Digital Implementation of Oscillatory Neural Networks on FPGA for Handwritten Digit Recognition

Chinmay Kulkarni, Sandip Lashkare

Abstract

Traditional computing systems based on the von Neumann architecture have a separate memory and processing unit, which causes memory bottleneck issues for fast and energy-efficient computing for AI tasks. Neuromorphic computing offers an effective solution to this problem. Neuromorphic computing is inspired by the functioning of brain neurons, which are known for their energy and time efficiency. Oscillatory neural networks (ONNs) also draw inspiration from this neuromorphic computing approach. ONNs overcome this bottleneck by combining memory and computation through phase-based interactions between coupled oscillators, and information is processed through phase synchronisation of these Oscillators. In this project, we implemented a digital ONN on an FPGA for handwritten digit recognition. The system was trained using images of digits '0' and '1' from the MNIST dataset. We used the Hebbian learning rule to train the synaptic weights offline. This work shows the potential of ONNs for low-power, parallel, and scalable edge AI applications.

INTRODUCTION

In recent years, artificial intelligence (AI) has grown rapidly. especially in areas like image recognition and edge computing. Neural networks are the key part of this process, while for pattern recognition, Artificial neural networks (ANN) are used most often. These neural networks are based on Von Neumann architecture. There are limitations of traditional computing systems based on the von Neumann architecture. These systems keep memory and processing units separate, which means that data must constantly move back and forth between them. This causes delays (high latency) and more energy use. As a result, these systems are not well-suited for tasks that need to be fast, efficient, and run on low-power devices, such as real-time AI on edge devices like embedded systems.

Neuromorphic computing can be a solution to this problem. Neuromorphic computing is a method of computation inspired by the human brain's processing. The human brain is very energy-efficient and fast compared to supercomputers. Oscillatory Neural Networks (ONN) are one of the neuromorphic computing paradigms. Unlike traditional artificial neural networks, ONNs leverage the inherent dynamics of oscillatory systems rather than purely arithmetic operations. In ONN, we use phase synchronisation of the oscillating waves, which have different phases. These oscillators interact through coupling mechanisms, where the strength of interaction is determined by the learned synaptic weights. This allows many computations to happen simultaneously (parallel processing), which helps make the process faster and uses much less power. Such properties make ONNs particularly suitable for edge computing and real-time pattern recognition applications

We implemented a fully digital ONN on an FPGA board. Implementing a fully digital version eliminates the variability issues present in analog designs while offering better scalability and portability. The goal was to recognise handwritten digits using this brain-inspired model. We built a 10×10 ONN, meaning the network had 100 oscillators. The size of the image was chosen as a trade-off between hardware resource constraints and maintaining sufficient feature representation for digit recognition. Each oscillator represents one pixel of a 10×10 grayscale image. Here, we have used **the Hebbian learning**

rule for training synaptic weights. We calculated the coupling weights between neurons off-chip using Python. For training and testing, we used images of the digits '0' and '1' from the MNIST dataset. These images were resized to 10×10 to match the network.

Our system shows that ONNs can be a powerful solution for recognising patterns quickly and efficiently. The design was synthesised and deployed on an Xilinx Nexys 4 DDR FPGA board, achieving real-time inference with minimal latency. This makes them especially useful for AI applications in edge devises.

Design and working of ONN

1. Basics of ONN

Oscillatory Neural Networks (ONNs) are inspired by how the human brain works. They follow the brain's way of processing information in a distributed and parallel manner, which helps in doing complex tasks quickly and efficiently. ONNs use the idea of coupled oscillators that work together in synchronisation. In these networks, each neuron is an oscillator, and its state is shown by a phase value. Using phase instead of amplitude helps reduce the amount of computation needed, making ONNs a good choice for low-power systems. When a network of such oscillators is interconnected via synaptic weights, the system evolves dynamically until it reaches a stable, synchronised state that corresponds to a learned pattern.

In ONNs, information is encoded not in the amplitude but in the phase of oscillations. A logical white pixel is typically represented by an out-of-phase condition (e.g., 180°), and a black pixel by an in-phase condition (e.g., 0°) with respect to a reference oscillator.

2. Weight Training Using Hebbian Learning Rule

The weights in ONNs define the strength of coupling between oscillators (neurons). These weights play a critical role in determining the synchronisation dynamics of the network. For pattern recognition, weights are trained using the **Hebbian learning rule**, which helps to achieve associative memory behaviour. This rule strengthens the connection between neurons that exhibit similar activity, mimicking the learning principle of biological neural systems in brain.

Let $\sigma_i^p \in \{-1, +1\}$ be the value of the i-th neuron in the p-th pattern. Then the synaptic weight between neuron i and neuron j is calculated as:

$$W_{ij} = \sum_p \sigma_i^p \times \sigma_j^p \text{ for } i \neq j; \text{ and } W_{ii} = 0$$

Where:
- P = number of training patterns
- W_{ij} = weight (coupling strength) between neuron i and j

In this formulation, self-connections are explicitly set to zero to avoid feedback loops that could destabilise the network dynamics
This results in a symmetric weight matrix that stores multiple patterns.

This results in a symmetric weight matrix that stores multiple patterns. Such symmetry ensures that the energy function of the network converges to stable states corresponding to stored patterns.

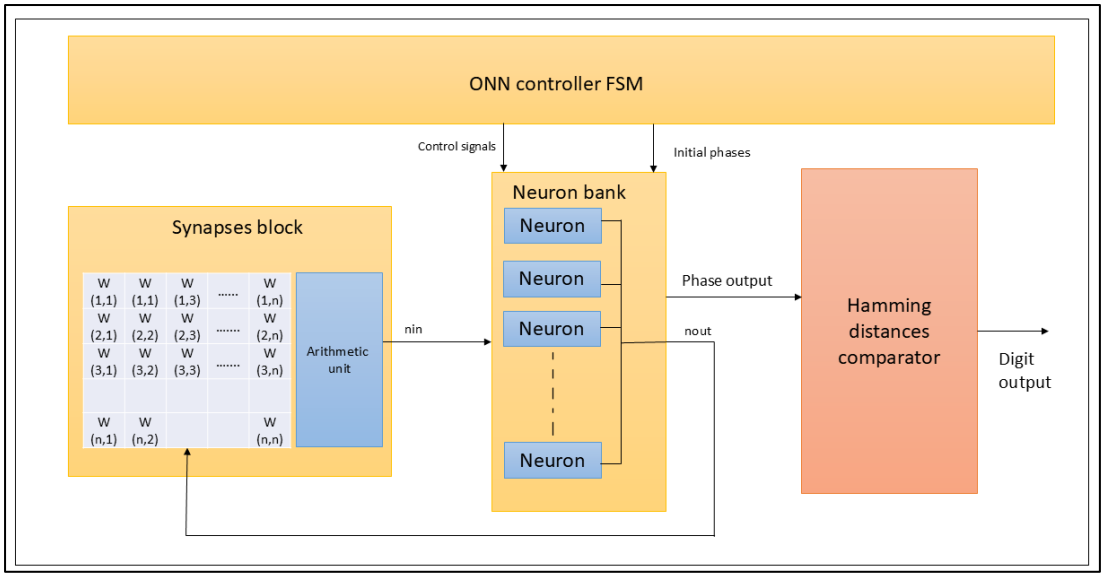
3. Architecture of Digital ONN

The Architecture of digital ONN is inspired by a research paper [5], as shown the Figure 1 below

The architecture of a Digital Oscillatory Neural Network (ONN) implemented on an FPGA consists of several functional blocks that operate in parallel:

- Neuron Block (Oscillator): Each neuron is implemented as a digital oscillator, using a circular shift register and a mux which represents 16 phases (from 0° to 180°). These phases are updated based on input signals from the synapse. Neurons typically operate at uniform frequency and update their phase based on weighted inputs from connected neurons.
- Synapse Block: This block stores the synaptic weights, which define the coupling strength between pairs of neurons. The weights are typically stored in memory as signed values (e.g., 5-bit registers). During operation, the neuron phases are multiplied by these weights to compute the net input for each neuron. This is typically performed in a parallel manner to maintain high speed. The summation unit computes the total weighted input by accumulating the contributions from all other neurons based on their current output and the respective synaptic weights.
- Phase calculator:- This block calculates the phase of the input wave based on the input and output waves. This phase is then used as the new phase for the neuron.
- Control Unit: This block orchestrates the operation of the ONN. It handles initialisation (e.g., loading weights and initial phases), controls the number of iteration cycles, checks for convergence, and manages the reset and input loading logic.
- hamming distance comparator : The interface handles digit input (e.g., a 10x10 grayscale image converted to phase input) and extracts the final phase configuration after convergence to perform pattern matching.

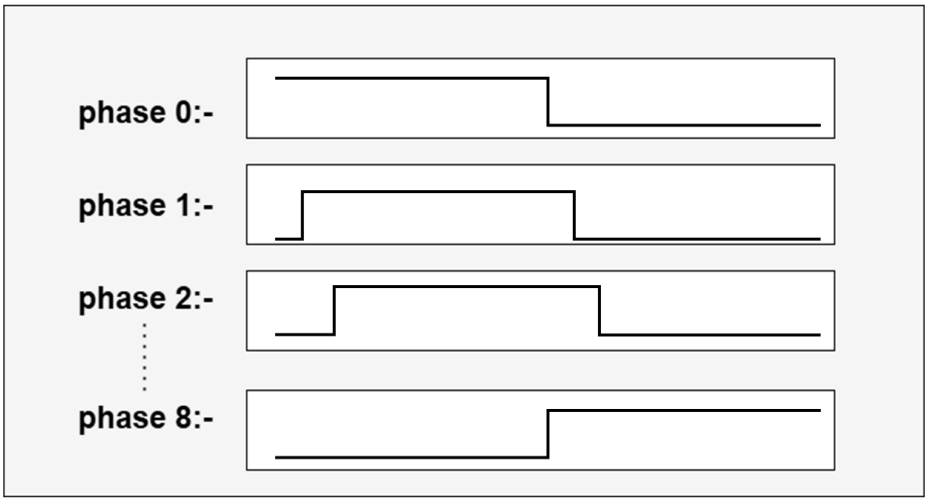
This entire architecture is designed to exploit the parallelism and reconfigurability of FPGAs, enabling fast and energy-efficient inference. Multiple neurons and synapses operate simultaneously, and the system settles into a stable phase state that may correspond to a trained pattern



4. Neuron Phase Behaviour

The phase-controlled oscillator generates a square wave signal with a specific phase, allowing each ONN neuron to have its own wave with a particular phase digitally. It mainly consists of a **16-bit circular shift register** and a **multiplexer.**

Here we use a circular right-shift register with a fixed binary pattern like `[0000000011111111]`, which shifts continuously to produce a square waveform of 16 bits. To select a particular phase, a multiplexer picks one of the 16 outputs from the shift register, based on a phase stored in a phase register. This design enables fast phase switching without requiring complex arithmetic operations and without changing the stored pattern in the register. When a neuron updates its phase, this control register is incremented or decremented accordingly. The selected shift register output is then connected to the neuron's output, which is connected to a synapse.



5. Digit Recognition using ONN

During inference:

1. A test digit pattern is applied to the network as an initial phase to each neuron based on the grey scale value of each pixel.
2. The network generates an initial wave and synchronises based on stored weights in synapses after some repetitions.
3. After stabilisation, the final phase states are compared with all stored base patterns of images using **Hamming distance**:

$$\text{Hamming Distance} = \sum |\sigma_i(\text{test}) - \sigma_i(\text{stored})|$$

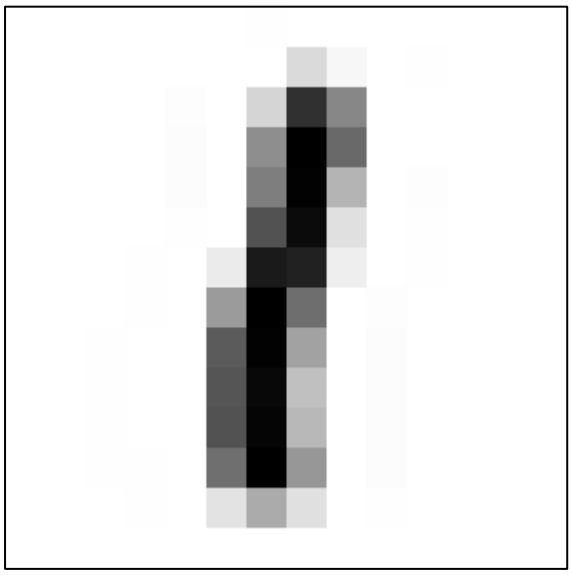
4. The pattern with the minimum Hamming distance is chosen as the recognised digit.

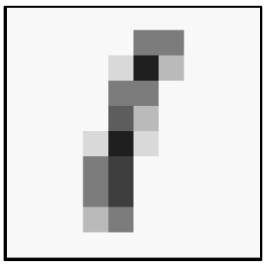
Results and Learnings

1. Image Resizing for ONN Compatibility

Here, we have used the MNIST dataset for training and testing of our ONN model. This dataset is widely used for testing and training digit recognition systems due to its diversity and real-world handwriting

variability. The MNIST dataset provides handwritten digit images in a standard 28×28 grayscale format. While suitable for large-scale neural networks, this resolution significantly exceeds the hardware capacity of our target platform the Nexys 4 DDR FPGA board, particularly for Oscillatory Neural Networks (ONNs), where each pixel corresponds to a neuron. Unlike convolutional networks that can leverage weight sharing, ONNs require a fully connected structure, resulting in quadratic growth of synaptic connections with network size. A direct 28×28 implementation would require 784 oscillators and a prohibitively large number of synaptic connections (over 300,000), leading to excessive memory and logic usage.





To overcome this, we downscaled the images to 10×10 resolution, reducing the ONN to a manageable 100 neurons with a 100×100 synapse matrix. This downscaling allows the entire ONN, including neurons, synapses, and control logic to fit within the available resources of the FPGA. This reduction maintains a balance between hardware feasibility and adequate feature representation for digit classification

2. Constraining Image Variability for Robust Training

One of the biggest challenges during training was the significant variation in how the same digit appeared—its shape and position often changed due to differences in individual handwriting. This includes variations like tilt, size, or off-centre placement. Convolutional Neural Networks (CNNs) handle this well because they learn patterns across different styles and locations. On the other hand, for Oscillatory Neural Networks (ONNs) using Hebbian learning, this kind of variation becomes a problem.

ONNs cannot generalise features like CNNs. Instead, they rely on fixed pixel-to-phase relationships. That means the pixel contributions get mixed up when the same digit appears in different positions or shapes. This weakens the learned weights and makes it hard for the network to recognise patterns consistently. The result is poor convergence and unreliable output.

To fix this, we added strict rules for image preprocessing so that all training samples look more consistent. The idea is to reduce differences within the same class of digits—something that's very important for memory-based learning models like ONNs.

For example, when working with the digit '1', we picked only those images where the digit is centered and upright, removing slanted or shifted versions. This helps preserve the strong vertical line that defines the digit '1'.

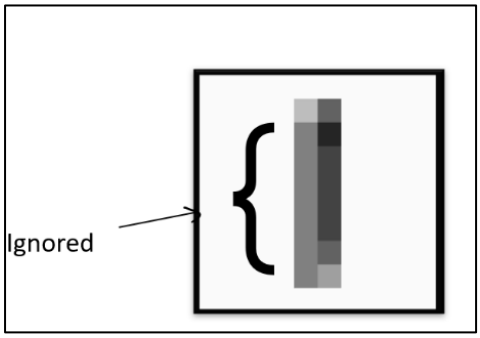
For digit' 0', we have chosen the images to be approximately circular and of similar size to avoid elliptical or distorted shapes. These preprocessing rules ensure that the most important pixel patterns are repeatedly learned during training, leading to a cleaner and more reliable weight matrix.

3. Adjusting the Threshold for Better Binarisation

In Hebbian learning, grayscale images need to be binarised by applying a threshold, turning pixel values into either +1 or -1. Typically, a threshold of 128 is used to separate dark and light areas. But when images are resized, The overall brightness of pixel values often decreases. This dimming effect occurs because interpolation during downscaling spreads pixel intensity across fewer pixels, leading to a reduction in contrast.. As a result, many pixels that should be part of the digit fall below the threshold and are wrongly classified as -1. Such misinterpretation leads to incorrect weight updates and poor learning.

To fix this issue, we raised the threshold value to 220. This means only the darkest and most important strokes in the digit are marked as active (+1). This adjustment makes the binarisation better match the actual shape of the digit. Although using a high threshold could make the system more sensitive to noise, we found that the impact was very small.

In fact, this change clearly improved the learning process. It helped preserve the key parts of the digit while filtering out background noise. As a result, the weight matrix became more stable and reliable, leading to better performance when testing the network with new images.



4. Weight Training Strategy for Better Convergence

Our early attempts to train the ONN using several different samples of the same digit led to unstable results during testing. This instability arises because ONNs lack a feature-extraction hierarchy like CNNs, making them highly sensitive to the variability of the input image used for training. Because Hebbian learning adds weight updates from each sample, training with varied examples caused the weights to spread out across many different pixel combinations. This made it hard for the network to form a strong memory of any one specific pattern. As a result, during recall, the ONN struggled to converge properly and often misclassified digits when comparing Hamming distances.

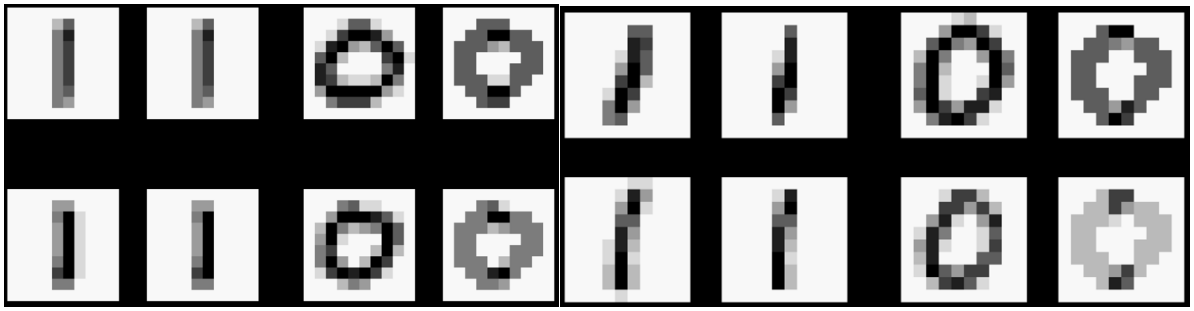
To address this, we used a more focused training method. Instead of feeding in many different samples, we picked one clear, well-shaped image for each digit (see Fig. 4) and repeated the Hebbian update process 15 times using just that image. This repetition helped reinforce the important pixel-to-pixel relationships in the chosen pattern, making the network strongly converge toward that specific

memory. This repetitive reinforcement is analogous to over-sampling a prototype pattern to strengthen its representation within the network.

With this approach, the ONN performed much better during testing, even when the input image was noisy or partially incomplete. After synchronisation of ONN, the pattern we get as output is more generalised for a particular digit, which helps to match it to the closest stored pattern based on Hamming distance. This strategy greatly improved both the stability and accuracy of digit recognition.

5. Convergence and Classification Accuracy

In most cases, the ONN output converges closely to the base image used for training, resulting in correct classification through Hamming distance. Even when full convergence isn't visually achieved—due to noise or slight variation—the network still outputs the correct digit. This shows that the system is robust and can reliably classify inputs based on the nearest stored pattern, even with partial phase mismatches



output images from ONN

Refrence

1. Jackson, S. Pagliarini, and L. Pileggi, "An Oscillatory Neural Network with Programmable Resistive Synapses in 28 nm CMOS," in *2018 IEEE International Conference on Rebooting Computing (ICRC)*, Nov. 2018, pp. 1--7.

2. E. Luhulima, M. Abernot, F. Corradi, and A. Todri-Sanial, "Digital Implementation of On-Chip Hebbian Learning for Oscillatory Neural Network," in *2023 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2023, pp. 1- 6.

3. C. Delacour, S. Carapezzi, M. Abernot, G. Boschetto, N. Azemard, J. Salles, T. Gil, and A. Todri-Sanial, "Oscillatory Neural Networks for Edge AI Computing," in *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, July 2021, pp. 326--331.

4. F. C. Hoppensteadt and E. M. Izhikevich, "Pattern recognition via synchronization in phase-locked loop neural networks," *IEEE Transl. J. Magn. Japan*, vol. 11, no. 3, pp. 734--738, May 2000.

5. M. Abernot, T. Gil, M. Jiménez, J. Núñez, M. J. Avellido, B. Linares-Barranco, T. Gonos, T. Hardelin, and A. Todri-Sanial, "Digital Implementation of Oscillatory Neural Network for Image Recognition Applications," *Front. Neurosci.*, vol. 15, Art. ID 713054, Aug. 2021.

6. B. Haverkort and A. Todri-Sanial, "Overcoming Quadratic Hardware Scaling for a Fully Connected Digital Oscillatory Neural Network," unpublished, 2024.