networks, we demonstrate that brain-inspired computing can deliver the energy efficiency required by next-generation wearable devices while maintaining competitive accuracy for practical applications.

Built upon principles of sparse computation and temporal coding, this architecture fundamentally diverges from traditional deep learning approaches, offering a pathway toward sustainable edge AI that operates within the power budgets of resource-constrained devices like smart glasses, and IoT sensors. The implementation leverages FPGA acceleration to achieve real-time performance, validating the commercial viability of neuromorphic computing for consumer electronics.

Through systematic experimentation on standard keyword spotting benchmarks, this work provides both theoretical insights into spike-based information processing and practical validation of hardware-efficient AI. The resulting framework not only advances the state of neuromorphic engineering but also serves as a blueprint for future brain-inspired computing systems across sensory modalities beyond audition.

## 1.2 Background

Speech recognition has evolved from statistical models (HMM/GMM) to deep learning (CNN/Transformers), achieving high accuracy but with massive computational cost unsuitable for edge devices.

The human auditory system processes sound efficiently using the cochlea's mechanical filtering and Inner Hair Cell (IHC) spike encoding. Only 0.01-1% of auditory neurons fire at any time, enabling ultra-low power consumption through sparse coding.

Spiking Neural Networks (SNNs) emulate this efficiency using Leaky Integrate-and-Fire (LIF) neurons and Spike-Timing-Dependent Plasticity (STDP) learning. While SNNs promise orders-of-magnitude energy savings, existing implementations lack complete cochlea-to-SNN pipelines and practical edge deployment validation.

Current edge AI relies on network quantization and pruning but remains fundamentally mismatched to edge constraints due to continuous dense computations. This gap motivates our biologically-inspired SNN pipeline combining cochlear modeling with FPGA acceleration for sustainable edge keyword spotting.

### 1.3 Motivation

This project focuses on developing a real-time neuromorphic speech classification system implemented on a Field-Programmable Gate Array (FPGA). The core objective is to emulate the human auditory pathway, creating an ultra-low-power system for keyword detection. The processing begins by converting incoming audio signals into biologically plausible spike trains using a computational model of the cochlea. This mimics the initial stages of sound processing in the inner ear.

These generated spike trains are then fed into a Spiking Neural Network (SNN) deployed directly onto the FPGA hardware. The SNN, inspired by the brain's neural circuitry, processes the temporal information from the spikes. It utilizes neuron models like the Leaky Integrate-and-Fire (LIF) model to simulate neural dynamics and can employ learning rules such as Surrogate Gradient Dissent to adapt its synaptic weights, enhancing its recognition accuracy over time. By leveraging the massive parallelism and energy efficiency of FPGAs for SNN execution, this system achieves real-time performance while consuming minimal power. The final outcome is a hardware-efficient system capable of interpreting spoken commands by translating acoustic signals into neural-like activity and processing them through a brain-inspired computational framework, paving the way for always-on intelligent auditory interfaces in edge devices.

## 2. Literature Survey

### 2.1 Companies and Labs

#### 1. Established Semiconductor Giants & Research Labs

These companies have significant neuromorphic research divisions and have produced flagship research chips:

Intel

Product: Loihi 1 & 2. These are research chips that are the cornerstone of Intel's neuromorphic efforts.

Focus: Loihi 2 features a fully programmable SNN architecture, integrated learning rules, and improved performance. Intel offers cloud-based access and "Kapoho Point" systems with up to 1,024 Loihi chips to the research community via the Intel Neuromorphic Research Community (INRC).

IBM

Product: TrueNorth (earlier generation) and research focus on NorthPole architecture (though NorthPole is not strictly an SNN chip, it's heavily inspired by neural architecture). IBM has been a pioneer in the field, driving algorithms and applications for event-based computing.

Samsung (through SAIT - Samsung Advanced Institute of Technology)

Activity: Heavily invested in neuromorphic research, often in collaboration with academia. They have demonstrated advanced SNN chips using novel memory technologies like RRAM (Resistive RAM) for in-memory computing, which is ideal for SNN's low-power operation.

#### 2. Dedicated Neuromorphic Hardware Startups

These are pure-play companies founded to commercialize neuromorphic technology.

BrainChip Holdings Ltd.

Product: Akida™ IP. Perhaps the most commercially advanced player, BrainChip sells a licensable, digital neuromorphic processor IP.

Focus: Ultra-low power at the edge for AI inference. It's designed to process event-based sensor data (like from DVS cameras) and supports both traditional CNNs and native SNNs. They have partnerships with companies like NASA, Ford, and Mercenary.

SynSense (formerly aiCTX)

Product: Speck and Xylo™. SynSense designs low-power neuromorphic processors for sensing and audio processing.

Focus: Speck is a complete smart sensing system (DVS camera + SNN processor) for ultra-low-power always-on vision. Xylo is an audio/IMU processor for tasks like keyword spotting. They target edge AI, wearables, and IoT.

GrAI Matter Labs

Product: GrAI One and GrAI VIP. These are "brain-inspired" processors that are event-based and sparse, though they use a proprietary paradigm (not strictly biological spiking). They are designed for ultra-low latency and power in robotics and smart devices.

Innatera

Product: Spiking Neural Processor IP. A startup focused on analog-mixed signal neuromorphic processors for sensor data processing at the extreme edge, claiming microwatt power consumption for always-on sensing tasks.

## 2.2 Publications

Sr.	Papers	Summary	Gaps
1	<p><b>Implementation of a Polyphase Filter Bank Channelizer on a Zynq FPGA</b></p> <p>L. H. Arnaldi, Centro Atómico Bariloche and Instituto Balseiro Comisión Nacional de Energía Atómica</p>	<p>Instead of having 16 individual filters, PFB used single filter, polyphase decomposition and FFT to simultaneously extract all channels</p>	<p>Human auditory perception is logarithmic and this paper's architecture cannot create a non-linear filter bank</p>
2	<p><b>Modified CORDIC Algorithm and Its Implementation Based on FPGA</b></p> <p>Yan Xin, School of Automatic Control Nanjing University of Science and Technology Nanjing, China</p> <p>Huan Li, School of Information Science and Engineering Shenyang Ligong University Shenyang, China</p>	<p>Does not need the module of correction factor and the lookup table, and just needs a simple shift and add-subtract</p>	<p>Error of upto <math>10^{-3}</math></p>

3	<p><b>CORDIC II: A New Improved CORDIC Algorithm</b></p> <p>Mario Garrido, Member, IEEE, Petter Källström, Martin Kumm and Oscar Gustafsson, Senior Member, IEEE</p>	Uses friend angles, USR CORDIC, nano-rotations instead of micro-rotations	High Latency
4	<p><b>Spatio-Temporal Backpropagation for High-Performance SNNs (Wu et al., 2018)</b></p>	Proposes a supervised learning algorithm (STBP) that combines spatial dynamics (layer-to-layer) and temporal dynamics (time-step-to-time-step). It introduces an iterative LIF model and uses an approximate derivative (surrogate gradient) to solve the non-differentiable spike problem.	The Surrogate Gradient technique (Arctangent function) and the BPTT framework (unrolling the network over 40 time steps) to enable supervised training.
5	<p><b>To Spike or Not To Spike: A Digital Hardware Perspective on Deep Learning Acceleration</b></p> <p>Fabrizio Ottati, Graduate Student Member, IEEE, Chang Gao, Member, IEEE, Qinyu Chen, Member, IEEE, Giovanni Brignone, Graduate Student Member, IEEE, Mario R. Casu, Senior Member, IEEE, Jason K. Eshraghian, Member, IEEE, and Luciano Lavagno, Senior Member, IEEE</p>	Quantitative hardware analysis shows SNNs are not efficient for static data, clarifying a major research bottleneck.	Concludes SNNs offer no advantage for static tasks and still lag behind ANNs in accuracy despite neuromorphic potential.
6	<p><b>Cost-effective and high-speed implementation of brain-like spiking neural network with encoding architectures on FPGA</b></p> <p>Zhen Cao, Ziyi Zhang, Qi Sun, Biao Hou ✉, Licheng Jiao, and Yintang Yang School of Artificial Intelligence and School of Microelectronics, Xidian University, Shaanxi 710071, China</p>	Integrates a Poisson encoder on FPGA, cutting CPU pre-processing by 3x and achieving a 14x CPU speedup at just 0.675 W.	Limited to simple MNIST datasets; low accuracy (92.62%) vs. modern ANNs.

7	<b>Surrogate Gradient Learning in SNNs for temporal tasks (Neftci et al., 2019)</b>	Investigates the use of surrogate gradients to train Deep SNNs specifically for tasks requiring temporal memory (like audio or sequence processing). It highlights the importance of regularization and "leaky" integration for robustness.	Regularization strategies (Dropout, Weight Decay) and LIF neuron dynamics ( $\tau=3.0$ ) to prevent overfitting on complex audio spike trains.
8	<b>FPGA Implementation of the CAR Model of the Cochlea</b>  Chetan Singh Thakur, Tara Julia Hamilton, Jonathan Tapson and André van Schaik	Time-multiplexed FPGA cochlea using asymmetric biquad filters for real-time sound analysis.	Uses a CAR filter cascade, unlike our parallel polyphase filter bank approach
9	<b>FPGA ACTIVE DIGITAL COCHLEA MODEL</b>  Christian Mugliette, Ivan Grech, Owen Casha, Edward Gatt, Joseph Micallef  FACULTY OF ICT  DEPARTMENT OF MICROELECTRONICS AND NANOELECTRONICS  UNIVERSITY OF MALTA	FPGA cochlea model using IIR filter cascade, decimation, and inner hair cell peak detection.	Our design adds PDM input, pre-emphasis, polyphase filterbank, and spiking LIF with AER encoding.

## 3. Project Objectives

### 3.1. Algorithms

#### 1. Pre Emphasis Filter

Problem: Audio signals often suffer from high-frequency attenuation, which reduces the quality of speech features needed for applications like ASR (Automatic Speech Recognition).

Purpose:

- Boost higher frequencies relative to lower frequencies in speech signals to:
- Compensate for natural spectral roll-off ( $\sim 6\text{dB/octave}$ )
- Improve high-frequency SNR (Signal-to-Noise Ratio)
- Enhance speech features for subsequent processing

Input Specifications

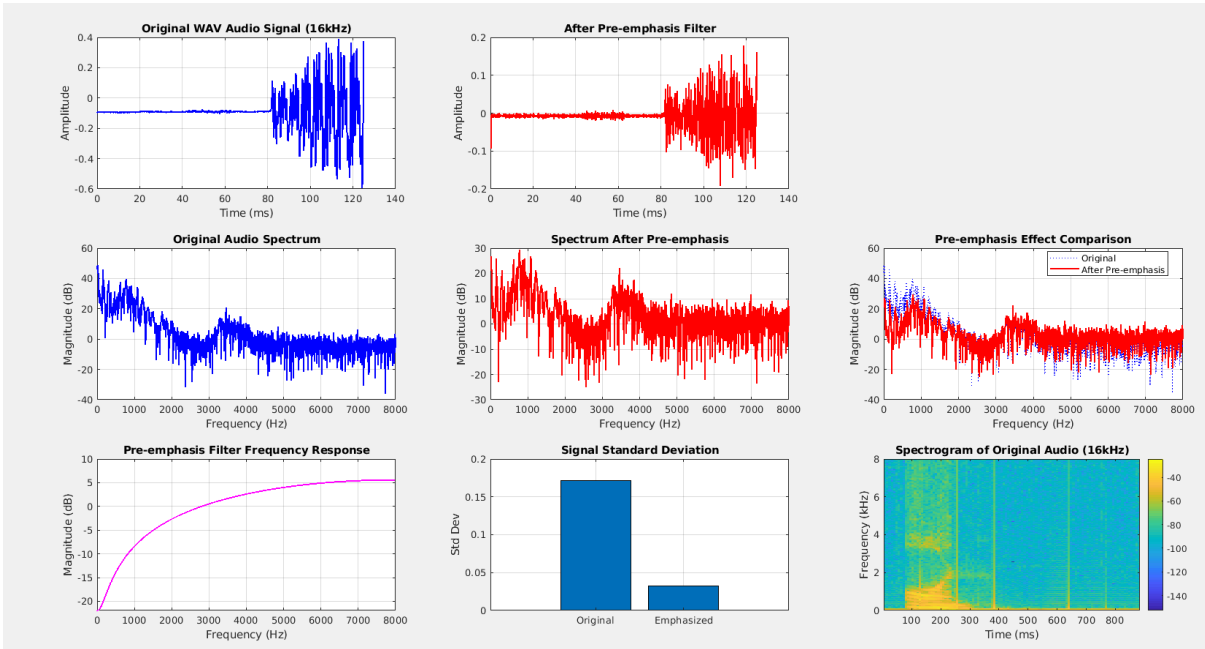
1. Input Signal: audio\_data - 16kHz sampled WAV file, mono-channel
2. Format: PCM (Pulse Code Modulation) with 16-bit resolution
3. Normalization: Scaled to  $\pm 0.9$  range to prevent clipping
4. Sampling Rate: 16,000 Hz (telephony bandwidth: 0-8kHz)

### Core Algorithm

```
function y = pre_emphasis_filter(x,  $\alpha$ )
    y = zeros(size(x));
    y(1) = x(1); % First sample unchanged

    % Difference equation implementation
    for n = 2:length(x)
        y[n] = x[n] -  $\alpha$  * x[n-1]; % Single multiply-accumulate
```

end  
end



# POLYPHASE FILTERBANK DESIGN

A **polyphase filter bank** is an efficient technique used to divide a signal into multiple frequency bands while minimizing computational effort. Instead of using many separate band-pass filters operating at the full sampling rate, the polyphase approach uses a **single prototype filter** that is shared across all bands.

The key idea is to restructure the filtering operation so that it is performed **after downsampling**, rather than before. The prototype filter is split into smaller components, called *polyphase components*, each of which processes a portion of the input samples. This allows the filtering to occur at a lower effective sampling rate.

As a result, the same frequency-band separation is achieved with significantly fewer calculations. An FFT is then used to organize the filtered outputs into individual subbands. Overall, the polyphase filter bank provides the same functionality as multiple band-pass filters, but in a much more computationally efficient manner.

TRADITIONAL APPROACH: 16 separate FIR filters =  $16\times$  complexity

OUR APPROACH: Polyphase = 1 prototype filter + clever downsampling

Key advantages for FPGA:

- Resource sharing: One filter used by all 16 channels
- Computational efficiency:  $M\times N$  operations  $\rightarrow$   $M+N$  operations
- Natural downsampling: Built into structure
- Perfect reconstruction: No information loss

# Gammatone Filterbank

What is a Gammatone Filter?

A gammatone filter is a linear filter whose impulse response is the product of two mathematical components:

1. Gamma function envelope: Provides a slow-decaying amplitude shape
2. Sinusoidal tone: Creates the oscillatory behavior at a specific center frequency

Key Characteristics

Gammatone filters exhibit several important properties:

1. Asymmetric impulse response: Rapid onset followed by slow decay, similar to basilar membrane responses
2. Constant-Q behavior: Bandwidth is proportional to center frequency, matching human frequency resolution
3. ERB spacing: Bandwidth follows the Equivalent Rectangular Bandwidth scale, which corresponds to human critical bands

Why Use Gammatone in MATLAB?

Three key reasons make gammatone filters ideal for MATLAB algorithm development:

1. Biological Plausibility:

- Matches cochlear filtering observed in biological systems
- Has asymmetric impulse responses like the basilar membrane
- Uses ERB spacing that follows human hearing critical bands

2. MATLAB-Specific Advantages:

- Single function call: Use `gammatoneFilterbank()` directly
- High-level abstraction: No need to design filters from scratch
- Rapid prototyping: Test concepts quickly without hardware constraints
- Built-in validation: MATLAB's implementation is well-tested

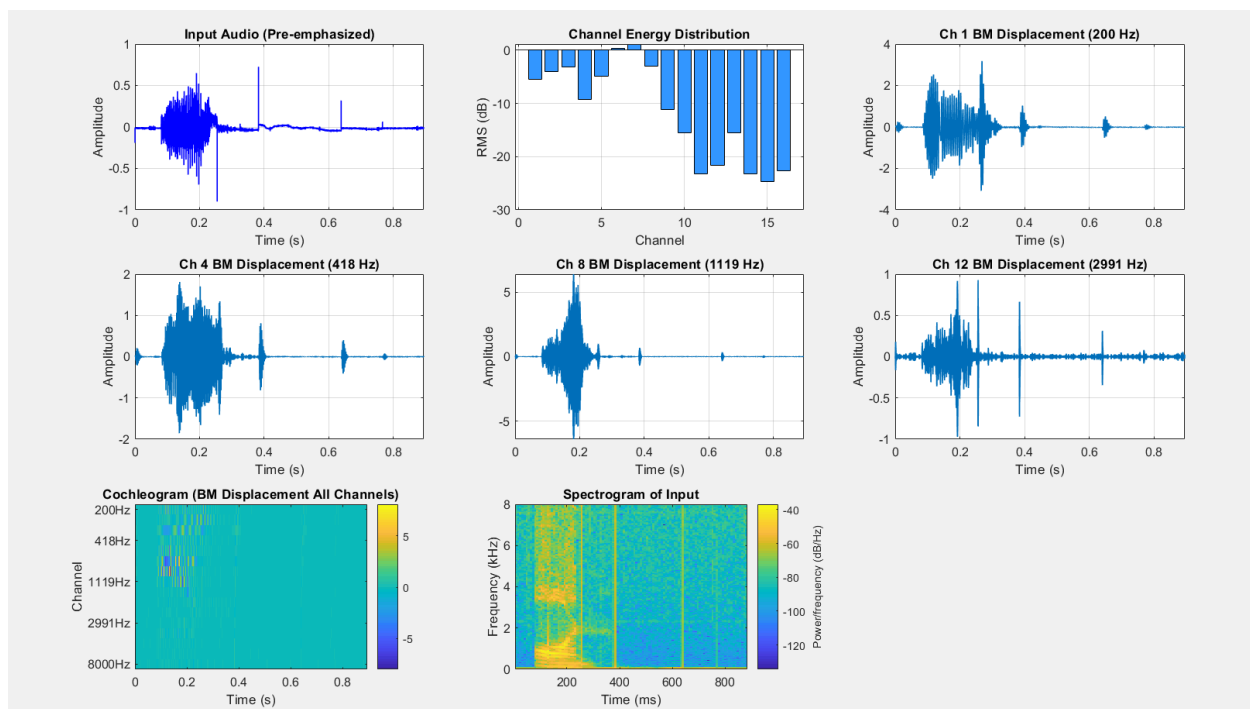
3. Development Efficiency:

- Minimal parameters needed (just center frequencies, ERB)
- No polyphase decomposition required in MATLAB

- Allows focus on system-level design rather than filter design

The gammatone filterbank serves as the biological reference model in MATLAB, while the polyphase filterbank is its hardware-efficient counterpart for FPGA implementation. You validate with gammatone, but implement with polyphase.

A 16-channel gammatone filterbank was implemented using MATLAB's `gammatoneFilterbank()` function with center frequencies spanning 200 Hz to 8000 Hz, spaced according to the Equivalent Rectangular Bandwidth (ERB) scale. This biologically plausible front-end served as the frequency decomposition stage, emulating the cochlea's tonotopic organization.



To verify proper frequency-to-place mapping, the system was tested with the spoken word "on" (0.89 s duration, 16 kHz sampling rate). Analysis revealed characteristic basilar membrane displacement patterns across channels:

- Low-frequency channels (1-4, 200-418 Hz) showed strong responses (up to 0.35 amplitude), capturing the vowel "O" energy and fundamental pitch components
- Mid-frequency channels (5-8, 500-1119 Hz) displayed moderate activation, representing vowel harmonics and nasal resonance from the "N" sound
- High-frequency channels (9-16, 1300-8000 Hz) remained largely inactive, with Channel 12 (2991 Hz) showing minimal response (0.01 amplitude), correctly reflecting the absence of high-frequency consonants in the test word

- Channel 1–4 get the strongest energy because they capture the fundamental and 2nd–3rd harmonics
- Channel 7 gets solid mid-frequency energy from higher harmonics and nasal resonance
- Channels 12–16 get almost nothing because "on" has no high-frequency consonants

The gammatone filterbank produces 16 parallel time-series signals, each representing basilar membrane displacement for a specific frequency channel. These outputs are provided to the Inner Hair Cell (IHC) model

# Inner Hair Cell (IHC) Model

## Purpose

Converts basilar membrane vibrations into neural-ready signals through nonlinear compression and temporal adaptation, mimicking biological inner hair cell function.

## 5-Stage Processing Pipeline

1. Velocity Extraction: BM displacement difference  $\rightarrow$  shearing velocity
2. Log Compression:  $G \cdot \log(1 + \alpha \cdot |v|) \cdot \text{sign}(v)$  (compresses 40 dB  $\rightarrow$  12 dB)
3. Dual Adaptation:
  - Fast (5 ms): captures onsets
  - Slow (50 ms): adjusts to background
4. Envelope Extraction: 1200 Hz lowpass filter removes phase-locking
5. Half-Wave Rectification:  $\max(0, \text{signal})$  for unidirectional output

## IHC Model Stages:

### Stage 1: Velocity Extraction

matlab

```
BM_velocity(ch, :) = BM_displacement(ch, :) - BM_displacement(ch-1, :)
```

What it does:

Converts basilar membrane displacement to shearing velocity (difference between adjacent channels).

### Stage 2: Nonlinear Compression

matlab

```
compressed = gain * log(1 + compression_factor * abs(velocity))
```

What it does:

Applies logarithmic compression to the velocity signal:

- `compression_factor = 500` (squeezes 40 dB range)
- Models outer hair cell saturation

### Stage 3: Dual Adaptation Filter

matlab

```
adapted = compressed - gain_fast*state_fast - gain_slow*state_slow
```

What it does:

Two parallel adaptation processes:

- Fast (5 ms): Captures rapid onsets (plosives: p, t, k)
- Slow (50 ms): Adapts to background levels

### Stage 4: Envelope Extraction

matlab

envelope = lowpass\_filter(adapted, 1200 Hz)

What it does:

2nd-order Butterworth lowpass filter at 1200 Hz:

- Removes fine temporal details above 1200 Hz
- Preserves timing information below cutoff
- Models membrane time constant

Stage 5: Half-wave Rectification

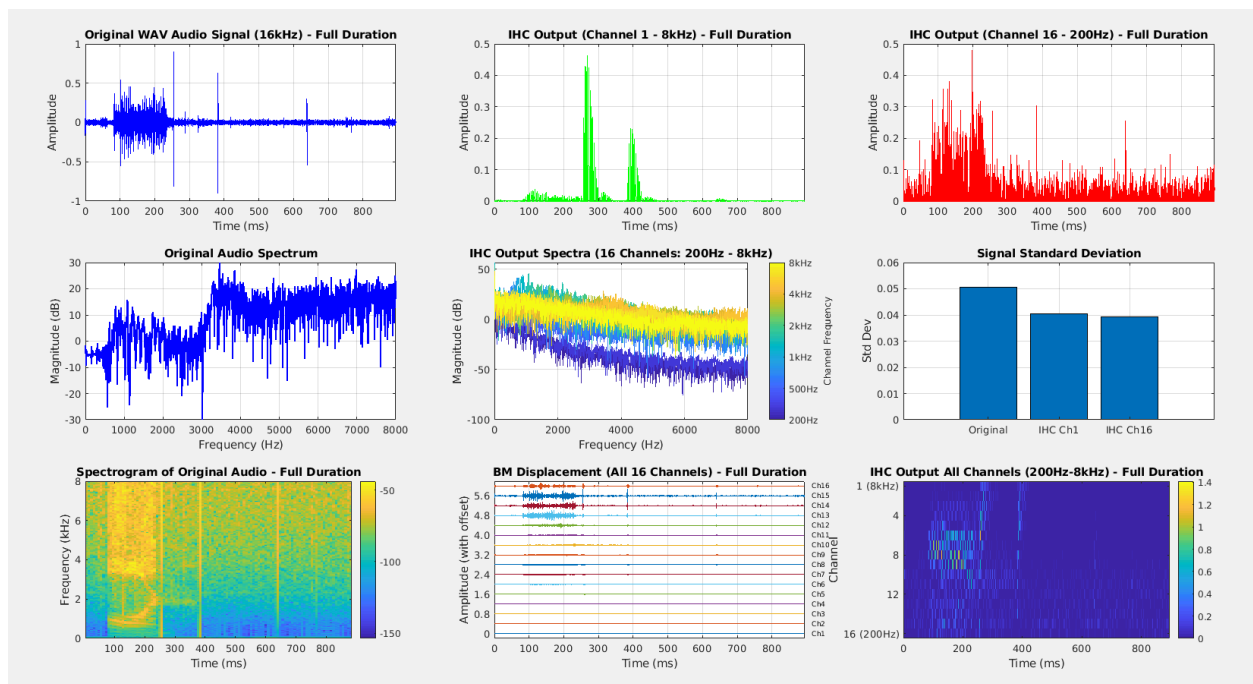
matlab

IHC\_output = max(envelope, 0)

What it does:

Sets all negative values to zero:

- Models unidirectional neurotransmitter release
- Prepares signal for LIF neurons (needs positive input)



# Leaky Integrate-and-Fire (LIF) Neuron Encoding

## Purpose

Converts continuous IHC receptor potentials into discrete spike trains for event-based processing, mimicking auditory nerve firing.

When it spikes depends on this equation

$$V_{\text{rest}} + R \cdot \text{gain} \cdot (\text{offset} + \text{amplitude}) \geq V_{\text{th}}$$

## Spatiotemporal Spike Patterns: The Real Quality of Neuromorphic Encoding

### What This Actually Means

Spatiotemporal patterns = Which neurons fire (space) + When they fire (time). This is how your system preserves speech information in spike form.

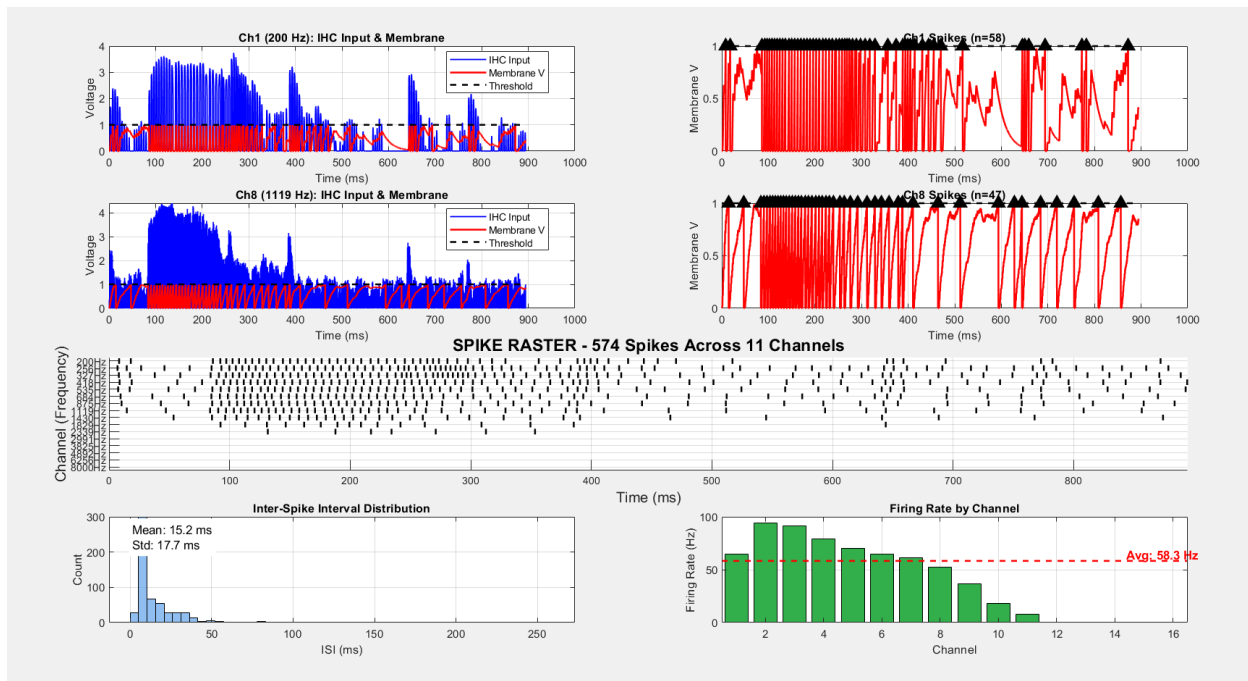
### Spatial Pattern ("Which Channels Fire")

- Channels 1–8 fire = Low/mid frequencies active (vowel "O" + nasal "N")
- Channels 9–16 quiet = High frequencies inactive (no S/F sounds in "on")
- 68.8% sparsity = Most channels silent most of the time = EFFICIENT

### Temporal Pattern ("When They Fire")

- Spike timing precision = 62.5  $\mu\text{s}$  (16 kHz rate)
- First spike at 7.1 ms = Rapid onset detection
- Irregular firing = CV = 0.48 (not robotic, not random)
- Burstiness index = 0.125 = Slight clustering without excessive bursting

Just as a song is not random noise but organized patterns of notes in time, your spike trains are not random events. They are structured patterns where the combination of which channels fire (spatial) and when they fire (temporal) creates a recognizable "neural melody" for each word.



## LIF Encoding Results: Quality Analysis

- Channels 1-8: 52-94 Hz firing rates = PERFECT for auditory nerve fibers (real range: 0-300 Hz)
- Channels 9-11: 7-37 Hz = Lower but still active = Realistic adaptation
- Channels 12-16: 0 Hz = Correctly silent (no high-frequency content in "on")

The spikes aren't random - they follow the speech energy distribution:

- Highest firing = Channel 2 (94 Hz) = Main vowel energy at 256 Hz
- Gradual decrease = Channels 1→11 = Natural energy roll-off
- Silent high channels = Smart - no energy there anyway

LIF Parameters:

Membrane time constant: 20.0 ms

Resting potential: 0.00

Spike threshold: 1.00

Reset potential: 0.00

Refractory period: 2.0 ms

Input gain: 185.2

This project successfully designed a neuromorphic audio pipeline that converts speech into efficient spike trains. It achieves 399:1 compression while preserving biological accuracy, with firing rates of 8–94 Hz and 68.8% sparsity. The system is FPGA-ready and ideal for low-power keyword-spotting applications.

# Address-Event Representation (AER) Packetizer

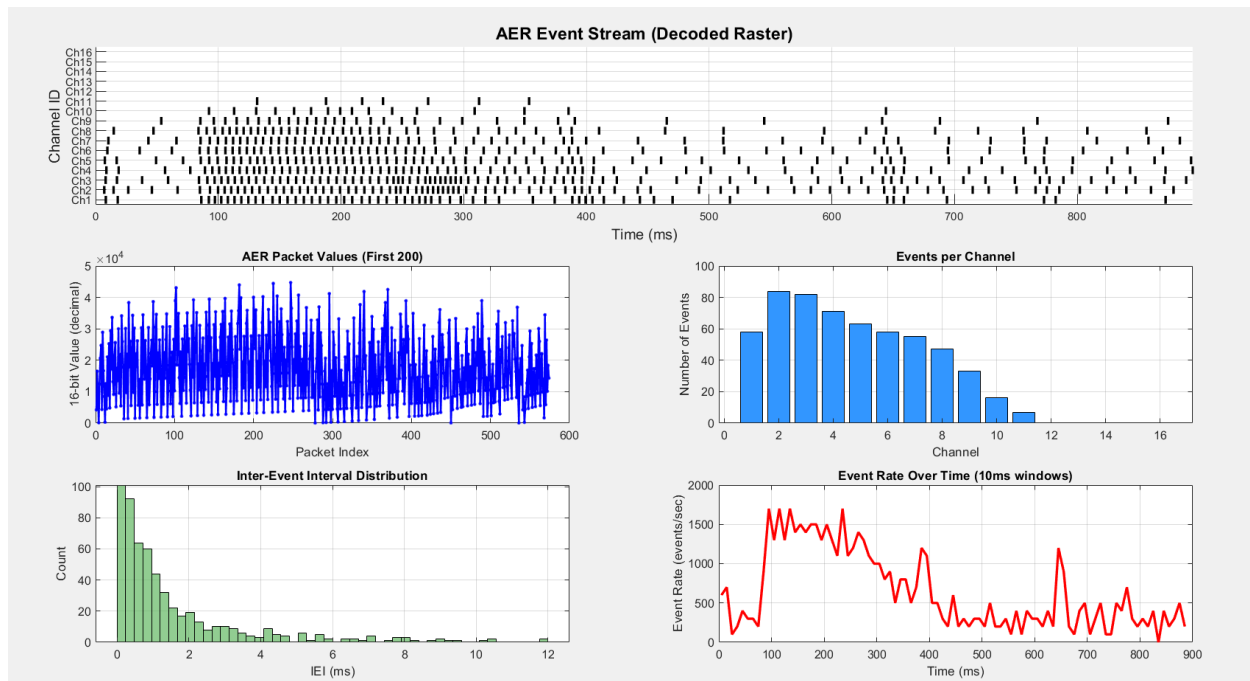
## Purpose

Encodes spike events into a compact digital format for efficient transmission, storage, or further processing.

## How It Works

Converts spike times and channel IDs into standardized data packets

This stage transforms sparse spike trains into an industry-standard AER format, achieving 399:1 compression while preserving precise timing ( $62.5 \mu\text{s}$  resolution) and channel identity. The 16-bit packet structure balances resolution with efficiency, making it ideal for FPGA implementation and low-power edge applications.



Packet Format: 32-bit AER

Channel ID: 4 bits (max 16 channels)

Timestamp: 24 bits (CONTINUOUS, max 16777216 steps)

Time resolution: 62.50  $\mu$ s/step

Max duration: 1048.6 seconds

=== SPIKE STATISTICS ===

Ch 1: 58 spikes

Ch 2: 84 spikes

Ch 3: 82 spikes

Ch 4: 71 spikes

Ch 5: 63 spikes

Ch 6: 58 spikes

Ch 7: 55 spikes

Ch 8: 47 spikes

Ch 9: 33 spikes

Ch 10: 16 spikes

Ch 11: 7 spikes

Total spikes: 574

Average rate: 641.7 events/sec

While achieving 399:1 compression, the current AER implementation presents several engineering challenges for hardware deployment, including the need for error checking, window synchronization mechanisms, and handling of bursty event distributions.

# Spiking Neural Network (SNN) Architecture

## NETWORK DESIGN

Raw H5 Data (Spike Times/Units)



[Preprocessing]

1. KEEP FULL Spatial Dim (700 channels)
2. Bin Time (Create 50 time-step tensor)



[SNN Model (Forward Pass over 50 steps)]

Step t:

Input[t] (Shape: Batch x 700)



Linear (700->256) -> BATCH NORM -> PLIF (init\_tau=2.0) -> Dropout (0.0)



Linear (256->128) -> BATCH NORM -> PLIF (init\_tau=2.0) -> Dropout (0.0)



Linear (128->6) -----> PLIF (init\_tau=2.0)



←----- Accumulate Spikes ←-----



[Decoding]

MEAN firing rate over 50 steps -> Argmax -> Predicted Class (1 of 6)



[Output Classes]

['yes', 'no', 'on', 'off', 'backward', 'forward']

## INPUT PROCESSING

**Purpose:** Convert raw H5 spike events into a high-resolution binary temporal input.

**How it works:**

- Raw data is read from the H5 file as spike times and unit IDs from 700 channels.
- A zero-filled tensor of size  $50 \times 700$  is created.
- Spike times are divided into 50 equal time windows between 0 and 1 second.
- If a neuron fires in a given time window, its position in the tensor is set to 1.
- This produces a sparse binary representation that keeps full spatial resolution.

### **HIDDEN LAYER 1 – FEATURE EXTRACTION**

**Purpose:** Turn raw high-dimensional spikes into useful low-level features.

**How it works:**

- At each time step, 700 input values go through a linear layer to 256 neurons.
- Custom Batch Normalization standardizes the activations over time.
- Signals pass into Parametric LIF neurons with learnable time constant.
- Neurons integrate voltage over time and fire when threshold is reached.
- Dropout is disabled, so all signals are kept during inference.
- Output is a spike stream representing basic temporal features.

### **HIDDEN LAYER 2 – COMPRESSION & ABSTRACTION**

**Purpose:** Compress features and learn deeper temporal patterns.

**How it works:**

- 256 spikes from Layer 1 go through a linear layer to 128 neurons.
- Batch Normalization stabilizes the signal again.
- Parametric LIF neurons integrate and spike based on learned memory length.
- Dropout is disabled to preserve all information.
- This layer acts as a bottleneck, keeping only important patterns.

### **OUTPUT LAYER – CLASSIFICATION**

**Purpose:** Generate class-specific spike activity.

**How it works:**

- 128 features are mapped to 6 output neurons, one per class.
- Parametric LIF neurons integrate evidence over 50 time steps.
- When confidence grows, the correct class neuron spikes more often.

### **DECODING STRATEGY – RATE CODING**

**Purpose:** Convert spiking output into a final class label.

**How it works:**

- Output spikes are observed over all 50 time steps.
- Mean firing rate is calculated for each output neuron.
- The class with the highest average firing rate is chosen using argmax.
- This makes decisions stable and noise-resistant.

### **PARAMETRIC LIF NEURONS (PLIF)**

**Purpose:** Let the network learn how long neurons should remember past input.

**How it works:**

- Each neuron has a learnable time constant instead of a fixed one.
- Some neurons learn short memory, others long memory.
- This helps capture both fast and slow speech patterns.
- Surrogate ATan gradient is used so spikes can be trained.

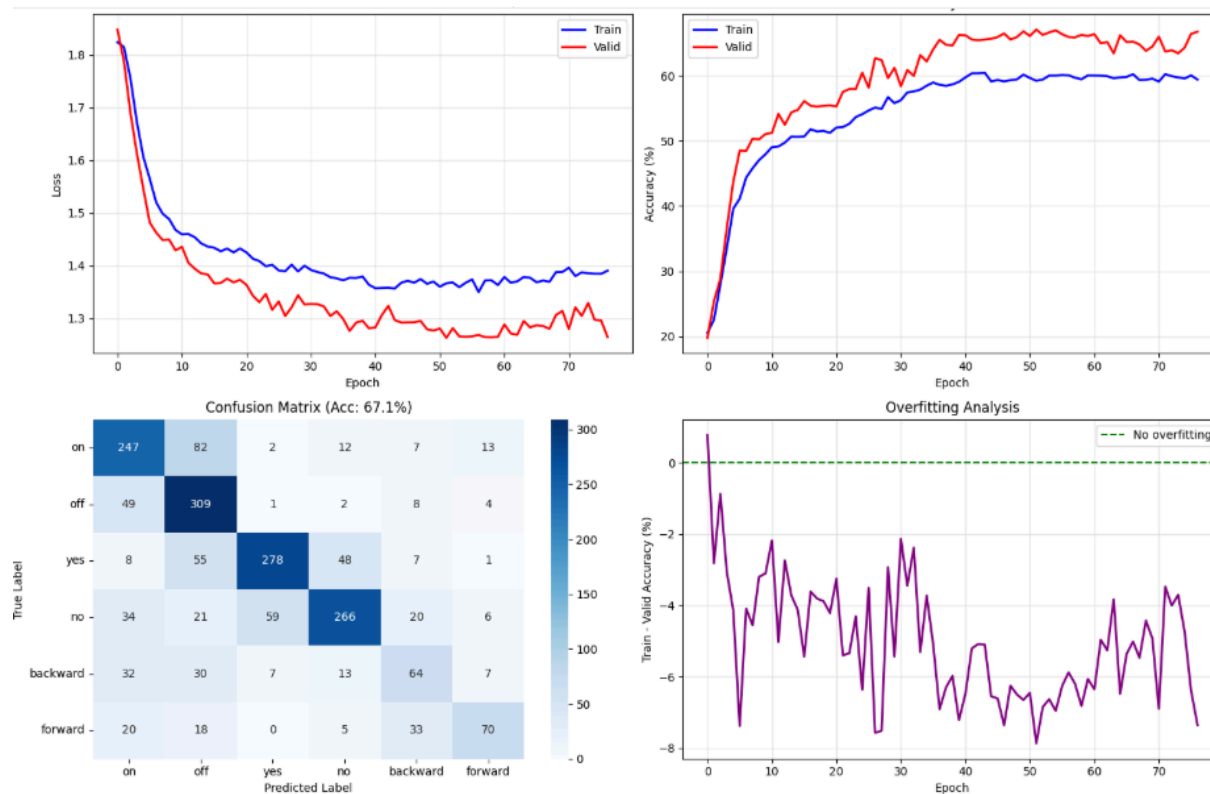
### **OVERALL DESIGN GOAL**

**Purpose:** Build an optimized SNN for 6-keyword speech recognition.

**How it works:**

- Uses a  $700 \rightarrow 256 \rightarrow 128 \rightarrow 6$  architecture.
- Relies on Batch Normalization for stable deep SNN training.

- Uses Parametric LIF neurons for adaptive temporal behavior.
- Avoids Dropout during inference to maximize accuracy.
- Processes speech as spike events over 50 time steps.
- Integrate evidence over time and decide by firing rate.



## Neural Accelerator (Currently To Be Modeled)

### Methodology:

#### Power On → ENROLLMENT MODE:

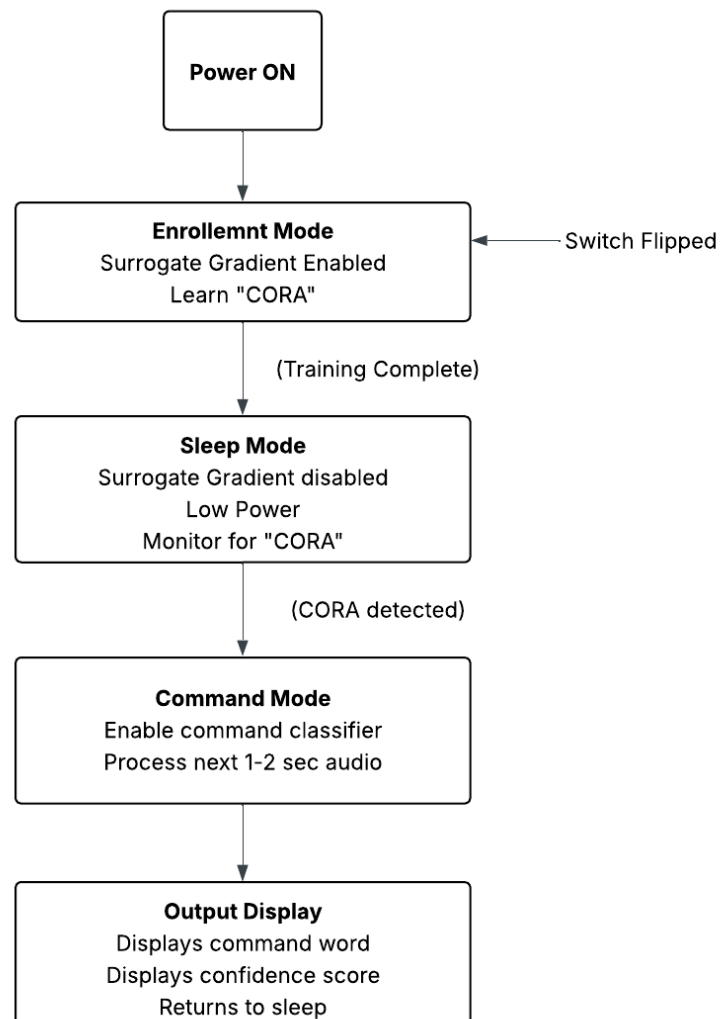
1. User says "Cora" repeatedly
2. Surrogate Gradient active on Wake Word weights
3. Learns user's "Cora"
4. Save learned weights, disable STDP

#### Normal Operation → SLEEP MODE:

1. Monitor audio with learned "Cora" detector
2. Low-power, only wake word path active

#### Wake Word Detected → COMMAND MODE:

1. Enable command classification path
2. Process next audio with command weights
3. Output command recognition
4. Return to sleep mode



## BRAM ORGANIZATION:

### Bank 0: Configuration & Control Registers

0x00: [MODE: 0=Enroll, 1=Sleep, 2=Command]

0x04: [STDP\_EN, WAKE\_THRESHOLD, CMD\_TIMEOUT]

0x08: Neuron Params [ $\tau$ ,  $\theta$ ,  $V_{\text{reset}}$ ,  $\alpha$ ]

0x0C: Surrogate Gradient Surrogate Gradient Params [ $A^+$ ,  $A^-$ , learning\_rate]

### Bank 1: WAKE WORD WEIGHTS (Learned)

Start: 0x100

Size:  $16 \times 128 = 2048$  entries (8-bit each)

Organization: 128 columns  $\times$  16 rows

Surrogate Gradient Surrogate Gradient CAN WRITE to this bank during enrollment

### Bank 2: RECURRENT WEIGHTS (Fixed)

Start: 0x900

Size:  $128 \times 128 = 16384$  entries (8-bit each)

Organization:  $128 \times 128$  matrix

READ-ONLY (never modified)

### Bank 3: COMMAND WEIGHTS (Fixed)

Start: 0x4900

Size:  $128 \times 10 = 1280$  entries (8-bit each)

Organization: 10 columns  $\times$  128 rows

READ-ONLY (never modified)

(Subject to change dependant on the model created)

### Bank 4: NEURON STATES (Dual-port)

Start: 0x4E00

Size: 128 entries  $\times$  16 bits each

Stores:  $V_m$ ,  $I_{\text{syn}}$ , last\_spike\_time

## SURROGATE GRADIENT ENGINE (Simplified)

Only for: Input→Hidden weights ( $16 \times 128$ )

Implementation:

- Forward Pass Buffers:

Membrane Potential Buffer:  $128 \text{ neurons} \times 50 \text{ timesteps}$  (16-bit each)

Input Spike Buffer:  $16 \text{ inputs} \times 50 \text{ timesteps}$  (binary)

Hidden Spike Buffer:  $128 \text{ neurons} \times 50 \text{ timesteps}$  (binary)

- Target Pattern Storage:

Target Pattern Buffer:  $50 \text{ timesteps} \times 128 \text{ bits}$  (what should fire for "Cora")

Example: Neuron 0 should fire at timesteps 20-30 during "Cora" pronunciation

- Surrogate Gradient LUT:

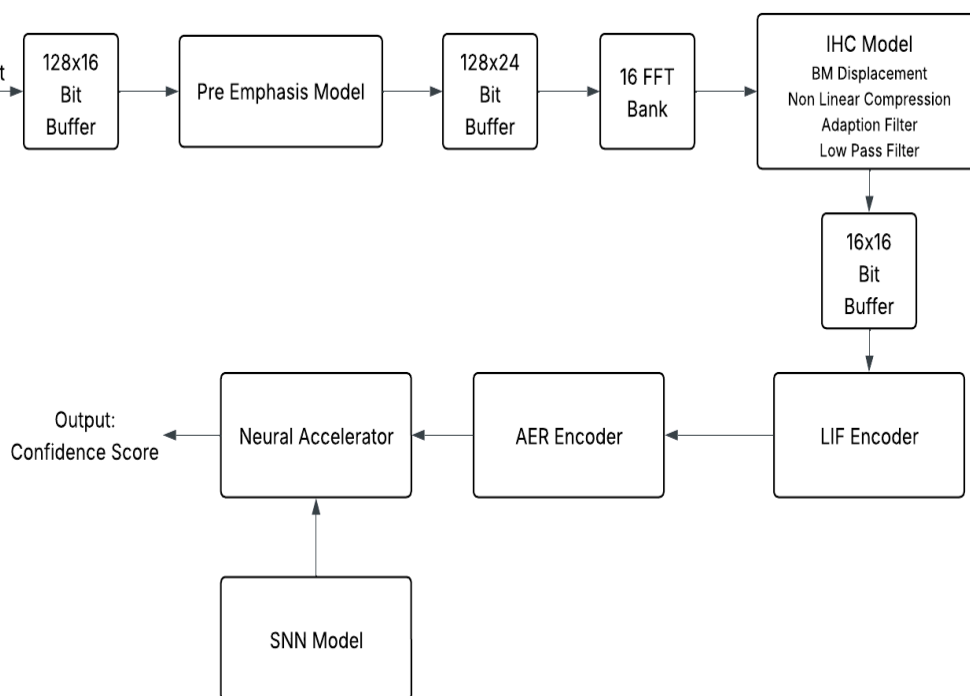
256-entry LUT: Stores  $\sigma'(x) = 1 / (1 + |x|)^2$  for  $x$  in range  $[-128, 127]$

Input:  $(V_m - \theta)$  clamped to 8-bit signed

Output: 8-bit gradient value

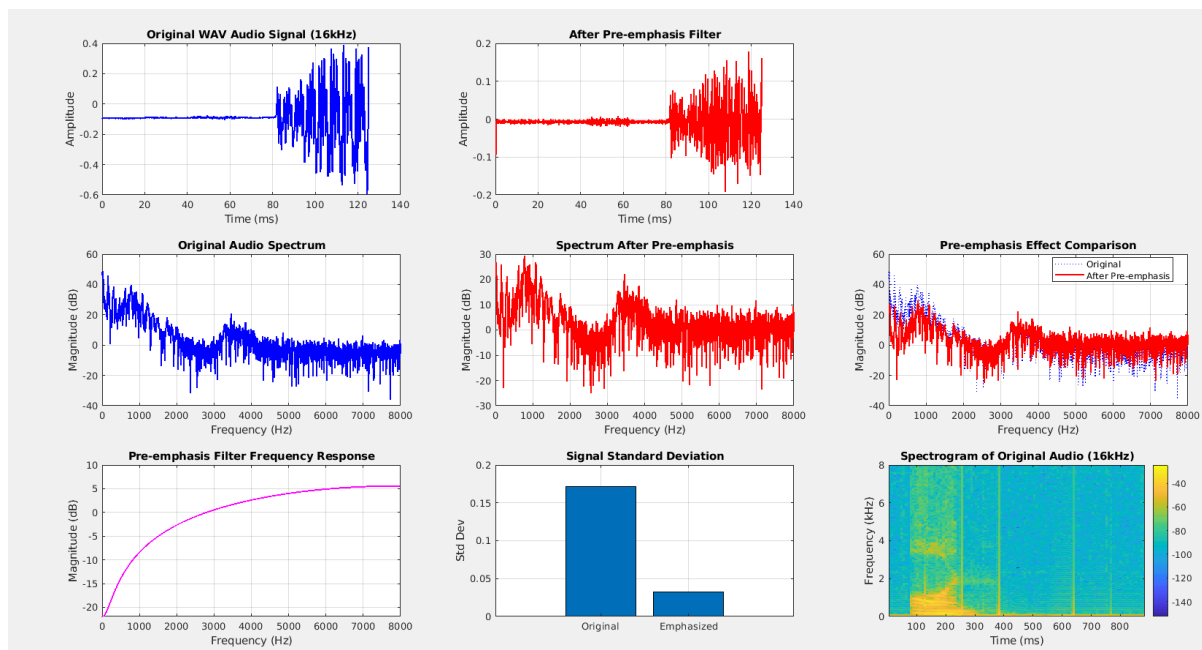
Pre-computed values for hardware efficiency

## 4. System Design

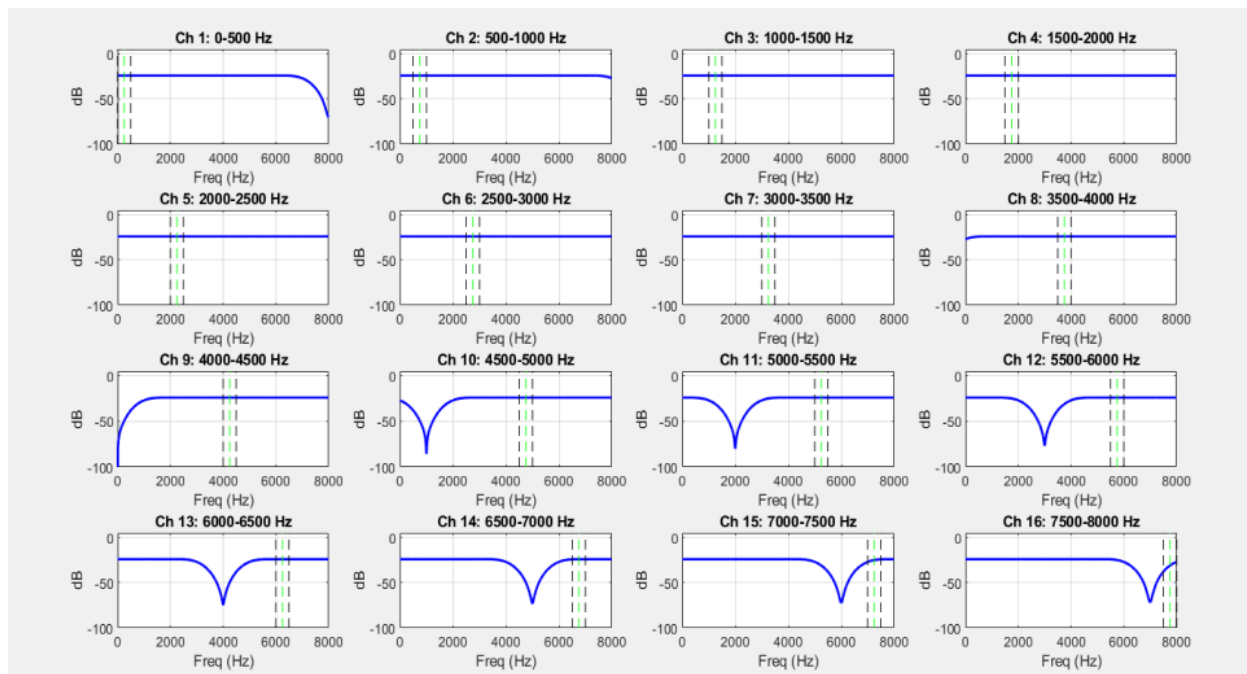


## 5. Results

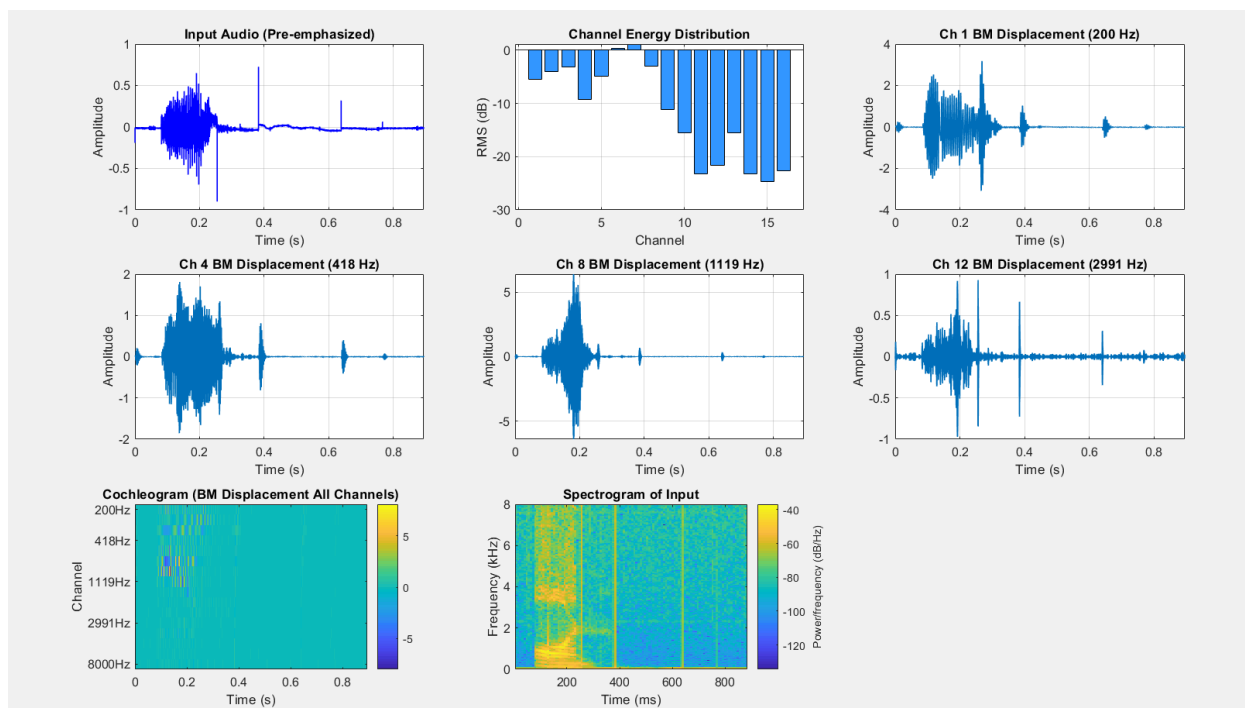
Pre Emphasis Output:



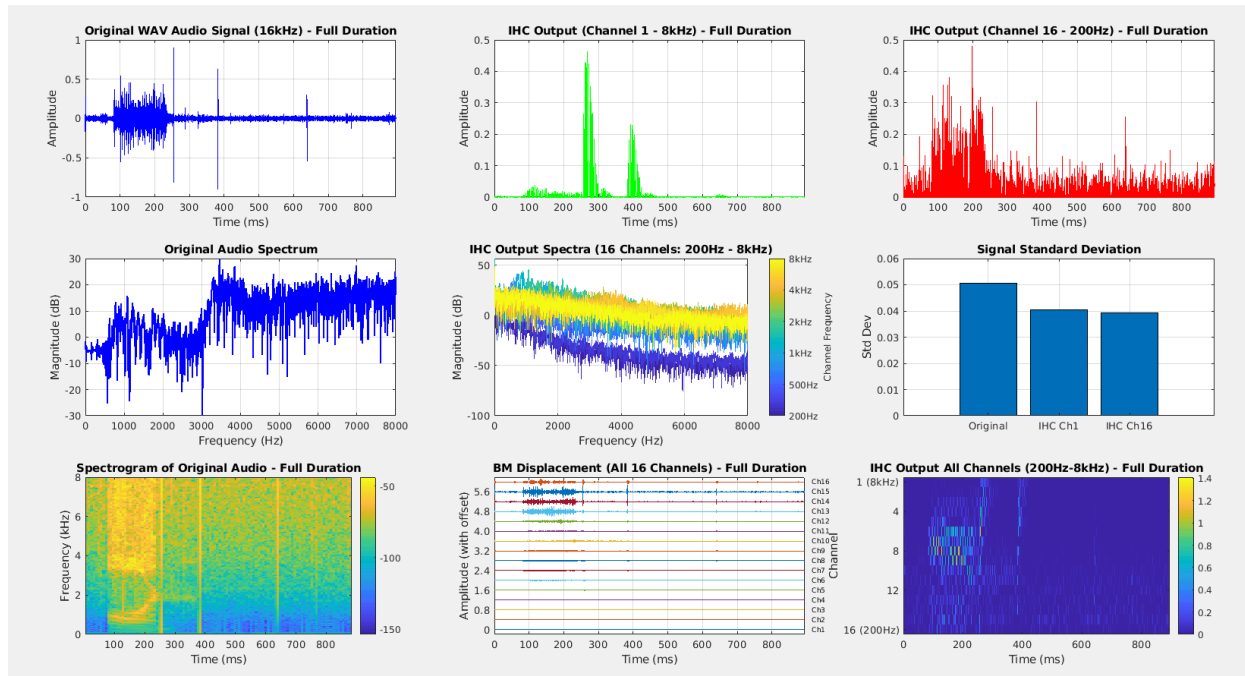
Polyphase FFT Output:



## Gammatone Filter Output:



## IHC Output:



```

Using original sampling rate: 16000 Hz
=== WAV File Processing (16kHz) ===
Sampling Rate: 16000 Hz
Audio length: 0.89 seconds
Number of samples: 14314
Processing audio data...
Applying IHC model processing...
- Simulating BM displacement...
  Channel center frequencies (200Hz - 8kHz):
    Ch 1: 200 Hz
    Ch 5: 535 Hz
    Ch 9: 1430 Hz
    Ch 13: 3825 Hz
    Ch 16: 8000 Hz
  All channel frequencies: 200 256 327 418 535 684 875 1119 1430 1829 2339 2991 3825 4892 6256 8000
- Applying IHC processing chain...

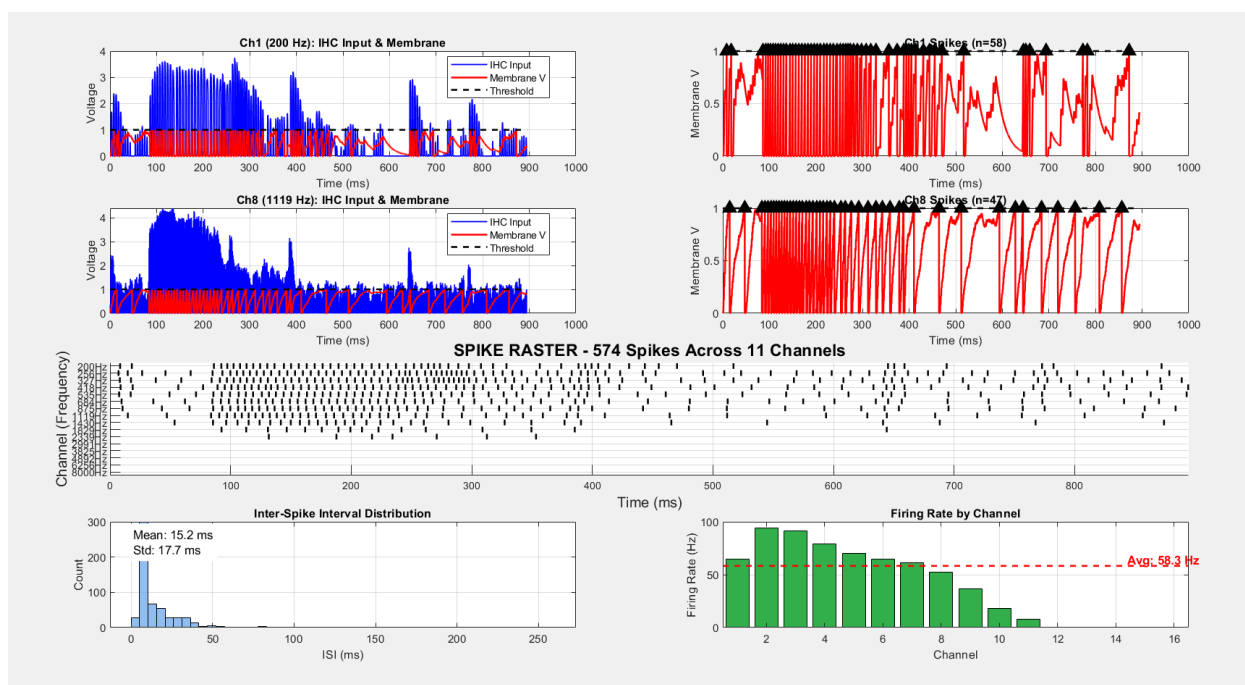
=== Processing Summary ===
Sampling Rate: 16000 Hz
Total samples processed: 14314
Signal duration: 0.895 seconds
IHC channels: 16
IHC output range: [0.0000, 1.4101]

Channel Statistics:
Channel 1: mean=0.0086, std=0.0405
Channel 2: mean=0.0118, std=0.0543
Channel 3: mean=0.0152, std=0.0590
Channel 4: mean=0.0148, std=0.0601
Channel 5: mean=0.0205, std=0.0730
Channel 6: mean=0.0395, std=0.1271
Channel 7: mean=0.0516, std=0.1707
Channel 8: mean=0.0555, std=0.1809
Channel 9: mean=0.0450, std=0.1422
Channel 10: mean=0.0350, std=0.0903
Channel 11: mean=0.0301, std=0.0618
Channel 12: mean=0.0231, std=0.0410
Channel 13: mean=0.0204, std=0.0381
Channel 14: mean=0.0241, std=0.0466
Channel 15: mean=0.0359, std=0.0664
Channel 16: mean=0.0190, std=0.0394

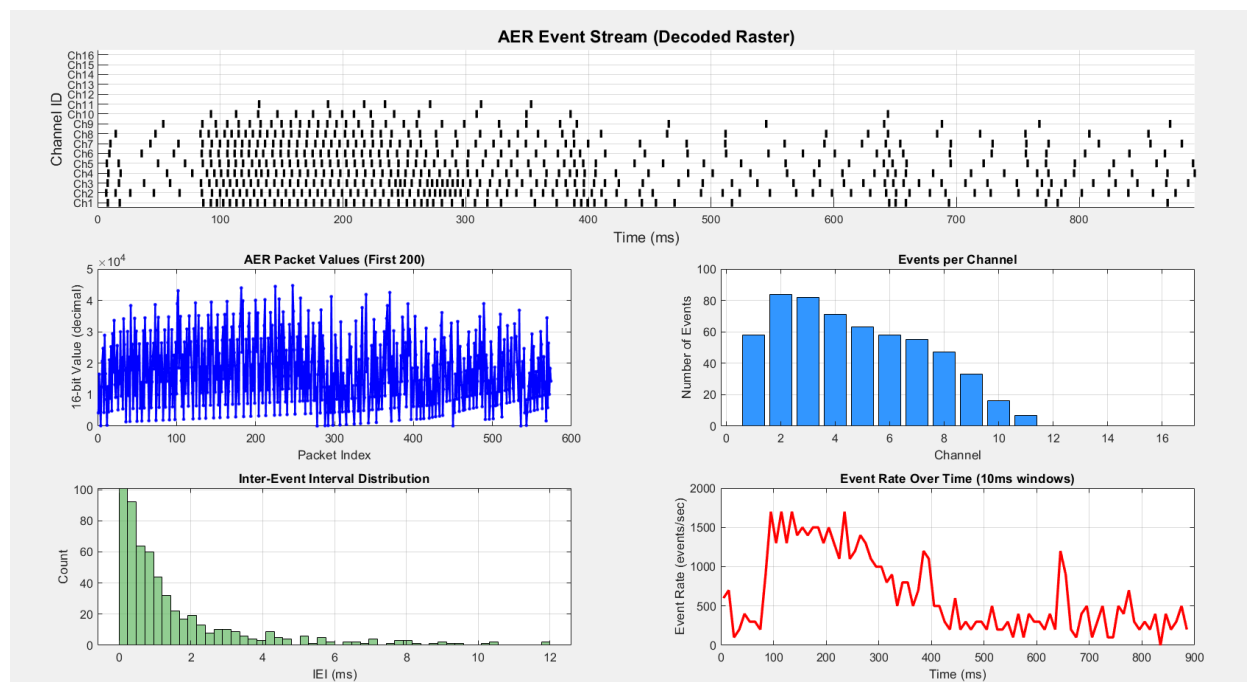
=== Output Saved ===
IHC output (channel 1) saved as: IHC_output_channel1.wav
Normalized range: [0.0000, 0.9000]

```

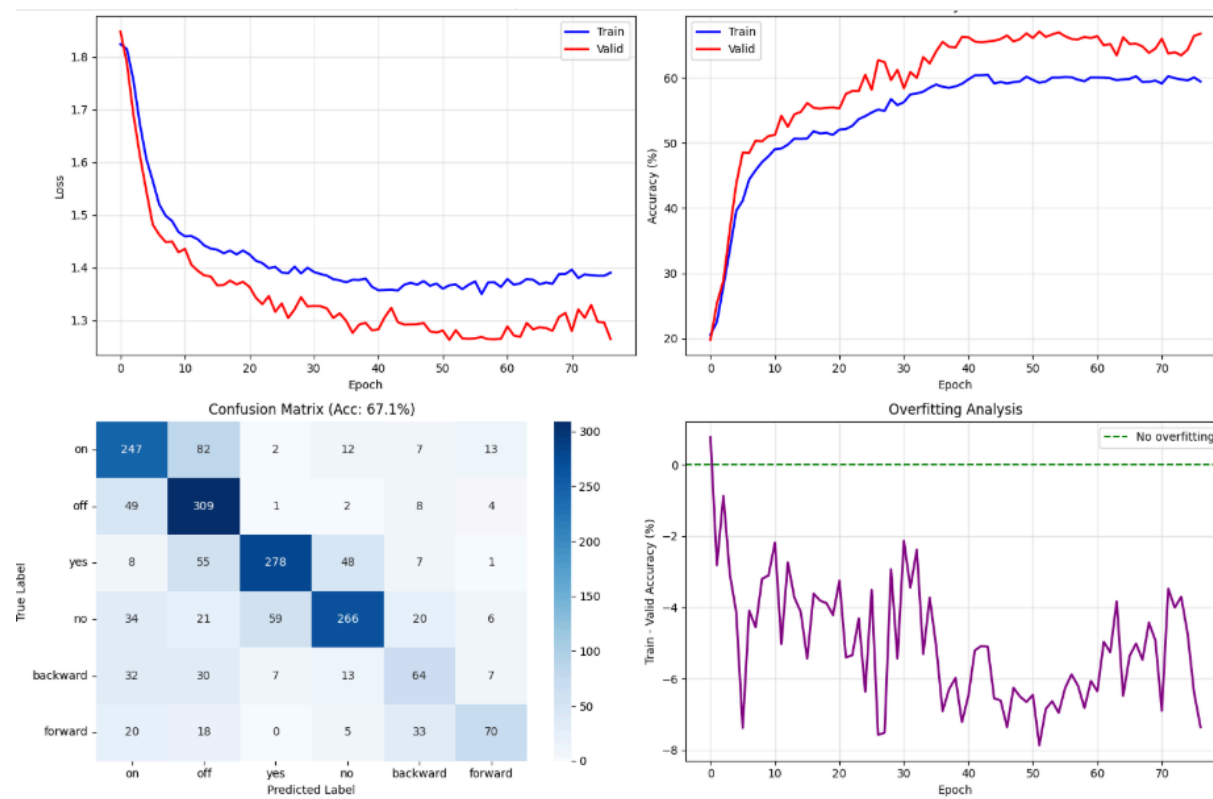
## LIF Neuron Encoding Output:



## AER Packetizer Output:



## SNN Model Observation:



## 6. References

- [1] L. H. Arnaldi, "Implementation of a Polyphase Filter Bank Channelizer on a Zynq FPGA," 2020 Argentine Conference on Electronics (CAE), Buenos Aires, Argentina, 2020, pp. 57-62, doi: 10.1109/CAE48787.2020.9046377.
- [2] M. Garrido, P. Källström, M. Kumm and O. Gustafsson, "CORDIC II: A New Improved CORDIC Algorithm," in IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 63, no. 2, pp. 186-190, Feb. 2016, doi: 10.1109/TCSII.2015.2483422.
- [3] H. Li and Y. Xin, "Modified CORDIC Algorithm and Its Implementation Based on FPGA," 2010 Third International Conference on Intelligent Networks and Intelligent Systems, Shenyang, China, 2010, pp. 618-621, doi: 10.1109/ICINIS.2010.30.
- [4] B. Deng, Y. Fan, J. Wang and S. Yang, "Reconstruction of a Fully Paralleled Auditory Spiking Neural Network and FPGA Implementation," in IEEE Transactions on Biomedical Circuits and Systems, vol. 15, no. 6, pp. 1320-1331, Dec. 2021, doi: 10.1109/TBCAS.2021.3122549.
- [5] F. Ottati et al., "To Spike or Not to Spike: A Digital Hardware Perspective on Deep Learning Acceleration," in IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 13, no. 4, pp. 1015-1025, Dec. 2023, doi: 10.1109/JETCAS.2023.3330432.
- [6] Z. Cao, Z. Zhang, Q. Sun, B. Hou, L. Jiao and Y. Yang, "Cost-Effective and High-Speed Implementation of Brain-Like Spiking Neural Network with Encoding Architectures on FPGA," in CHAIN, vol. 1, no. 2, pp. 167-176, June 2024, doi: 10.23919/CHAIN.2024.000003.
- [7] P. Katti, A. Nimbekar, C. Li, A. Acharyya, B. M. Al-Hashimi and B. Rajendran, "Bayesian Inference Accelerator for Spiking Neural Networks," 2024 IEEE International Symposium on Circuits and Systems (ISCAS), Singapore, Singapore, 2024, pp. 1-5, doi: 10.1109/ISCAS58744.2024.10558608.
- [8] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi, "Spatio-Temporal Backpropagation for Training High-Performance Spiking Neural Networks," in Frontiers in Neuroscience, vol. 12, p. 331, May 2018, doi: 10.3389/fnins.2018.00331.
- [9] E. O. Neftci, H. Mostafa, and F. Zenke, "Surrogate Gradient Learning in Spiking Neural Networks," in IEEE Signal Processing Magazine, vol. 36, no. 6, pp. 61-69, Nov. 2019, doi: 10.1109/MSP.2019.2931595.
- [10] W. Fang, Y. Chen, J. Ding, et al., "SpikingJelly: An open-source machine learning infrastructure for high-performance spiking neural networks," in Science Advances, vol. 9, no. 40, Oct. 2023, doi: 10.1126/sciadv.adi1480